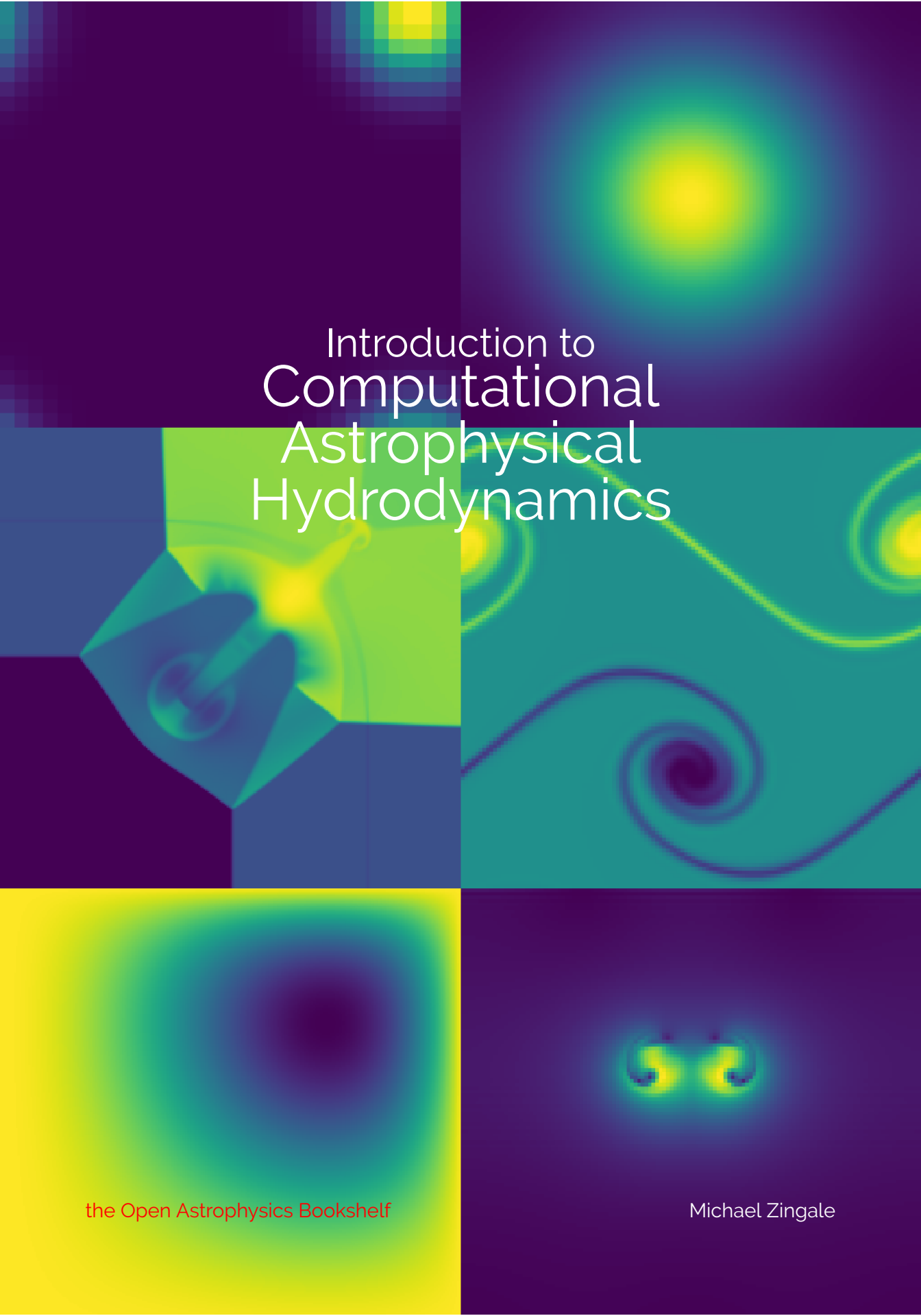
Introduction to
Computational
Astrophysical
Hydrodynamics
the Open Astrophysics Bookshelf
Michael Zingale

document git version: 9d511eeb0172 . . .
January 31, 2023

the source for these notes are available online (via `git`):
`https://github.com/Open-Astrophysics-Bookshelf/numerical_exercises`

Chapter Listing

Table of Contents

List of Figures

List of Exercises

preface

This text started as a set of notes to help new students at Stony Brook University working on projects in computational astrophysics. They focus on the common methods used in computational hydrodynamics for astrophysical flows and are written at a level appropriate for upper-level undergraduates. Problems integrated in the text help demonstrate the core ideas. An underlying principle is that source code is provided for all the methods described here (including all the figures). This allows the reader to explore the routines themselves.

These notes are very much a work in progress, and new chapters will be added with time. The page size is formatted for easy reading on a tablet or for 2-up printing in a landscape orientation on letter-sized paper.

This text is part of the Open Astrophysics Bookshelf. Contributions to these notes are welcomed. The LaTeX source for these notes is available online on github at:

`https://github.com/Open-Astrophysics-Bookshelf/numerical_exercises`

Simply fork the notes, hack away, and submit a pull-request to add your contributions. All contributions will be acknowledged in the text.

A PDF version of the notes is always available at:

`http://bender.astro.sunysb.edu/hydro_by_example/CompHydroTutorial.pdf`

These notes are updated at irregular intervals, usually when I have a new student working with me, or if I am using them for a course.

The source (usually python) for all the figures is also contained in the main git repo. The line drawings of the grids are done using the classes in `grid_plot.py`. This needs to be in your `PYTHONPATH` if you wish to run the scripts.

The best way to understand the methods described here is to run them for yourself. There are several sets of example codes that go along with these notes:

1. hydro_examples is a set of simple 1-d, standalone python scripts that illustrate some of the basic solvers. Many of the figures in these notes were created using

these codes—the relevant script will be noted in the figure caption.

You can get this set of scripts from github at:
`https://github.com/zingale/hydro_examples/`

References to the scripts in hydro_examples are shown throughout the text as:

hydro_examples: `scriptname`

Clicking on the name of the script will bring up the source code to the script (on github) in your web browser.

More details on the codes available in hydro_examples are described in Appendix A.

2. The pyro code [86] is a 2-d simulation code with solvers for advection, diffusion, compressible and incompressible hydrodynamics, as well as multigrid. A gray flux-limited diffusion radiation hydrodynamics solver is in development. pyro is designed with clarity in mind and to make experimentation easy.

   You can download pyro at:
   `https://github.com/python-hydro/pyro2/`

   A brief overview of pyro is given in Appendix B, and more information can be found at:
   `http://python-hydro.github.io/pyro2/`

3. hydro1d is a simple one-dimensional compressible hydrodynamics code that implements the piecewise parabolic method from Chapter 8. It can be obtained from github at:
   `https://github.com/zingale/hydro1d/`

   Details on it are given in Appendix C.

Wherever possible we try to use standardized notation for physical quantities, as listed in Table 1.

These notes benefited *immensely* from numerous conversations and an ongoing collaboration with Ann Almgren, John Bell, Andy Nonaka, & Weiqun Zhang—pretty much everything I know about projection methods comes from working with them. Discussions with Alan Calder, Sean Couch, Max Katz, Chris Malone, and Doug Swesty have also been influential in the presentation of these notes.

If you find errors, please e-mail me at michael.zingale@stonybrook.edu, or issue a pull request to the git repo noted above.

Michael Zingale
Stony Brook University

Table 1: Definition of symbols.

symbol	meaning	units
$\mathbf{A}$	Jacobian matrix	N/A
C	Lagrangian sound speed, $C = \sqrt{\Gamma_1 p \rho}$	g m^{-2} s^{-1}
$\mathcal{C}$	CFL number	–
c	sound speed, $c = \sqrt{\Gamma_1 p/\rho}$	m s^{-1}
c_p	specific heat at constant pressure ($c_p \equiv \partial h/\partial T\|_{p,X_k}$)	erg g^{-1} K^{-1}
c_v	specific heat at constant density ($c_v \equiv \partial e/\partial T\|_{\rho,X_k}$)	erg g^{-1} K^{-1}
E	specific total energy	erg g^{-1}
e	specific internal energy	erg g^{-1}
$\mathbf{F}$	flux vector	N/A
$\mathbf{g}$	gravitational acceleration	cm s^{-2}
Γ_1	first adiabatic exponent ($\Gamma_1 \equiv d\log p/d\log\rho\|_s$)	–
γ	ratio of specific heats, $\gamma = c_p/c_v$	–
γ_e	the quantity $p/(\rho e) + 1$	–
H	heat sources	erg g^{-1} s^{-1}
$H_{\rm nuc}$	nuclear energy source	erg g^{-1} s^{-1}
h	specific enthalpy	erg g^{-1}
$\mathcal{I}$	integral under a (piecewise) parabolic polynomial reconstruction	N/A
$k_{\rm th}$	thermal conductivity	erg cm^{-1} s^{-1} K^{-}1
$\mathbf{L}$	matrix of left eigenvectors	–
$\mathbf{\Lambda}$	diagonal matrix of eigenvalue	–
$\mathbf{l}$	left eigenvector	N/A
λ	eigenvalue	N/A
M	Mach number, $M = \|\mathbf{U}\|/c$	–

continued on next page

Table 1—continued

symbol	meaning	units
$\dot{\omega}_k$	species creation rate	s^{-1}
p	pressure	erg cm^{-3}
$\mathbf{q}$	primitive variable vector	N/A
$\mathbf{R}$	matrix of right eigenvectors	–
$\mathcal{R}(q_L, q_R)$	Riemann problem between states q_L and q_R	N/A
$\mathbf{r}$	right eigenvector	N/A
ρ	mass density	g cm^{-3}
S	source term to the divergence constraint	s^{-1}
s	specific entropy	erg g^{-1} K^{-1}
T	temperature	K
t	time	s
τ	specific volume ($\tau = 1/\rho$)	cm^3 g^{-1}
$\mathbf{U}$	total velocity vector, $\mathbf{U} = (u, v)^\intercal$ in 2-d	cm s^{-1}
$\mathcal{U}$	conserved variable vector	N/A
u	x-velocity	cm s^{-1}
v	y-velocity	cm s^{-1}
$\mathbf{w}$	characteristic variables vector	N/A
X_k	mass fraction of the species ($\sum_k X_k = 1$)	–

Authorship

Primary Author

Michael Zingale (Stony Brook)

Contributions

Thank you to the following people for pointing out typos or confusing remarks in the text:

- Chen-Hung
- Rixin Li (Arizona)
- Zhi Li (Shanghai Astronomical Observatory)
- Chris Malone
- Sai Praneeth (Waterloo)
- Donald Willcox (Stony Brook)

Material on WENO schemes was contributed by Ian Hawke (Southampton).

See the git log for full details on contributions. All contributions via pull-requests will be acknowledged here.

Part I

Basics

Chapter 1

Simulation Overview

1.1 What is simulation?

Astronomy is an observational science. Unlike in terrestrial physics, we do not have the luxury of being able to build a model system and do physical experimentation on it to understand the core physics. We have to take what nature gives us. Simulation enables us to build a model of a system and allows us to do virtual experiments to understand how this system reacts to a range of conditions and assumptions.

It's tempting to think that one can download a simulation code, set a few parameters, maybe edit some initial conditions, run, and then have a virtual realization of some astrophysical system that you are interested in. Just like that. In practice, it is not this simple. All simulation codes make approximations—these start even before one turns to the computer, simply by making a choice of what equations are to be solved. The main approximation that we will follow here, is the *fluid approximation* (see Figure 1.1). We don't want to focus on the motions of the individual atoms, nuclei, electrons, and photons in our system, so we work on a scale that is much larger than the mean free path of the system. This allows us to describe the bulk properties of a fluid element, which in turn is small compared to the system of interest.

Within the fluid approximation, additional approximations are made, both in terms of the physics included and how we represent a continuous fluid in the finite-memory of a computer (the *discretization* process).

Typically, we have a system of PDEs, and we need to convert the continuous functional form of our system into a discrete form that can be represented in the finite memory of a computer. This introduces yet more approximation.

Blindly trusting the numbers that come out of the code is a recipe for disaster. You don't stop being a physicist the moment you execute the code—your job as a com-

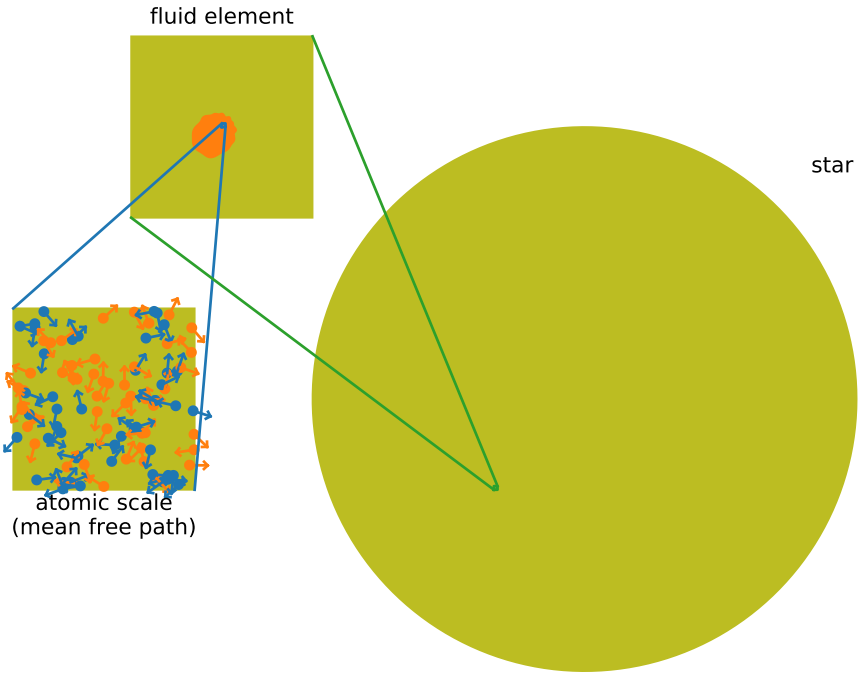

Figure 1.1: The fluid scale sits in an intermediate range—much smaller than the system of interest (a star in this case), but much larger than the mean free path.

putational scientist is to make sure that the code is producing reasonable results, by testing it against known problems and your physical intuition.

Simulations should be used to gain insight and provide a physical understanding. Because the systems we solve are so nonlinear, small changes in the code or the programming environment (compilers, optimization, etc.) can produce large differences in the numbers coming out of the code. That's not a reason to panic. As such it is best not to obsess about precise numbers, but rather the trends our simulations reveal. To really understand the limits of your simulations, you should do parameter and convergence studies.

There is no "über-code". Every algorithm begins with approximations and has limitations. Comparisons between different codes are important and common in our field (see, for example, [30, 31, 33]), and build confidence in the results that we are on the right track.

To really understand your simulations, you need to know what the code your are using is doing under the hood. This means understanding the core methods used in our field. These notes are designed to provide a basic tour of some of the more popular methods, referring to the key papers for full derivations and details. The best

way to learn is to code up these methods for yourself. A companion python code, pyro is available to help, and most of the exercises (or corresponding figures) have links to simple codes that are part of the hydro_examples repository*. Descriptions and links to these codes are found in the appendices.

1.2 Numerical basics

We assume a familiarity with basic numerical methods, which we summarize below. Any book on numerical methods can provide a deeper discussion of these methods. Some good choices are the texts by Garcia [35], Newman [56], and Pang [60].

1.2.1 Sources of error

With any algorithm, there are two sources of error we are concerned with: *roundoff error* and *truncation error*.

Roundoff arises from the error inherent in representing a floating point number with a finite number of bits in the computer memory. An excellent introduction to the details of how computers represent numbers is provided in [38].

Exercise 1.1

In your choice of programming language, create a floating point variable and initialize it to 0.1. Now, print it out in full precision (you may need to use format specifiers in your language to get all significant digits the computer tracks).
You should see that it is not exactly 0.1 to the computer—this is the floating point error. The number 0.1 is not exactly representable in the binary format used for floating point.

Exercise 1.2

To see roundoff error in action, write a program to find the value of ϵ for which $1 + \epsilon = 1$. Start with $\epsilon = 1$ and iterate, halving ϵ each iteration until $1 + \epsilon = 1$. This last value of ϵ for which this was not true is the machine epsilon. *You will get a different value for single- vs. double-precision floating point arithmetic.*

Some reorganization of algorithms can help minimize roundoff, e.g. avoiding the subtraction of two very large numbers by factoring as:

$$x^3 - y^3 = (x - y)(x^2 + xy + y^2) \ , \tag{1.1}$$

but roundoff error will always be present at some level.

*look for the ⌨ symbol.

Truncation error is a feature of an algorithm—we typically approximate an operator or function by expanding about some small quantity. When we throw away higher-order terms, we are truncating our expression, and introducing an error in the representation. If the quantity we expand about truly is small, then the error is small. A simple example is to consider the Taylor series representation of $\sin(x)$:

$$\sin(x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} \tag{1.2}$$

For $|x| \ll 1$, we can approximate this as:

$$\sin(x) \approx x - \frac{x^3}{6} \tag{1.3}$$

in this case, our truncation error has the leading term $\propto x^5$, and we say that our approximation is $\mathcal{O}(x^5)$, or 5^{th}-order accurate.

Exercise 1.3

We will be concerned with the order-of-accuracy of our methods, and a good way to test whether our method is behaving properly is to perform a convergence test. Consider our 5^{th}-order accurate approximation to $\sin(x)$ above. Pick a range of x's (< 1), and compute the error in our approximation as

$$\epsilon \equiv \sin(x) - [x - x^3/6]$$

and show that as you cut x in half, $|\epsilon|$ reduces by 2^5—demonstrating 5^{th}-order accuracy.

This demonstration of measuring the error as we vary the size of our small parameter is an example of a *convergence test*.

1.2.2 Differentiation and integration

For both differentiation and integration, there are two cases we might encounter:

1. We have function values, f_0, f_1, ..., at a discrete number of points, x_0, x_1, ..., and we want to compute the derivative at a point or integration over a range of points.
2. We know a function analytically and we want to construct a derivative or integral of this function numerically.

In these notes, we will mainly be concerned with the first case.

Differentiation of discretely-sampled function

Consider a collection of equally spaced points, labeled with an index i, with the physical spacing between them denoted Δx. We can express the first derivative of a

quantity a at i as:

$$\left.\frac{\partial a}{\partial x}\right|_i \approx \frac{a_i - a_{i-1}}{\Delta x} \tag{1.4}$$

or

$$\left.\frac{\partial a}{\partial x}\right|_i \approx \frac{a_{i+1} - a_i}{\Delta x} \tag{1.5}$$

(Indeed, as $\Delta x \to 0$, this is the definition of a derivative we learned in calculus.) Both of these are *one-sided differences*. By Taylor expanding the data about x_i, we see

$$a_{i+1} = a_i + \Delta x \left.\frac{\partial a}{\partial x}\right|_i + \frac{1}{2}\Delta x^2 \left.\frac{\partial^2 a}{\partial x^2}\right|_i + \ldots \tag{1.6}$$

Solving for $\partial a/\partial x|_i$, we see

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_i - a_{i-1}}{\Delta x} - \frac{1}{2}\Delta x \left.\frac{\partial^2 a}{\partial x^2}\right|_i + \ldots \tag{1.7}$$

$$= \frac{a_i - a_{i-1}}{\Delta x} + \mathcal{O}(\Delta x) \tag{1.8}$$

where $\mathcal{O}(\Delta x)$ indicates that the leading term in the error for this approximation is $\sim \Delta x$[†]. We say that this is *first order accurate*. This means that we are neglecting terms that scale as Δx or to higher powers. This is fine if Δx is small. This error is our truncation error—just as discussed above, it arises because our numerical approximation throws away higher order terms. The approximation $\partial a/\partial x|_i = (a_{i+1} - a_i)/\Delta x$ has the same order of accuracy.

Exercise 1.4

Show that a centered difference,

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_{i+1} - a_{i-1}}{2\Delta x}$$

is second order accurate, i.e. its truncation error is $\mathcal{O}(\Delta x^2)$.

Figure 1.2 shows the left- and right-sided first-order differences and the central difference as approximations to $\sin(x)$. Generally speaking, higher-order methods have lower numerical error associated with them, and also involve a wider range of data points.

Second- and higher-order derivatives can be constructed in the same fashion.

Exercise 1.5

Using the Taylor expansions for a_{i+1} *and* a_{i-1}*, find a difference approximation to the second derivative at* i.

[†]in some texts, you see this $\mathcal{O}(\Delta x^n)$ referred to as "big O notation"

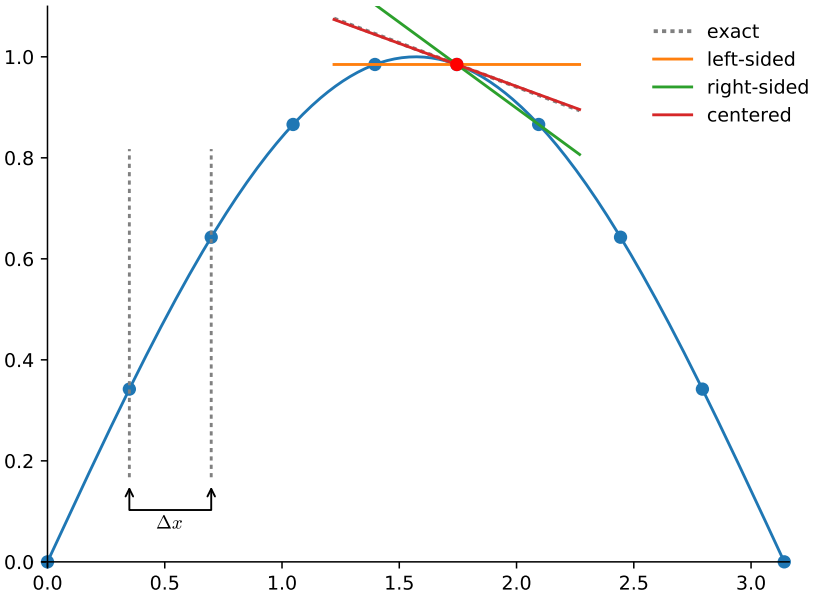

Figure 1.2: A comparison of one-sided and centered difference approximations to the derivative of $\sin(x)$. hydro_examples: `derivatives.py`

Differentiation of an analytic function

An alternate scenario is when you know the analytic form of the function, $f(x)$, and are free to choose the points where you evaluate it. Here you can pick a δx and evaluate the derivative as

$$\left.\frac{df}{dx}\right|_{x=x_0} = \frac{f(x_0+\delta x) - f(x_0)}{\delta x} \tag{1.9}$$

An optimal value for δx requires a balance of truncation error (which wants a small δx) and roundoff error (which becomes large when δx is close to machine ϵ). Figure 1.3 shows the error for the numerical derivative of $f(x) = \sin(x)$ at the point $x_0 = 1$, as a function of δx. A nice discussion of this is given in [85]. A good rule-of-thumb is to pick $\delta x \approx \sqrt{\epsilon}$, where ϵ is machine epsilon, to balance roundoff and truncation error.

Comparing the result with different choices of δx allows for error estimation and an improvement of the result by combining the estimates using δx and $\delta x/2$ (this is the basis for a method called *Richardson extrapolation*).

Integration

In numerical analysis, any integration method that is composed as a weighted sum of the function evaluated at discrete points is called a *quadrature rule*.

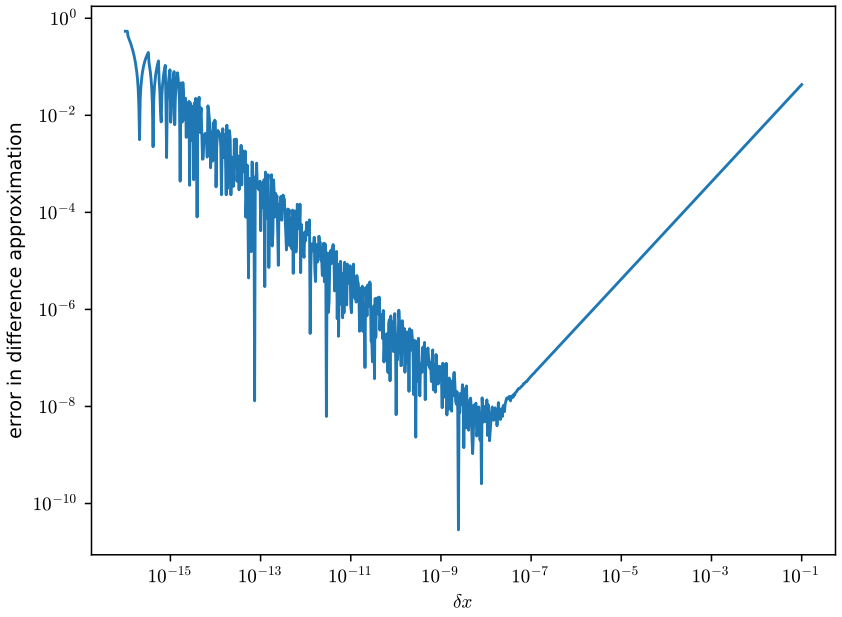

Figure 1.3: Error in the numerical approximation of the derivative of $f(x) = \sin(x)$ at $x_0 = 1$ as a function of the spacing δx. For small δx, roundoff error dominates the error in the approximation. For large δx, truncation error dominates.
hydro_examples: `deriv_error.py`

If we have a function sampled at a number of equally-spaced points, $x_0 \equiv a, x_1, \ldots, x_N \equiv b$‡, we can construct a discrete approximation to an integral as:

$$I \equiv \int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{N-1} f(x_i) \tag{1.10}$$

where $\Delta x \equiv (b-a)/N$ is the width of the intervals (see the top-left panel in Figure 1.4). This is a very crude method, but in the limit that $\Delta x \to 0$ (or $N \to \infty$), this will converge to the true integral. This method is called the *rectangle rule*. Note that here we expressing the integral over the N intervals using a simple quadrature rule in each interval. Summing together the results of the integral over each interval to get the result in our domain is called *compound* integration.

We can get a more accurate answer for I by interpolating between the points. The simplest case is to connect the sampled function values, $f(x_0), f(x_1), \ldots, f(x_N)$ with a line, creating a trapezoid in each interval, and then simply add up the area of all of the trapezoids:

$$I \equiv \int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \tag{1.11}$$

‡Note that this is N intervals and $N+1$ points

This is called the *trapezoid rule* (see the top-right panel in Figure 1.4). Note here we assume that the points are equally spaced.

One can keep going, but practically speaking, a quadratic interpolation is as high as one usually encounters. Fitting a quadratic polynomial requires three points.

Exercise 1.6

Consider a function, $f(x)$, sampled at three equally-spaced points, α, β, γ, with corresponding function values $f_\alpha, f_\beta, f_\gamma$. Derive the expression for Simpson's rule by fitting a quadratic $\hat{f}(x) = A(x-\alpha)^2 + B(x-\alpha) + C$ to the three points (this gives you A, B, and C), and then analytically integrating $\hat{f}(x)$ in the interval $[\alpha, \gamma]$. You should find

$$I = \frac{\gamma - \alpha}{6}(f_\alpha + 4f_\beta + f_\gamma) \tag{1.12}$$

Note that $(\gamma - \alpha)/6 = \Delta x/3$

For a number of samples, N, in $[a, b]$, we will consider every two intervals together. The resulting expression is:

$$I \equiv \int_a^b f(x)dx \approx \frac{\Delta x}{3} \sum_{i=0}^{(N-2)/2} [f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})] \tag{1.13}$$
$$= f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)$$

This method is called *Simpson's rule*. Note that for 2 intervals / 3 sample points ($N = 2$), we only have 1 term in the sum, $(N-2)/2 = 0$, and we get the result derived in Exercise 1.6.

Figure 1.4 shows these different approximations for the case of two intervals (three points).

Analogous expressions exist for the case of unequally-spaced points.

The compound trapezoid rule converges as second-order over the interval $[a, b]$, while Simpson's rule converges as fourth-order.

As with differentiation, if you are free to pick the points where you evaluate $f(x)$, you can get a much higher-order accurate result. *Gaussian quadrature* is a very powerful technique that uses the zeros of different polynomials as the evaluation points for the function to give extremely accurate results. See the text by Garcia [35] for a nice introduction to these methods.

1.2.3 Root finding

Often we want to find the root of a function (or the vector that zeros a vector of functions). The most popular method for root finding is the *Newton-Raphson method*.

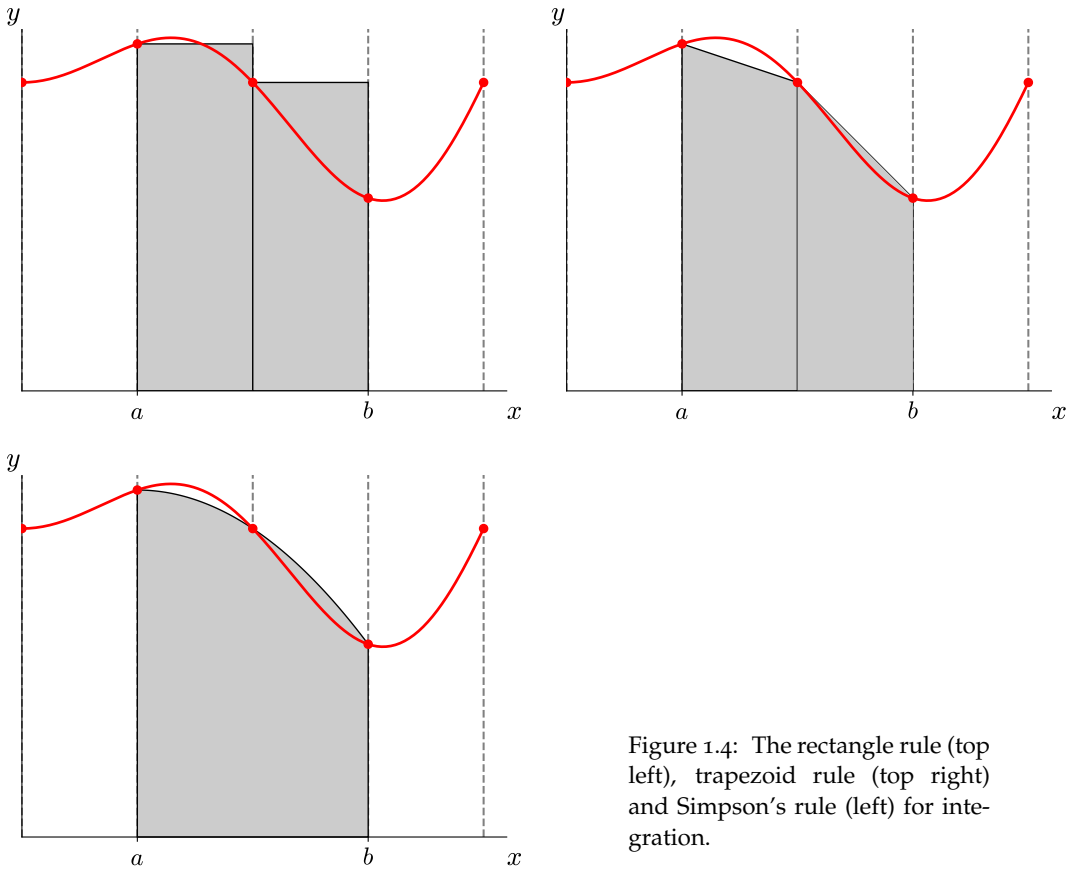

Figure 1.4: The rectangle rule (top left), trapezoid rule (top right) and Simpson's rule (left) for integration.

We want to find x, such that $f(x) = 0$. Start with an initial guess x_0 that you believe is close to the root, then you can improve the guess to the root by an amount δx by looking at the Taylor expansion about x_0:

$$f(x_0 + \delta x) \sim f(x_0) + f'(x_0)\delta x + \ldots = 0 \tag{1.14}$$

Keeping only to $\mathcal{O}(\delta x)$, we can solve for the correction, δx:

$$\delta x = -\frac{f(x_0)}{f'(x_0)} \tag{1.15}$$

This can be used to correct out guess as $x_0 \leftarrow x_0 + \delta x$, and we can iterate on this procedure until δx falls below some tolerance. Figure 1.5 illustrates this iteration.

The main "got-ya" with this technique is that you need a good initial guess. In the Taylor expansion, we threw away the δx^2 term, but if our guess was far from the root, this (and higher-orders) term may not be small. Obviously, the derivative must be non-zero in the region around the root that you search as well.

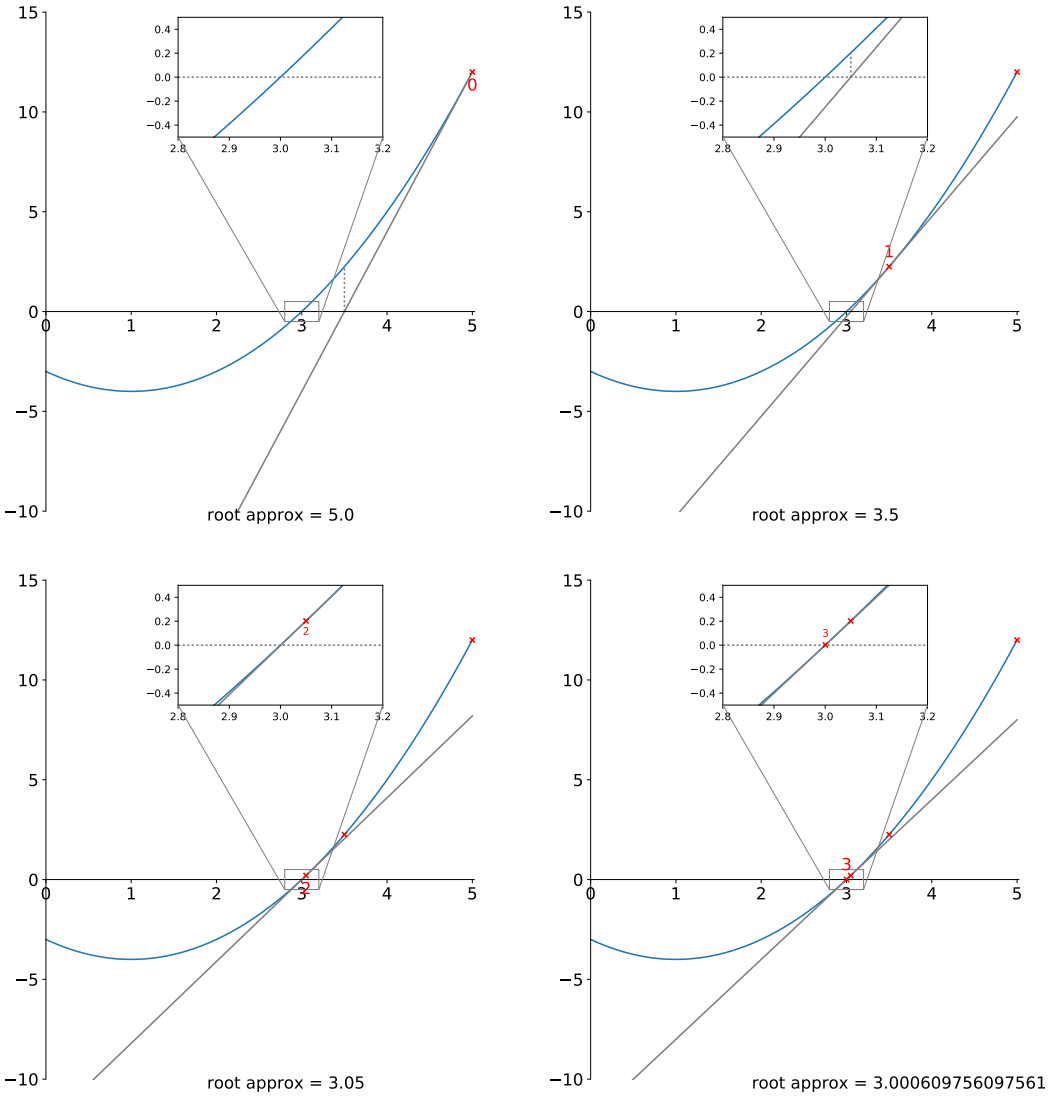

Figure 1.5: The convergence of Newton's method for finding the root. In each pane, the red point is the current guess for the root. The solid gray line is the extrapolation of the slope at the guess to the x-axis, which defines the next approximation to the root. The vertical dotted line to the function shows the new slope that will be used for extrapolation in the next iteration.
hydro_examples: `roots_plot.py`

Exercise 1.7

Code up Newton's method for finding the root of a function $f(x)$ and test it on several different test functions.

The *secant method* for root finding is essentially Newton-Raphson, but instead of using an analytic derivative, f' is estimated as a numerical difference.

Newton's method has pathologies—it is possible to get into a cycle where you don't converge but simply pass through the same set of root approximations. Many other root finding methods exist, including *bisection*, which iteratively halves an interval known to contain a root by looking for the change in sign of the function $f(x)$. *Brent's method* combines several different methods to produce a robust procedure for root-finding. Most numerical analysis texts will give a description of these.

1.2.4 Norms

Often we will need to measure the "size" of an error to a discrete approximation. For example, imagine that we know the exact function, $f(x)$ and we have have an approximation, f_i defined at N points, $i = 0, \ldots, N-1$. The error at each point i is $\epsilon_i = |f_i - f(x_i)|$. But this is N separate errors—we want a single number that represents the error of our approximation. This is the job of a vector norm.

There are many different norms we can define. For a vector $\mathbf{q}$, we write the norm as $\|q\|$. Often, we'll put a subscript after the norm brackets to indicate which norm was used. Some popular norms are:

- *inf norm*:

$$\|q\|_\infty = \max_i |q_i| \tag{1.16}$$

- *L1 norm*:

$$\|q\|_1 = \frac{1}{N} \sum_{i=0}^{N-1} |q_i| \tag{1.17}$$

- *L2 norm*:

$$\|q\|_2 = \left[\frac{1}{N} \sum_{i=0}^{N-1} |q_i|^2 \right]^{1/2} \tag{1.18}$$

- *general p-norm*

$$\|q\|_p = \left[\frac{1}{N} \sum_{i=0}^{N-1} |q_i|^p \right]^{1/p} \tag{1.19}$$

Note that these norms are defined such that they are normalized—if you double the number of elements (N), the normalization gives you a number that can still be meaningfully compared to the smaller set. For this reason, we will use these norms when we look at the convergence with resolution of our numerical methods.

We'll look into how the choice of norm influences you convergence criterion, but inspection shows that the inf-norm is local—a element in your vector is given the entire weight, whereas the other norms are more global.

1.2.5 ODEs

Consider a system of first-order ordinary differential equations,

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}(t)) \tag{1.20}$$

If k represents an index into the vector $\mathbf{y}$, then the k^{th} ODE is

$$\dot{y}_k = f_k(t, \mathbf{y}(t)) \tag{1.21}$$

We want to solve for the vector $\mathbf{y}$ as a function of time. Note higher-order ODEs can always be written as a system of first-order ODEs by introducing new variables[§].

Broadly speaking, methods for integrating ODEs can be broken down into explicit and implicit methods. Explicit methods difference the system to form an update that uses only the current (and perhaps previous) values of the dependent variables in evaluating f. For example, a first-order explicit update (*Euler's method*) appears as:

$$y_k^{n+1} = y_k^n + \Delta t f_k(t^n, \mathbf{y}^n) \tag{1.22}$$

where Δt is the stepsize. Implicit methods instead evaluate the righthand side, $\mathbf{f}$ using the new-time value of $\mathbf{y}$ we are solving for.

Perhaps the most popular explicit method for ODEs is the 4th-order Runge-Kutta method (RK4). This is a multistage method where several extrapolations to the midpoint and endpoint of the interval are made to estimate slopes and then a weighted average of these slopes are used to advance the solution. The various slopes are illustrated in Figure 1.6 and the overall procedure looks like:

$$y_k^{n+1} = y_k^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{1.23}$$

where the slopes are:

$$k_1 = f(t^n, y_k^n) \tag{1.24}$$
$$k_2 = f(t^n + \tfrac{\Delta t}{2}, y_k^n + \tfrac{\Delta t}{2} k_1) \tag{1.25}$$
$$k_3 = f(t^n + \tfrac{\Delta t}{2}, y_k^n + \tfrac{\Delta t}{2} k_2) \tag{1.26}$$
$$k_4 = f(t^n + \Delta t, y_k^n + \Delta t k_3) \tag{1.27}$$

Note the similarity to Simpson's method for integration. This is fourth-order accurate overall.

[§]As an example, consider Newton's second law, $\ddot{x} = F/m$. We can write this as a system of two ODEs by introducing the velocity, v, giving us: $\dot{v} = F/m$, $\dot{x} = v$.

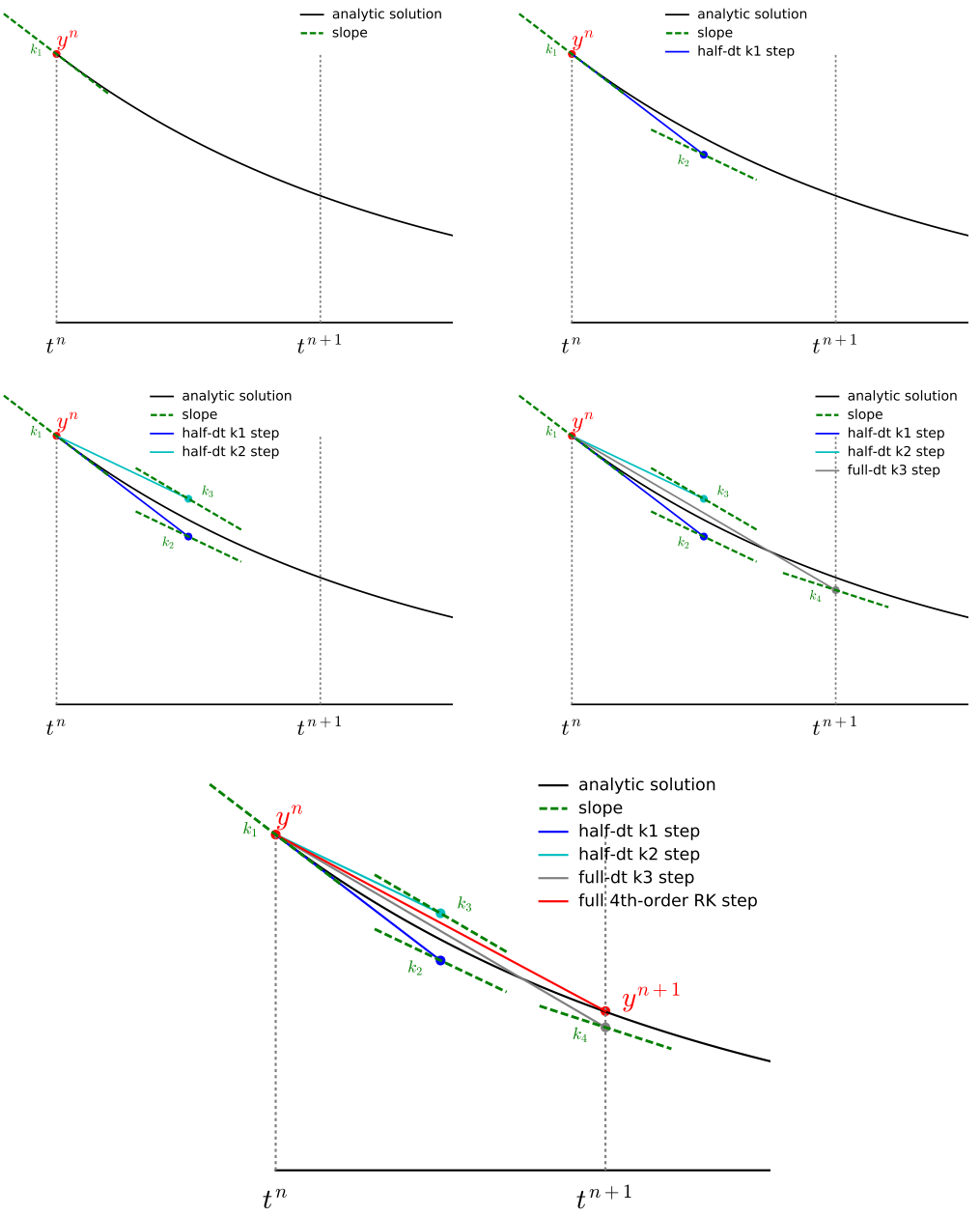

Figure 1.6: A graphical illustration of the four steps in the 4th-order Runge-Kutta method. This example is integrating $dy/dt = -y$.

Exercise 1.8

Consider the orbit of Earth around the Sun. If we work in the units of astronomical units, years, and solar masses, then Newton's gravitational constant and the solar mass together are simply $GM = 4\pi^2$ (this should look familiar as Kepler's third law). We can write the ODE system describing the motion of Earth as:

$$\dot{\mathbf{x}} = \mathbf{v} \tag{1.28}$$

$$\dot{\mathbf{v}} = -\frac{GM\mathbf{r}}{r^3} \tag{1.29}$$

If we take the coordinate system such that the Sun is at the origin, then, $\mathbf{x} = (x, y)^\intercal$ is the position of the Earth and $\mathbf{r} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}}$ is the radius vector pointing from the Sun to the Earth.
Take as initial conditions the planet at perihelion:

$$\begin{aligned} x_0 &= 0 \\ y_0 &= a(1-e) \\ (\mathbf{v}\cdot\hat{\mathbf{x}})_0 &= -\sqrt{\frac{GM}{a}\frac{1+e}{1-e}} \\ (\mathbf{v}\cdot\hat{\mathbf{y}})_0 &= 0 \end{aligned}$$

where a is the semi-major axis and e is the eccentricity of the orbit (these expressions can be found in any introductory astronomy text that discusses Kepler's laws).
Integrate this system for a single orbital period with the first-order Euler and the RK4 method and measure the convergence by integrating at a number of different Δt's. Note: you'll need to define some measure of error, you can consider a number of different metrics, e.g., the change in radius after a single orbit.

The choice of the stepsize, Δt, in the method greatly affects the accuracy. In practice you want to balance the desire for accuracy with the expense of taking lots of small steps. A powerful technique for doing this is to use error estimation and an adaptive stepsize with your ODE integrator. This monitors the size of the truncation error and adjusts the stepsize, as needed, to achieve a desired accuracy. A nice introduction to how this works for RK4 is given in [35].

Implicit methods

Implicit methods difference the system in a way that includes the new value of the dependent variables in the evaluation of $f_k(t, \mathbf{y}(t))$—the resulting implicit system is usually solved using, for example, Newton-Raphson iteration.

A first-order implicit update (called *backward Euler*) is:

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}) \tag{1.30}$$

This is more complicated to solve than the explicit methods above, and generally will require some linear algebra. If we take Δt to be small, then the change in the solution, $\Delta \mathbf{y}$ will be small as well, and we can Taylor-expand the system.

To solve this, we pick a guess, $\mathbf{y}_0^{n+1}$, that we think is close, to the solution and we will solve for a correction, $\Delta \mathbf{y}_0$ such that

$$\mathbf{y}^{n+1} = \mathbf{y}_0^{n+1} + \Delta \mathbf{y}_0 \tag{1.31}$$

Using this approximation, we can expand the righthand side vector,

$$\mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}) = \mathbf{f}(t^{n+1}, \mathbf{y}_0^{n+1}) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_0 \Delta \mathbf{y}_0 + \ldots \tag{1.32}$$

Here we recognize the Jacobian matrix, $\mathbf{J} \equiv \partial \mathbf{f}/\partial \mathbf{y}$,

$$\mathbf{J} = \begin{pmatrix} \partial f_1/\partial y_1 & \partial f_1/\partial y_2 & \partial f_1/\partial y_3 & \ldots & \partial f_1/\partial y_n \\ \partial f_2/\partial y_1 & \partial f_2/\partial y_2 & \partial f_2/\partial y_3 & \ldots & \partial f_2/\partial y_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial y_1 & \partial f_n/\partial y_2 & \partial f_n/\partial y_3 & \ldots & \partial f_n/\partial y_n \end{pmatrix} \tag{1.33}$$

Putting Eqs. 1.31 and 1.32 into Eq. 1.30, we have:

$$\mathbf{y}_0^{n+1} + \Delta \mathbf{y}_0 = \mathbf{y}^n + \Delta t \left[\mathbf{f}(t^{n+1}, \mathbf{y}_0^{n+1}) + \mathbf{J}|_0 \, \Delta \mathbf{y}_0 \right] \tag{1.34}$$

Writing this as a system for the unknown correction, $\Delta \mathbf{y}_0$, we have

$$(\mathbf{I} - \Delta t \, \mathbf{J}|_0) \, \Delta \mathbf{y}_0 = \mathbf{y}^n - \mathbf{y}_0^{n+1} + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}_0^{n+1}) \tag{1.35}$$

This is a linear system (a matrix $\times$ vector $=$ vector) that can be solved using standard matrix techniques (numerical methods for linear algebra can be found in most numerical analysis texts). After solving, we can correct our initial guess:

$$\mathbf{y}_1^{n+1} = \mathbf{y}_0^{n+1} + \Delta \mathbf{y}_0 \tag{1.36}$$

Written this way, we see that we can iterate. To kick things off, we need a suitable guess—an obvious choice is $\mathbf{y}_0^{n+1} = \mathbf{y}^n$. Then we correct this guess by iterating, with the k-th iteration looking like:

$$\left(\mathbf{I} - \Delta t \, \mathbf{J}|_{k-1}\right) \Delta \mathbf{y}_{k-1} = \mathbf{y}^n - \mathbf{y}_{k-1}^{n+1} + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}_{k-1}^{n+1}) \tag{1.37}$$

$$\mathbf{y}_k^{n+1} = \mathbf{y}_{k-1}^{n+1} + \Delta \mathbf{y}_{k-1} \tag{1.38}$$

We will iterate until we find $\|\Delta \mathbf{y}_k\| < \epsilon \|\mathbf{y}^n\|$. Here ϵ is a small tolerance, and we use $\mathbf{y}^n$ to produce a reference scale for meaningful comparison. Note that here we use a vector norm to give a single number for comparison.

Note that the role of the Jacobian here is the same as the first derivative in the scalar Newton's method for root finding (Eq. 1.15)—it points from the current guess to the solution. Sometimes an approximation to the Jacobian, which is cheaper to evaluate, may work well enough for the method to converge.

Explicit methods are easier to program and run faster (for a given Δt), but implicit methods work better for *stiff* problems—those characterized by widely disparate timescales over which the solution changes [20][¶]. A good example of this issue in astrophysical problems is with nuclear reaction networks (see, e.g., [78]). As with the explicit case, higher-order methods exist that can provide better accuracy at reduced cost.

1.2.6 FFTs

The discrete Fourier transform converts a discretely-sampled function from real space to frequency space, and identifies the amount of power associate with discrete wavenumbers. This is useful for both analysis, as well as for solving certain linear problems (see, e.g., § 9.2).

For a function, $f(x)$ sampled at N equally-spaced points (such that $f_n = f(x_n)$), the discrete Fourier transform, $\mathcal{F}_k$ is written as:

$$\mathcal{F}_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i n k/N} \qquad k \in [0, N-1] \tag{1.39}$$

The exponential in the sum brings in a real (cosine terms, symmetric functions) and imaginary (sine terms, antisymmetric functions) part.

$$\mathrm{Re}(\mathcal{F}_k) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{2\pi n k}{N}\right) \tag{1.40}$$

$$\mathrm{Im}(\mathcal{F}_k) = \sum_{n=0}^{N-1} f_n \sin\left(\frac{2\pi n k}{N}\right) \tag{1.41}$$

Alternately, it is sometimes useful to combine the real and imaginary parts into an amplitude and phase.

The inverse transform is:

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} \mathcal{F}_k e^{2\pi i n k/N} \qquad n \in [0, N-1] \tag{1.42}$$

[¶]Defining whether a problem is stiff can be tricky (see [20] for some definitions). For a system of ODEs, a large range in the eigenvalues of the Jacobian usually means it is stiff.

The $1/N$ normalization is a consequence of Parseval's theorem—the total power in real space must equal the total power in frequency space. One way to see that this must be the case is to consider the discrete transform of $f(x) = 1$, which should be a delta function.

The FFT of a discrete function has the same amount of information as the original discrete function. Note that if $f(x)$ is real-valued, then the transform $\mathcal{F}_k$ has 2 numbers (the real and imaginary parts) for each of our original N real values. This would mean that we have $2N$ pieces of information in frequency space where we only had N pieces of information in real space. Since we cannot create information in frequency space where there was no corresponding real-space information, half of the $\mathcal{F}_k$'s are redundant (and $\mathcal{F}_{-k} = \mathcal{F}_k^\star$).

Directly computing $\mathcal{F}_k$ for each k takes $\mathcal{O}(N^2)$ operations. The fast Fourier transform (FFT) is a reordering of the sums in the discrete Fourier transform to reuse partial sums and compute the same result in $\mathcal{O}(N \log N)$ work. Many numerical methods books can give a good introduction to how to design an FFT algorithm.

Exercise 1.9

Learn how to use a FFT library or the built-in FFT method in your programming language of choice. There are various ways to define the normalization in the FFT, that can vary from one library to the next. To ensure that you are doing things correctly, compute the following transforms:

- $\sin(2\pi k_0 x)$ *with* $k_0 = 0.2$. *The transform should have all of the power in the imaginary component only at the frequency* 0.2.
- $\cos(2\pi k_0 x)$ *with* $k_0 = 0.2$. *The transform should have all of the power in the real component only at the frequency* 0.2.
- $\sin(2\pi k_0 x + \pi/4)$ *with* $k_0 = 0.2$. *The transform should have equal power in the real and imaginary components, only at the frequency* 0.2. *Since the power is 1, the amplitude of the real and imaginary parts will be* $1/\sqrt{2}$.

An example of the transform of a single-frequency sine wave with a phase shift is shown in Figure 1.7.

The FFT assumes that a function is periodic and that the points are evenly spaced. If the function is not periodic, then a signal will show up in the FFT at a wavenumber corresponding to the size of the domain. Related to this, you need to ensure that your points are evenly spaced even at the boundary, e.g., if you have a function on $[0, 1]$ represented by 5 points, you want the points to be $\{0, 0.2, 0.4, 0.6, 0.8\}$ and not $\{0, 0.25, 0.5, 0.75, 1\}$. In the latter case, 0 and 1 are the same function value (due to periodicity), but the FFT assumes that all points are evenly spaced (even across the boundary), so it will think that there is a spacing of 0.25 between these end points.

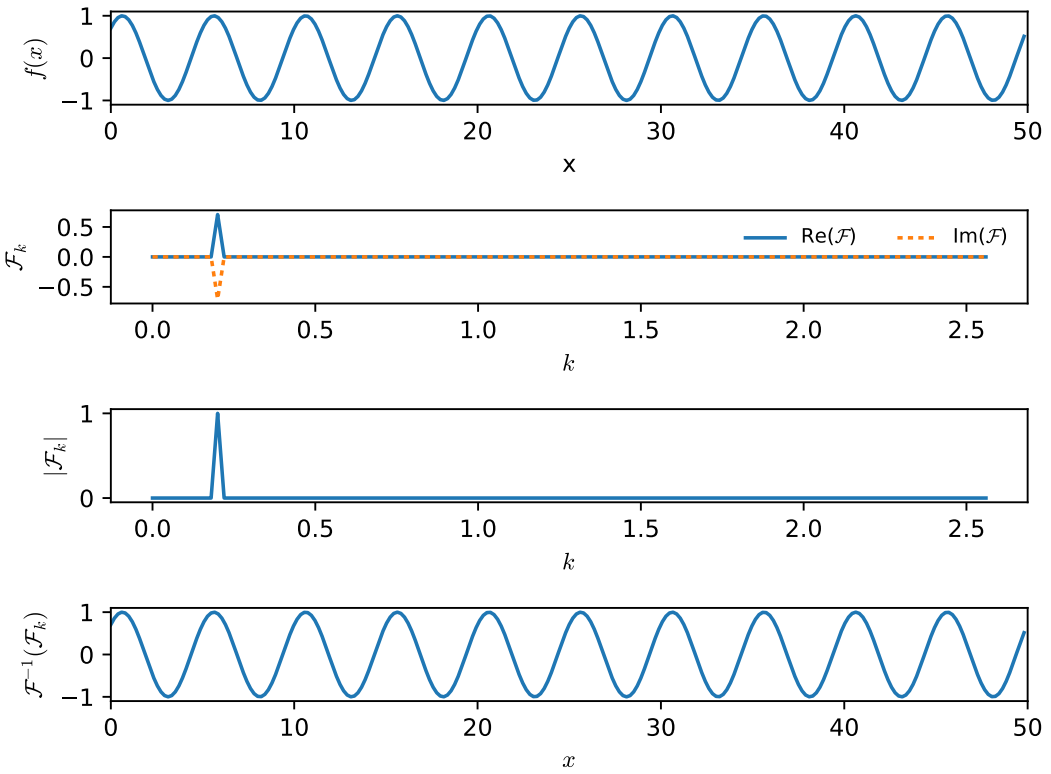

Figure 1.7: The Fourier transform of a sine wave with a phase of $\pi/4$, $f(x) = \sin(2\pi k_0 x + \pi/4)$ with $k_0 = 0.2$. The top shows the original function. The second panel shows the real and imaginary components—we see all of the power is at our input wavenumber, split equally between the real and imaginary parts. The third pane shows the power (the absolute value of the transform). Finally, the bottom panel shows the inverse transform of our transform, giving us back our original function.

hydro_examples: `fft_simple_examples.py`

Chapter 2

Classification of PDEs

2.1 Introduction

Partial differential equations (PDEs) are usually grouped into one of three different classes: *hyperbolic*, *parabolic*, or *elliptic*. You can find the precise mathematical definition of these classifications in most books on PDEs, but this formal definition is not very intuitive or useful. Instead, it is helpful to look at some prototypical examples of each type of PDE.

When we are solving multiphysics problems, we will see that our system of PDEs spans these different types. Nevertheless, we will look at solutions methods for each type separately first, and then use what we learn to solve more complex systems of equations.

2.2 Hyperbolic PDEs

The canonical hyperbolic PDE is the wave equation:

$$\frac{\partial^2 \phi}{\partial t^2} = c^2 \frac{\partial^2 \phi}{\partial x^2} \tag{2.1}$$

The general solution to this is traveling waves in either direction:

$$\phi(x,t) = \alpha f_0(x - ct) + \beta g_0(x + ct) \tag{2.2}$$

Here f_0 and g_0 are set by the initial conditions, and the solution propagates f_0 to the right and g_0 to the left at a speed c.

Exercise 2.1

Show by substitution that Eq. 2.2 is a solution to the wave equation

A simple first-order hyperbolic PDE is the linear advection equation:

$$a_t + ua_x = 0 \tag{2.3}$$

This simply propagates any initial profile to the right at the speed u. We will use linear advection as our model equation for numerical methods for hyperbolic PDEs.

A system of first-order hyperbolic PDEs takes the form:

$$\mathbf{a}_t + \mathbf{A}\mathbf{a}_x = 0 \tag{2.4}$$

where $\mathbf{a} = (a_0, a_1, \ldots a_{N-1})^\intercal$ and $\mathbf{A}$ is a matrix. This system is hyperbolic if the eigenvalues of A are real (see [46] for an excellent introduction).

An important concept for hyperbolic PDEs are *characteristics*—these are curves in a space-time diagram along which the solution is constant. Associated with these curves is a speed—this is the wave speed at which information on how the solution changes is communicated. For a linear PDE (or system of PDEs), these will tell you everything you need to know about the solution.

2.3 Elliptic PDEs

The canonical elliptic PDE is the Poisson equation:

$$\nabla^2 \phi = f \tag{2.5}$$

Note that there is no time-variable here. This is a pure boundary value problem. The solution, ϕ is determined completely by the source, f, and the boundary conditions.

In contrast to the hyperbolic case, there is no propagation of information here. The potential, ϕ, is known instantaneously everywhere in the domain. For astrophysical flows, this commonly arises as the Poisson equation describing the gravitational potential.

2.4 Parabolic PDEs

The canonical parabolic PDE is the heat equation:

$$\frac{\partial \phi}{\partial t} = k \frac{\partial^2 f}{\partial x^2} \tag{2.6}$$

This has aspects of both hyperbolic and elliptic PDEs.

The heat equation represents diffusion—an initially sharp feature will spread out into a smoother profile on a timescale that depends on the coefficient k. We'll encounter parabolic equations for thermal diffusion and other types of diffusion (like species, mass), and with viscosity.

Exercise 2.2

Using dimensional analysis, estimate the characteristic timescale for diffusion from Eq. 2.6.

Chapter 3

Finite-Volume Grids

3.1 Discretization

The physical systems we model are described by continuous mathematical functions, $f(x, t)$ and their derivatives in space and time. To represent this continuous system on a computer we must discretize it—convert the continuous function into a discrete number of points in space at one or more discrete instances in time. There are many different discretization methods used throughout the physical sciences, engineering, and applied mathematics fields, each with their own strengths and weaknesses. Broadly speaking, we can divide these methods into grid-based and gridless methods.

Gridless methods include those which represent the function as a superposition of continuous basis functions (e.g. sines and cosines). This is the fundamental idea behind *spectral methods*. A different class of methods are those that use discrete particles to represent the mass distribution and produce continuous functions by integrating over these particles with a suitable kernel—this is the basis of *smoothed particle hydrodynamics* (SPH) [55]. SPH is a very popular method in astrophysics.

For grid-based methods, we talk about both the style of the grid (structured vs. unstructured) and the discretization method, e.g. the finite-difference, finite-volume, and finite-element methods.

Structured grids are logically Cartesian. This means that you can reference the location of any cell in the computational domain via an integer index in each spatial dimension. From a programming standpoint, the grid structure can be represented exactly by a multi-dimensional array. Unstructured grids don't have this simple pattern. A popular type of unstructured grid is created using triangular cells (in 2-d) or tetrahedra (in 3-d). The main advantage of these grids is that you can easily represent irregularly-shaped domains. The disadvantage is that the data structures

required to describe the grid are more complicated than a simple array (and tend to have more inefficient memory access).

Once a grid is established, the system of PDEs is converted into a system of discrete equations on the grid. Finite-difference and finite-volume methods can both be applied to *structured grids*. The main difference between these methods is that the finite-difference methods build from the differential form of PDEs while the finite-volume methods build from the integral form of the PDEs. The attractiveness of finite-volume methods is that conservation is a natural consequence of the discretization—this is why they are popular in astrophysics.

In these notes, we will focus on finite-volume techniques on structured grids.

3.2 Grid basics

The grid is the fundamental object for representing continuous functions in a discretized fashion, making them amenable to computation. In astrophysics, we typically use structured grids—these are logically Cartesian, meaning that the position of a quantity on the grid can be specified by a single integer index in each dimension. This works for our types of problems because we don't have irregular geometries—we typically use boxes, disks, or spheres.

We represent derivatives numerically by discretizing the domain into a finite number of zones (a numerical grid). This converts a continuous derivative into a difference of discrete data. Different approximations have different levels of accuracy.

There are two main types of structured grids used in astrophysics: *finite-difference* and *finite-volume*. These differ in way the data is represented. On a finite-difference grid, the discrete data is associated with a specific point in space. On a finite-volume grid, the discrete data is represented by averages over a control volume. Nevertheless, these methods can often lead to very similar looking discrete equations.

Consider the set of grids show in Figure 3.1. On the top is a classic finite-difference grid. The discrete data, f_i, are stored as points regularly spaced in x. With this discretization, the spatial locations of the points are simply $x_i = i\Delta x$, where $i = 0, \ldots, N$*. Note that for a finite-sized domain, we would put a grid point precisely on the physical boundary at each end.

The middle grid is also finite-difference, but now we imagine first dividing the domain into N cells or zones, and we store the discrete data, f_i, at the center of the zone. This is often called a *cell-centered finite-difference* grid. The physical coordinate of the zone centers (where the data lives) are: $x_i = (i + 1/2)\Delta x$, where $i = 0, \ldots, N - 1$. Note that now for a finite-sized domain, the left edge of the first cell will be on the

*When you see f_{i+1}, you can think of this as meaning $f((i+1)\Delta x)$ or $f(x_i + \Delta x)$

boundary and the first data value will be associated at a point $\Delta x/2$ inside the boundary. A similar situation arises at the right physical boundary. Some finite-difference schemes stagger the variables, e.g., putting velocity on the boundaries and density at the center.

Finally, the bottom grid is a finite-volume grid. The layout looks identical to the cell-centered finite difference grid, except now instead of the discrete data being associated at a single point in space, keep track of the total amount of f in the zone (indicated as the shaded regions). Since we generally don't know how f varies in the zone, we will typically talk about the average of f, $\langle f \rangle_i$, over the zone, and represent this by a horizontal. The total amount of f in the zone is then simply $\Delta x \langle f \rangle_i$. We label the left and right edges of a zone with half-integer indices $i - 1/2$ and $i + 1/2$. The physical coordinate of the center of the zone is the same as in the cell-centered finite-difference case.

In all cases, for a *regular* structured grid, we take Δx to be constant. For the finite-difference grids, the discrete value at each point is obtained from the continuous function $f(x)$ as:

$$f_i = f(x_i) \tag{3.1}$$

3.3 Finite-volume grids

In the finite-volume discretization, f_i represents the average of $f(x,t)$ over the interval $x_{i-1/2}$ to $x_{i+1/2}$, where the half-integer indices denote the zone edges (i.e. $x_{i-1/2} = x_i - \Delta x/2$):

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx \tag{3.2}$$

The lower panel of Figure 3.1 shows a finite-volume grid, with the half-integer indices bounding zone i marked. Here we've drawn $\langle f \rangle_i$ as a horizontal line spanning the entire zone—this is to represent that it is an average within the volume defined by the zone edges. To second-order accuracy,

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx \sim f(x_i) \tag{3.3}$$

Exercise 3.1

Show that Eq. 3.3 is true to $O(\Delta x^2)$ by expanding $f(x)$ as a Taylor series in the integral, e.g., as:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2 + \frac{1}{6} f'''(x_i)(x - x_i)^3 + \ldots \tag{3.4}$$

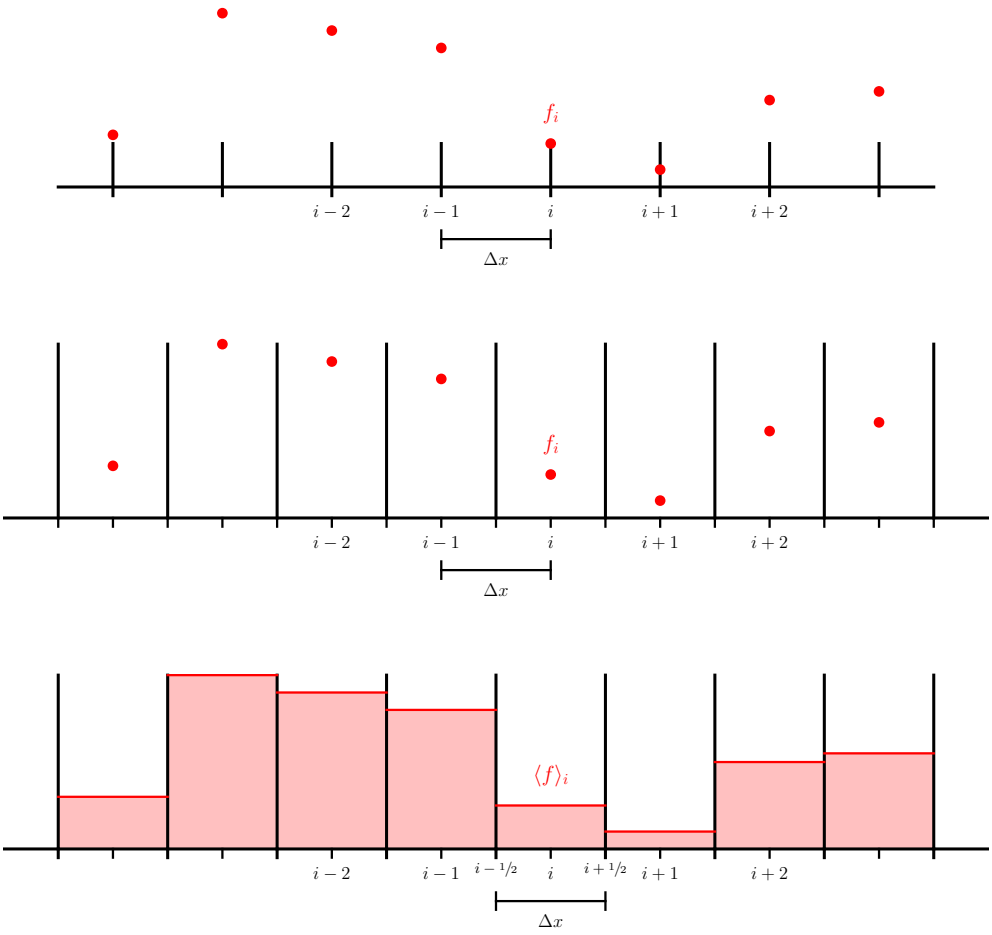

Figure 3.1: Different types of structured grids showing the same data. Top: a finite-difference grid—the discrete data are associated with a specific point in space. Middle: a cell-centered finite-difference grid—again the data is at a specific point, but now we imagine the domain divided into zone with the data living at the center of each zone. Bottom: a finite-volume grid—here the domain is divided into zones and we store the average value of the function within each zone.

This means that we can treat the average of f over a zone as simply $f(x)$ evaluated at the zone center if we only want second-order accuracy. Using the subscript notation, we can express the average of the zone to the right as $\langle f \rangle_{i+1}$.

When we want to interpolate data on a finite-volume grid, we want to construct an interpolating polynomial that is conservative. A *conservative interpolant* reconstructs a continuous functional form, $f(x)$, from a collection of cell-averages subject to the requirement that when $f(x)$ is averaged over a cell, it returns the original cell-average.

Exercise 3.2

Consider three cell averages: $\langle f \rangle_{i-1}$, $\langle f \rangle_i$, $\langle f \rangle_{i+1}$. Fit a quadratic polynomial through these points,

$$f(x) = A(x - x_i)^2 + B(x - x_i) + C \tag{3.5}$$

where the coefficients, A, B, and C are found by applying the constraints:

$$\langle f \rangle_{i-1} = \frac{1}{\Delta x} \int_{x_{i-3/2}}^{x_{i-1/2}} f(x) dx \tag{3.6}$$

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx \tag{3.7}$$

$$\langle f \rangle_{i+1} = \frac{1}{\Delta x} \int_{x_{i+1/2}}^{x_{i+3/2}} f(x) dx \tag{3.8}$$

Show that the conservative interpolant is:

$$\begin{aligned} f(x) = &\frac{\langle f \rangle_{i-1} - 2\langle f \rangle_i + \langle f \rangle_{i+1}}{2\Delta x^2}(x - x_i)^2 + \\ &\frac{\langle f \rangle_{i+1} - \langle f \rangle_{i-1}}{2\Delta x}(x - x_i) + \\ &\frac{-\langle f \rangle_{i-1} + 26\langle f \rangle_i - \langle f \rangle_{i+1}}{24} \end{aligned} \tag{3.9}$$

The Jupyter notebook hydro_examples: `conservative-interpolation.ipynb` shows how to derive these interpolants using SymPy, and gives higher-order interpolants.

3.3.1 Differences and order of accuracy

In practice, when computing derivatives in a finite-volume discretization, we can use the second-order centered difference from § 1.2.2 treating the finite-volume data as living at the cell-centers and still be second-order accurate. For higher accuracy, we can fit a *conservative interpolant* (as illustrated in exercise 3.2) to a collection of points and then differentiate the interpolant itself.

Notice that the righthand side of all derivative approximations involve some linear combinations of f_i's. We call this the *stencil*. The *width* of the stencil is a measure of how many zones on either side of zone i we reach when computing our approximation.

For example, a second derivative can be discretized as:

$$\left.\frac{d^2 f}{dx^2}\right|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \tag{3.10}$$

The stencil on the righthand side encompasses 3 zones.

3.3.2 Conservation

The finite-volume grid is useful when dealing with conservation laws. Consider the following system:

$$\frac{\partial \mathcal{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathcal{U}) = 0 \tag{3.11}$$

where $\mathcal{U}$ is a vector of conserved quantities and $\mathbf{F}(\mathcal{U})$ is the flux of each quantity. This type of system arises, for example, in fluid flow, where the system will represent conservation of mass, momentum, and energy.

Consider one-dimension. Integrating this system over a zone, and normalizing by Δx, we have:

$$\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial \mathcal{U}}{\partial t} dx = -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial \mathbf{F}}{\partial x} dx \tag{3.12}$$

On the left, we can take the time derivative out of the integral, and we are left with the definition of a zone average, so this becomes simply $\partial \langle \mathcal{U} \rangle_i / \partial t$. On the right, we apply the divergence theorem, giving:

$$\frac{\partial \langle \mathcal{U} \rangle_i}{\partial t} = -\frac{1}{\Delta x}\left\{ \mathbf{F}(\mathcal{U})|_{x_{i+1/2}} - \mathbf{F}(\mathcal{U})|_{x_{i-1/2}} \right\} \tag{3.13}$$

Independent of how we discretize in time, notice that we have the cell-average on the left and that the righthand side is simply a difference of fluxes through the interfaces of the zone. For the $i+1$ zone, the update would be:

$$\frac{\partial \langle \mathcal{U} \rangle_{i+1}}{\partial t} = -\frac{1}{\Delta x}\left\{ \mathbf{F}(\mathcal{U})|_{x_{i+3/2}} - \mathbf{F}(\mathcal{U})|_{x_{i+1/2}} \right\} \tag{3.14}$$

Notice that this shares the flux at the $x_{i+1/2}$ interface, but with the opposite sign. When all of the updates are done, the flux through each boundary adds to one zone and subtracts from its neighbor, exactly conserving (to round-off error) the quantity $\mathcal{U}$. This cancellation of the sums is an example of a *telescoping property*, and is the main reason why finite-volume methods are attractive—conserved quantities are conserved to machine (roundoff) precision.

Note that conservation is not the same as accuracy. We can construct the fluxes for our discretized equation by calling a random number generator and we'd still be conservative, but not at all accurate.

3.3.3 Boundary conditions with finite-volume grids

Imagine that we wish to compute the derivative in every zone using:

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_i - f_{i-1}}{\Delta x} \quad . \tag{3.15}$$

If we denote the index corresponding to the leftmost zone in our domain as 'lo', then when we try to compute $\partial f/\partial x|_{\text{lo}}$, we will "fall-off" the grid, i.e., we need a value of

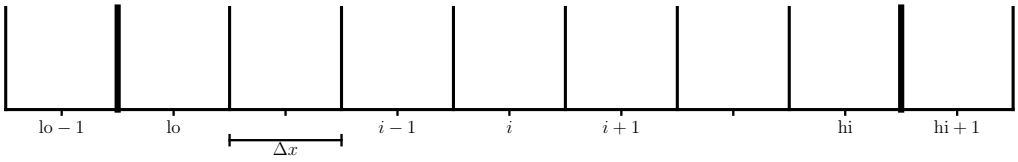

Figure 3.2: A simple 1-d finite-volume grid with a single ghost cell at each end of the domain. The domain boundaries are indicated by the heavy vertical lines. Here there are hi − lo + 1 zones used in the discretization of the domain, with the first zone in the domain labeled lo and the last in the domain labeled hi.

f for zone lo − 1, which is outside the domain. This is where boundary conditions for our grid come into play.

We implement boundary conditions by extending the computational grid beyond the physical domain (see Figure 3.2). These additional zones are called *ghost cells*. They exist solely to handle the boundary conditions and allow us to use the same update equation (e.g. Eq. 3.15) for all zones in the domain.

The number of ghostcells needed for the grid depends on how wide the stencils used in the computation are. The wider the stencil, the more ghostcells are needed.

Periodic boundary conditions would be implemented as:

$$f_{\mathrm{hi}+1} = f_{\mathrm{lo}} \tag{3.16}$$
$$f_{\mathrm{lo}-1} = f_{\mathrm{hi}} \tag{3.17}$$

A simple outflow (zero-gradient) boundary condition would be implemented as:

$$f_{\mathrm{hi}+1} = f_{\mathrm{hi}} \tag{3.18}$$
$$f_{\mathrm{lo}-1} = f_{\mathrm{lo}} \tag{3.19}$$

3.4 Numerical implementation details

The computational grids used for finite-volume techniques maps nicely onto multi-dimensional arrays. For the cell-average data, a typical array would be dimensioned as

```
ilo = ng
ihi = nx + ng - 1
state = np.zeros((nx + 2*ng, nvar))
```

in python (using NumPy), where `nx` is the number of zones in the domain and `ng` is the number of ghost cells. For some problems, there might be more than one state

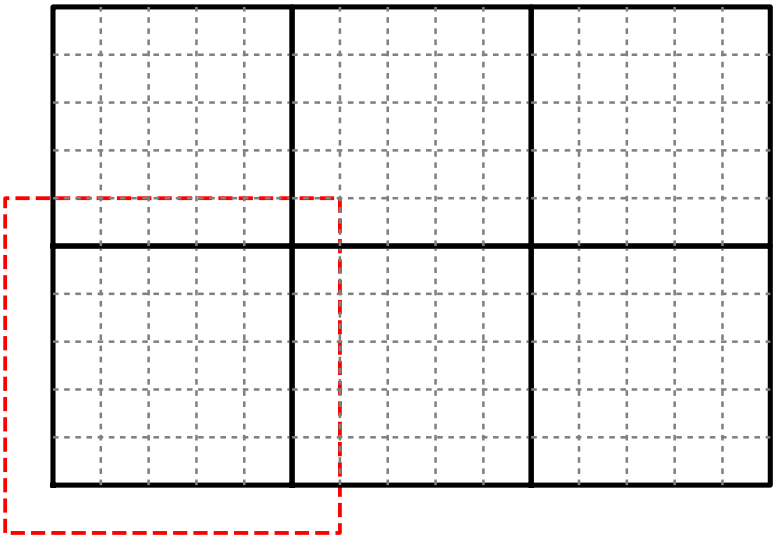

Figure 3.3: Domain decomposition of the computational domain into 6 separate sub-domains. Each sub-domain here has 5×5 zones. For one of the sub-domains, the perimeter of ghost cells is illustrated as the red boundary.

variable, so `nvar`, is the number of state variables that live in each zone. We would then loop from `ilo` to `ihi` to access the state data in the grid interior.

Fluxes are also stored in arrays, but since we cannot index an array with half-integer indices, the standard convention is that the index `i` in an array refers to the flux on the *left* edge of that zone. I.e.,

$$\texttt{flux[i]} \quad \longleftrightarrow \quad F_{i-1/2}$$

3.5 Going further

- *Domain decomposition*: when running on a parallel computer, the work is divided up across processor using domain decomposition. Here, we break the computational domain into smaller sub-domains, and put one (or more) sub-domains on each processor. Each sub-domain has its own perimeter of ghost cells. These are now filled by copying information from the neighboring sub-domains or using the physical boundary conditions for the full domain, depending on where the ghost cells lie. Figure 3.3 shows a simple decomposition of a domain into 6 sub-domains.

- *AMR for structured grids*: adaptive mesh refinement uses a hierarchy of grids to focus resolution in regions of complex flow. For finite-volume codes, the

standard reference for AMR is Berger & Colella [15]. Each level is an even integer multiple finer in resolution, and the grid cells line up with one another (i.e. in two-dimensions, four fine cells will be completely enclosed by a single coarse cell, when using a jump in resolution of $2\times$.) This provides a natural way to enforce conservation. At coarse-fine interfaces, corrections are done to ensure consistency.

- *Mapped grids*: mapped grids are (usually) logically Cartesian grids that use a function (map) to transform from a rectangular mesh (that corresponds to the array in memory) to a general curvilinear grid. These maintain the performance characteristics of structured grids, since any zone can be directly indexed with an integer for each spatial dimension.
- *Lagrangian grids*: a hybrid of particle and grid methods is provided by methods that move particles in a Lagrangian fashion and use a Voronoi tessellation of these particles to define the grid that finite-volume methods are applied to. See, for example, the Arepo code paper [75].

Part II

Advection and Hydrodynamics

Chapter 4

Advection Basics

4.1 The linear advection equation

The linear advection equation is simply:

$$a_t + u a_x = 0 \tag{4.1}$$

where $a(x,t)$ is some scalar quantity and u is the velocity at which it is advected ($u > 0$ advects to the right). The solution to Eq. 4.1 is to simply take the initial data, $a(x, t = 0)$, and displace it to the right at a speed u. The shape of the initial data is preserved in the advection. Many hyperbolic systems of PDEs, e.g. the equations of hydrodynamics, can be written in a form that looks like a system of (nonlinear) advection equations, so the advection equation provides important insight into the methods used for these systems.

Exercise 4.1

Show via substitution that $a(x - ut)$ is a solution to Eq. 4.1 for any choice of a. This means that the solution is constant along the lines $x = ut$ (the curves along which the solution is constant are called the characteristics).

Figure 4.1 shows an initial profile, $a(x)$, and the corresponding characteristics in the t-x plane. With time, since the solution is constant along these characteristics, it simply each point simply follows the characteristic curve, resulting in a shift of the profile to the right.

An important concept that we will discuss shortly is *stability*. Not every discretization that we write down will be well behaved. For some, our initial state will begin to "blow-up", and take on obscenely large and unphysical values after just a few steps. This is the hallmark of a method that is unstable. Some methods have restrictions on the size of the timestep that result in a stable methods as well.

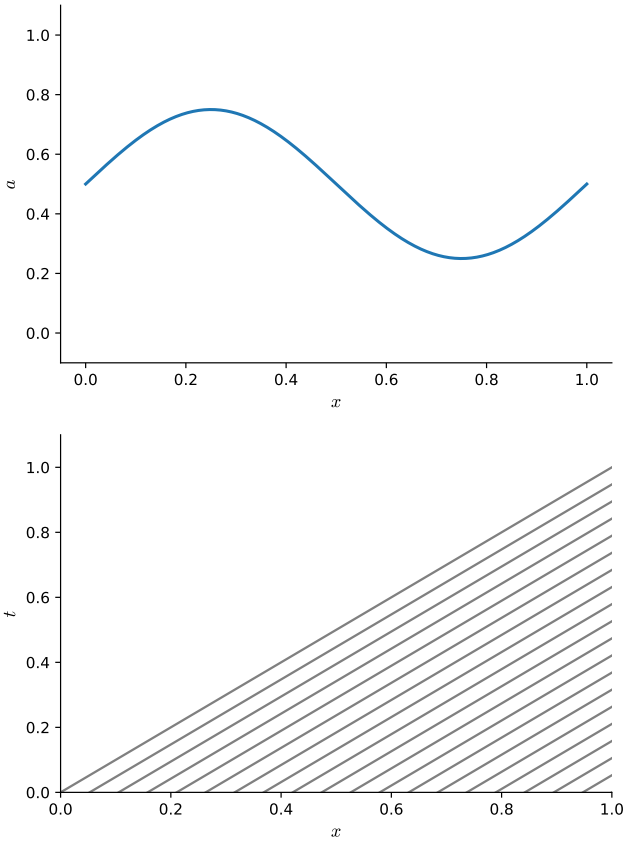

Figure 4.1: (top) Initially sinusoidal distribution (bottom) Characteristic structure for the linear advection equation $u = 1$. Note that for this linear equation the characteristics are all parallel (they have the same slope in the t-x plane.

4.2 First-order advection in 1-d and finite-differences

To get a flavor of the methods for advection, we will use a simple finite-difference discretization—here, the domain is divided into a sequence of points where we store the solution. We will solve Eq. 4.1 numerically by discretizing the solution at these points. The index i denotes the point's location, and a_i denotes the discrete value of $a(x)$ in zone i. The data in each zone can be initialized as $a_i = a(x_i)$. Figure 4.2 shows the grid.

We also need to discretize in time. We denote the time-level of the solution with a superscript, so $a_i^n = a(x_i, t^n)$. For a fixed Δt, time level n corresponds to a time of $t = n\Delta t$.

A simple first-order accurate discretization is:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_i^n - a_{i-1}^n}{\Delta x} \tag{4.2}$$

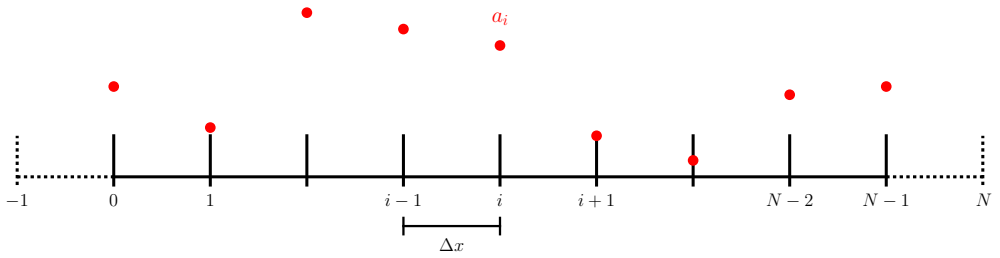

Figure 4.2: A simple finite-difference grid. The solution is stored at each of the labeled points. The dotted lines show the ghost points used to extend our grid past the physical boundaries to accommodate boundary conditions. Note that if we are periodic, then points 0 and $N-1$ are at the same physical point in space, so we would only need to update one of them.

This is an *explicit* method, since the new solution, a_i^{n+1}, depends only on information at the old time level, n.

Finally, we also need to specify a boundary condition for this. Our choice of spatial derivative is one-sided—it uses information from the zone to the left of the zone we are updating. This is because information is flowing from left to right in this problem ($u > 0$). This choice of the derivative is called *upwinding*—this choice of derivative results in a stable method. Notice that if we use Eq. 4.2 to update the data in the first zone inside the boundary, we need data to the left of this zone—outside of the domain. This means that we need a single *ghost point* to implement the boundary conditions for our method. The presence of the ghost points allow us to use the same update equation (e.g. Eq. 4.2) for all zones in the domain.

The last piece of information needed to update the solution is the timestep, Δt. It can be shown that for the solution to be *stable*, the timestep must be less than the time it takes information to propagate across a single zone. That is:

$$\Delta t \leq \frac{\Delta x}{u} \quad . \tag{4.3}$$

This is called the *Courant-Friedrichs-Lewy* or *CFL* condition. A dimensionless quantity called the *CFL number* is defined as

$$\mathcal{C} = \frac{\Delta t u}{\Delta x} \tag{4.4}$$

Stability requires $\mathcal{C} \leq 1$. We traditionally write the timestep as

$$\Delta t = \mathcal{C} \frac{\Delta x}{u} \tag{4.5}$$

and specify $\mathcal{C}$ as part of the problem (a typical value may be $\mathcal{C} = 0.7$).

Exercise 4.2

Show analytically that when you use $\mathcal{C} = 1$ in the first-order differenced advection equation (Eq. 4.2) that you advect the profile exactly, without any numerical error.

Keep in mind that, in general, we will be solving a non-linear system of equations, so it is not possible to run with $\mathcal{C} = 1$, since u (and therefore $\mathcal{C}$) will change from zone to zone. Instead, one looks at the most restrictive timestep over all the zones and uses that for the entire system.

Exercise 4.3

Write a code to solve the 1-d linear advection equation using the discretization of Eq. 4.2 on the domain $[0, 1]$ with $u = 1$ and periodic boundary conditions. For initial conditions, try both a Gaussian profile and a top-hat:

$$a = \begin{cases} 0 & \text{if} \quad x < 1/3 \\ 1 & \text{if} \quad 1/3 \leq x < 2/3 \\ 0 & \text{if} \quad 2/3 \leq x \end{cases} \tag{4.6}$$

Note: For a general treatment of boundary conditions, you would initialize the ghost points to their corresponding periodic data and apply the difference equations to zones $0, \ldots, N-1$. However, for periodic BCs on this grid, points 0 and $N-1$ are identical, so you could do the update in this special case on points $1, \ldots, N-1$ without the need for ghost points and then set $a_0 = a_{N-1}$ after the update.

Run you program for one or more periods (one period is $T = 1/u$) with several different CFL numbers and notice that there is substantial numerical dissipation (see Figure 4.3).

This method is first-order accurate.

Ultimately we will want higher-order accurate methods. The most obvious change from our initial discretization is to try a higher-order spatial derivative.

Exercise 4.4

You may think that using a centered-difference for the spatial derivative, $a_x \sim (a_{i+1} - a_{i-1})/(2\Delta x)$ would be more accurate. This method is called FTCS (forward-time, centered-space). Try this on the same test problems.

You will find that no matter what value of $\mathcal{C}$ you choose with the FTCS method, the solution is unconditionally *unstable* (see Figure 4.4). If you continue to evolve the equation with this method, you would find that the amplitude grows without bound.

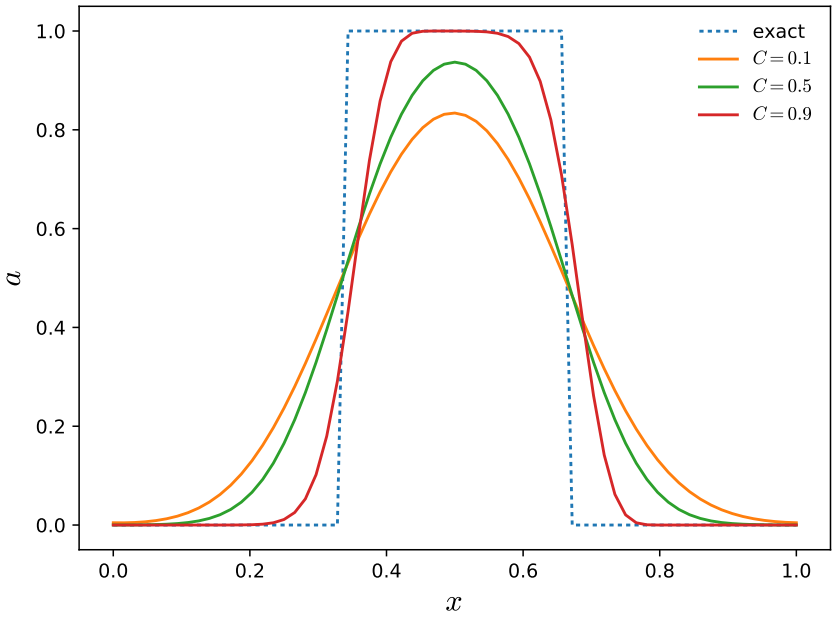

Figure 4.3: Finite-difference solution to the first-order finite-difference upwind method for advection, using 65 points and a variety of CFL numbers.
hydro_examples: `fdadvect.py`

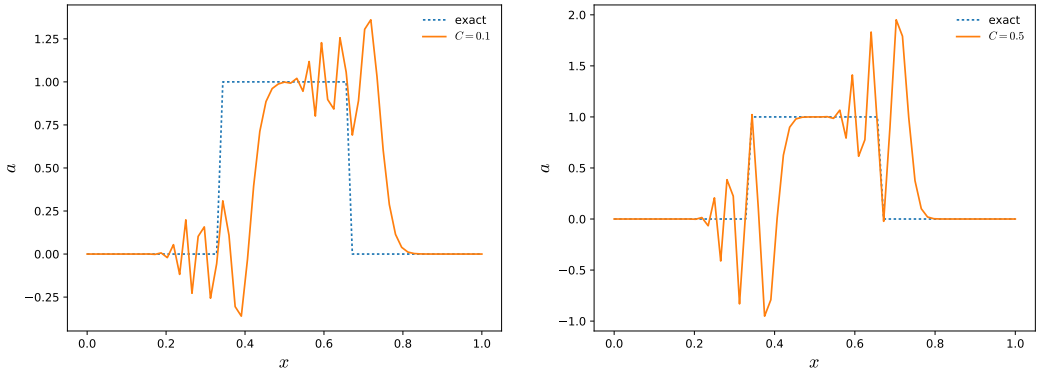

Figure 4.4: Finite-difference solution using the FTCS finite-difference method for advection using 65 points, modeling for only $1/10^{\text{th}}$ of a period. The panel of the left is with $\mathcal{C} = 0.1$ and the panel on the right is $\mathcal{C} = 0.5$.
hydro_examples: `fdadvect.py`

There is something about that discretization that simply gets the physics wrong.

4.3 Stability

The classic method for understanding stability is to consider the growth of a single Fourier mode in our discretization. That is, substitute in $a_i^n = A^n e^{Ii\theta}$, where $I = \sqrt{-1}$*, and θ represents a phase. A method is stable if $|A^{n+1}/A^n| \leq 1$. FTCS appears as:

$$a_i^{n+1} = a_i^n - \frac{\mathcal{C}}{2}(a_{i+1}^n - a_{i-1}^n) \tag{4.7}$$

Examining a Fourier mode shows that:

$$A^{n+1}e^{Ii\theta} = A^n e^{Ii\theta} - \frac{\mathcal{C}}{2}\left(A^n e^{I(i+1)\theta} - A^n e^{I(i-1)\theta}\right) \tag{4.8}$$

$$A^{n+1} = A^n - \frac{\mathcal{C}}{2}A^n\left(e^{I\theta} - e^{-I\theta}\right) \tag{4.9}$$

$$A^{n+1} = A^n\left(1 - I\mathcal{C}\sin\theta\right) \tag{4.10}$$

so the magnitude of the amplification is

$$\left|\frac{A^{n+1}}{A^n}\right|^2 = 1 + \mathcal{C}^2\sin^2\theta \tag{4.11}$$

We see that there is no value of $\mathcal{C}$ that can make the method stable ($|A^{n+1}/A^n| > 1$ always). Doing the same analysis for Eq. 4.2 would show that the upwind method is stable for $0 \leq \mathcal{C} \leq 1$.

Exercise 4.5

Using the above stability analysis, considering the amplitude of a single Fourier mode, show that the growth of a mode for the upwind method (Eq. 4.2) is:

$$\left|\frac{A^{n+1}}{A^n}\right|^2 = 1 - 2\mathcal{C}(1-\mathcal{C})(1-\cos\theta) \tag{4.12}$$

and stability requires $2\mathcal{C}(1-\mathcal{C}) \geq 0$ *or* $0 \leq \mathcal{C} \leq 1$.

It is important to note that this stability analysis only works for linear equations, where the different Fourier modes are decoupled, nevertheless, we use its ideas for nonlinear advection problems as well.

Truncation analysis can also help us understand stability. The idea here is to keep the higher order terms in the Taylor series to understand how they modify the actual equation you are trying to solve.

*our use of i and j as spatial indices presents an unfortunate clash of notation here, hence the use of I for the imaginary unit

Exercise 4.6

To get an alternate feel for stability, we can ask what the terms left out by truncation look like. That is, we can begin with the discretized equation:

$$a_i^{n+1} - a_i^n = -\frac{u\Delta t}{\Delta x}(a_i^n - a_{i-1}^n) \tag{4.13}$$

and replace a_i^{n+1} with a Taylor expansion in time, and replace a_{i-1}^n with a Taylor expansion in space, keeping terms to $O(\Delta t^3)$ and $O(\Delta x^3)$. Replacing $\partial a/\partial t$ with $-u\partial a/\partial x$ in the higher-order terms, show that our difference equation more closely corresponds to

$$\begin{aligned} a_t + ua_x &= \frac{u\Delta x}{2}\left(1 - \frac{\Delta t u}{\Delta x}\right)\frac{\partial^2 a}{\partial x^2} && (4.14)\\ &= \frac{u\Delta x}{2}(1-\mathcal{C})\frac{\partial^2 a}{\partial x^2} && (4.15) \end{aligned}$$

Notice that the righthand side of Eq. 4.14 looks like a diffusion term, however, if $\mathcal{C} > 1$, then the coefficient of the diffusion is negative—this is unphysical. This means that the diffusion would act to take smooth features and make them more strongly peaked—the opposite of physical diffusion.

For FTCS, a similar truncation analysis would show that the diffusion term is always negative.

4.3.1 Domain of dependence

Another important view of our numerical difference scheme is to look at the *domain of dependence*. Figure 4.5 illustrates this for the updated point a_i^{n+1} (shown at the top of center of the space-time diagram). The numerical domain of dependence shows the points that can influence the updated value of a_i using our difference method. For the upwind scheme, we see that this is a triangle that includes a_{i-1}^n and a_i^n. The physical domain of dependence is shown as the orange triangle—this is formed by tracing backwards in time from a_i^{n+1} along a characteristic, reaching out to the point $x_i - C\Delta x = x_i - u\Delta t$ over Δt.

Any stable numerical method must have a numerical domain of dependence that includes the physical domain of dependence. If it does not, then the update to the solution simply does not see the points that contribute to the solution over the timestep Δx. Notice that this is a necessary, but not sufficient condition for stability. FTCS has a domain of dependence that includes the physical domain of dependence, but it is not stable.

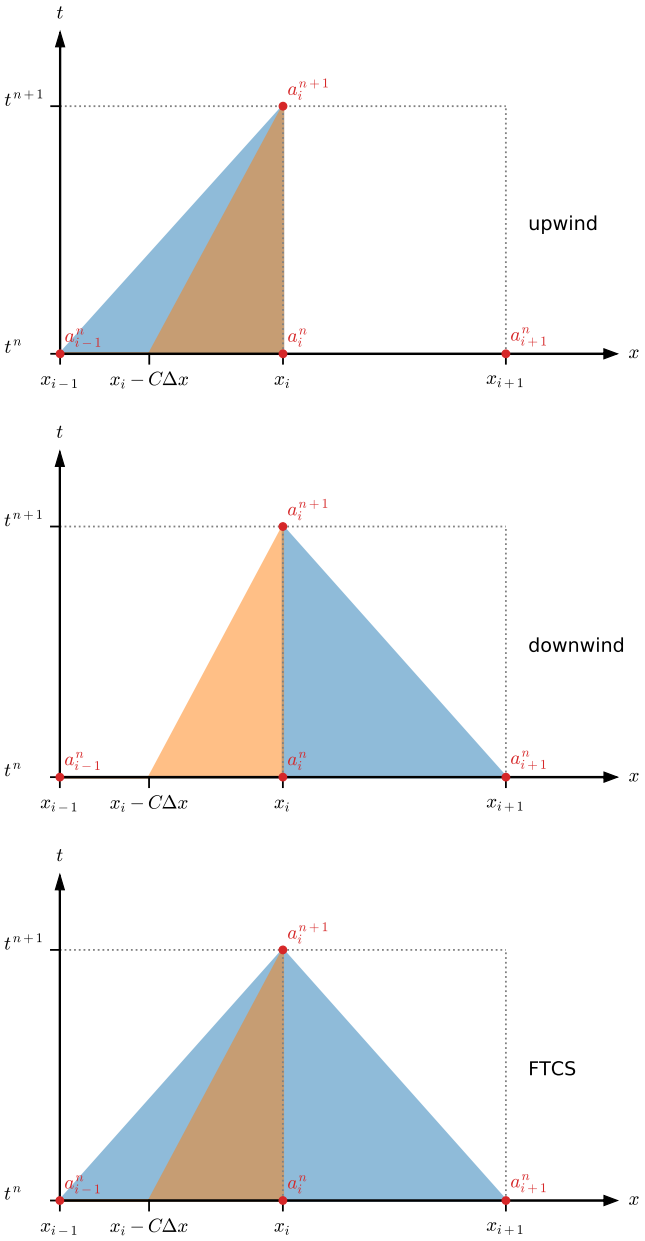

Figure 4.5: Space-time diagrams showing the numerical domain of dependence (blue region) for three different difference methods. Also show is the physical domain of dependence—that formed by tracing backwards from a_i^{n+1} along a characteristic.

4.4 Implicit-in-time

An alternate approach to time-discretization is to do an implicit discretization. Here our upwind method would appear as:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_i^{n+1} - a_{i-1}^{n+1}}{\Delta x} \tag{4.16}$$

The only change here is that the righthand side is evaluated at the new timelevel, $n+1$. We can write this as a linear system with coupled equations:

$$-\mathcal{C} a_{i-1}^{n+1} + (1+\mathcal{C}) a_i^{n+1} = a_i^n \tag{4.17}$$

If we use periodic boundary conditions, then point 0 and $N-1$ are identical, so we only need to update one of these. Taking $a_0^{n+1} = a_{N-1}^{n+1}$, our system in matrix form appears as:

$$\begin{pmatrix} 1+\mathcal{C} & & & & & & & -\mathcal{C} \\ -\mathcal{C} & 1+\mathcal{C} & & & & & & \\ & -\mathcal{C} & 1+\mathcal{C} & & & & & \\ & & -\mathcal{C} & 1+\mathcal{C} & & & & \\ & & & \ddots & \ddots & & & \\ & & & & -\mathcal{C} & 1+\mathcal{C} & & \\ & & & & & -\mathcal{C} & 1+\mathcal{C} & \end{pmatrix} \begin{pmatrix} a_1^{n+1} \\ a_2^{n+1} \\ a_3^{n+1} \\ a_4^{n+1} \\ \vdots \\ a_{N-2}^{n+1} \\ a_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} a_1^n \\ a_2^n \\ a_3^n \\ a_4^n \\ \vdots \\ a_{N-2}^n \\ a_{N-1}^n \end{pmatrix} \tag{4.18}$$

This requires a matrix solve—this makes implicit methods generally more expensive than explicit methods. However, stability analysis would show that this implicit discretization is stable for any choice of $\mathcal{C}$. (But one must not confuse stability with accuracy—the most accurate solutions with this method will still have a small $\mathcal{C}$). Also note that the form of the matrix will change depending on the choice of boundary conditions. Figure 4.6 shows the result of solving this implicit system.

Exercise 4.7

Code up the implicit advection scheme, but using outflow instead of periodic boundary conditions. This will change the form of the matrix. You can use the code from Figure 4.6 as a starting point.

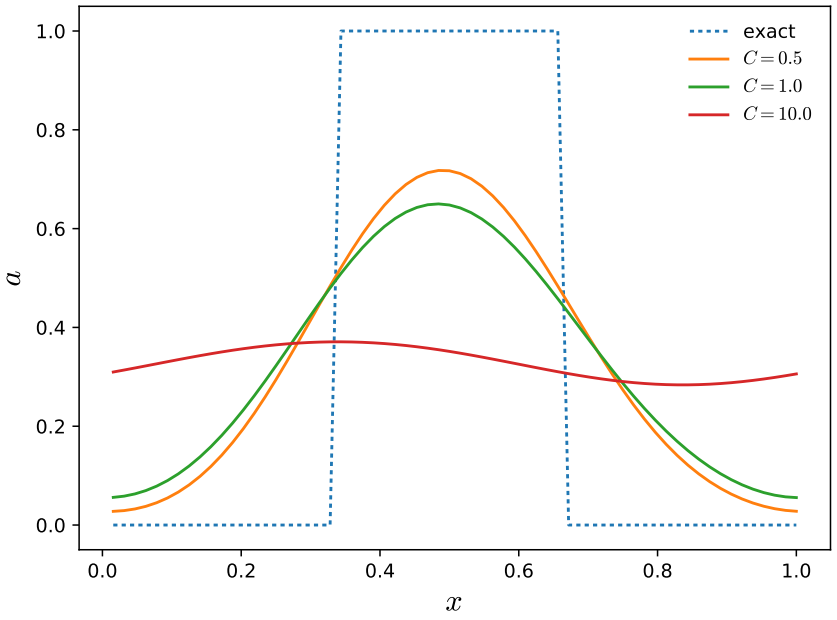

Figure 4.6: Finite-difference solution to the implicit first-order finite-difference upwind method for advection, using 65 points and a variety of CFL numbers.
hydro_examples: `fdadvect_implicit.py`

4.5 Eulerian vs. Lagrangian frames

It is useful to think about how our advected quantity, $a(x,t)$, changes in time. The full time derivative is:

$$\frac{da(x,t)}{dt} = \frac{\partial a}{\partial t} + \frac{\partial a}{\partial x}\frac{dx}{dt} \tag{4.19}$$

So the value of this derivative depends on the path, $x(t)$, that we choose to follow.

Consider an observer who is stationary. They will watch the flow move past them, so $dx/dt = 0$, and $da/dt = \partial a/\partial t$. This fixed frame is called *Eulerian frame*.

Instead imagine an observer who moves with the flow, at the velocity u. This way they keep pace with an individual feature in the flow and track the changes it experiences. In this case, $dx/dt = u$, and our derivative, commonly written as D/Dt is:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} \tag{4.20}$$

This is the *Lagrangian frame*, and the derivative, D/Dt is called the *Lagrangian derivative*, *material derivative*, *convective derivative*, or *advective derivative*[†].

Our linear advection equation can be written simply as $Da/Dt = 0$. We've been solving the equations in the Eulerian frame—our grid is fixed and the fluid moves

[†]and there are actually many more names...

through it. For hydrodynamics, it will be useful conceptually to consider the Lagrangian frame to understand how the fluid properties change in a particular fluid element over time.

4.6 Errors and convergence rate

For the advection problem (with $u > 0$), the analytic solution is to simply propagate the initial profile to the right. This means that with periodic boundary conditions, after advecting for one period, our numerical solution should be identical to the initial conditions. Any differences are our numerical error. We can quantify the error by taking the norm of error[‡] as:

$$\epsilon^{\text{abs}} = \|a^{\text{final}} - a^{\text{init}}\|_2 \equiv \left[\frac{1}{N}\sum_{i=1}^{N}(a_i^{\text{final}} - a_i^{\text{init}})^2\right]^{1/2} \tag{4.21}$$

It is sometimes useful to compare to the norm of the original solution to get a measure of the relative error:

$$\epsilon^{\text{rel}} \equiv \frac{\|a^{\text{final}} - a^{\text{init}}\|_2}{\|a^{\text{init}}\|_2} \tag{4.22}$$

Note that for the absolute norm, it is important in these definitions to normalize by the number of zones, N, otherwise our error will be resolution-dependent. For the relative norm, since we scale by a norm on the same grid, this normalization will cancel.

[‡]see § 1.2.4 for the definition of the norms

Chapter 5

Second- (and Higher-) Order Advection

5.1 Advection and the finite-volume method

In these notes, we will typically use a *finite-volume* discretization. Here we explore this method for the advection equation. First we rewrite the advection equation in *conservation form*:

$$a_t + [f(a)]_x = 0 \tag{5.1}$$

where $f(a) = ua$ is the flux of the quantity a. In conservation form, the time derivative of a quantity is related to the divergence of its flux.

Recall that in the finite-volume discretization, $\langle a \rangle_i$ represents the average of $a(x,t)$ over the interval $x_{i-1/2}$ to $x_{i+1/2}$, where the half-integer indexes denote the zone edges (i.e. $x_{i-1/2} = x_i - \Delta x/2$). Figure 5.1 shows an example of such a grid with 2 ghost cells at each end. (For simplicity of notation, we drop the $\langle\rangle$ going forward). To discretize Eq. 5.1, we integrate it over a zone, from $x_{i-1/2}$ to $x_{i+1/2}$, normalizing by the zone width, Δx:

$$\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} a_t\, dx = -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial}{\partial x} f(a)\, dx \tag{5.2}$$

$$\frac{\partial}{\partial t} a_i = -\frac{1}{\Delta x}\left\{ [f(a)]_{i+1/2} - [f(a)]_{i-1/2} \right\} \tag{5.3}$$

This is an evolution equation for the zone-average of a, and shows that it updates in time based on the fluxes through the boundary of the zone.

We now have a choice on how to proceed with the time-discretization:

1. We can discretize $\partial a_i/\partial t$ directly as $(a_i^{n+1} - a_i^n)/\Delta t$. Then to achieve second-order accuracy, we need to evaluate the righthand side of Eq. 5.3 at the midpoint

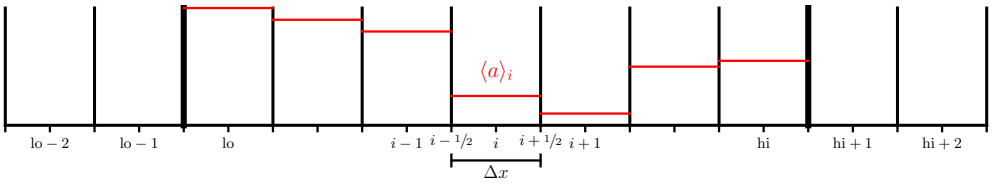

Figure 5.1: A finite-volume grid running from lo, . . . , hi, with two ghost cells at each end.

in time $(n + 1/2)$. This gives rise to a predictor-corrector method (as described in [24]. Sometimes this is called a characteristic tracing method (a name which will be more clear when we discuss the Euler equations).

2. We can recognize that with the spatial discretization done, our PDE has now become an ODE, and we can use standard ODE methods (like Runge-Kutta) to integrate the system in time. This is a method-of-lines integration.

We'll look at both of these in the next sections.

5.2 Second-order predictor-corrector scheme

We discretize Eq. 5.3 in time by evaluating the righthand side at the midpoint in time—this gives a centered-difference in time, which is second-order accurate:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -\frac{[f(a)]_{i+1/2}^{n+1/2} - [f(a)]_{i-1/2}^{n+1/2}}{\Delta x} \tag{5.4}$$

To evaluate the fluxes at the half-time, we need the state at the half-time, that is, we do :

$$[f(a)]_{i+1/2}^{n+1/2} = f(a_{i+1/2}^{n+1/2}) \ . \tag{5.5}$$

We construct a second-order accurate approximation to $a_{i+1/2}^{n+1/2}$ by Taylor expanding the data in the cell to the interface. The construction of the interface state at the midpoint in time is the prediction and the conservative update in the correction here.

Notice that for each interface, there are two possible interface states we can construct—one using the data to the left of the interface (which we will denote with a "L" subscript) and the other using the data to the right of the interface (denoted with an "R"

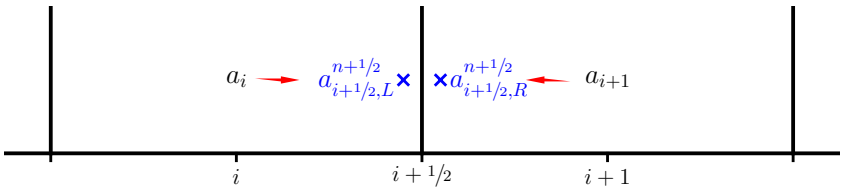

Figure 5.2: The left and right interface state at the $i+1/2$ interface. Here, the left state, $a^{n+1/2}_{i+1/2,L}$, was predicted to the interface from the zone to the left of the interface, using a_i, and the right state, $a^{n+1/2}_{i+1/2,R}$, was predicted to the interface from the zone to the right, using a_{i+1}.

subscript)—see Figure 5.2. These states are:

$$\begin{aligned} a^{n+1/2}_{i+1/2,L} &= a^n_i + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_i + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_i + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_i + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_i + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_i\right) + \ldots \\ &= a^n_i + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_i + \ldots \end{aligned} \tag{5.6}$$

$$\begin{aligned} a^{n+1/2}_{i+1/2,R} &= a^n_{i+1} - \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i+1} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i+1} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_{i+1} - \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i+1} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i+1}\right) + \ldots \\ &= a^n_{i+1} - \frac{\Delta x}{2}\left(1 + \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i+1} + \ldots \end{aligned} \tag{5.7}$$

A suitable estimate is needed for the slope of a that appears in these expressions (as $\partial a/\partial x$). We can approximate this simply as

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_{i+1} - a_{i-1}}{2\Delta x} \tag{5.8}$$

We can think of this method as reconstructing the function form of the data from the cell-average data in each cell using a piecewise linear polynomial. Don't be worried that this looks like FTCS—we'll do upwinding next.

We now have two states, $a^{n+1/2}_{i+1/2,L}$ and $a^{n+1/2}_{i+1/2,R}$ separated by an interface—this is called the *Riemann problem*. The solution to this will depend on the equation being solved, and results in a single state at the interface:

$$a^{n+1/2}_{i+1/2} = \mathcal{R}(a^{n+1/2}_{i+1/2,L}, a^{n+1/2}_{i+1/2,R}) \tag{5.9}$$

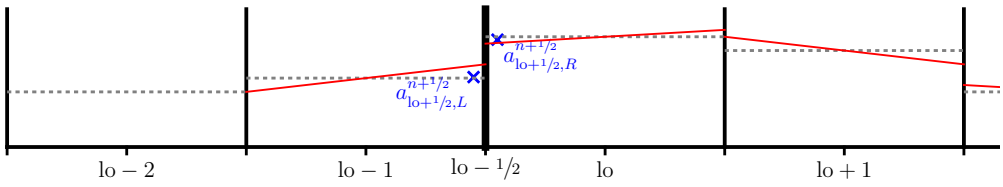

Figure 5.3: Reconstruction near the boundary, showing the need for two ghostcells. Here we see the left and right state at the left physical boundary of the domain (marked as lo $-$ 1/2). The gray dotted lines are the piecewise constant cell averages and the red lines are the reconstructed slopes. Note that we need the slope in lo $-$ 1 to get the left interface state at lo $-$ 1/2, and that slope in turn needed the data in zone lo $-$ 2 to construct a centered-difference.

In our case, the advection equation simply propagates the state to the right (for $u > 0$), so the solution to the Riemann problem is to take the left state (this is another example of upwinding). That is we do:

$$\mathcal{R}(a^{n+1/2}_{i+1/2,L}, a^{n+1/2}_{i+1/2,R}) = \begin{cases} a^{n+1/2}_{i+1/2,L} & u > 0 \\ a^{n+1/2}_{i+1/2,R} & u < 0 \end{cases} \tag{5.10}$$

To complete the update, we use this interface state to evaluate the flux and update the advected quantity via Eq. 5.4 and 5.5.

Boundary conditions are implemented by filling the ghost cells outside each end of the domain based on data in the interior. Note that at the very left edge of the domain, the state $a^{n+1/2}_{\mathrm{lo}-1/2}$ requires the construction of states on the left and right. The left state at that interface, $a^{n+1/2}_{\mathrm{lo}-1/2,L}$ depends on the slope reconstructed in the lo $-$ 1 ghost cell, $\partial a/\partial x|_{\mathrm{lo}-1}$. This in turn is constructed using a limited center-difference that will consider the value in the cell to the left, lo $-$ 2. Therefore, we need two ghost cells at each end of the domain for this method—figure 5.3 illustrates this. Higher-order limiters may require even more ghost cells.

Exercise 5.1

Write a second-order solver for the linear advection equation. To mimic a real hydrodynamics code, your code should have routines for finding initializing the state, filling boundary conditions, computing the timestep, constructing the interface states, solving the Riemann problem, and doing the update. The problem flow should look like:

- *set initial conditions*
- *main evolution loop—loop until final time reached*
  - *fill boundary conditions*

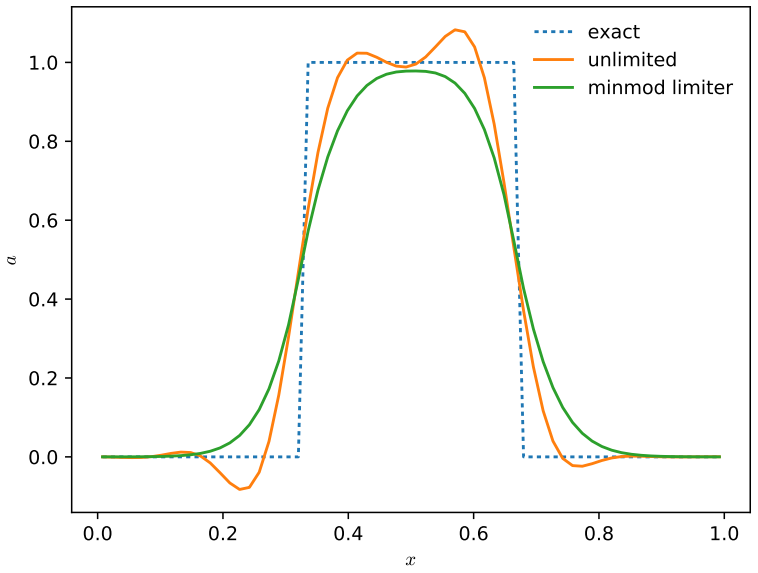

Figure 5.4: Second-order finite volume advection showing the result of advecting a tophat profile through five periods with both unlimited and limited slopes. This calculation used 64 zones and $C = 0.7$.
hydro_examples: `advection.py`

- *get timestep (Eq. 4.5)*
- *compute interface states (Eqs. 5.6 and 5.7)*
- *solve Riemann problem at all interfaces (Eq. 5.10)*
- *do conservative update (Eqs. 5.4 and 5.5)*

Use both the top-hat and Gaussian initial conditions and periodic boundary conditions and compare to the first-order method. See Figure 5.4.

5.2.1 Limiting

The second-order method likely showed some oscillations in the solution, especially for the top-hat initial conditions. *Godunov's theorem* says that any monotonic linear method for advection is first-order accurate (see, e.g., [44]). In this context, monotonic means that no new minima or maxima are introduced. The converse is true too, which suggests that in order to have a second-order accurate method for this linear equation, the algorithm itself must be nonlinear.

Exercise 5.2

To remove the oscillations in practice, we limit the slopes to ensure that no new minima or maxima are introduced during the advection process. There are many choices for limited slopes. A popular one is the minmod *limiter. Here, we construct the slopes in the interface states as:*

$$\left.\frac{\partial a}{\partial x}\right|_i = \texttt{minmod}\left(\frac{a_i - a_{i-1}}{\Delta x}, \frac{a_{i+1} - a_i}{\Delta x}\right) \tag{5.11}$$

instead of Eq. 5.8. with

$$\texttt{minmod}(a,b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } a \cdot b > 0 \\ b & \text{if } |b| < |a| \text{ and } a \cdot b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

Use this slope in your second-order advection code and notice that the oscillations go away—see Figure 5.4.

We can get a feel for what happens with and without limiting pictorially. Figures 5.5 and 5.6 show the evolution of an initial discontinuity with and without limiting. Without limiting, we see an overshoot behind the discontinuity and an undershoot ahead of it—these develop after only a single step. With each additional step, the overshoots and undershoots move further away from the discontinuity. In contrast, the case with limiting shows no over- or undershoots around the initial discontinuity. By the end of the evolution, we see that the profile is much narrower in the limiting case than in the case without limiting.

See the text by LeVeque [46] for alternate choices of limiters. Note: most limiters will have some sort of test on the product of a left-sided and right-sided difference ($a \cdot b$ above)—this is < 0 at an extremum, which is precisely where we want to limit.

A slightly more complex limiter is the MC limiter (monotonized central difference). First we define an extrema test,

$$\xi = (a_{i+1} - a_i) \cdot (a_i - a_{i-1}) \tag{5.13}$$

Then the limited difference is

$$\left.\frac{\partial a}{\partial x}\right|_i = \begin{cases} \min\left[\frac{|a_{i+1}-a_{i-1}|}{2\Delta x}, 2\frac{|a_{i+1}-a_i|}{\Delta x}, 2\frac{|a_i-a_{i-1}|}{\Delta x}\right] \text{sign}(a_{i+1} - a_{i-1}) & \xi > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

Note that a slightly different form of this limiter is presented in [46], where all quantities are in a `minmod`, which appears to limit a bit less. This is second-order accurate for smooth flows.

The main goal of a limiter is to reduce the slope near extrema. Figure 5.7 shows a finite-volume grid with the original data, cell-centered slopes, and MC limited slopes.

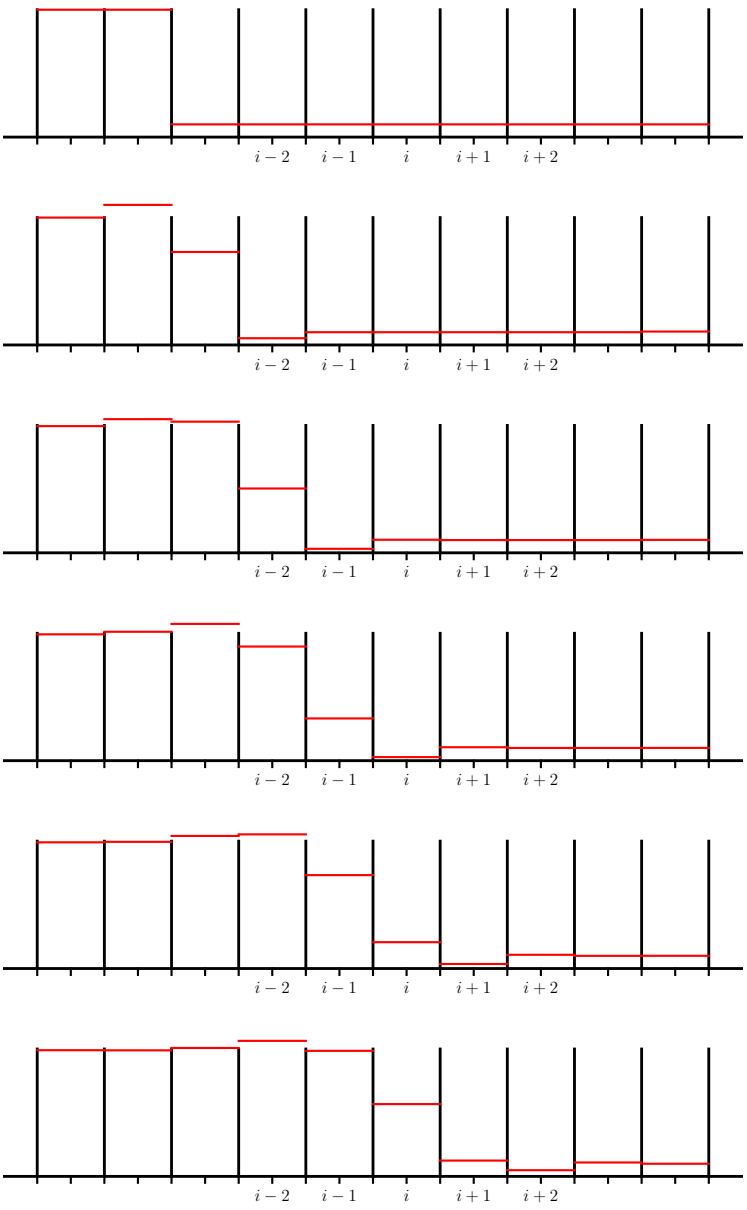

Figure 5.5: Initially discontinuous data evolved for several steps with no limiting. Notice that there are overshoots and undershoots surrounding the discontinuity.

Note that near the strong gradients is where the limiting kicks in. The different limiters are all constructed by enforcing a condition requiring the method to be *total variation diminishing*, or TVD. More details on TVD limiters can be found in [46, 82].

A popular extension of the MC limiter is the 4^{th}-order MC limiter, which is more accurate in smooth flows (this is shown in [23], Eqs. 2.5 and 2.6; and [24], Eq. 191).

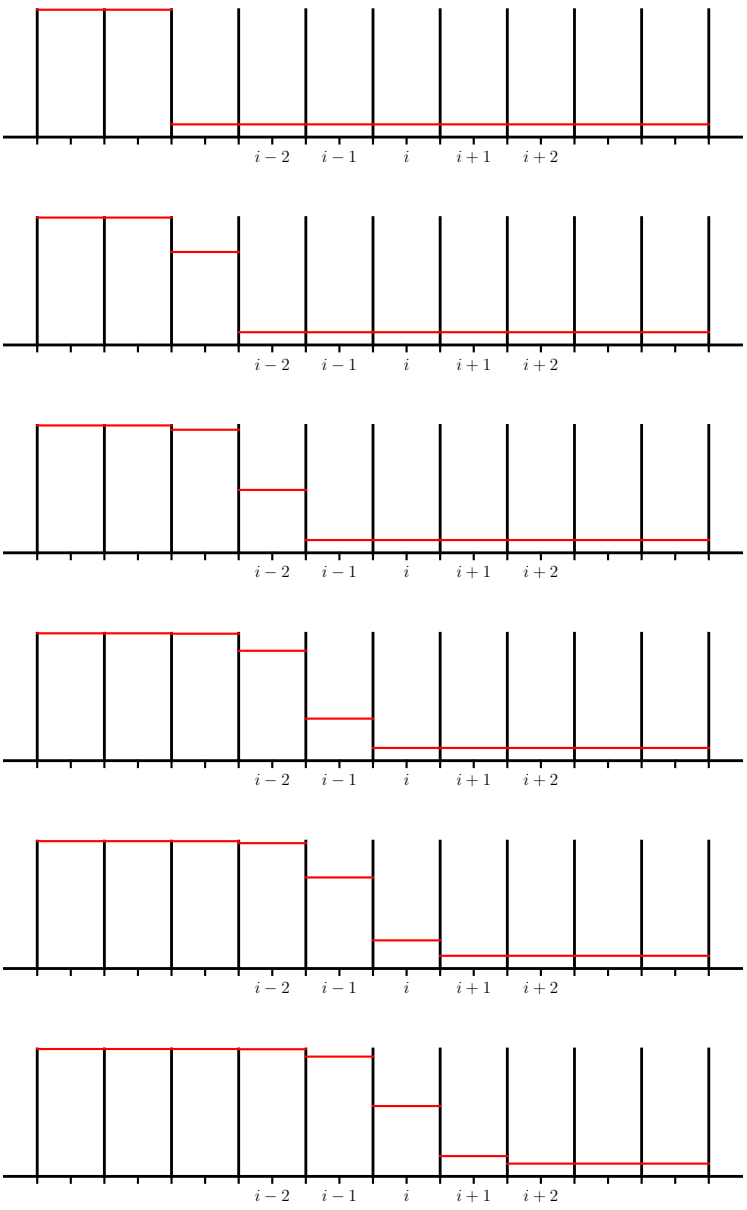

Figure 5.6: Initially discontinuous data evolved for several steps with limiting. Note that unlike the sequence without limiting (Figure 5.5), the discontinuity remains sharper with limiting and there are no over- or undershoots.

Exercise 5.3

Show analytically that if you fully limit the slopes (i.e. set $\partial a/\partial x|_i = 0$, that the second-order method reduces to precisely our first-order finite-difference discretization, Eq. 4.2.

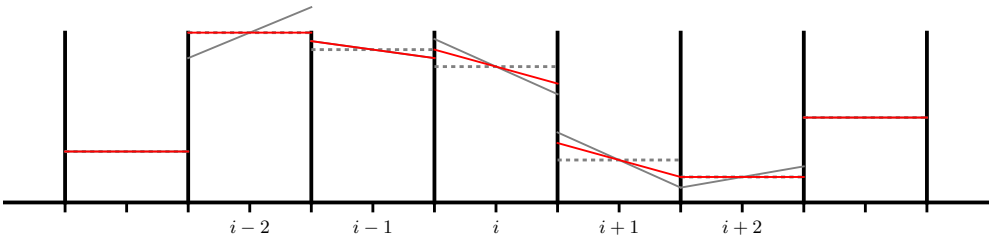

Figure 5.7: A finite-volume grid showing the cell averages (gray, dotted, horizontal lines), unlimited center-difference slopes (gray, solid) and MC limited slopes (red). Note that in zones i and $i+1$, the slopes are limited slightly, so as not to overshoot or undershoot the neighboring cell value. 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.

It is common to express the slope simply as the change in the state variable:

$$\Delta a_i = \left.\frac{\partial a}{\partial x}\right|_i \Delta x \tag{5.15}$$

and to indicate the limited slope as $\overline{\Delta a}_i$.

5.2.2 Reconstruct-evolve-average

Another way to think about these methods is as a reconstruction, evolve, and average (R-E-A) process (see Figure 5.8).

We can write the conservative update as:

$$a_i^{n+1} = a_i^n + \frac{\Delta t}{\Delta x}(ua_{i-1/2}^{n+1/2} - ua_{i+1/2}^{n+1/2}) \tag{5.16}$$

$$= a_i^n + \mathcal{C}(a_{i-1/2}^{n+1/2} - a_{i+1/2}^{n+1/2}) \tag{5.17}$$

If we take $u > 0$, then the Riemann problem will always choose the left state, so we can write this as:

$$a_i^{n+1} = a_i^n + \mathcal{C}\left[\underbrace{\left(a_{i-1}^n + \frac{1}{2}(1-\mathcal{C})\Delta a_{i-1}\right)}_{a_{i-1/2,L}} - \underbrace{\left(a_i^n + \frac{1}{2}(1-\mathcal{C})\Delta a_i\right)}_{a_{i+1/2,L}}\right] \tag{5.18}$$

If we instead look at this via the R-E-A procedure, we write the reconstructed a in each zone in the form of a piecewise linear polynomial

$$a_i(x) = a_i + \frac{\Delta a_i}{\Delta x}(x - x_i) \tag{5.19}$$

Consider zone i. If we are advecting with a CFL number $\mathcal{C}$, then that means that the fraction $\mathcal{C}$ of the zone immediately to the left of the $i-1/2$ interface will advect into

zone i over the timestep. And only the fraction $1 - \mathcal{C}$ in zone i immediately to the right of the interface will stay in that zone. This is indicated by the shaded regions in Figure 5.8.

The average of the quantity a from zone $i-1$ that will advect into zone i is

$$\begin{aligned}
\mathcal{I}_< &= \frac{1}{\mathcal{C}\Delta x} \int_{x_{i-1/2}-\mathcal{C}\Delta x}^{x_{i-1/2}} a_{i-1}(x)dx && (5.20)\\
&= \frac{1}{\mathcal{C}\Delta x} \int_{x_{i-1/2}-\mathcal{C}\Delta x}^{x_{i-1/2}} \left[a_{i-1} + \frac{\Delta a_{i-1}}{\Delta x}(x - x_{i-1})\right] dx && (5.21)\\
&= a_{i-1} + \frac{1}{2}\Delta a_{i-1}(1-\mathcal{C}) && (5.22)
\end{aligned}$$

And the average of the quantity a in zone i that will remain in zone i is

$$\begin{aligned}
\mathcal{I}_> &= \frac{1}{(1-\mathcal{C})\Delta x} \int_{x_{i-1/2}}^{x_{i-1/2}+(1-\mathcal{C})\Delta x} a_i(x)dx && (5.23)\\
&= \frac{1}{(1-\mathcal{C})\Delta x} \int_{x_{i-1/2}}^{x_{i-1/2}+(1-\mathcal{C})\Delta x} \left[a_i + \frac{\Delta a_i}{\Delta x}(x - x_i)\right] dx && (5.24)\\
&= a_i - \frac{1}{2}\Delta a_i \mathcal{C} && (5.25)
\end{aligned}$$

The final part of the R-E-A procedure is to average the over the advected profiles in the new cell. The weighted average of the amount brought in from the left of the interface and that that remains in the cell is

$$\begin{aligned}
a_i^{n+1} &= \mathcal{C}\mathcal{I}_< + (1-\mathcal{C})\mathcal{I}_> && (5.26)\\
&= \mathcal{C}\left[a_{i-1}^n + \frac{1}{2}\Delta a_{i-1}(1-\mathcal{C})\right] + (1-\mathcal{C})\left[a_i^n - \frac{1}{2}\Delta a_i \mathcal{C}\right] && (5.27)\\
&= a_i^n + \mathcal{C}\left[a_{i-1}^n + \frac{1}{2}(1-\mathcal{C})\Delta a_{i-1}\right] - \mathcal{C}\left[a_i^n + \frac{1}{2}(1-\mathcal{C})\Delta a_i\right] && (5.28)
\end{aligned}$$

This is identical to Eq. 5.18. This demonstrates that the R-E-A procedure is equivalent to our reconstruction, prediction of the interface states, solving the Riemann problem, and doing the conservative flux update.

Exercise 5.4

Run the first-order solver for several different values of Δx, each a factor of 2 smaller than the previous. Compute ϵ for each resolution and observe that it converges in a first-order fashion (i.e. ϵ decreases by 2 when we decrease Δx by a factor of 2).
Do the same with the second-order solver and observe that it converges as second-order. However: you may find less than second-order if your initial conditions have discontinuities and you are limiting. Figure 5.9

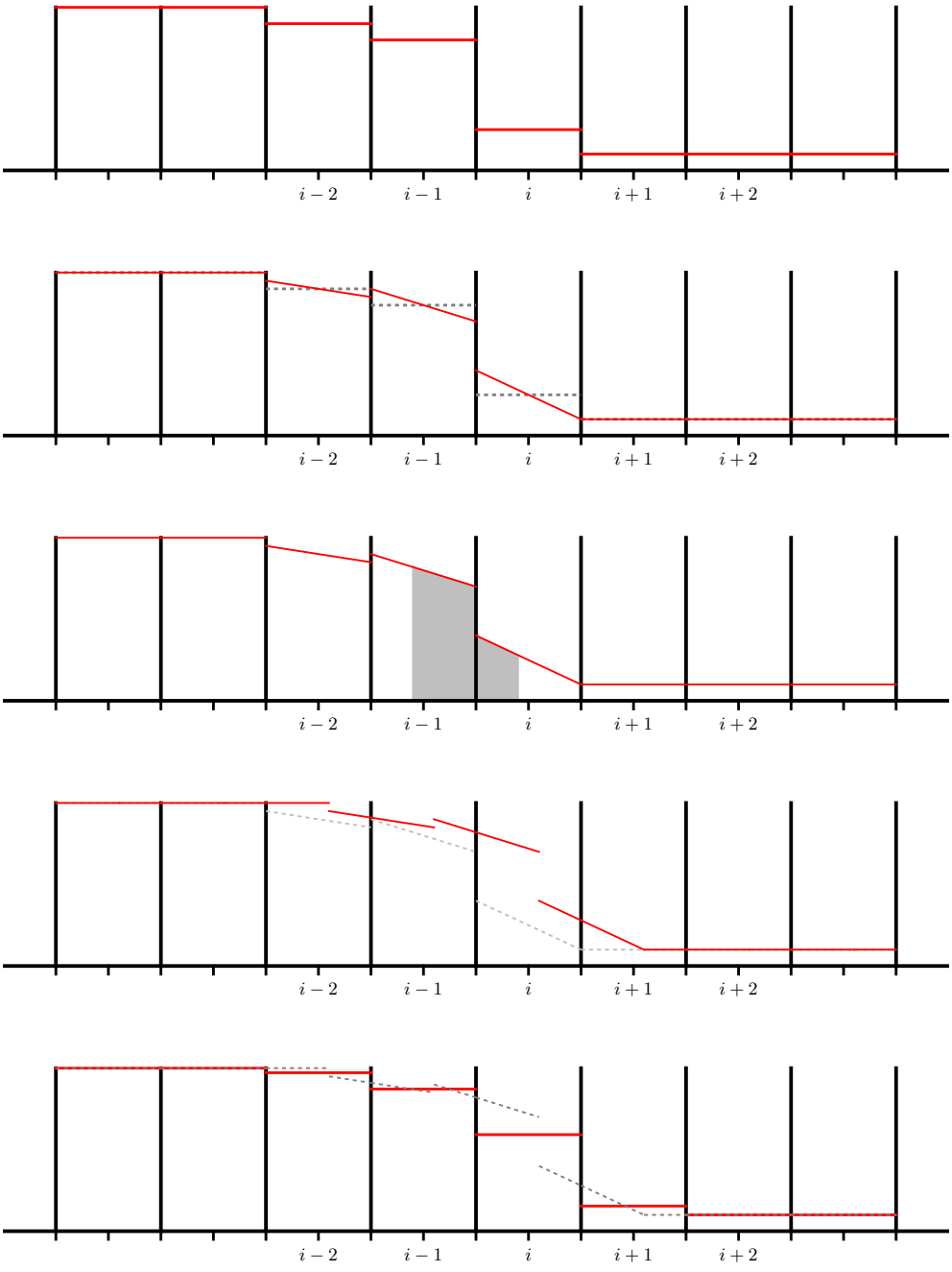

Figure 5.8: Reconstruct-Evolve-Average. The top panel shows the original cell-average data. The second panel shows the (limited) piecewise linear reconstruction of the data. Assuming a CFL number of 0.6 and advection to the right, the shaded regions in the third panel show the data that will wind up in cell i after advecting for a single step. The fourth panel shows the piecewise-linear data advected to the right by 0.6 of a cell-width (corresponding to a CFL of 0.6). The final panel shows the new averages of the data, constructed by averaging the advected piecewise linear data in each cell.

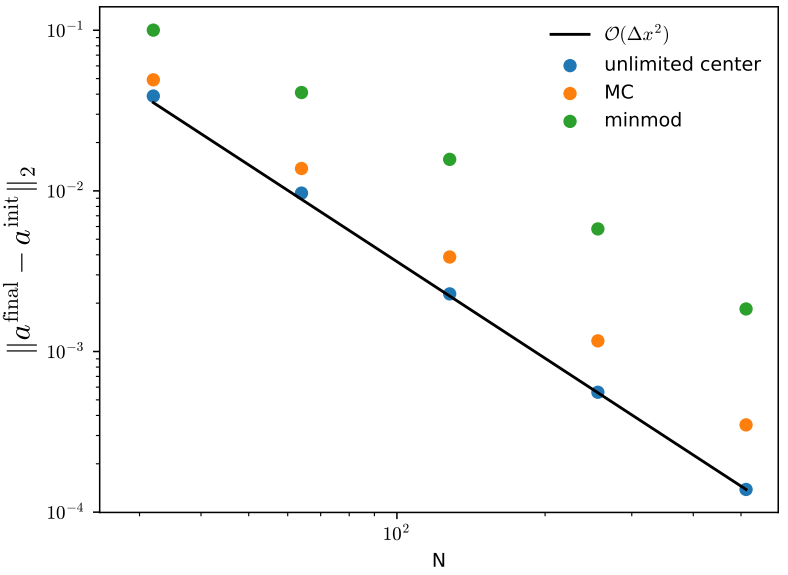

Figure 5.9: Convergence for the second-order finite-volume method with no limiting, MC, and minmod limiting advecting a Gaussian initial profile with $C = 0.8$ for 5 periods.
hydro_examples: `advection.py`

> *shows the convergence of the method with no limiting, MC, and minmod limiting, $C = 0.8$, and a Gaussian initial condition for 5 periods.*

As seen in figure 5.9, the choice of limiters can have a great effect on the accuracy. Figure 5.10 shows the Gaussian and tophat profiles with several different limiters.

5.3 Method of lines approach

In the above constructions, we computed a time-centered flux by predicting the interface state to the half-time (by Taylor-expanding in time through $\Delta t/2$). Instead, we can recognize that with the spatial discretization done in Eq. **??**, we are left with an ordinary differential equation in time that can then be solved using standard ODE techniques.

Substituting in the flux as $f(a) = ua$, we have the ODE:

$$\frac{\partial a_i}{\partial t} = -\frac{ua_{i+1/2} - ua_{i-1/2}}{\Delta x} \tag{5.29}$$

Note that there is no time-level indicated on the righthand side. We find the interface values of a by interpolating in space only. For a second-order accurate method, we

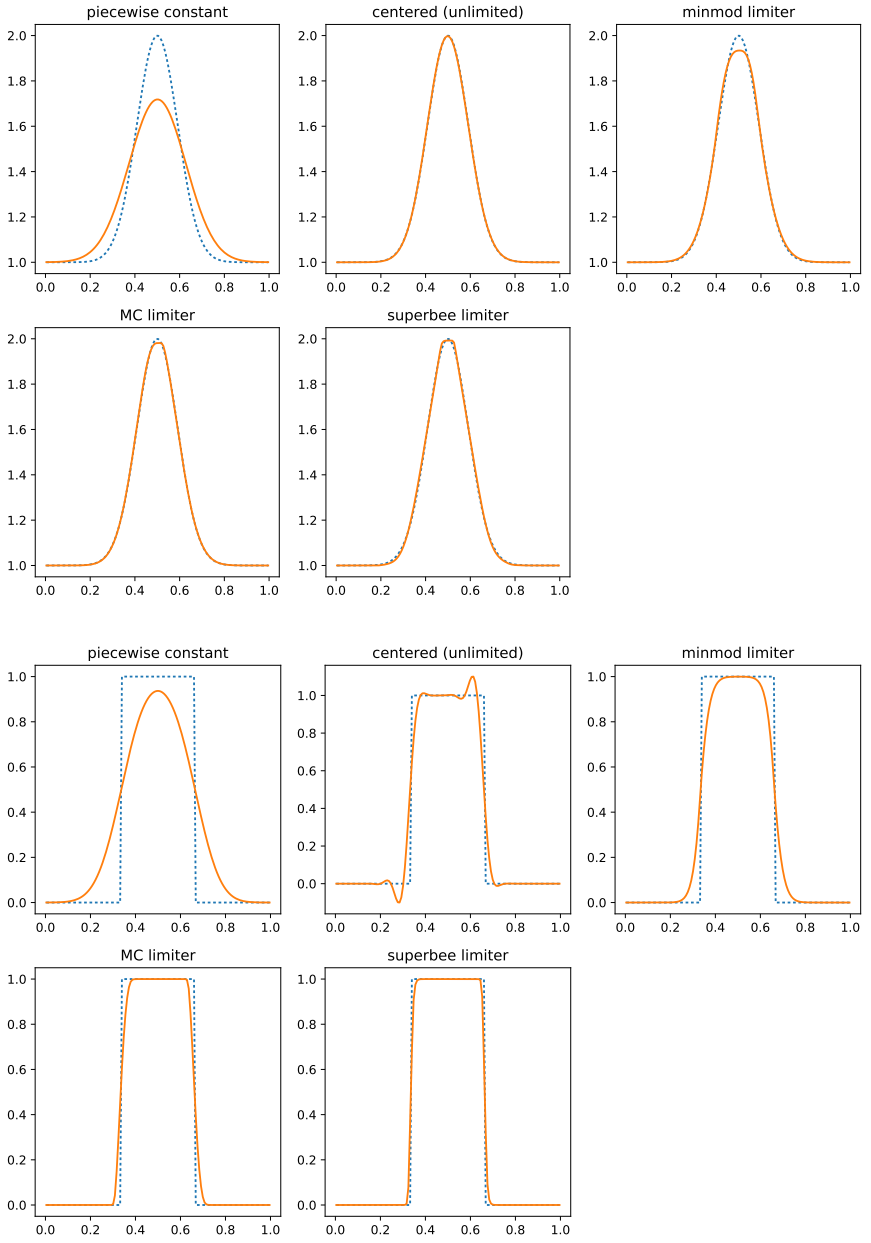

Figure 5.10: The effect of different limiters on the evolution of a Gaussian initial profile (top) and a tophat initial profile (bottom), using $C = 0.8$ and 5 periods.
hydro_examples: `advection.py`

can use a piecewise linear reconstruction (see Figure 5.11), and evaluate the interface

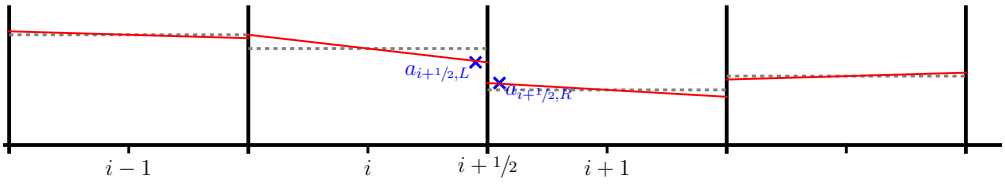

Figure 5.11: Piecewise linear reconstruction of the zones showing the interface states used with the method of lines approach.

states as simply the point on the reconstructed line on the interface, e.g.,

$$a_{i+1/2,L} = a_i + \frac{1}{2}\Delta a_i \tag{5.30}$$

$$a_{i+1/2,R} = a_{i+1} - \frac{1}{2}\Delta a_{i+1} \tag{5.31}$$

The same limiting ideas discussed above apply here. As usual, we resolve the left-right degeneracy on the interface by solving the Riemann problem:

$$a_{i+1/2,j} = \mathcal{R}(a_{i+1/2,j,L}, a_{i+1/2,j,R}) \tag{5.32}$$

Once we have this interface state, we can integrate the ODE. A Runge-Kutta method works fine here (it should be atleast second-order to match the spatial accuracy).

5.4 Multi-dimensional advection

The two-dimensional linear advection equation is:

$$a_t + ua_x + va_y = 0 \tag{5.33}$$

where u is the velocity in the x-direction and v is the velocity in the y-direction. We denote the average of $a(x, y, t)$ in a zone i, j as $a_{i,j}$. Here, i is the index in the x-direction and j is the index in the y-direction. A 2-d grid is shown in Figure 5.12. Just as in the one-dimensional case, we will extend the domain with a perimeter of ghost cells to set the boundary conditions.

To derive the finite-volume form of the update, we start by writing this in conservative form. Since u and v are constant, we can move them inside the divergences:

$$a_t + (ua)_x + (va)_y = 0 \tag{5.34}$$

This is the form we will integrate over zones. As before, we will define the average of a in a zone by integrating it over the volume:

$$a_{i,j} = \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} a(x, y, t)\, dx\, dy \tag{5.35}$$

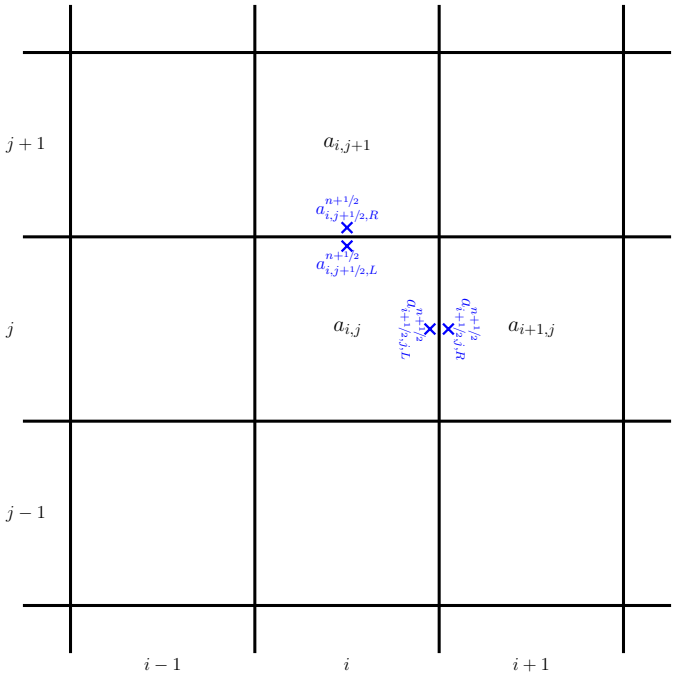

Figure 5.12: A simple 2-d grid with the zone-centered indexes. The $\times$s mark the left and right interface states at the upper edge of the i, j zone in each coordinate direction. For a finite-volume discretization, $a_{i,j}$ represents the average of $a(x, y)$ over the zone.

Integrating Eq. 5.34 over x and y, we have:

$$\frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} a_t \, dx \, dy = -\frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} (ua)_x \, dx \, dy - \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} (va)_y \, dx \, dy \tag{5.36}$$

or using the divergence theorem,

$$\frac{\partial a_{i,j}}{\partial t} = -\frac{1}{\Delta x \Delta y} \int_{y_{j-1/2}}^{y_{j+1/2}} \left\{ (ua)_{i+1/2,j} - (ua)_{i-1/2,j} \right\} dy - \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \left\{ (va)_{i,j+1/2} - (va)_{i,j-1/2} \right\} dx \tag{5.37}$$

Now we integrate over time—the left side of our expression becomes just the differ-

ence between the new and old state.

$$a_{i,j}^{n+1} - a_{i,j}^{n} = -\frac{1}{\Delta x \Delta y}\int_{t^n}^{t^{n+1}}\int_{y_{j-1/2}}^{y_{j+1/2}} \left\{(ua)_{i+1/2,j} - (ua)_{i-1/2,j}\right\} dydt - \frac{1}{\Delta x \Delta y}\int_{t^n}^{t^{n+1}}\int_{x_{i-1/2}}^{x_{i+1/2}} \left\{(va)_{i,j+1/2} - (va)_{i,j-1/2}\right\} dxdt \tag{5.38}$$

We define the flux through the interface as the average over the face of that interface and time:

- x-face:

$$\left\langle (ua)_{i+1/2,j} \right\rangle_{(t)} = \frac{1}{\Delta y \Delta t}\int_{t^n}^{t^{n+1}}\int_{y_{j-1/2}}^{y_{j+1/2}} (ua)_{i+1/2,j}\, dydt \tag{5.39}$$

- y-face

$$\left\langle (va)_{i,j+1/2} \right\rangle_{(t)} = \frac{1}{\Delta x \Delta t}\int_{t^n}^{t^{n+1}}\int_{x_{i-1/2}}^{x_{i+1/2}} (va)_{i,j+1/2}\, dxdt \tag{5.40}$$

where $\langle . \rangle_{(t)}$ denotes the time-average over the face. For a second-order accurate method in time, we replace the average in time with the flux at the midpoint in time and the average over the face with the flux at the center of the face:

$$\left\langle (ua)_{i+1/2,j} \right\rangle_{(t)} \approx (ua)_{i+1/2,j}^{n+1/2} \tag{5.41}$$

Then we have:

$$a_{i,j}^{n+1} = a_{i,j}^{n} - \Delta t \left[\frac{(ua)_{i+1/2,j}^{n+1/2} - (ua)_{i-1/2,j}^{n+1/2}}{\Delta x} + \frac{(va)_{i,j+1/2}^{n+1/2} - (va)_{i,j-1/2}^{n+1/2}}{\Delta y} \right] \tag{5.42}$$

For the advection problem, since u and v are constant, we need to find the interface states of a alone. There are two methods for computing these interface states, $a_{i\pm1/2,j}^{n+1/2}$ on x-interfaces and $a_{i,j\pm1/2}^{n+1/2}$ on y-interfaces: dimensionally split and unsplit. Dimensionally split methods are easier to code, since each dimension is operated on independent of the others, so you can simply call a one-dimensional method for each direction. Unsplit methods, however, are more accurate and less susceptible to grid effects.

5.4.1 Dimensionally split

In a split method, we update the state in each coordinate direction independently. This is simple and a straightforward way to use one-dimensional methods in multi-d. To be second-order accurate in time, we do *Strang splitting* [77], where we alternate

the order of the dimensional updates each timestep. An update through Δt consists of x and y sweeps and appears as:

$$\frac{a^\star_{i,j} - a^n_{i,j}}{\Delta t} = -\frac{ua^{n+1/2}_{i+1/2,j} - ua^{n+1/2}_{i-1/2,j}}{\Delta x} \tag{5.43}$$

$$\frac{a^{n+1}_{i,j} - a^\star_{i,j}}{\Delta t} = -\frac{va^{\star,n+1/2}_{i,j+1/2} - va^{\star,n+1/2}_{i,j-1/2}}{\Delta y} \tag{5.44}$$

Here, Eq. 5.43 is the update in the x-direction. In constructing the interface states, $a^{n+1/2}_{i+1/2,j}$ and $a^{n+1/2}_{i-1/2,j}$, we do the exact same procedure as the one-dimensional case, constructing the left and right states at each interface and then solving the same Riemann problem to find the unique state on the interface. Each dimensional sweep is done without knowledge of the other dimensions. For example, in the x-update, we are solving:

$$a_t + ua_x = 0 \tag{5.45}$$

and in the y-update, we are solving:

$$a_t + va_y = 0 \tag{5.46}$$

The construction of the interface states largely mimics the one-dimensional case (Eq. 5.6 and 5.7). For example, the $a^{n+1/2}_{i+1/2,j,L}$ state is:

$$\begin{aligned} a^{n+1/2}_{i+1/2,j,L} &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i,j} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i,j}\right) + \dots \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i,j} + \dots \end{aligned} \tag{5.47}$$

Notice that when substituting for $\partial a/\partial t$, we use the one-dimensional split version of the advection equation (Eq. 5.45) instead of the full multi-dimensional equation. There are no y-direction terms that come into play in the split method when considering the x-direction.

The x-update (Eq. 5.43) updates the state only accounting for the x-fluxes—we denote this intermediate state with the '$\star$' superscript. For the y-update, we construct our interface states in the analogous way as in the x-direction, but begin with the '$\star$' state instead of the old-time state. In this fashion, the y-update 'sees' the result of the x-update and couples things together.

To achieve second-order accuracy in time, it is necessary to alternate the directions of the sweeps each timestep, i.e., x-y then y-x. Furthermore, this pair of sweeps should use the same timestep, Δt.

5.4.2 Unsplit multi-dimensional advection

The unsplit case differs from the dimensionally split case in two ways: (1) in predicting the interface states, we use knowledge of the flow in the transverse direction, and (2), only a single conservative update is done per timestep, with all directions updating simultaneously. See [24] for more details. This idea is sometimes called the "corner transport upwind" or CTU method.

The construction of the $a^{n+1/2}_{i+1/2,j,L}$ interface state appears as

$$\begin{aligned} a^{n+1/2}_{i+1/2,j,L} &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i,j} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i,j} - v\left.\frac{\partial a}{\partial y}\right|_{i,j}\right) + \ldots \\ &= \underbrace{a^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i,j}}_{\hat{a}^{n+1/2}_{i+1/2,j,L}} \underbrace{-\frac{\Delta t}{2}v\left.\frac{\partial a}{\partial y}\right|_{i,j}}_{\text{"transverse flux difference"}} + \ldots \end{aligned} \tag{5.48}$$

The main difference between the split and unsplit interface states is the explicitly appearance of the "transverse flux difference" in the unsplit interface state. We rewrite this as:

$$a^{n+1/2}_{i+1/2,j,L} = \hat{a}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}v\left.\frac{\partial a}{\partial y}\right|_{i,j} \tag{5.49}$$

Here, the $\hat{a}^{n+1/2}_{i+1/2,j,L}$ term is just the normal prediction without considering the transverse direction (e.g., Eq. 5.47). The basic update procedure is:

- Construct the normal predictor term, $\hat{a}^{n+1/2}_{i+1/2,j,L}$, in a fashion identical to the one-dimensional and split methods. We compute these one-dimensional $\hat{a}$'s at the left and right every interface in both coordinate directions. Note that these states are still one-dimensional, since we have not used any information from the transverse direction in their computation.

- Solve a Riemann problem at each of these interfaces:

$$a^T_{i+1/2,j} = \mathcal{R}(\hat{a}^{n+1/2}_{i+1/2,j,L}, \hat{a}^{n+1/2}_{i+1/2,j,R}) \tag{5.50}$$

$$a^T_{i,j+1/2} = \mathcal{R}(\hat{a}^{n+1/2}_{i,j+1/2,L}, \hat{a}^{n+1/2}_{i,j+1/2,R}) \tag{5.51}$$

$$\tag{5.52}$$

  These states are given the '^{T}' superscript since they are the states that are used in computing the transverse flux difference.

- Correct the previously computed normal interface states (the $\hat{a}$'s) with the transverse flux difference:

$$a^{n+1/2}_{i+1/2,j,L} = \hat{a}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}v\frac{a^T_{i,j+1/2} - a^T_{i,j-1/2}}{\Delta y} \tag{5.53}$$

Notice that the fluxes that are differenced for the left state are those that are transverse, but to the left of the interface. Similarly, for the right state, it would be those that are transverse, but to the right of the interface:

$$a^{n+1/2}_{i+1/2,j,R} = \hat{a}^{n+1/2}_{i+1/2,j,R} - \frac{\Delta t}{2} v \frac{a^T_{i+1,j+1/2} - a^T_{i+1,j-1/2}}{\Delta y} \tag{5.54}$$

A similar procedure happens at the y-interfaces.

Figure 5.13 illustrates the steps involved in the construction of the $a^{n+1/2}_{i+1/2,j,L}$ state.

Once all of the full states (normal prediction + transverse flux difference) are computed to the left and right of all the interfaces (x and y), we solve another Riemann problem to find the final state on each interface.

$$a^{n+1/2}_{i+1/2,j} = \mathcal{R}(a^{n+1/2}_{i+1/2,j,L}, a^{n+1/2}_{i+1/2,j,R}) \tag{5.55}$$

The final conservative update is then done via Eq. 5.38.

See [24] for more details on this unsplit method, and [66] for details of the extension to 3-d.

Figures 5.14 and 5.15 show the advection of a smooth Gaussian and a discontinuous tophat diagonally on a coarse 32×32 zone domain. Each show a diffusion of the initial function, similar to what we saw in 1-d. In the tophat, we also see a slight undershoot on the trailing side of the tophat after one period.

5.4.3 Timestep limiter for multi-dimensions

The timestep criterion we used in one-dimension (Eq. 4.5) needs to be generalized for multi-dimensions. For the dimensionally split case, since we are basically piecing together two one-dimensional sweeps, the timestep limit is usually taken as:

$$\Delta t = \mathcal{C} \min \left\{ \frac{\Delta x}{|u|}, \frac{\Delta y}{|v|} \right\} \tag{5.56}$$

This is, for example, the construction that is used with the dimensionally-split hydrodynamics solver in the original Flash code [34]*. Stability requires picking a CFL number, $\mathcal{C} \leq 1$.

A general extension of the timestep restriction for a fully-multi-dimensional algorithm is often written in the form (see, e.g., [71])

$$\Delta t = \mathcal{C} \left(\sum_{i=1}^{d} \frac{|\mathbf{U} \cdot \mathbf{e}_d|}{\Delta x_d} \right)^{-1} \tag{5.57}$$

*although, the paper does not explicitly write this out

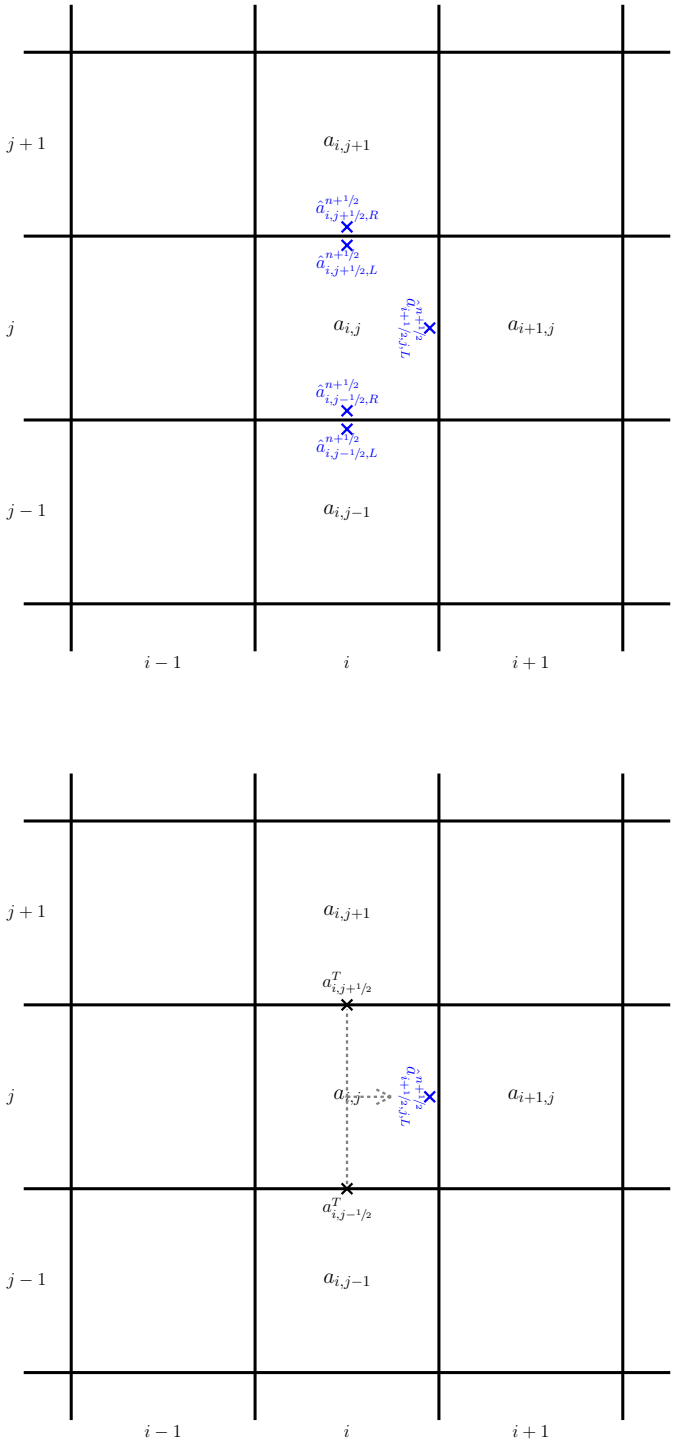

Figure 5.13: The construction of the $a^{n+1/2}_{i+1/2,j,L}$ state. Top: first we compute the $\hat{a}$'s—here we show all of the $\hat{a}$'s that will be used in computing the full left interface state at $(i+1/2, j)$. Bottom: after the transverse Riemann solves, we have the two transverse states ($a^T_{i,j+1/2}$ and $a^T_{i,j-1/2}$) that will be differenced and used to correct $\hat{a}^{n+1/2}_{i+1/2,j,L}$ (illustrated by the dotted lines) to make $a^{n+1/2}_{i+1/2,j,L}$.

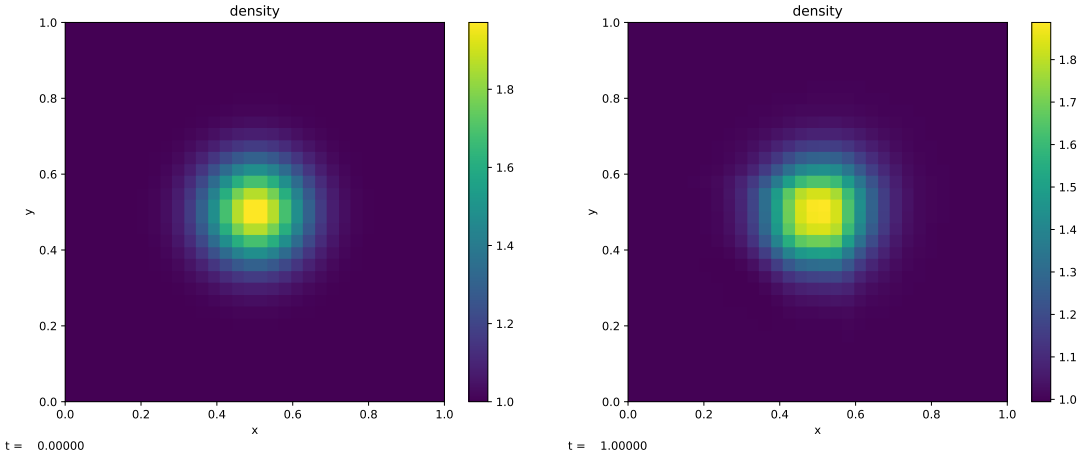

Figure 5.14: Advection of a Gaussian profile with $u = v = 1$ for one period on a 32×32 grid, with $\mathcal{C} = 0.8$. This was run with pyro as `./pyro.py advection smooth inputs.smooth`

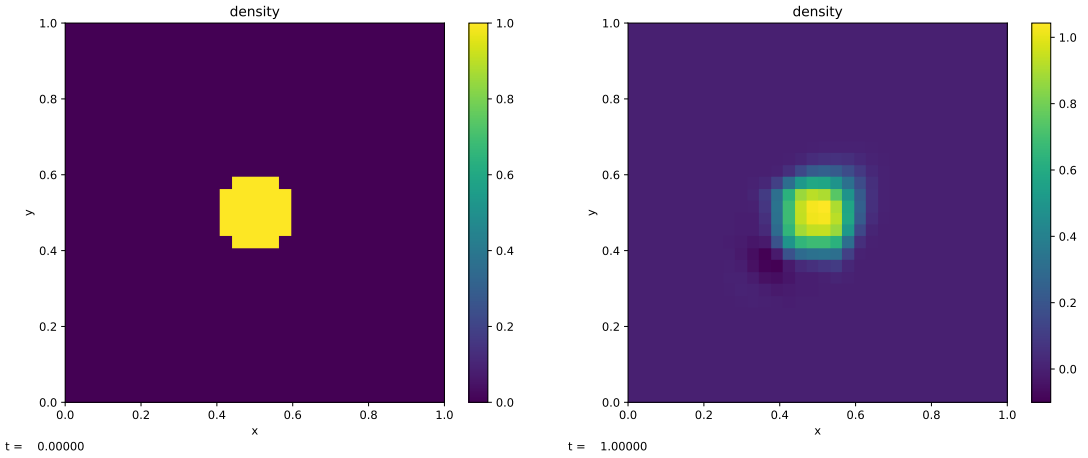

Figure 5.15: Advection of a tophat function with $u = v = 1$ for one period on a 32×32 grid, with $\mathcal{C} = 0.8$. This was run as `./pyro.py advection tophat inputs.tophat`

where d is the coordinate direction, $\mathbf{U} = (u, v)$ is the velocity vector, and $\mathbf{e}_d$ is the unit vector in direction d. For these methods $\mathcal{C} \lesssim 1$. This is usually more restrictive than Eq. 5.56.

For the CTU method described above, [24] argues that the inclusion of the transverse information removes some of the some of the instability inherent in simpler schemes, allowing for a larger timestep, restricted by Eq. 5.56.

Note, because of the different form of the timestep limiters here, it is not enough to simply cite a CFL number when describing results, you must also explain which form of the timestep calculation (Eq. 5.56 or Eq. 5.57) is being used.

5.4.4 Method-of-lines in multi-dimensions

The multidimensional version of method-of-lines integration follows the ideas from the one-dimensional case. We can start with Eq. 5.37, and define the fluxes over though a face as:

- x-face:

$$\langle (ua)_{i+1/2,j} \rangle = \frac{1}{\Delta y} \int_{y_{j-1/2}}^{y_{j+1/2}} (ua)_{i+1/2,j}\, dy \tag{5.58}$$

- y-face

$$\langle (va)_{i,j+1/2} \rangle = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} (va)_{i,j+1/2}\, dx \tag{5.59}$$

These are similar to the expressions above, except there is no integral over time. Again, to second order in space, we can just use the flux evaluated at the midpoint in the face:

$$\langle (ua)_{i+1/2,j} \rangle \approx (ua)_{i+1/2,j} \tag{5.60}$$

The result is the ODE:

$$\frac{da_{i,j}}{dt} = -\frac{(ua)_{i+1/2,j} - (ua)_{i-1/2,j}}{\Delta x} - \frac{(va)_{i,j+1/2} - (va)_{i,j-1/2}}{\Delta y} \tag{5.61}$$

For the interface states, we separately construct slopes in each dimension: $\Delta^{(x)} a_{i,j}$ is the slope in the x-direction, and $\Delta^{(y)} a_{i,j}$ is the slope in the y-direction. Then the interface states are:

$$a_{i+1/2,j,L} = a_{i,j} + \frac{1}{2} \Delta a_{i,j}^{(x)} \tag{5.62}$$

$$a_{i+1/2,j,R} = a_{i+1,j} - \frac{1}{2} \Delta a_{i+1,j}^{(x)} \tag{5.63}$$

and

$$a_{i,j+1/2,L} = a_{i,j} + \frac{1}{2} \Delta a_{i,j}^{(y)} \tag{5.64}$$

$$a_{i,j+1/2,R} = a_{i,j+1} - \frac{1}{2} \Delta a_{i,j+1}^{(y)} \tag{5.65}$$

With these interface states, we have the method for evaluating the righthand side of our ODE. We can then evolve the ODE using a Runge-Kutta integration method. At each stage of the RK integration, we do the same construction of the interface states, solve the Riemann problem, etc. Notice that unlike the directionally-split method, we are doing the update in both directions at the same time—we are relying on the ODE integrator to do the coupling in the different directions for us.

This construction is a lot simpler than the unsplit method described previously. The cost of this simplicity is a reduced range of stability.

Performaing the stability analysis here is much more complicated. For simplicity, consider a simple first-order upwind scheme in two-dimensions. Higher order time integrators (like RK4) will have separate stages that look like a first-order-in-time update, so this stability analysis can be thought of as applying to a single stage. Our difference equation is:

$$\frac{a_{i,j}^{n+1} - a_{i,j}^{n}}{\Delta t} = -u\frac{a_{i,j}^{n} - a_{i-1,j}^{n}}{\Delta x} - v\frac{a_{i,j}^{n} - a_{i,j-1}^{n}}{\Delta y} \tag{5.66}$$

We introduce a single-mode Fourier function, as before, but this time with a mode in each direction:

$$a_{i,j}^{n} = A^{n} e^{Ii\theta} e^{Ij\phi} \tag{5.67}$$

Next, to simplify things, we'll assume that $u = v$, and $\Delta x = \Delta y$, then $\mathcal{C} = u\Delta t/\Delta x = v\Delta t/\Delta y$. Inserting this into our difference equation gives:

$$\frac{A^{n+1}}{A^{n}} = 1 - \mathcal{C}\left(1 - e^{-I\theta}\right) - \mathcal{C}\left(1 - e^{-I\phi}\right) \tag{5.68}$$

This form looks very close to the one-dimensional case, but it is difficult to analytically find the values of $\mathcal{C}$, but we can numerically find the maximum $|A^{n+1}/A^{n}|^2$ for $\theta \in [0, 2\pi]$ and $\phi \in [0, 2\pi]$ as a function of $\mathcal{C}$. In doing this, we find that this discretization is unstable ($|A^{n+1}/A^{n}|^2 > 1$) for $\mathcal{C} > 1/2$[†].

This restriction is $1/d$, where d is the dimensionality, and it is analogous to the difference between the timestep limits for stability from the more generous Eq. 5.56 to the more restrictive Eq. 5.57. This reduced $\mathcal{C}$ for stability is discussed in [71] (see section 4), and is also discussed in [52, 81]. Different reconstruction and integration methods can vary this limit some, for instance, the fourth-order method in [52] allows for $\Delta t \sum_{i=1}^{d} (|\mathbf{U} \cdot \mathbf{e}_d|)/\Delta x_d \lesssim 1.4$. Total variation diminishing Runge-Kutta methods are popular for this application [39].

5.5 High-Order Finite difference methods

Chapter 4 introduced numerical methods for the linear advection equation

$$a_t + ua_x = 0 \tag{5.69}$$

that were first or second order in space. The advantages of using numerical methods that are higher order can be large, especially when in regions where the flow is smooth. However, these must be balanced against the additional computational cost and complexity. In addition, many higher order methods do no better than second order methods in cases with discontinuities.

[†]See the script cfl.py

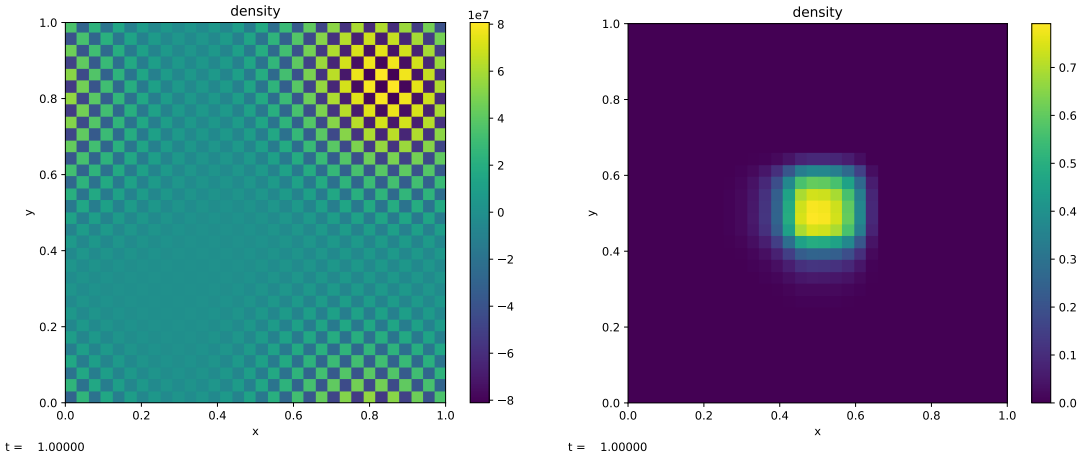

Figure 5.16: Advection of a tophat function using the method-of-lines approach with 4th-order Runge-Kutta time integration. We use $u = v = 1$ for one period on a 32×32 grid. On the left is $\mathcal{C} = 0.8$—this is clearly unstable. On the right is $\mathcal{C} = 0.4$, which is stable. This was run as `./pyro.py advection_rk tophat inputs.tophat`

5.5.1 The problem with higher-order finite volume methods

The finite volume approach outlined in chapter 3 matches the physics of hyperbolic balance laws perfectly, by maintaining the conservation of appropriate quantities using the fluxes through the surfaces of each control volume or computational cell. However, this leads to significant computational costs and complexities when going beyond second order methods.

The inter-cell fluxes are computed by integrals over the face of the computational cell. In one dimension the face is a single point and the integral needs the flux evaluated at a single point, as in equation 3.14. In higher dimensions the face is one dimensional (e.g., a line) or two dimensional (e.g., a square) and so the integral must be approximated by evaluating the flux at *multiple* points. If we use Gauss quadrature, this means that a third order method would require at least two flux evaluations per face in two dimensions, and typically four evaluations per face in three dimensions.

5.5.2 Finite differences

The conservative finite difference method starts from the endpoint for finite volume methods, by considering the update formula (3.14)

$$\frac{\partial \langle \boldsymbol{U} \rangle_i}{\partial t} = -\frac{1}{\Delta x} \left\{ \mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2} \right\}. \tag{3.14}$$

In the finite difference method a interpretation used is different. We are now thinking of

$$\frac{1}{\Delta x} \left\{ \mathbf{F}_{i+1/2} - \mathbf{F}_{i-1/2} \right\} \tag{5.70}$$

as representing a direct approximation to the derivative of the flux at x_i. In particular, we are *not* thinking of $\mathbf{F}_{i+1/2}$ as being directly linked to the flux. Crucially the update still ensures global conservation, as the "flux" term is re-used for neighbouring points in the same way as in the finite volume case.

When we combine this viewpoint with the Method of Lines approach in section 5.3, the implementation of a high-order finite boils down to

- using a high-order time integration scheme (such as a high order Runge-Kutta method);
- finding a way that the finite difference in equation (5.70) approximates the derivative of the flux both stably, and to a sufficiently high order.

5.5.3 WENO reconstruction

For the advection equation with constant advection speed u, our higher-order method requires us to find a high-order approximation to $a_{i+1/2}$. If we considered the five points $a_j, j \in [i-2, \ldots, i+2]$ then we could use the information from all five of these points to approximate $a_{i+1/2}$ as

$$a_{i+1/2}^{(\text{Fifth})} = \tfrac{1}{60}\left(2\langle a\rangle_{i-2} - 13\langle a\rangle_{i-1} + 47\langle a\rangle_i + 27\langle a\rangle_{i+1} - 3\langle a\rangle_{i+2}\right) + \mathcal{O}(\Delta x^5). \tag{5.71}$$

Alternatively we could use only three of the points, in three different ways:

$$a_{i+1/2}^{(\text{Third},2)} = \tfrac{1}{6}\left(2\langle a\rangle_{i-2} - 7\langle a\rangle_{i-1} + 11\langle a\rangle_i\right) + \mathcal{O}(\Delta x^3), \tag{5.72a}$$

$$a_{i+1/2}^{(\text{Third},1)} = \tfrac{1}{6}\left(-\langle a\rangle_{i-1} + 5\langle a\rangle_i + 2\langle a\rangle_{i+1}\right) + \mathcal{O}(\Delta x^3), \tag{5.72b}$$

$$a_{i+1/2}^{(\text{Third},0)} = \tfrac{1}{6}\left(2\langle a\rangle_i + 5\langle a\rangle_{i+1} - \langle a\rangle_{i+2}\right) + \mathcal{O}(\Delta x^3). \tag{5.72c}$$

Which is better? The five point stencil gives higher order accuracy, but includes information from both sides of the point. This ignores information from the characteristics and, near discontinuities, will lead to Gibbs oscillations. The three point stencils give lower accuracy. However, they give us the flexibility to choose a stencil to avoid discontinuities, possibly using the characteristic information. The advantages of the higher-order methods are illustrated by reconstructing a smooth function in figure 5.17. The disadvantages of using a fixed stencil are illustrated by reconstructing a non-smooth function in figure 5.18.

Exercise 5.5

Check that equations (5.71) *and* (5.72) *give the claimed order of accuracy results.*

The method of choosing the "best" stencil (to avoid discontinuities as far as possible) is called *Essentially Non-Oscillatory* (ENO) reconstruction. It is not as accurate as it

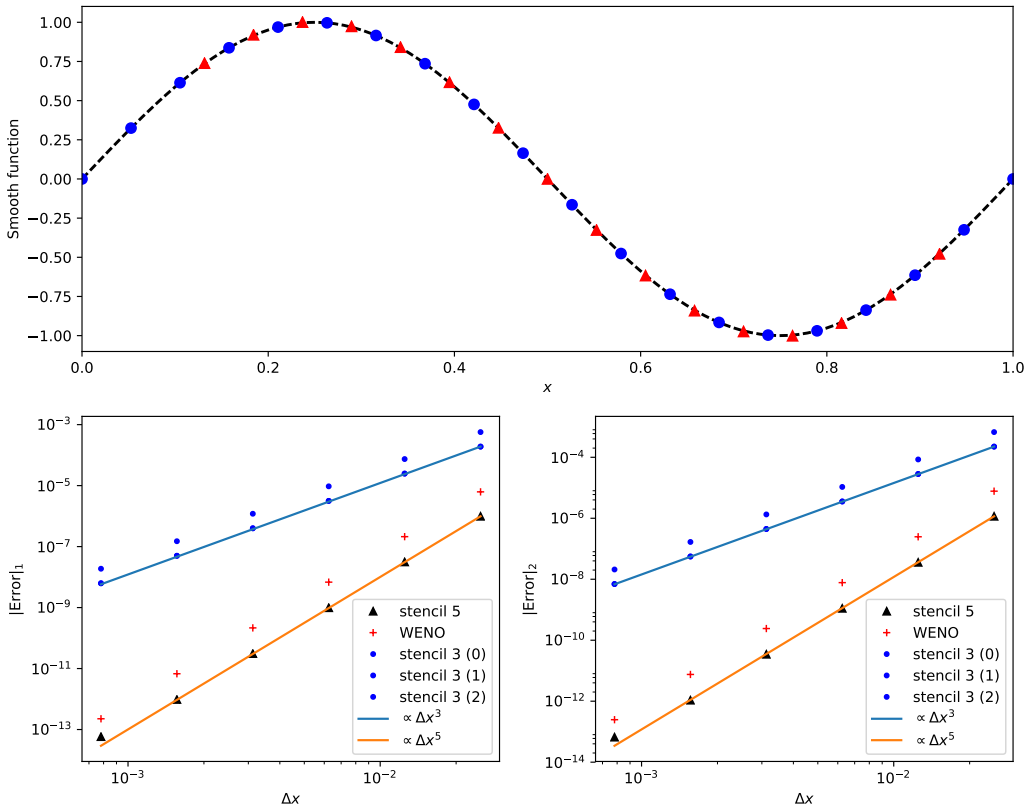

Figure 5.17: A smooth sine function is reconstructed on the unit interval from the (integral-average) data, at the blue circles in the top plot, to the right, at the red triangles in the top plot. The convergence rate with grid resolution for the 5 point stencil of equation (5.71) and the 3 point stencils of equation (5.72) are shown to match expectations in the bottom plots. The error from the WENO reconstruction described in section 5.5.3 is, on this smooth function, converging with the expected order but larger in magnitude.

could be given the size of the stencil it uses, and the logical branches required to find the "best" stencil can reduce computational performance.

An alternative that avoids the problems of ENO schemes are the *Weighted* ENO, or WENO, schemes. These use a combination of the possible ENO stencils to reconstruct $a_{i+1/2}$. This relies on the observation that there are constants C_k such that, for example,

$$a_{i+1/2}^{(\text{Fifth})} = \sum_{k=0}^{2} C_k a_{i+1/2}^{(\text{Third},r)}. \tag{5.73}$$

Exercise 5.6

Check that the constants $C_0 = \frac{3}{10}, C_1 = \frac{3}{5}, C_2 = \frac{1}{10}$ *make equation* (5.73) *consistent with equations* (5.71) *and* (5.72).

The WENO methods work by retaining the sum over all stencils as in equation (5.73), but adjusting the weights to avoid oscillations. Ideally, in regions where the $a^{(0,1)}$ stencils include a shock but the $a^{(2)}$ stencil does not the weights should be $\tilde{C}_0 = 0 = \tilde{C}_1$ and $\tilde{C}_2 = 1$. In order for this sum to make sense we will need the weights to add to 1.

WENO algorithm

We can now write out the full WENO algorithm of order $(2r-1)$. The constant r sets the size of the individual stencils as well as the (optimal) order of the scheme. This section mostly uses the notation of Gerolymos et al. [36], and we will drop the integral average notation (so $\langle a \rangle_i$ will be denoted a_i).

First compute the individual stencils

$$a_{r,k,i+1/2} = \sum_{\ell=0}^{r-1} A_{r,k,\ell} a_{i+k-\ell}, \quad k = 0, \ldots, r-1. \tag{5.74}$$

Here the $A_{r,k,\ell}$ terms are constants which, for given scheme accuracy r are needed to compute the k^{th} stencil.

We then want to combine these individual stencils to compute the WENO result. For this we write

$$a_{r,WENO,i+1/2} = \sum_{k=0}^{r-1} \omega_{r,k,i+1/2} a_{r,k,i+1/2}. \tag{5.75}$$

Here the $\omega_{r,k,i+1/2}$ terms are the weights that vary from point to point. They must sum to one, so that we have a convex combination of the stencils:

$$\sum_{k=1}^{r-1} \omega_{r,k,i+1/2} = 1. \tag{5.76}$$

We also want the weights to match the *optimal* weights $C_{r,k}$ in smooth regions. The optimal weights are defined such that

$$a_{r,WENO,i+1/2} = \sum_{k=0}^{r-1} C_{r,k} a_{r,k,i+1/2} \tag{5.77}$$

is accurate to order $\mathcal{O}(\Delta x^{(2r-1)})$ when q is smooth.

The *choice* of how to get from the optimal weights $C_{r,k}$ to the nonlinear weights $\omega_{r,k,i+1/2}$ defines the WENO scheme. The standard choice of Jiang and Shu is to introduce a measure of the smoothness of the k^{th} stencil by computing the sum of the integral averages of its derivatives from order 1 up to order $2r-1$. For practical implementation purposes this means computing the *smoothness indicators* $\beta_{r,k,i+1/2}$ as

$$\beta_{r,k,i+1/2} = \sigma_{r,k,\ell,m} a_{i+k-\ell} a_{i+k-m}. \tag{5.78}$$

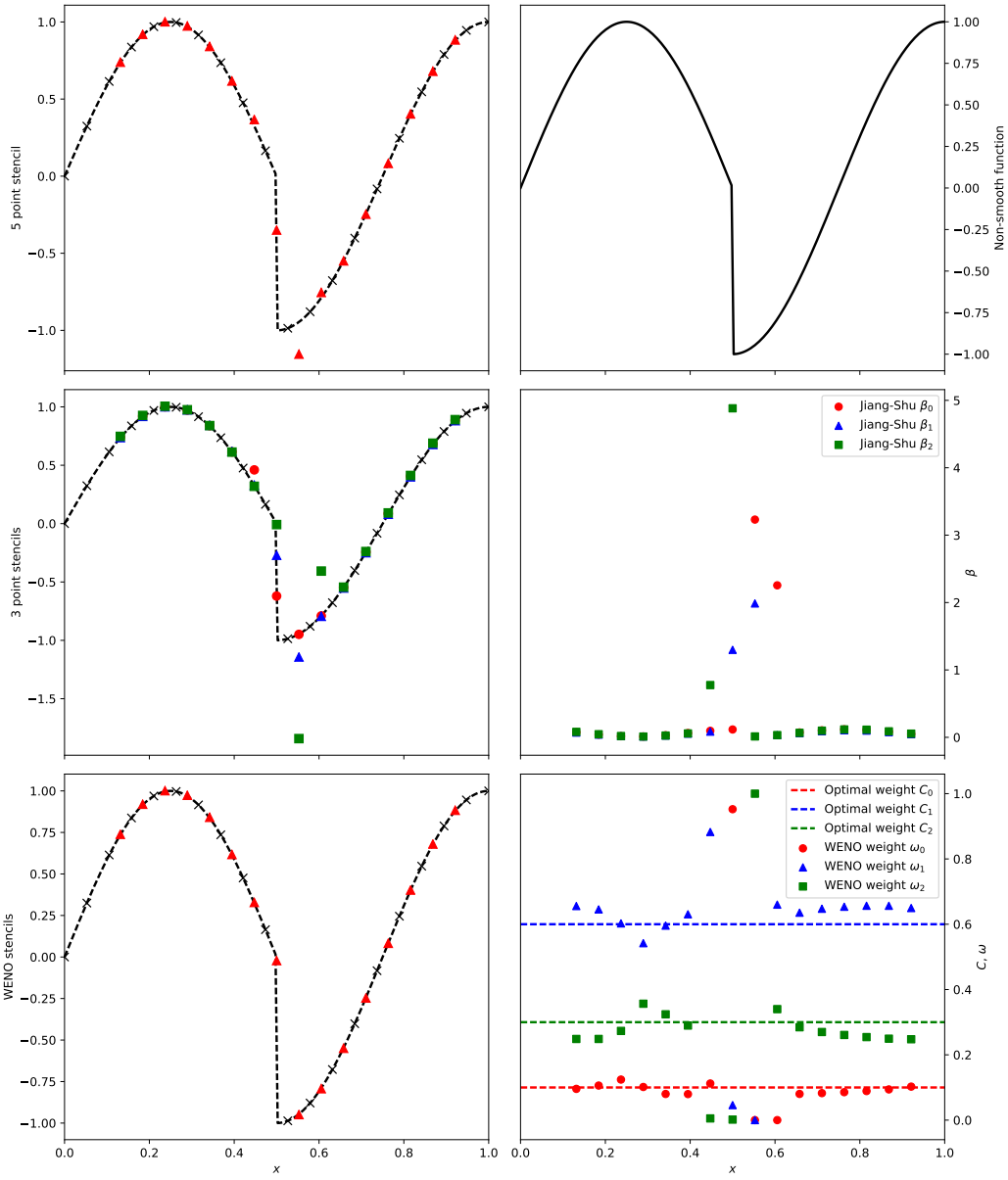

Figure 5.18: A non-smooth function is reconstructed on the unit interval. The 5 point stencil of equation (5.71) and the 3 point stencils of equation (5.72) both show oscillations near the discontinuity, whilst the WENO reconstruction does not. The WENO algorithm depends on measuring the smoothness of the function. The Jiang-Shu smoothness indicators β are mostly zero, but spike near the discontinuity. This feeds into the weight ω that the WENO algorithm gives to each 3 point stencil it uses. Away from the discontinuity the weights revert to the optimal C weights, but near the discontinuity a single ω weight dominates meaning only a single 3 point stencil is used.

The terms $\sigma_{r,k,\ell,m}$ are pre-computed constants which give the coefficients in the quadratic form for the k^{th} stencil of width r in the reconstruction. These smoothness indicators are non-negative, and will be large when the derivatives are large in magnitude. The Jiang and Shu weights are then set by

$$\omega_{r,k,i+1/2} = \frac{\alpha_{r,k,i+1/2}}{\sum_{k=0}^{r-1} \alpha_{r,k,i+1/2}}, \tag{5.79a}$$

$$\alpha_{r,k,i+1/2} = \frac{C_{r,k}}{\epsilon + \beta^2_{r,k,i+1/2}}. \tag{5.79b}$$

Here ϵ is a small number introduced to avoid division-by-zero problems. The form of $\omega_{r,k,i+1/2}$ in equation (5.79a) guarantees that the convex sum condition in equation (5.76) holds. The form of $\alpha_{r,k,i+1/2}$ ensures that when all the smoothness indicators $\beta_{r,k,i+1/2}$ are the same magnitude the weights $\omega_{r,k,i+1/2}$ will match the optimal weights $C_{r,k}$, but when a smoothness indicator is large (i.e., when the associated stencil has large derivatives, typically associated with discontinuities), the contribution of its associated stencil will be small. An example of this is shown in figure 5.18.

Finally, we note that this method has reconstructed $a_{i+1/2}$ from the left. For the advection equation case where the advection speed u is positive this is all we need: the characteristic information tells us that we should use this reconstruction. In the case where the advection speed is negative we should reconstruct from the right. In the implementation it is easiest to implement only one reconstruction direction and pass the data in in reverse order.

Exercise 5.7

Taking the values for the C, A and σ constants from Gerolymos et al. or Shu's review [72], construct a WENO reconstruction method for $r = 2$. Check your results on smooth and non-smooth functions.

Implementation issues with WENO schemes

A convergence test on the WENO schemes with $r = 3$ and $r = 5$ is shown in figure 5.19. This should be compared to figure 5.9, as in both a Gaussian profile is advected five times around a periodic domain. The WENO schemes show higher absolute accuracy for moderate size grids ($N \gtrsim 100$) and show faster convergence.

However, it is clear that whilst the $r = 3$ method converges at fifth order as expected, the $r = 5$ WENO method does not converge at ninth order but at fourth order. This is explained by the time integrator. We remember that any solver has an error both from the spatial and the time discretization. In figure 5.19 the time integrator is the classic fourth order Runge-Kutta, and it is the error from this integrator that is dominating.

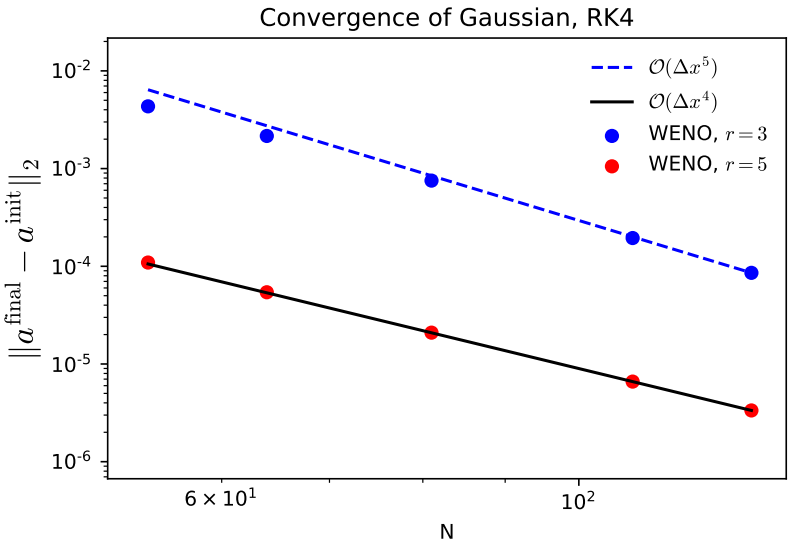

Figure 5.19: WENO solutions for advecting a Gaussian five periods, using two different orders. A fourth order Runge-Kutta method is used for time evolution. For $r = 3$ we see the expected fifth $(2r - 1)$ order convergence. For $r = 5$ we see fourth order, rather than the expected ninth order, convergence. This is as the error from the time integrator dominates.
hydro_examples: `weno.py`

To confirm this, figure 5.20 uses the eigth order Dormand-Price Runge-Kutta method with adaptive step size control (using the `scipy.integrate.ode` routine). Here we see high order convergence for all r, and in each case the convergence rate is $2r - 2$ for **no reason I can understand**.

5.6 Going further

- *Slope limiting*: there are a wide variety of slope limiters. All of them are designed to reduce oscillations in the presence of discontinuities or extrema, but some are higher-order and can be less restrictive when dealing with smooth flows. Most hydro texts (e.g. [46, 82]) provide an introduction to the design of such limiters.

- *Multi-dimensional limiting*: the procedure described above still does the limiting in each dimension independent of the other when doing the unsplit reconstruction. This can lead to overshoots/ undershoots. An example of a method that considers the limiting in multi-dimensions is [11, 51].

- *Spatially-varying velocity field*: if we consider a spatially varying velocity field, $u(x, y)$ and $v(x, y)$ that is specified externally, then we can describe the advec-

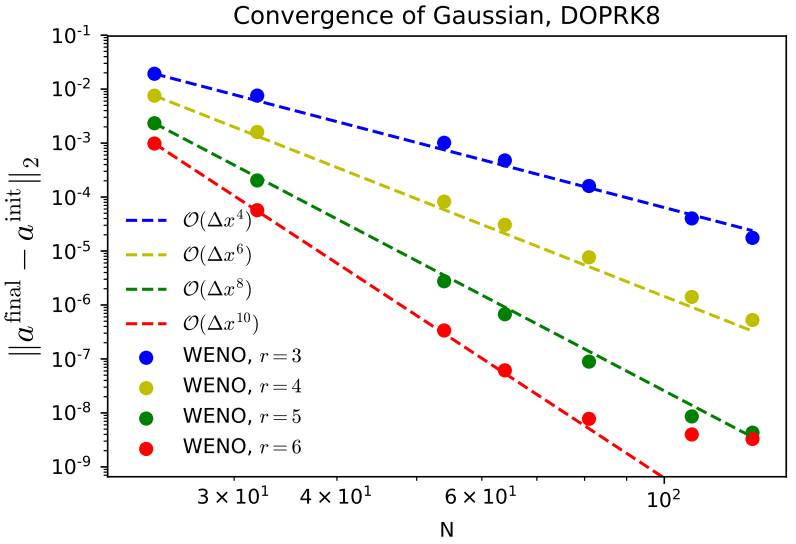

Figure 5.20: WENO solutions for advecting a Gaussian one periods, using four different orders. An eighth order Dormand-Price Runge-Kutta method is used for time evolution. This minimizes the time integrator error and we see convergence at order $2r-2$ for all schemes, although the time integrator error eventually shows in the highest order case.
hydro_examples: `weno.py`

tion of a quantity ϕ as:

$$\phi_t + (\phi u)_x + (\phi v)_y = 0 \tag{5.80}$$

The solution procedure is largely the same as described above. We write:

$$\begin{aligned}
\phi^{n+1/2}_{i+1/2,j,L} &= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x} + \frac{\Delta t}{2}\frac{\partial \phi}{\partial t} + \ldots \\
&= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x} + \frac{\Delta t}{2}\left[-(\phi u)_x - (\phi v)_y\right]_{i,j} \\
&= \underbrace{\phi^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\frac{\partial \phi}{\partial x}}_{\hat{\phi}^{n+1/2}_{i+1/2,j,L}} - \frac{\Delta t}{2}\left[\phi u_x\right]_{i,j} - \frac{\Delta t}{2}\left[(\phi v)_y\right]_{i,j}
\end{aligned} \tag{5.81}$$

and upwinding is used to resolve the Riemann problem for both the transverse and normal interface states. This type of construction comes up in low Mach number flows, where the density can be advected according to the velocity field in much the fashion shown here, and is described in § 15.3.2.

For compressible hydrodynamics, we often have density-weighted quantities that we advect. This extension is described in § **??**.

5.7 pyro experimentation

To gain some experiences with these ideas, we can use the advection solver in pyro (see Appendix B to get started). The pyro advection solver implements the second-order unsplit advection algorithm described in the previous sections. To run this solver on the Gaussian advection problem, do:

```
./pyro.py advection smooth inputs.smooth
```

By default, this will advect a Gaussian profile diagonally across the domain for a single period.

To get a feel for the advection algorithm, here are some suggested exercises:

Exercise 5.8

Implement a tophat initial profile and run with and without limiters (this is controlled by the `advection.limiter` *runtime parameter).*

Exercise 5.9

Look at the solution when you advect purely in the x- or y-direction and compare to the diagonal case—notice how the direction affects the error in the solution. This is the imprint of the grid we discretize on.

Exercise 5.10

Implement a dimensionally-split version of the advection algorithm and compare the results to the unsplit version. Pick a suitable norm and measure the convergence of the methods.

Chapter 6

Burgers' Equation

6.1 Burgers' equation

The inviscid Burgers' equation is the simplest *nonlinear* hyperbolic equation:

$$u_t + uu_x = 0 \tag{6.1}$$

Here u is both the quantity being advected and the speed at which it is moving. Recall that for the linear advection equation, we saw that the solution was constant along lines $x = ut + x_0$, which are parallel, since u is spatially constant. For Burgers' equation, this is no longer the case, and the characteristic lines are now given by $dx/dt = u$, with $x(0) = x_0$. Since $u = u(t)$, we cannot integrate this directly.

Exercise 6.1

To find the lines along which the solution to Burgers' equation is constant, consider the full change in u:

$$du = \left.\frac{\partial u}{\partial t}\right|_x dt + \left.\frac{\partial u}{\partial x}\right|_t dx = 0 \tag{6.2}$$

Here, we seek $u =$ constant, *so we set* $du = 0$. *Now, using the Burgers' equation itself, show that this implies that the lines along which the solution is constant are:*

$$\frac{dx}{dt} = u(x,t) \tag{6.3}$$

If we take $u_0 = u(t = 0)$, then we can look at how the characteristic behave over a small time interval (before $u(x,t)$ changes significantly). Figure 6.1 shows the behavior for an initial velocity with sinusoidal profile. We see that after a short period of time, the characteristics intersect. At the point, (x_s, t_s) where they intersect, there is

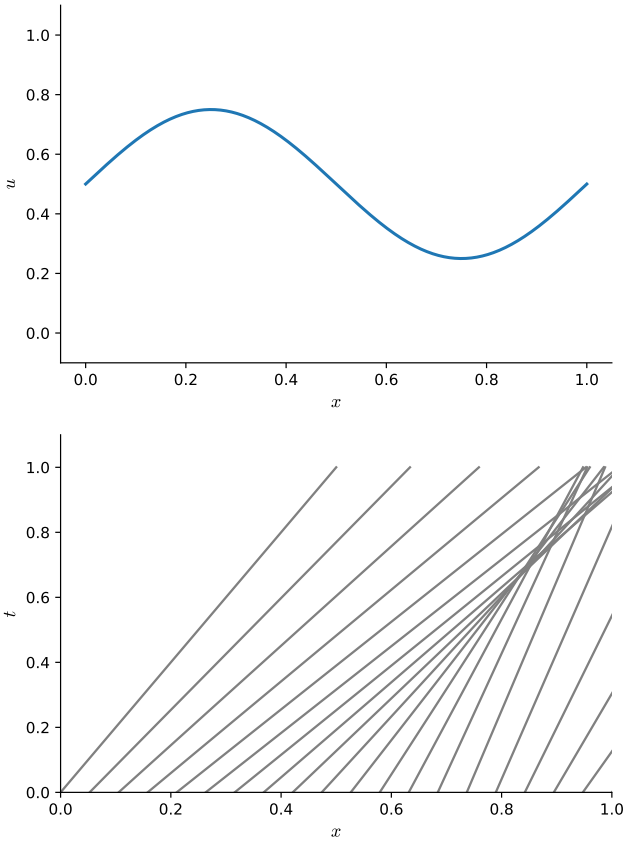

Figure 6.1: (top) Initially sinusoidal velocity distribution (bottom) Approximate characteristic structure for Burgers' equation, using $u_0 = u(t)$. Note that after a short period of time, the characteristics intersect, signaling the formation of a shock.

no way to trace backwards along the characteristics to find a unique initial state. This merging of the characteristics in the x-t plane is a *shock*, and represents just one way that nonlinear problems can differ from linear ones.

Another type of wave not present in a linear system is a *rarefaction*. Figure 6.2 shows initial conditions of slower velocity to the left of faster velocity. We see that the characteristics diverge in this case, and we will be left with having to fill in the solution inbetween as some intermediate state.

In conservative form, Burgers' equation appears as:

$$u_t + \left[\tfrac{1}{2}u^2\right]_x = 0 \tag{6.4}$$

The solution of this follows the same methodology as described in Chapter 4. The interface states are predicted as to second order by Taylor expanding in space and

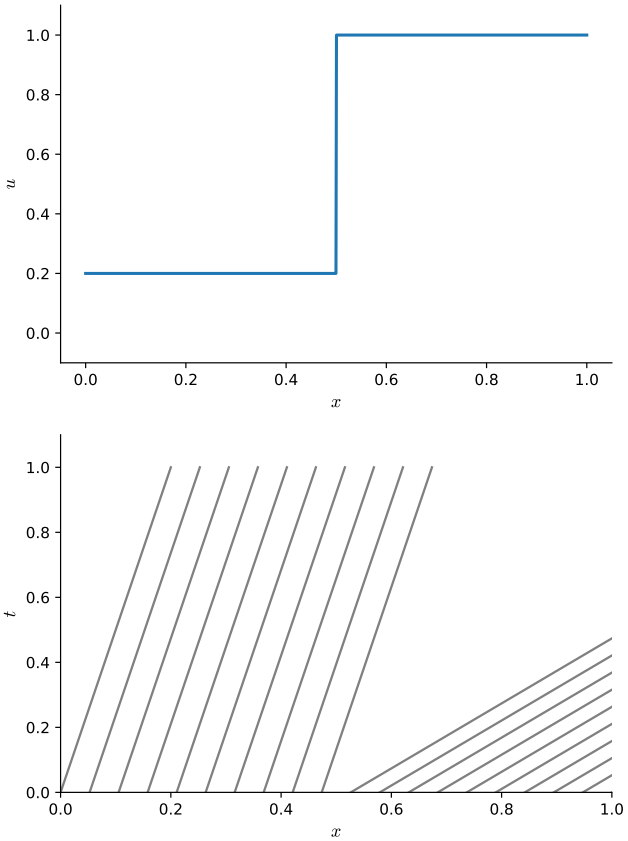

Figure 6.2: (top) Initially discontinuous velocity distribution with low velocity left of high velocity (bottom) Approximate characteristic structure for Burgers' equation, using $u_0 = u(t)$. Note that after a short period of time, the characteristics diverge—this is a rarefaction.

time:

$$u^{n+1}_{i+1/2,L} = u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left.\frac{\partial u}{\partial t}\right|_i + \ldots \tag{6.5}$$

$$= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left.\left(-u_i\frac{\partial u}{\partial x}\right)\right|_i + \ldots \tag{6.6}$$

$$= u^n_i + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_i\right)\left.\frac{\partial u}{\partial x}\right|_i + \ldots \tag{6.7}$$

The only difference with the linear advection equation is that now $u_i\Delta t/\Delta x$ varies from zone to zone, whereas with linear advection, it is the constant C. The slopes are computed using the same limiters as with linear advection.

The Riemann problem differs from linear advection. As we saw above, the characteristic curves can intersect in the x-t plane, and it is not possible to trace backward from time to learn where the flow originated. This is the condition for a *shock*.

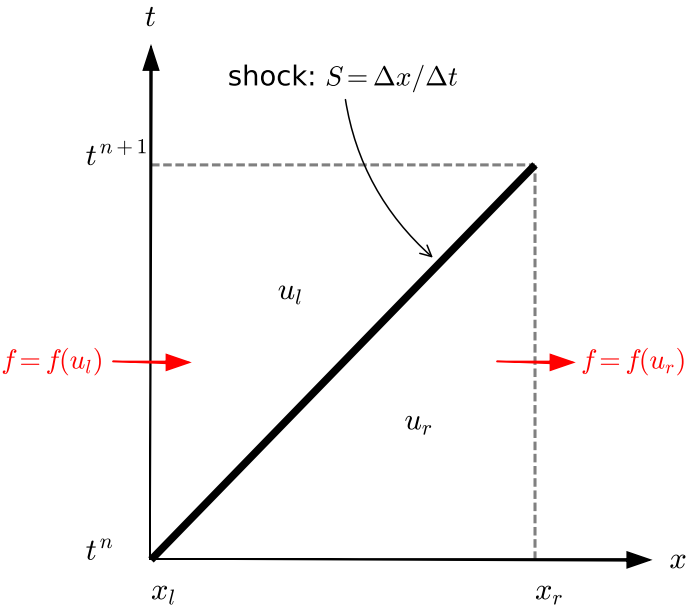

Figure 6.3: A rightward moving shock in the x-t plane separating two states: u_l and u_r.

The shock speed is computed through the *Rankine-Hugoniot* jump conditions. For a scalar equation, these are easy to construct. We'll follow the method of [46]. Figure 6.3 shows two states separated by a rightward moving shock in the x-t plane. At time t^n, the state in our interval ($x \in [x_l, x_r]$) is entirely u_r. As time evolves, we imagine our interval $[x_l, x_r]$ moving vertically upwards in the diagram, and we see that it contains a mix of states u_l and u_r. Finally, at time, t^{n+1} it is entirely u_l. The shock moves with a speed $S = \Delta x/\Delta t$ in this figure. To determine the speed, we integrate our conservation law over both space and time (and normalize by $\Delta x = x_r - x_l$):

$$\frac{1}{\Delta x}\int_{x_l}^{x_r} dx \int_{t^n}^{t^{n+1}} dt\, u_t = -\frac{1}{\Delta x}\int_{x_l}^{x_r} dx \int_{t^n}^{t^{n+1}} dt\, [f(u)]_x \tag{6.8}$$

Doing the t integral on the left and x integral on the right, we have

$$\frac{1}{\Delta x}\int_{x_l}^{x_r} \left\{u(t^{n+1}) - u(t^n)\right\} dx = -\frac{1}{\Delta x}\int_{t^n}^{t^{n+1}} \left\{f(u)|_{x=x_r} - f(u)|_{x=x_l}\right\} dt \tag{6.9}$$

Recognizing that at $t = t^n$, $u = u_r$ and at $t = t^{n+1}$, $u = u_l$, in the left side becomes

$$\frac{1}{\Delta x}\int_{x_l}^{x_r} \left\{u(t^{n+1}) - u(t^n)\right\} dx = \left\{u(t^{n+1}) - u(t^n)\right\} = u_l - u_r \ . \tag{6.10}$$

For the right side, we see that all along $x = x_l$ the flux is $f = f(u_l)$ for $t \in [t^n, t^{n+1}]$. Likewise, all along $x = x_r$, the flux is $f = f(u_r)$ in the same time interval (see the

figure). Therefore, our expression becomes:

$$(u_l - u_r) = -\frac{\Delta t}{\Delta x}\left[f(u_r) - f(u_l)\right] \tag{6.11}$$

and using $S = \Delta x/\Delta t$, we see

$$S = \frac{f(u_r) - f(u_l)}{u_r - u_l} \tag{6.12}$$

For Burgers' equation, substituting in $f(u) = u^2/2$, we get

$$S = \frac{1}{2}(u_l + u_r) \tag{6.13}$$

With the shock speed known, the Riemann problem is straightforward. If there is a shock (compression, so $u_l > u_r$) then we compute the shock speed and check whether the shock is moving to the left or right, and then use the appropriate state. If there is no shock, then we can simply use upwinding, as there is no ambiguity as to how to trace backwards in time to the correct state. Putting this together, we have:

$$\underset{\text{(shock)}}{\text{if } u_l > u_r:} \qquad u_s = \begin{cases} u_l & \text{if } S > 0 \\ u_r & \text{if } S < 0 \\ 0 & \text{if } S = 0 \end{cases} \tag{6.14}$$

$$\text{otherwise:} \qquad u_s = \begin{cases} u_l & \text{if } u_l > 0 \\ u_r & \text{if } u_r < 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.15}$$

Once the interface states are constructed, the flux is calculated as:

$$F^{n+1/2}_{i+1/2} = \frac{1}{2}\left(u^{n+1/2}_{i+1/2}\right)^2 \tag{6.16}$$

and the conservative update is

$$u^{n+1}_i = u^n_i + \frac{\Delta t}{\Delta x}\left(F^{n+1/2}_{i-1/2} - F^{n+1/2}_{i+1/2}\right) \tag{6.17}$$

The timestep constraint now must consider the most restrictive Courant condition over all the zones:

$$\Delta t = \min_i \{\Delta x/u_i\} \tag{6.18}$$

Figure 6.4 shows the solution to Burgers' equation using the 2nd-order piecewise linear method described here, with the MC limiter. The initial conditions chosen are all positive velocity, with a lower velocity to the left of the higher velocity. As the

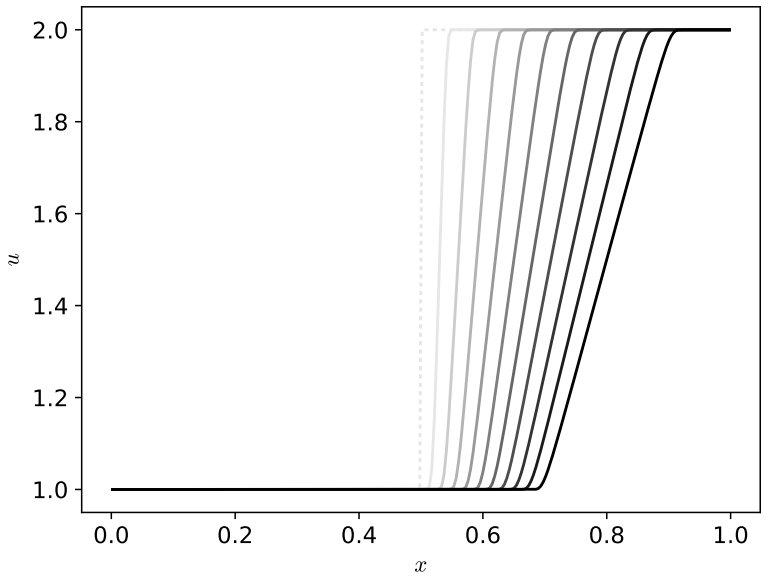

Figure 6.4: Solution to the inviscid Burgers' equation with 256 zones and a Courant number, $C = 0.8$ for initial conditions that generate a rarefaction: the left half of the domain was initialized with $u = 1$ and the right half with $u = 2$. This initial velocity state creates a divergent flow. The curves are shown 0.02 s apart, with the darker grayscale representing later in time.
hydro_examples: `burgers.py`

solution evolves, the state on the right will rush away from the state on the left, and spread out like a fan. This is called a *rarefaction wave* or simply a *rarefaction*.

Figure 6.5 shows the solution to Burgers' equation with initially sinusoidal data:

$$u(x, t = 0) = \begin{cases} 1 & x < 1/3 \\ 1 + \frac{1}{2} \sin\left(\frac{2\pi(x-1/3)}{1/3}\right) & 1/3 \le x \le 2/3 \\ 1 & x > 2/3 \end{cases} \tag{6.19}$$

This is analogous to the case shown in Figure 6.1—we see the solution steepen and form a shock which propagates to the right. The shock is practically infinitesimally thin here, since there is no explicitly viscosity to smear it out*.

Exercise 6.2

Extend your 1-d finite-volume solver for advection (from Exercise 5.1) to solve Burgers' equation. You will need to change the Riemann solver and

*Numerical diffusion, in the form of truncation error of our method, will smear things a little.

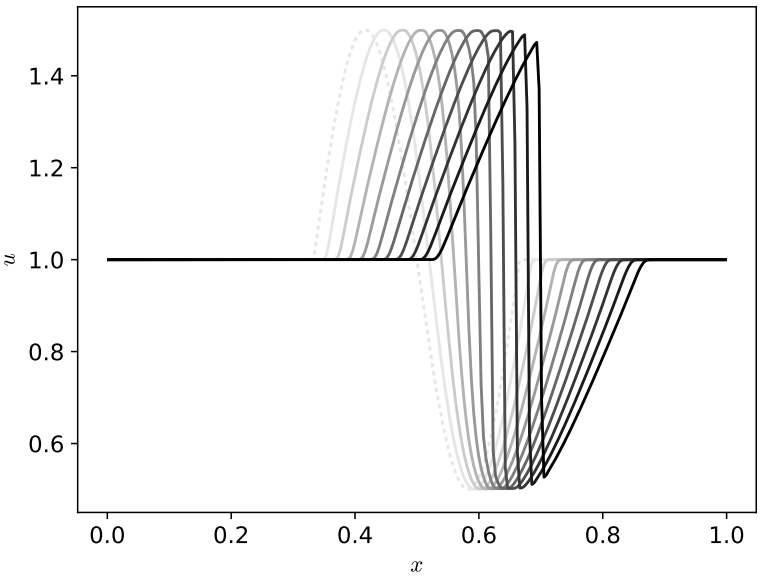

Figure 6.5: Solution to the inviscid Burgers' equation with 256 zones and a Courant number, $C = 0.8$. The initial conditions here are sinusoidal and the solution quickly steepens into a shock.The curves are shown 0.02 s apart, with the darker grayscale representing later in time.
hydro_examples: `burgers.py`

> *use the local velocity in the construction of the interface states. Run the examples shown in Figures 6.4 and 6.5.*

As we'll see shortly, these two types of waves can also appear in the Euler equations for hydrodynamics.

A final thing to note is that we solved Burgers' equation in conservative form. For shock solutions, this is essential, since as we noted earlier, that finite-volume method relates to the integral form of the PDEs, the discontinuity is nicely handled by the Riemann solver. We could also imagine differencing Burgers' equation in non-conservative form, i.e., starting with:

$$u_t + uu_x = 0 \tag{6.20}$$

If you did this (for instance, using a finite difference scheme and an upwind difference for u_x), you would find that you get the wrong shock speed.

Exercise 6.3

Using a simple first-order finite-difference method like we described in Ch. 4 for linear advection, difference the conservative and non-conservation formulations of Burgers' equation as:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\frac{1}{2}\frac{(u_i^n)^2 - (u_{i-1}^n)^2}{\Delta x} \tag{6.21}$$

and

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\frac{u_i^n(u_i^n - u_{i-1}^n)}{\Delta x} \tag{6.22}$$

(Note: these discretizations are upwind so long as $u > 0$).
Run these with the shock initial Riemann conditions:

$$u(x, t = 0) = \begin{cases} 2 & x < 1/2 \\ 1 & x > 1/2 \end{cases} \tag{6.23}$$

and measure the shock speed from your solution by simply differencing the location of the discontinuity at two different times. Compare to the analytic solution for a shock for the Riemann problem.

6.2 Characteristic tracing

A concept that will be useful in the next section is *characteristic tracing*. The idea is that we only include information in the interface states if the characteristic that carries that information is moving toward the interface. For Burgers equation, this is simple, since there is only a single characteristic—the velocity. So for the left state on an interface, we'd only add the change if the velocity $u_i > 0$ (moving toward the interface $i + 1/2$):

$$u_{i+1/2,L}^{n+1} = u_i^n + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}\max(0, u_i)\right)\left.\frac{\partial u}{\partial x}\right|_i + \ldots \tag{6.24}$$

Notice that the effect of this is to set the interface state simply to the value given by the piecewise linear reconstruction on the interface if the wave isn't moving to the interface.

6.3 Going further

- The equation we've been dealing with here is the *inviscid* Burgers' equation. The full Burgers' equation includes viscosity (a velocity diffusion):

$$u_t + uu_x = \epsilon u_{xx} \tag{6.25}$$

To solve this, we need to first learn about techniques for diffusion, and then how to solve equations that span multiple PDE types. This will be described in §11.3.

- Aside from pedagogical interest, Burgers' equation can be used as a simple model of traffic flow (where shocks can arise from people slamming on the brakes). Many sources discuss this application, including the text by [46].

6.4 WENO methods, nonlinear equations, and flux-splitting

For a nonlinear scalar conservation law

$$u_t + f(u)_x = 0 \tag{6.26}$$

the WENO method introduced in section **??** above does not work, as we have assumed the characteristic information travels in one direction only. We have also reconstructed the variable q and from that constructed the "flux" which we feed into the differencing formula of equation (5.5) to update the solution: this will not work either.

In the nonlinear case, we instead reconstruct the *flux* directly. That is, given the state u_i at location x_i, we compute the flux $f_i = f(u_i)$ and reconstruct the flux $f_{i+1/2}$ using the WENO reconstruction above.

This has the significant problem that we have no characteristic information for the flux: does it propagate to the left or to the right? To get around this we introduce *flux splitting*. We write

$$f(u) = f^{(+)}(u) + f^{(-)}(u) \tag{6.27}$$

where we choose the functions such that the information contained in $f^{(+)}$ propagates to the right and that in $f^{(-)}$ propagates to the left. There are many ways of doing this, but the simplest is the *Lax-Friedrichs* flux splitting

$$f^{(\pm)}(u) = \tfrac{1}{2}\left(f(u) \pm \alpha u\right) \tag{6.28}$$

where $\alpha \geq \max|\partial_u f|$. With α being larger than the maximum propagation speed, this means that

$$\partial_u f^{(+)} \geq 0, \quad \partial_u f^{(-)} \leq 0, \tag{6.29}$$

and the characteristic information contained in $f^{(\pm)}$ will propagate as required.

The simplest flux-split algorithm is then

1. Compute the maximum characteristic speed α over the entire computational grid;
2. Compute the split fluxes $f^{(\pm)}$ from equation (6.28);

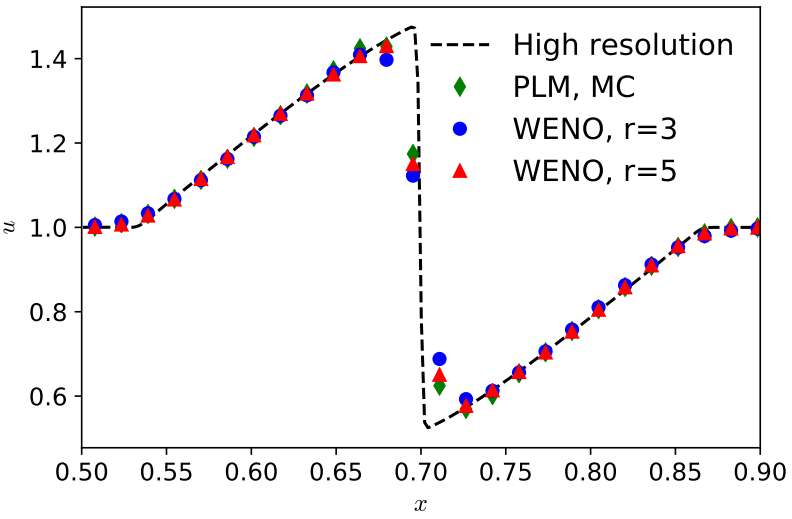

Figure 6.6: Numerical solutions to Burgers' equation using the piecewise linear method as in figure 6.5 and two flux-split WENO methods as outlined in section 6.4. This uses the same sinusoidal data as figure 6.5 shortly after shock formation. Note that the WENO methods are not obviously any better at resolving the shock than the simpler piecewise linear approach.
hydro_examples: `weno_burgers.py`

3. Reconstruct $f^{(+)}$ to the right using, e.g., the WENO algorithm of section 5.5.3;
4. Reconstruct $f^{(-)}$ to the left using the same method;
5. Compute $f_{i+1/2} = f^{(+)}_{i+1/2} + f^{(-)}_{i+1/2}$;
6. Update the state using equation (3.14).

As a direct comparison between the piecewise linear method used at the start of this chapter, and the more complex WENO methods introduced here, we start with looking at a shock forming from smooth initial data. Figure 6.6 shows the evolution of the sinusoidal initial data as used in figure 6.5, but only showing the result at one time, shortly after characteristic crossing leads to a shock. We see that there is very little difference between the different numerical methods. The characteristic focusing means that all the methods have a similar number of points within the shock. The piecewise linear method appears to capture the edges of the shock slightly more accurate than the (higher order) WENO method with $r = 3$. This is likely as the piecewise linear method is using the exact solution to the Riemann Problem, whilst the WENO method is using the more diffusive Lax-Friedrichs flux splitting. At the higher $r = 5$ order there is a marginal advantage to the WENO method.

The advantages of the WENO method arise when the profile is still continuous, as in the rarefaction problem, or at times before shocks have formed. To confirm that

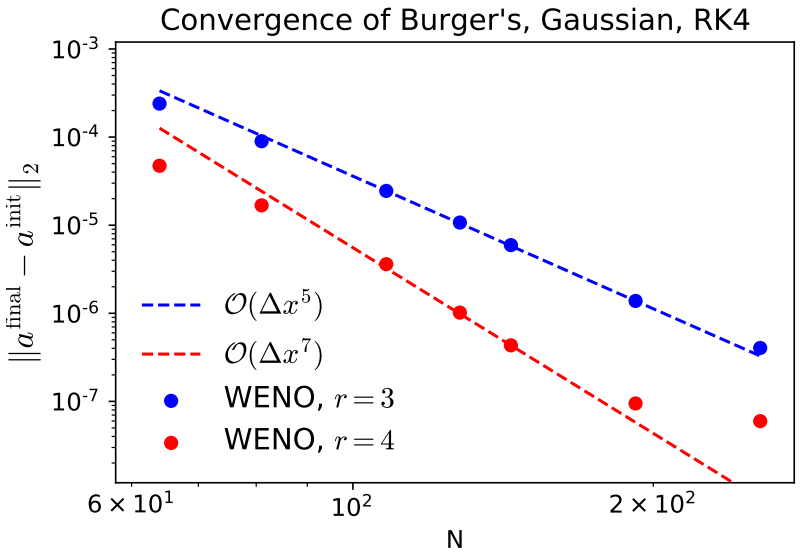

Figure 6.7: WENO solutions from evolving Burgers' equation for an initial Gaussian profile, stopping before the solution is discontinuous. A fourth order Runge-Kutta method is used for time evolution. For $r = 3$ we see the expected fifth $(2r - 1)$ order convergence. For $r = 5$ we see seventh order convergence briefly, before the error from the time integrator dominates.
hydro_examples: `weno_burgers.py`

high-order convergence is still possible for a nonlinear problem, an initially Gaussian profile is evolved for a short time and compared to the exact solution (computed from characteristic tracing). The resulting error convergence is shown in figure 6.7. We see the expected fifth and seventh order convergence rates, at least for some of the resolutions. At too low a resolution the higher order terms in the error expansion affect the solution. At too high a resolution, for the higher-order method, the time integration error starts to dominate.

Chapter 7

Euler Equations: Theory

7.1 Euler equation properties

The Euler equations* describe conservation of mass, momentum, and energy in the fluid approximation. Their general form, without any source terms, is:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{7.1}$$

$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p = 0 \tag{7.2}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{U} + p\mathbf{U}) = 0 \tag{7.3}$$

Here ρ is the density, $\mathbf{U}$ is the velocity vector, $\mathbf{U} = u\hat{x} + v\hat{y}$, p is the pressure, and E is the total energy / mass, and can be expressed in terms of the specific internal energy, e, and kinetic energy as:

$$E = e + \frac{1}{2}|\mathbf{U}|^2 \tag{7.4}$$

The equations are closed with the addition of an equation of state. A common choice is the gamma-law EOS:

$$p = \rho e(\gamma - 1) \tag{7.5}$$

where γ is the ratio of specific heats for the gas/fluid (for an ideal, monatomic gas, $\gamma = 5/3$), but any relation of the form:

$$p = p(\rho, e) \tag{7.6}$$

*We focus on the Euler equations, which are the most commonly modeled set of fluid equations in astrophysics. The more general equation set, the Navier-Stokes equations, includes dissipative terms. However, for astrophysical flows, the scales on which these dissipative terms operate are usually much smaller than the system of interest (equivalently, Reynolds numbers of astrophysical flows are very large).

will work. For many astrophysical environments, we may not be able to express this relation analytically, but instead will solve it via numerical integration or by interpolating from tabulated results.

For a derivation of the equations of hydrodynamics using moments of the Boltzmann equation see [22, 73]. For a physically-motivated derivation from conservation, see [22, 46].

In one dimension, they appear as[†]:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0 \tag{7.7}$$

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho u u + p)}{\partial x} = 0 \tag{7.8}$$

$$\frac{\partial (\rho E)}{\partial t} + \frac{\partial (\rho u E + u p)}{\partial x} = 0 \tag{7.9}$$

One thing that we can notice immediately is that there is no need for temperature in this equation set, although often, when source terms are present, we will need to obtain temperature from the equation of state.

In this form, the equations are said to be in *conservative form*, i.e. they can be written as:

$$\boldsymbol{\mathcal{U}}_t + [\mathbf{F}(\boldsymbol{\mathcal{U}})]_x = 0 \tag{7.10}$$

with

$$\boldsymbol{\mathcal{U}} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix} \qquad \mathbf{F}(\boldsymbol{\mathcal{U}}) = \begin{pmatrix} \rho u \\ \rho u u + p \\ \rho u E + u p \end{pmatrix} \tag{7.11}$$

We can write this in *quasi-linear* form by first expressing the flux vector in terms of the conserved variables directly. Taking $m \equiv \rho u$, $\mathcal{E} \equiv \rho E$, and assuming a gamma-law EOS[‡],

$$p = \rho e(\gamma - 1) = \left(\mathcal{E} - \frac{1}{2} \frac{m^2}{\rho} \right) (\gamma - 1) \tag{7.12}$$

we have

$$\mathbf{F}(\boldsymbol{\mathcal{U}}) = \begin{pmatrix} m \\ \frac{1}{2} \frac{m^2}{\rho} (3 - \gamma) + \mathcal{E}(\gamma - 1) \\ \frac{m\mathcal{E}}{\rho} \gamma - \frac{1}{2} \frac{m^3}{\rho^2} (\gamma - 1) \end{pmatrix} \tag{7.13}$$

[†]assuming Cartesian coordinates

[‡]we can relax this assumption by writing $p = p(\rho, e)$, and then taking the derivatives of this as needed: $\partial p/\partial \rho$, $\partial p/\partial m = \partial p/\partial e|_\rho \partial e/\partial m$, and $\partial p/\partial \mathcal{E} = \partial p/\partial e|_\rho \partial e/\partial \mathcal{E}$, with $e = (\mathcal{E} - 1/2 m^2/\rho)/\rho$. But as we'll see, there are simpler systems to work with.

The Jacobian[§] of this flux vector can now be computed as $\mathbf{A} = \partial\mathbf{F}/\partial\boldsymbol{\mathcal{U}}$:

$$\mathbf{A}(\boldsymbol{\mathcal{U}}) = \begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{2}u^2(3-\gamma) & u(3-\gamma) & \gamma-1 \\ \frac{1}{2}(\gamma-2)u^3 - \frac{uc^2}{\gamma-1} & \frac{3-2\gamma}{2}u^2 + \frac{c^2}{\gamma-1} & u\gamma \end{pmatrix} \tag{7.14}$$

where the speed of sound is $c = \sqrt{\gamma p/\rho}$. With this, our system can be written as:

$$\boldsymbol{\mathcal{U}}_t + \mathbf{A}(\boldsymbol{\mathcal{U}})\boldsymbol{\mathcal{U}}_x = 0 \tag{7.15}$$

This matrix is quite complex and difficult to work with. The eigenvectors of this matrix can be found in a variety of sources (e.g. [76, 82]).

An alternate way to express these equations is using the *primitive variables*: ρ, u, p.

Exercise 7.1

Show that the Euler equations in primitive form can be written as

$$\frac{\partial\rho}{\partial t} + u\frac{\partial\rho}{\partial x} + \rho\frac{\partial u}{\partial x} = 0 \tag{7.16}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0 \tag{7.17}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \gamma p\frac{\partial u}{\partial x} = 0 \tag{7.18}$$

Notice that the velocity equation looks like Burgers' equation, and is nonlinear. This nonlinearity will admit shock and rarefaction solutions like we saw with Burgers' equation.

The primitive variable system can be written compactly as:

$$\mathbf{q}_t + \mathbf{A}(\mathbf{q})\mathbf{q}_x = 0 \tag{7.19}$$

where

$$\mathbf{q} = \begin{pmatrix} \rho \\ u \\ p \end{pmatrix} \qquad \mathbf{A}(\mathbf{q}) = \begin{pmatrix} u & \rho & 0 \\ 0 & u & 1/\rho \\ 0 & \gamma p & u \end{pmatrix} \tag{7.20}$$

The eigenvalues of $\mathbf{A}$ can be found via $|\mathbf{A} - \lambda\mathbf{I}| = 0$, where $|\ldots|$ indicates the determinant and λ are the eigenvalues.

[§]The Jacobian, $\mathbf{J}$ of a vector $\mathbf{F}(\boldsymbol{\mathcal{U}})$ with $\mathbf{F} = (f_1, f_2, \ldots, f_n)^\mathsf{T}$ and $\boldsymbol{\mathcal{U}} = (u_1, u_2, \ldots, u_n)^\mathsf{T}$ is

$$\mathbf{J} \equiv \frac{\partial\mathbf{F}}{\partial\boldsymbol{\mathcal{U}}} = \begin{pmatrix} \partial f_1/\partial u_1 & \partial f_1/\partial u_2 & \ldots & \partial f_1/\partial u_n \\ \partial f_2/\partial u_1 & \partial f_2/\partial u_2 & \ldots & \partial f_2/\partial u_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial u_1 & \partial f_n/\partial u_2 & \ldots & \partial f_n/\partial u_n \end{pmatrix}$$

Exercise 7.2

Show that the eigenvalues of $\mathbf{A}$ *are* $\lambda^{(-)} = u - c$, $\lambda^{(\circ)} = u$, $\lambda^{(+)} = u + c$.

Note that both the conserved Jacobian matrix, $\mathbf{A}(\boldsymbol{\mathcal{U}})$, and the primitive variable matrix, $\mathbf{A}(\mathbf{q})$, have the same eigenvalues, since they represent the same physics.

In Eq. 7.18, we used the algebraic gamma-law equation of state to replace e with p, however, for a general equation of state, we can get the appropriate expression by writing $p = p(\rho, s)$:

$$\frac{Dp}{Dt} = \left.\frac{\partial p}{\partial \rho}\right|_s \frac{D\rho}{Dt} + \left.\frac{\partial p}{\partial s}\right|_\rho \cancelto{0}{\frac{Ds}{Dt}} \tag{7.21}$$

where $Ds/Dt = 0$ when no entropy sources are present (we will relax this assumption in § 15.1). Recognizing that $\Gamma_1 \equiv \partial \log p / \partial \log \rho|_s$, we have:

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \Gamma_1 p \frac{\partial u}{\partial x} = 0 \tag{7.22}$$

as the generalization of the pressure equation[¶]. The sound speed is then $c^2 = \Gamma_1 p/\rho$.

These eigenvalues are the speeds at which information propagates through the fluid. Since the eigenvalues are real, this system (the Euler equations) is said to be *hyperbolic*. Additionally, since $\mathbf{A} = \mathbf{A}(\mathbf{q})$, the system is said to be *quasi-linear*. Note, physically, these eigenvalues are telling us that information is communicated at the speed of the fluid (u), as well as the speed of sound moving with respect to the fluid ($u \pm c$).

We'll use the symbols $\{-, \circ, +\}$ to denote the eigenvalues and their corresponding eigenvectors throughout these notes. The right and left eigenvectors can be found via:

$$\mathbf{A}\,\mathbf{r}^{(\nu)} = \lambda^{(\nu)}\mathbf{r}^{(\nu)}\ ; \qquad \mathbf{l}^{(\nu)}\,\mathbf{A} = \lambda^{(\nu)}\mathbf{l}^{(\nu)} \tag{7.23}$$

where $\nu = \{-, \circ, +\}$ corresponding to the three waves, and there is one right and one left eigenvector for each of the eigenvalues.

Exercise 7.3

Show that the right eigenvectors are:

$$\mathbf{r}^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \qquad \mathbf{r}^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{r}^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix} \tag{7.24}$$

[¶] Γ_1 is one of the many adiabatic indices that describe the coupling between different thermodynamic quantities under compression/expansion. Any stellar structure book should give a good background, e.g., [40]. Note that for an ideal gas, $\Gamma_1 = \gamma$

and the left eigenvectors are:

$$\mathbf{l}^{(-)} = \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \tag{7.25}$$

$$\mathbf{l}^{(\circ)} = \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \tag{7.26}$$

$$\mathbf{l}^{(+)} = \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \tag{7.27}$$

Note that in general, there can be an arbitrary constant in front of each eigenvector. Here they are normalized such that $\mathbf{l}^{(i)} \cdot \mathbf{r}^{(j)} = \delta_{ij}$.

A Jupyter notebook using SymPy that derives these eigenvectors is available: hydro_examples: `euler.ipynb` [||]

A final form of the equations is called the *characteristic form*. Here, we wish to diagonalize the matrix $\mathbf{A}$. We take the matrix $\mathbf{R}$ to be the matrix of right eigenvectors,

$$\mathbf{R} = (\mathbf{r}^{(-)}|\mathbf{r}^{(\circ)}|\mathbf{r}^{(+)}) \tag{7.28}$$

(i.e., $\mathbf{R}$ is a square matrix whose columns are the eigenvectors), and $\mathbf{L}$ is the corresponding matrix of left eigenvectors:

$$\mathbf{L} = \begin{pmatrix} \mathbf{l}^{(-)} \\ \mathbf{l}^{(\circ)} \\ \mathbf{l}^{(+)} \end{pmatrix} \tag{7.29}$$

Note that $\mathbf{L}\,\mathbf{R} = \mathbf{I} = \mathbf{R}\,\mathbf{L}$, and $\mathbf{L} = \mathbf{R}^{-1}$.

Exercise 7.4

Show that $\mathbf{\Lambda} = \mathbf{LAR}$ *is a diagonal matrix with the diagonal elements simply the 3 eigenvalues we found above:*

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{pmatrix} \tag{7.30}$$

To transform our primitive variable system into the characteristic form, we start by multiplying by $\mathbf{L}$:

$$\mathbf{L}\mathbf{q}_t + \mathbf{L}\mathbf{A}\mathbf{q}_x = 0 \tag{7.31}$$

Next, recalling that $\mathbf{RL} = \mathbf{I}$ (we can insert the identity matrix where we please), we can rewrite our system as:

$$\mathbf{L}\mathbf{q}_t + \mathbf{LAR}\mathbf{L}\mathbf{q}_x = 0 \tag{7.32}$$

Finally, defining $d\mathbf{w} = \mathbf{L}d\mathbf{q}$,

$$\mathbf{w}_t + \Lambda\mathbf{w}_x = 0 \tag{7.33}$$

[||] if you don't have Jupyter, you can view this rendered online via the github link.

Here, the $\mathbf{w}$ are the *characteristic variables*. Note that we cannot in general integrate $d\mathbf{w} = \mathbf{L}d\mathbf{q}$ to write down the characteristic quantities. Since $\mathbf{\Lambda}$ is diagonal, this system is a set of decoupled advection-like equations:

$$w_t^{(-)} + \lambda^{(-)} w_x^{(-)} = 0 \tag{7.34}$$

$$w_t^{(\circ)} + \lambda^{(\circ)} w_x^{(\circ)} = 0 \tag{7.35}$$

$$w_t^{(+)} + \lambda^{(+)} w_x^{(+)} = 0 \tag{7.36}$$

If the system were linear, then the solution to each would simply be to advect the quantity $w^{(\nu)}$ at the wave speed $\lambda^{(\nu)}$. We could then get back the primitive variables as $d\mathbf{q} = \mathbf{L}^{-1}d\mathbf{w} = \mathbf{R}d\mathbf{w}$.

The basic idea of hyperbolic systems of PDEs is that there is one wave for each eigenvalue, and these define the characteristic curves, $dx/dt = \lambda^{(\nu)}$. As we discussed with linear advection and Burgers' equation, along the characteristics the solution is constant. Across them, the solution jumps. For a linear system, the characteristics are straight lines.

Imagine initial conditions consisting of a jump in the primitive variables, $\Delta\mathbf{q}$. The corresponding jumps in characteristic variables are $\Delta\mathbf{w} \sim \mathbf{L}\Delta\mathbf{q}$ (where the $\sim$ accounts for the fact that in a nonlinear system, $\mathbf{L} = \mathbf{L}(\mathbf{q})$). The characteristic form of the equations says that each of the waves will carry its respective jump, $\Delta\mathbf{w}$. Since $d\mathbf{q} = \mathbf{L}^{-1}d\mathbf{w} = \mathbf{R}d\mathbf{w}$, the jump in the primitive variable across each wave is proportional to the right-eigenvector associated with that wave. So, for example, since $\mathbf{r}^{(\circ)}$ is only non-zero for the density element (see Eq. 7.24), this then means that only density jumps across the $\lambda^{(\circ)} = u$ wave—pressure and velocity are constant across this wave (see for example, Toro [82], Ch. 2, 3 or LeVeque [46] for a thorough discussion).

Figure 7.1 shows the three waves emanating from an initial discontinuity. This is a classic problem called the Sod problem [74]. The initial conditions are

$$\begin{aligned} \rho_l &= 1 & \rho_r &= 1/8 \\ u_l &= 0 & u_r &= 0 \\ p_l &= 1 & p_r &= 1/10 \end{aligned} \tag{7.37}$$

Here we see the three waves propagating away from the initial discontinuity. The left $(u - c)$ wave is a rarefaction, the middle (u) is the contact discontinuity, and the right $(u + c)$ is a shock. Note that all 3 primitive variables jump across the left and right waves, but only the density jumps across the middle wave. This reflects the right eigenvectors. Also note that no waves have reached the far left and far right yet, the conditions there are the same as the initial conditions.

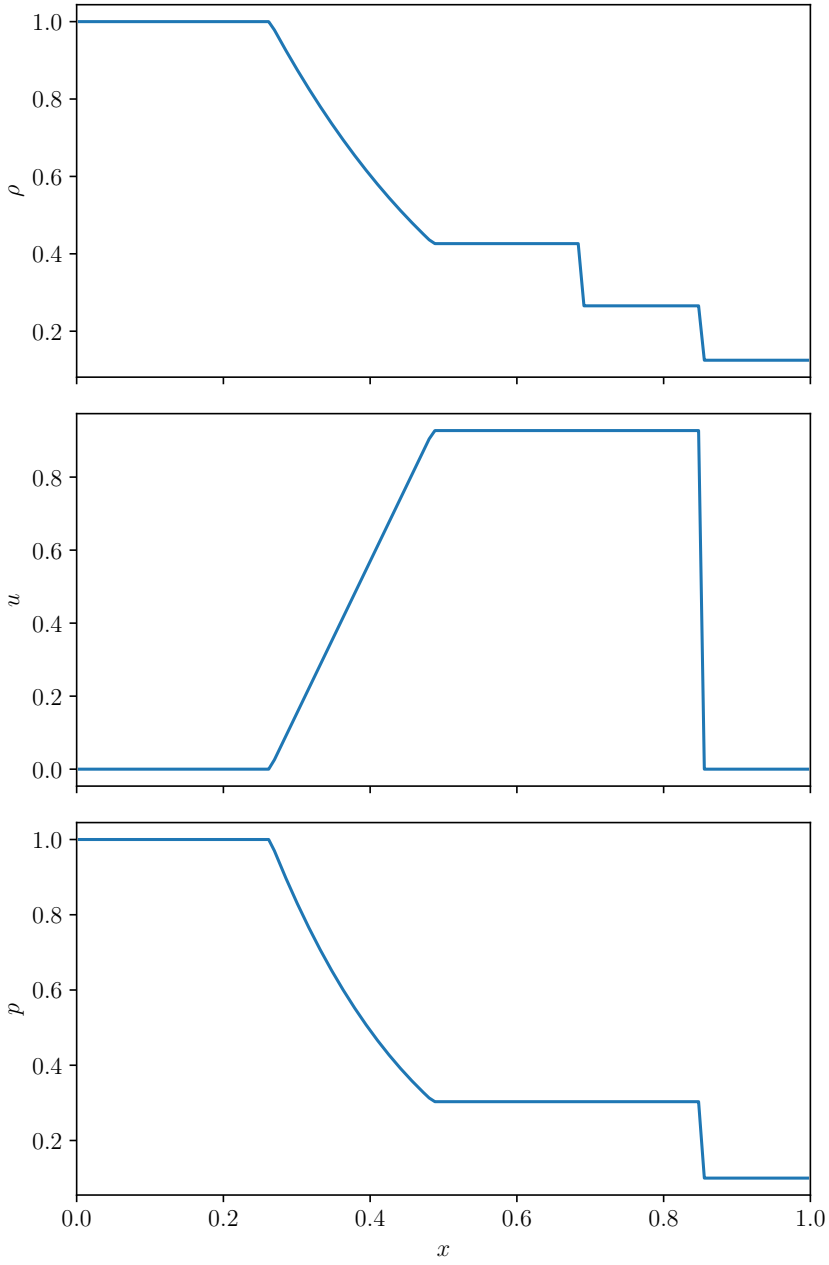

Figure 7.1: Evolution following from an initial discontinuity at $x = 0.5$. These particular conditions are called the *Sod problem*, and in general, a setup with two states separated by a discontinuity is called a shock-tube problem. We see three waves carrying changes in the solution.
hydro_examples: `riemann-sod.py`

7.2 The Riemann problem

Just like with advection and Burgers' equation, we will need to solve a Riemann problem. However, as our system is nonlinear, and has 3 waves, the solution is considerably more complex, but the general ideas are straightforward. Here we review the basic outline of operations, and refer to Toro [82] for full details on a variety of methods for solving the Riemann problem.

The Riemann problem consists of a left and right state separated by an interface. For the Euler equations, there are three eigenvalues, which are the speeds at which information propagates. Each of these correspond to a wave that will move out from the interface with time, and each wave will carry with it a jump in the characteristic variables. Figure 7.2 shows the three waves moving out from the interface, separating space into 4 regions, marked: L, L_*, R_*, and R. We typically work in terms of

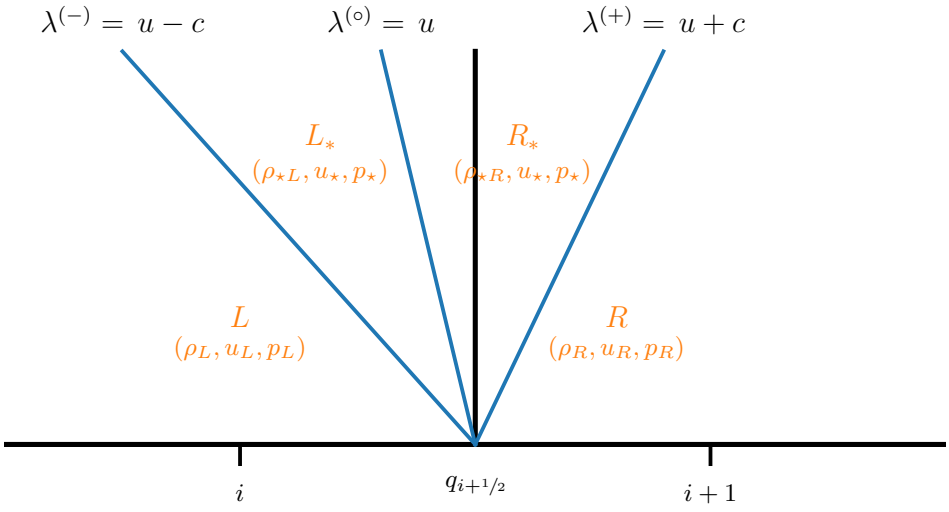

Figure 7.2: The wave structure and 4 distinct regions for the Riemann problem. Time can be thought of as the vertical axis here, so we see the waves moving outward from the interface.

primitive variables. The states in the L and R regions are simply the left and right input states—the waves have not had time to reach here, so they are unmodified.

Looking at the middle right eigenvector, $\mathbf{r}^{(\circ)}$, we identify the middle wave as a *contact discontinuity*. There is neither compression nor expansion across this wave (since the velocity does not jump across it) and it simply separates two states. We also know from the eigenvectors that the pressure is constant across the contact.

The left and right waves can be either a rarefaction or a shock. It will be a shock if

the flow is compressing and a rarefaction otherwise.

We already see that the only unknowns are $u_\star$, $p_\star$, $\rho_{\star,L}$, and $\rho_{\star,R}$.

7.2.1 Rarefactions

The conditions across a rarefaction can be understood by considering a system where entropy replaces pressure, $\mathbf{q}_s = (\rho, u, s)^\intercal$. The entropy equation is simply $Ds/Dt = 0$. We need to express the pressure gradient in the velocity equation in terms of $\mathbf{q}_s$.

$$\frac{\partial p(\rho, s)}{\partial x} = \left.\frac{\partial p}{\partial s}\right|_\rho \frac{\partial s}{\partial x} + \left.\frac{\partial p}{\partial \rho}\right|_s \frac{\partial \rho}{\partial x} = \left.\frac{\partial p}{\partial s}\right|_\rho \frac{\partial s}{\partial x} + \frac{p\Gamma_1}{\rho}\frac{\partial \rho}{\partial x} \tag{7.38}$$

giving

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\left[\left.\frac{\partial p}{\partial s}\right|_\rho \frac{\partial s}{\partial x} + \frac{p\Gamma_1}{\rho}\frac{\partial \rho}{\partial x}\right] = 0 \tag{7.39}$$

This gives a system:

$$\mathbf{q}_{s_t} + \mathbf{A}_s(\mathbf{q}_s)\mathbf{q}_{s_x} = 0 \tag{7.40}$$

where

$$\mathbf{A}_s = \begin{pmatrix} u & \rho & 0 \\ c^2/\rho & u & \frac{1}{\rho}\left.\frac{\partial p}{\partial s}\right|_\rho \\ 0 & 0 & u \end{pmatrix} \tag{7.41}$$

The eigenvalues of $\mathbf{A}_s$ are again u, $u - c$, and $u + c$. The right eigenvectors are**

$$\mathbf{r}_s^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ 0 \end{pmatrix} \qquad \mathbf{r}_s^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ -c^2/p_s \end{pmatrix} \qquad \mathbf{r}_s^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ 0 \end{pmatrix} \tag{7.42}$$

Here, we define $p_s \equiv \partial p/\partial s|_\rho$. Since the jump in $\mathbf{q}_s$ is proportional to these eigenvectors, we see that entropy does not change across the left, $(-)$, and right, $(+)$, waves.

Consider the $(+)$ wave, which moves at a speed $\lambda^{(+)} = u + c$. Across this wave, the characteristic variable $\mathbf{w}^{(+)}$ will jump, but the others will not. Similarly, $\mathbf{w}^{(+)}$ will be constant across the $(-)$ wave. So we can use this constancy to tell us how the evolution behaves across that wave. The constancy of the middle and right characteristic variables across the left wave gives us the conditions (using our original primitive variable system):

$$\mathbf{l}^{(+)} \cdot d\mathbf{q} = 0 \tag{7.43}$$

$$\mathbf{l}^{(\circ)} \cdot d\mathbf{q} = 0 \tag{7.44}$$

**Again, see the SymPy Jupyter notebook: hydro_examples: `euler.ipynb`

or

$$\begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \begin{pmatrix} d\rho \\ du \\ dp \end{pmatrix} = 0 \qquad \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \begin{pmatrix} d\rho \\ du \\ dp \end{pmatrix} = 0 \tag{7.45}$$

Expanding these, we have the system:

$$du + \frac{1}{\rho c} dp = 0 \tag{7.46}$$

$$d\rho - \frac{1}{c^2} dp = 0 \tag{7.47}$$

Defining the *Lagrangian sound speed*, $C \equiv \rho c$, and the specific volume, $\tau = 1/\rho$, we can rewrite this system as:

$$du = -\frac{dp}{C}, \;\; d\tau = -\frac{dp}{C^2} \quad \text{across the left wave} \tag{7.48}$$

Similar arguments would give the condition across the right wave as:

$$du = \frac{dp}{C}, \;\; d\tau = -\frac{dp}{C^2} \quad \text{across the right wave} \tag{7.49}$$

These are completely general at this point. Note that the condition of the second wave, $d\tau/dp$ is essentially the definition of the adiabatic index Γ_1, $dp/p - \Gamma_1 d\rho/\rho = 0$ for constant entropy. Finding the solution across the rarefaction then involves integrating the system from $p_{l,r}$ to $p_\star$, where l, r is the original left or right state, depending on which wave you are considering. Coupled with this, we need our equation of state to provide $C = C(\tau, p)$. These solutions are called the *Riemann invariants*.

Considerable simplification can be made if we assume a gamma-law gas. The eigensystem with entropy showed us that entropy is constant across the solution. This allows us to write our equation of state as:

$$p = K\rho^\gamma \tag{7.50}$$

where K is a constant that depends on the entropy, and do the integrals analytically.

Exercise 7.5

Assume a gamma-law gas, then you only need to integrate a single equation to find the solution across the left rarefaction:

$$u = -\int \frac{dp}{\rho c} \tag{7.51}$$

where $c = \sqrt{\gamma p/\rho}$. Using $p = K\rho^\gamma$, show that

$$u + \frac{2c}{\gamma - 1} = \text{constant} \quad \text{across the left rarefaction} \tag{7.52}$$

This allows you to link the state to the left of the rarefaction to the star state:

$$u_L + \frac{2c_L}{\gamma - 1} = u_\star + \frac{2c_\star}{\gamma - 1} \tag{7.53}$$

and

$$\frac{p_L}{\rho_L^\gamma} = \frac{p_\star}{\rho_\star^\gamma} \tag{7.54}$$

from our constant-entropy equation of state. Together, show that this gives:

$$u_{\star,L} = u_L + \frac{2c_L}{\gamma - 1}\left[1 - \left(\frac{p_\star}{p_L}\right)^{(\gamma-1)/2\gamma}\right] \tag{7.55}$$

A similar construction can be done for the right rarefaction, yielding:

$$u_{\star,R} = u_R - \frac{2c_R}{\gamma - 1}\left[1 - \left(\frac{p_\star}{p_R}\right)^{(\gamma-1)/2\gamma}\right] \tag{7.56}$$

For a general equation of state, there is not a closed form, but we would need to integrate our system to get $u_\star = u(p_\star)$ and $\rho_{\star,s} = 1/\tau(p_\star)$.

We'll denote the solution across the left rarefaction as $u_\star = u_L^{rare}(p_\star)$ and across the right rarefaction as $u_\star = u_R^{rare}(p_\star)$.

7.2.2 Shocks

Entropy is not constant across a shock—there is dissipation, so we need a different way to connect the states across the shock. As with Burgers' equation, we can understand the shock by looking at the Rankine-Hugoniot jump conditions. There will be one condition for each of our conservation laws, and together they tell us the speed of the shock and how density and pressure jump across it.

The Rankine-Hugoniot conditions are:

$$\frac{\mathbf{F}(\boldsymbol{\mathcal{U}}_\star) - \mathbf{F}(\boldsymbol{\mathcal{U}}_s)}{\boldsymbol{\mathcal{U}}_\star - \boldsymbol{\mathcal{U}}_s} = S \tag{7.57}$$

where S is the shock speed. It is easiest to work in the frame of the shock. If the shock speed is S, then we define the velocity in the shock frame as:

$$\hat{u}_s = u_s - S \tag{7.58}$$

$$\hat{u}_\star = u_\star - S \tag{7.59}$$

and the Rankine-Hugoniot conditions become:

$$\frac{\mathbf{F}(\hat{\boldsymbol{U}}_\star) - \mathbf{F}(\hat{\boldsymbol{U}}_s)}{\hat{\boldsymbol{U}}_\star - \hat{\boldsymbol{U}}_s} = 0 \tag{7.60}$$

For the one-dimensional system of Euler equations, we have the conditions:

$$\rho_\star \hat{u}_\star = \rho_s \hat{u}_s \tag{7.61}$$

$$\rho_\star \hat{u}_\star^2 + p_\star = \rho_s \hat{u}_s^2 + p_s \tag{7.62}$$

$$\rho_\star \hat{u}_\star e_\star + \frac{1}{2}\rho_\star \hat{u}_\star^3 + \hat{u}_\star p_\star = \rho_s \hat{u}_s e_s + \frac{1}{2}\rho_s \hat{u}_s^3 + \hat{u}_s p_s \tag{7.63}$$

Our goal is to find how each variable jumps across the shock. We'll work this out for the general EOS case, following the ideas in [25].

Starting with the mass flux, we can express:

$$\hat{u}_\star = \frac{\rho_s}{\rho_\star}\hat{u}_s \tag{7.64}$$

and then insert this into the momentum equation to get:

$$\rho_s^2 \left(\frac{1}{\rho_s} - \frac{1}{\rho_\star}\right) \hat{u}_s^2 = p_\star - p_s \tag{7.65}$$

Introducing compact notation for the jump across the shock:

$$[q] \equiv q_\star - q \tag{7.66}$$

we have:

$$-[\tau] = \frac{[p]}{\rho_s^2 \hat{u}_s^2} \tag{7.67}$$

We now introduce the mass flux, W_s. For a shock separating $L/L\star$ (the 1-shock), mass will be moving through the shock to the right, so, in the frame of the shock, the mass flux is

$$W_L \equiv \rho_L \hat{u}_L = \rho_\star \hat{u}_\star \tag{7.68}$$

Likewise, for a shock separating $R/R\star$ (the 3-shock), mass will move to the left passing through the shock, and in the frame of the shock, the mass flux is

$$W_R \equiv -\rho_R \hat{u}_R = -\rho_\star \hat{u}_\star \tag{7.69}$$

Thus, our first jump condition becomes

$$-[\tau] = \frac{[p]}{W_s^2} \tag{7.70}$$

Next, going back to the momentum equation, we can substitute in the mass flux. For the left (1-shock), we have:

$$W_L \hat{u}_\star + p_\star = W_L \hat{u}_L + p_L \tag{7.71}$$

giving

$$[u] = -\frac{[p]}{W_L} \tag{7.72}$$

where we recognized that $\hat{u}_\star - \hat{u}_L = u_\star - u_L = [u]$, since the shock speed is the same for $u_\star$ and u.

Likewise, for the right (3-shock), we have:

$$-W_R \hat{u}_\star + p_\star = -W_R \hat{u}_R + p_R \tag{7.73}$$

giving

$$[u] = \frac{[p]}{W_R} \tag{7.74}$$

The last jump condition is for energy. Since all the terms in the energy equation are proportional to velocity, there will be no sign difference between the left and right shock jump conditions. We start by introducing the mass flux:

$$W_s e_\star + \frac{1}{2} W_s \hat{u}_\star^2 + \frac{W_s}{\rho_\star} p_\star = W_s e_s + \frac{1}{2} W_s \hat{u}_s^2 + \frac{W_s}{\rho_s} p_s \tag{7.75}$$

The mass flux cancels, leaving

$$[e] + \frac{p_\star}{\rho_\star} - \frac{p_s}{\rho_s} + \frac{1}{2}\left(\hat{u}_\star^2 - \hat{u}_s^2\right) = 0 \tag{7.76}$$

getting rid of the velocities using $\hat{u}_\star^2 = W_s^2/\rho_\star^2$ and $\hat{u}_s^2 = W_s^2/\rho_s^2$, we have:

$$[e] + \frac{p_\star}{\rho_\star} - \frac{p_s}{\rho_s} + \frac{1}{2} W_s^2 \left(\frac{1}{\rho_\star^2} - \frac{1}{\rho_s^2}\right) = 0 \tag{7.77}$$

Then introducing $W_s^2 = -[p]/[\tau]$, and after a lot of algebra, we arrive at:

$$[e] = -\frac{p_\star + p_s}{2}[\tau] \tag{7.78}$$

Some sources write $\bar{p} \equiv (p_\star + p_s)/2$.

To summarize, our jump conditions across the shock are:

$$[\tau] = -\frac{[p]}{W_s^2} \tag{7.79}$$

$$[u] = \mp\frac{[p]}{W_s} \quad \text{'}-\text{' for left, '+' for right} \tag{7.80}$$

$$[e] = -\frac{p_\star + p_s}{2}[\tau] \tag{7.81}$$

As with the rarefaction, the goal is to express this as a function, $u_\star = u_s^{\rm shock}(p_\star)$. The general solution procedure is starts with a proposed value for $p_\star$, then:

1. root find to find $\rho_\star$ corresponding to the $p_\star$:
   (a) guess a value for $\rho_\star$
   (b) using the equation of state, express $e_\star = e(p_\star, \rho_\star)$
   (c) use Newton's method (or another technique) with $[e] = -\bar{p}[\tau]$ to find a correction to $\rho_\star$
2. compute
$$\frac{1}{W_s^2} = -\frac{[\tau]}{[p]} \tag{7.82}$$
3. find the star velocity:
$$u_\star = u_s \mp \frac{[p]}{W_s} \tag{7.83}$$
   where we use '$-$' for the left shock, and '+' for the right shock

The one other piece of information we need is the shock speed. We can get this from the mass flux, W_s, definition, e.g., $W_L = \rho_L \hat{u}_L = \rho_L(u_L - S)$,:

$$S = u_s \mp \frac{W_s}{\rho_s} \quad \text{'}-\text{' for left, '+' for right} \tag{7.84}$$

This procedure gives us $\rho_{\star,s}$, $p_\star$, and S, the shock speed.

Just as with the rarefaction, considerable simplification can be made if we assume a gamma-law gas.

Exercise 7.6

Introducing

$$e = \frac{p}{\rho}\frac{1}{\gamma - 1} \tag{7.85}$$

into the jump condition for energy, Eq. 7.81, show that we can express the jump in density in terms of the ratio of pressure, $p_\star / p_s$, as:

$$\rho_{\star,s} = \rho_s \left[\frac{\frac{p_\star}{p_s}(\gamma+1) + (\gamma-1)}{(\gamma+1) + \frac{p_\star}{p_s}(\gamma-1)} \right] \tag{7.86}$$

Now, compute the mass flux, W_s starting with

$$W_s^2 = -\frac{[p]}{[\tau]} \tag{7.87}$$

use the γ-law equation of state and show that

$$W_s^2 = \frac{1}{2} p_s \rho_s \left[\left(\frac{p_\star}{p_s} \right) (\gamma + 1) + (\gamma - 1) \right] \tag{7.88}$$

With this, show that the star velocity is:

$$u_\star = u_s \pm c \left[\frac{2}{\gamma(\gamma-1)} \right]^{1/2} \frac{1 - \frac{p_\star}{p_s}}{\left(\frac{p_\star}{p_s} \frac{\gamma+1}{\gamma-1} + 1 \right)^{1/2}} \tag{7.89}$$

with '+' for the left shock and '−' for the right shock. Also show that the shock speed is:

$$S = u_s \mp c \left[\left(\frac{p_\star}{p_s} \right) \frac{\gamma+1}{2\gamma} + \frac{\gamma-1}{2\gamma} \right]^{1/2} \tag{7.90}$$

with the '−' or the left shock and the '+' for the right shock.

7.2.3 Finding the Star State

The left and right states are connected to the state in the star region by a Hugoniot curve—this is a curve in the u-p plane that shows all of the possible states one can reach from the current state through either a shock or rarefaction. There are two such curves, one corresponding to the left and one to the right state:

$$u_{\star,L}(p) = \begin{cases} u_{\star,L}^{\text{shock}}(p) & p > p_L \\ u_{\star,L}^{\text{rare}}(p) & p \le p_L \end{cases} \qquad u_{\star,r}(p) = \begin{cases} u_{\star,R}^{\text{shock}}(p) & p > p_R \\ u_{\star,R}^{\text{rare}}(p) & p \le p_R \end{cases} \tag{7.91}$$

The solution to the Riemann problem is the point in the u-p plane where these two curves intersect, e.g., we solve for $p_\star$ in:

$$u_{\star,l}(p_\star) - u_{\star,r}(p_\star) = 0 \tag{7.92}$$

This is equivalent to saying that pressure and velocity do not jump across the contact wave.

For a gamma-law equation of state, using the results from the exercises, we have:

$$u_{\star,s}(p_\star) = \begin{cases} u_s \pm \frac{2c}{\gamma-1} \left[1 - \left(\frac{p_\star}{p_s} \right)^{(\gamma-1)/2\gamma} \right] & p_\star \le p_s \\ u_s \pm c \left[\frac{2}{\gamma(\gamma-1)} \right]^{1/2} \frac{1 - \frac{p_\star}{p_s}}{\left(\frac{p_\star}{p_s} \frac{\gamma+1}{\gamma-1} + 1 \right)^{1/2}} & p_\star > p_s \end{cases} \tag{7.93}$$

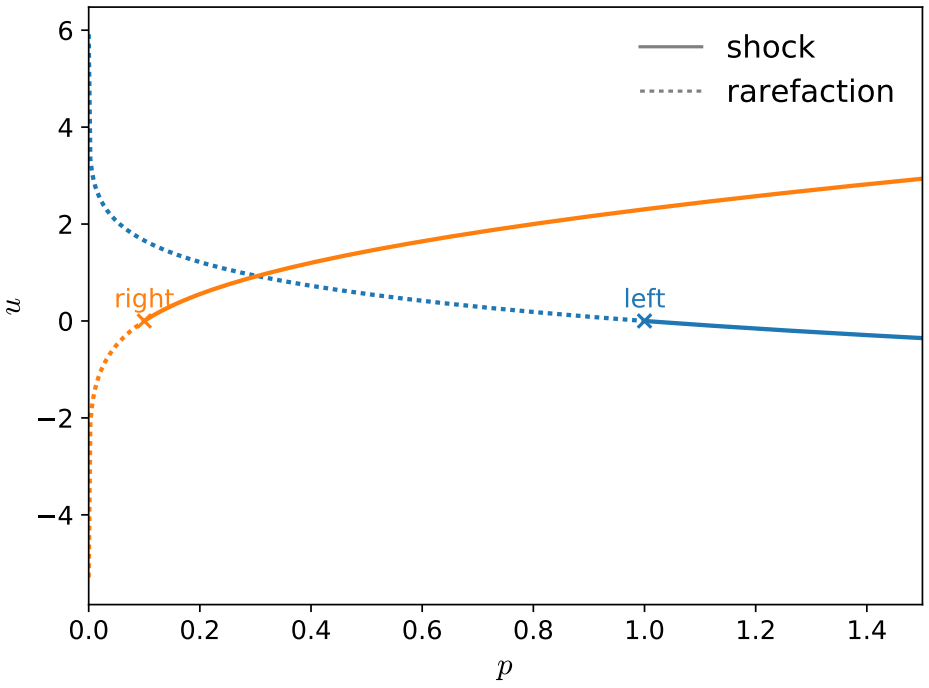

Figure 7.3: The Hugoniot curves corresponding to the Sod problem. The shock and rarefaction curves are shown. The solution to the Riemann problem is the point where the curves intersect. The line style of the curve indicates where we are a shock or rarefaction. Where $p > p_s$, where $s \in L, R$, we have a shock.
hydro_examples: `riemann-phase.py`

Figure 7.3 shows the Hugoniot curves for the Sod problem. Comparing to Figure 7.1, we see that the right state is linked to the star state with a shock while the left state is linked to the star state with a rarefaction.

Exercise 7.7

Download the code demonstrated in Figure 7.3 and experiment with different initial conditions to see how the solution changes based on the states. Try to create initial states that give rise to two rarefactions and a separate set of states that give rise to two shocks.

7.2.4 Complete Solution

To complete the solution, we need to find which of the 4 regions, $L, L\star, R\star, R$ we fall in. For hydrodynamics, we are usually interested only in the solution on the interface, but we can look along any ray in the x-t plane by defining $\xi = (x - x_{\text{interface}})/t$. If the initial discontinuity is on the interface, then $\xi = 0$.

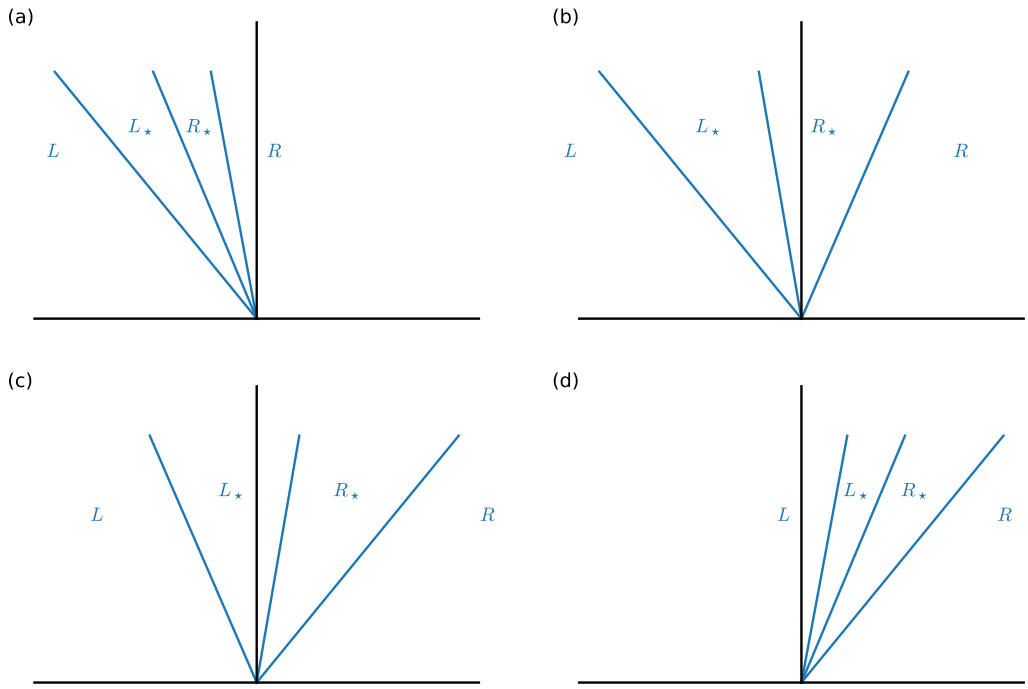

Figure 7.4: An illustration of the 3 waves emanating from an initial discontinuity (the origin of the axes). The 4 cases show all possible cases of waves being to the left or right of the interface (corresponding to the choice $\xi = 0$). In all cases, the middle wave is a contact and the left (1) and right (3) waves are either a shock or a rarefaction. The state on the interface is: R (case a); $R\star$ (case b); $L\star$ (case c); L (case d).

Figure 7.4 shows the possible configurations of the waves. Note that the middle wave is always a contact but the left (1) and right (3) waves can be either a shock or rarefaction. To find out which region the interface falls in, we simply look at the speeds.

The first speed to consider is the contact wave, that has a speed of simply $S_c = u_\star$. If $S_c < \xi$, then we are choosing between R and $R_\star$ states (cases a and b). If $S_c > \xi$, then we are choosing between the L and $L_\star$ states (cases c and d in Figure 7.4)[††].

The direction of the contact reduces the problem down to choosing between just two regions, either L-$L_\star$ or R-$R_\star$. This leaves just a single wave to consider: the right wave for cases a and b; and the left wave for cases c and d. We need the wave speed for this—it will depend on whether it is a shock or a rarefaction.

For a shock, the wave speed is given by Eq. 7.84. We do the same procedure as before. For example, for cases a and b, we look at S_R, the right-moving shock speed for the

[††]for the special case where $S_c = 0$, we usually take the solution to be the average of the $L_\star$ and $R_\star$ states. Testing for this explicitly rather than using $\geq$ in a check is important for maintaining symmetry (see § 8.9.1).

3-wave. If $S_R > 0$, then the interface is in the $R_\star$ state (case b), while if $S_R < 0$, then the interface is in the R state (case a). A similar process holds for the left-moving shock and cases c and d.

For a rarefaction, the relevant speeds are the characteristic speeds on either side of the rarefaction. We usually talk about the leading part of the rarefaction (the *head*) and trailing part (the *tail*). The corresponding wave speeds are:

- left (1) rarefaction:
  - $\lambda_{\text{head}} = u_L - c_L$
  - $\lambda_{\text{tail}} = u_\star - c_\star$
- right (3) rarefaction:
  - $\lambda_{\text{head}} = u_R + c_R$
  - $\lambda_{\text{tail}} = u_\star + c_\star$

The nature of the rarefaction is such that it spreads out, $|\lambda_{\text{head}}| > |\lambda_{\text{tail}}|$. To determine which state we are in, we look to see where both the head and tail are with respect to the interface. Figure 7.5 shows the possibilities for a left (1) rarefaction. In case (a), $\lambda_{\text{head}}, \lambda_{\text{tail}} < 0$, so the $L_\star$ state is on the interface. In case (c), $\lambda_{\text{head}}, \lambda_{\text{tail}} > 0$, so the L state is on the interface. Case (b) is more complicated—we have $\lambda_{\text{head}} < 0, \lambda_{\text{tail}} > 0$, so the rarefaction spans the interface. We need to therefore solve for the state in the rarefaction itself.

To find the structure inside of a rarefaction, consider some point, (x, t), in the x-t plane. For an initial discontinuity at the origin, we can imagine a line connecting this point and the origin, which has the form:

$$\frac{x}{t} = u - c \tag{7.94}$$

7.3 Other thermodynamic equations

At times we will want to use alternate forms of the energy equation. The internal energy is governed by the first law of thermodynamics. In the absence of any heat sources, we have:

$$dq = 0 = de + pd(1/\rho) \tag{7.95}$$

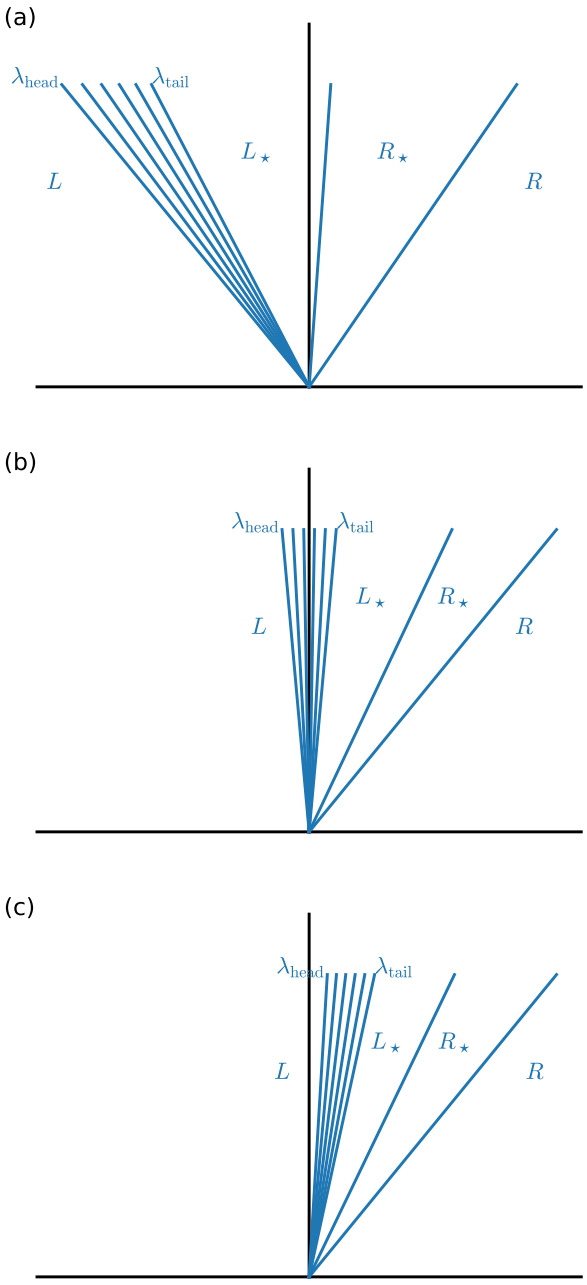

Figure 7.5: An illustration of the structure of the left rarefaction wave, for the case where the contact and right wave are to the right of the interface. Here, we are choosing between states L and $L_\star$. In (a), both the head and tail of the rarefaction are to the left of the interface, so state $L_\star$ is on the interface. In (c), both the head and tail of the rarefaction are to the right of the interface, so state L is on the interface. Inbetween, case (b), shows a rarefaction that spans the interface. For this case, we need to integrate the Riemann invariants to find the state on the interface.

where e is the specific internal energy. Applying this to a Lagrangian fluid element, we have:

$$\frac{De}{Dt} + p\frac{D(1/\rho)}{Dt} = 0 \tag{7.96}$$

$$\frac{De}{Dt} - \frac{p}{\rho^2}\frac{D\rho}{Dt} = 0 \tag{7.97}$$

$$\rho\frac{De}{Dt} + p\nabla\cdot\mathbf{U} = 0 \tag{7.98}$$

where we used the continuity equation in the last step to eliminate $D\rho/Dt$. This can be rewritten by adding $e\times$ the continuity equation to give:

$$\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho\mathbf{U}e) + p\nabla\cdot\mathbf{U} = 0 \tag{7.99}$$

Notice that internal energy, (ρe) is not a conserved quantity (in particular, the $p\nabla\cdot\mathbf{U}$ term is not in conservative form).

Another energy-like quantity that we can consider is specific enthalpy,

$$h = e + \frac{p}{\rho} \tag{7.100}$$

Differentiating this, and using the internal energy equation,

$$\frac{Dh}{Dt} = \frac{De}{Dt} - \frac{p}{\rho^2}\frac{D\rho}{Dt} + \frac{1}{\rho}\frac{Dp}{Dt} \tag{7.101}$$

$$= \frac{p}{\rho^2}\frac{D\rho}{Dt} - \frac{p}{\rho^2}\frac{D\rho}{Dt} + \frac{1}{\rho}\frac{Dp}{Dt} \tag{7.102}$$

so

$$\rho\frac{Dh}{Dt} = \frac{Dp}{Dt} \tag{7.103}$$

This form is useful for very subsonic flows (see, e.g. [13]), where $Dp/Dt = 0$, which then shows that enthalpy is conserved:

$$\frac{\partial(\rho h)}{\partial t} + \nabla\cdot(\rho h\mathbf{U}) = 0 \tag{7.104}$$

(here we added $h\times$ the continuity equation to transform from the Lagrangian derivative to a conservation equation).

We can also look at the temperature evolution. It is interesting to approach this from both the internal energy and enthalpy equations. Starting with internal energy, writing it as $e(\rho, T)$, we have:

$$\frac{De}{Dt} = \left.\frac{\partial e}{\partial\rho}\right|_T \frac{D\rho}{Dt} + \left.\frac{\partial e}{\partial T}\right|_\rho \frac{DT}{Dt} = -\frac{p}{\rho}\nabla\cdot\mathbf{U} \tag{7.105}$$

The specific heat at constant volume is defined as $c_v = \partial e / \partial T|_\rho$, allowing us to write:

$$c_v \frac{DT}{Dt} = \left(\left. \frac{\partial e}{\partial \rho} \right|_T - \frac{p}{\rho} \right) \nabla \cdot \mathbf{U} \tag{7.106}$$

If we alternately start with enthalpy, expressing it as $h = h(p, T)$, we have:

$$\frac{Dh}{Dt} = \left. \frac{\partial h}{\partial p} \right|_T \frac{Dp}{Dt} + \left. \frac{\partial h}{\partial T} \right|_p \frac{DT}{Dt} = \frac{1}{\rho} \frac{Dp}{Dt} \tag{7.107}$$

The specific heat at constant pressure is defined as $c_p = \partial h / \partial T|_p$, letting us write:

$$c_p \frac{DT}{Dt} = \left(\frac{1}{\rho} - \left. \frac{\partial h}{\partial p} \right|_T \right) \frac{Dp}{Dt} \tag{7.108}$$

These two temperature evolution equations are equivalent, but one describes the evolution in terms of density and the other in terms of pressure. Later, we'll see how these can be useful when we approximate the evolution of reactive flow under constant density or constant pressure behaviors.

Chapter 8

Euler Equations: Numerical Methods

8.1 Introduction

The numerical methods we use for the Euler equations mirror those used with advection in Ch. 4. The basic process is:

- Reconstruct the state variables to the interfaces.
- Solve the Riemann problem (as described in § 7.2) and construct the fluxes through the interfaces.
- Perform the conservative update on the state variables.

The Euler equations are much more complicated than linear advection—we already saw this with the Riemann problem. But we will also see that in different parts of the algorithm we will use conservative or primitive variables. Additional physics can enter as source terms, which we'll see how to add to both the interface states and conservative update here.

8.2 Reconstruction of interface states

We will solve the Euler equations using a high-order *Godunov method*—a finite volume method whereby the fluxes through the interfaces are computed by solving the Riemann problem for our system. The finite-volume update for our system appears as:

$$\mathcal{U}_i^{n+1} = \mathcal{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2}\right) \tag{8.1}$$

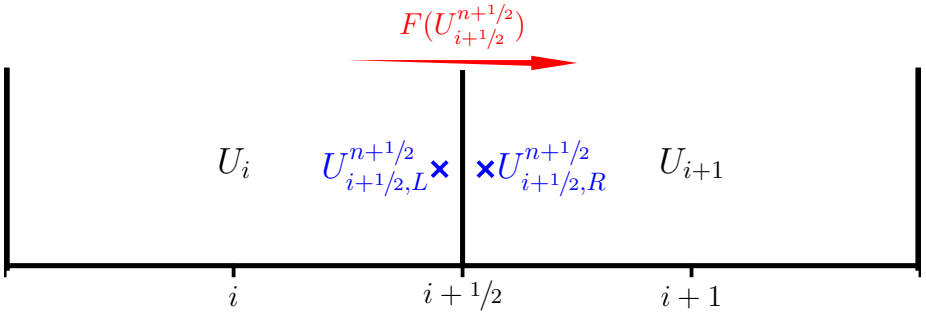

Figure 8.1: The left and right states at interface $i + 1/2$. The arrow indicates the flux through the interface, as computed by the Riemann solver using these states as input.

This says that each of the conserved quantities in $\boldsymbol{\mathcal{U}}$ change only due to the flux of that quantity through the boundary of the cell.

Instead of approximating the flux itself on the interface, we find an approximation to the state on the interface, $\boldsymbol{\mathcal{U}}^{n+1/2}_{i-1/2}$ and $\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2}$ and use this with the flux function to define the flux through the interface:

$$\mathbf{F}^{n+1/2}_{i-1/2} = \mathbf{F}(\boldsymbol{\mathcal{U}}^{n+1/2}_{i-1/2}) \tag{8.2}$$

$$\mathbf{F}^{n+1/2}_{i+1/2} = \mathbf{F}(\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2}) \tag{8.3}$$

To find this interface state, we predict left and right states at each interface (centered in time), which are the input to the Riemann solver. The Riemann solver will then look at the characteristic wave structure and determine the fluid state on the interface, which is then used to compute the flux. This is illustrated in Figure 8.1.

Finally, although we use the conserved variables for the final update, in constructing the interface states it is often easier to work with the primitive variables. These have a simpler characteristic structure. The interface states in terms of the primitive variables can be converted into the interface states of the conserved variables through a simple algebraic transformation,

$$\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,L} = \boldsymbol{\mathcal{U}}(\mathbf{q}^{n+1/2}_{i+1/2,L}) \tag{8.4}$$

As we saw with linear advection in § 5.1, at the start of a timestep, each zones contains cell-averages. Consider a cell average quantity, $\mathbf{q}_i$. Constructing the interface states requires reconstructing the cell-average data as a continuous function, $\mathbf{q}(x)$, defined for each cell. This polynomial is only *piecewise* continuous, since each cell has its own $\mathbf{q}(x)$. Standard choices are piecewise constant,

$$\mathbf{q}(x) = \mathbf{q}_i \tag{8.5}$$

piecewise linear,

$$\mathbf{q}(x) = \mathbf{q}_i + \frac{\Delta \mathbf{q}_i}{\Delta x}(x - x_i) \tag{8.6}$$

or piecewise parabolic,

$$\mathbf{q}(x) = \boldsymbol{\alpha}(x - x_i)^2 + \boldsymbol{\beta}(x - x_i) + \boldsymbol{\gamma} \tag{8.7}$$

where in each case, the average of $\mathbf{q}(x)$ over the cell gives $\mathbf{q}_i$. Most of the complexity of the method is then figuring out the coefficients of the polynomial.

Characteristic tracing is then done under this polynomial to see how much of each characteristic quantity comes to the interface over the timestep. The jump in the primitive variables is projected into the characteristic variables, and only jumps moving toward the interface are included in our reconstruction. We look at several methods below that build off of these ideas below.

8.2.1 Piecewise constant

The simplest possible reconstruction of the data is piecewise constant. This is what was done in the original Godunov method [37]. For the interface marked by $i + 1/2$, the left and right states on the interface are simply:

$$\mathcal{U}_{i+1/2,L} = \mathcal{U}_i \tag{8.8}$$
$$\mathcal{U}_{i+1/2,R} = \mathcal{U}_{i+1} \tag{8.9}$$

This does not take into account in any way how the state $\mathcal{U}$ may be changing through the cell. As a result, it is first-order accurate in space, and since no attempt was made to center it in time, it is first-order accurate in time.

8.2.2 Piecewise linear

For higher-order reconstruction, we first convert from the conserved variables, $\mathcal{U}$, to the primitive variables, $\mathbf{q}$. These have a simpler characteristic structure, making them easier to work with. Here we consider piecewise linear reconstruction—the cell average data is approximated by a line with non-zero slope within each cell. Figure 8.2 shows the piecewise linear reconstruction of some data.

Consider constructing the left state at the interface $i + 1/2$ (see Figure 8.1). Just like for the advection equation, we do a Taylor expansion through $\Delta x/2$ to bring us to the interface, and $\Delta t/2$ to bring us to the midpoint in time. Starting with $\mathbf{q}_i$, the cell-centered primitive variable, expanding to the right interface (to create the left state

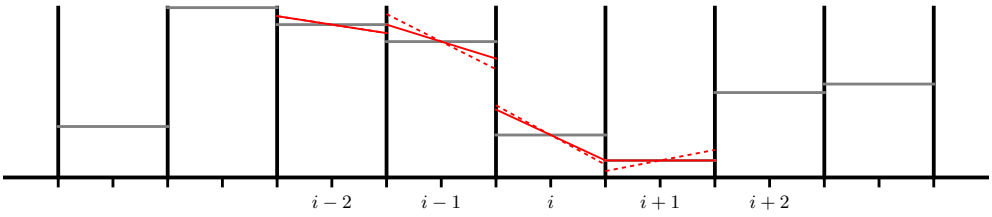

Figure 8.2: Piecewise linear reconstruction of the cell averages. The dotted line shows the unlimited center-difference slopes and the solid line shows the limited slopes.

there) gives:

$$\mathbf{q}_{i+1/2,L}^{n+1/2} = \mathbf{q}_i^n + \frac{\Delta x}{2}\frac{\partial \mathbf{q}}{\partial x}\bigg|_i + \frac{\Delta t}{2}\underbrace{\frac{\partial \mathbf{q}}{\partial t}\bigg|_i}_{=-\mathbf{A}\partial\mathbf{q}/\partial x} + \ldots \tag{8.10}$$

$$= \mathbf{q}_i^n + \frac{\Delta x}{2}\frac{\partial \mathbf{q}}{\partial x}\bigg|_i - \frac{\Delta t}{2}\left(\mathbf{A}\frac{\partial \mathbf{q}}{\partial x}\right)_i \tag{8.11}$$

$$= \mathbf{q}_i^n + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\mathbf{A}_i\right]\Delta\mathbf{q}_i \tag{8.12}$$

where $\Delta\mathbf{q}_i$ is the reconstructed slope of the primitive variable in that cell (similar to how we compute it for the advection equation). We note that the terms truncated in the first line are $O(\Delta x^2)$ and $O(\Delta t^2)$, so our method will be second-order accurate in space and time.

As with the advection equation, we limit the slope such that no new minima or maxima are introduced. Any of the slope limiters used for linear advection apply here as well. We represent the limited slope as $\overline{\Delta\mathbf{q}}_i$.

We can decompose $\mathbf{A}\Delta\mathbf{q}$ in terms of the left and right eigenvectors and sum over all the waves that move *toward* the interface. First, we recognize that $\mathbf{A} = R\Lambda L$ and recognizing that the '1' in Eq. 8.12 is the identity, $I = LR$, we rewrite this expression as:

$$\mathbf{q}_{i+1/2,L}^{n+1/2} = \mathbf{q}_i^n + \frac{1}{2}\left[\mathbf{RL} - \frac{\Delta t}{\Delta x}\mathbf{R\Lambda L}\right]_i \overline{\Delta\mathbf{q}}_i \tag{8.13}$$

We see the common factor of $\mathbf{L}\Delta\mathbf{q}$. We now write this back in component form. Consider:

$$\mathbf{R\Lambda L}\overline{\Delta\mathbf{q}} = \begin{pmatrix} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{pmatrix} \begin{pmatrix} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{pmatrix} \begin{pmatrix} l_1^{(-)} & l_2^{(-)} & l_3^{(-)} \\ l_1^{(\circ)} & l_2^{(\circ)} & l_3^{(\circ)} \\ l_1^{(+)} & l_2^{(+)} & l_3^{(+)} \end{pmatrix} \begin{pmatrix} \overline{\Delta\rho} \\ \overline{\Delta u} \\ \overline{\Delta p} \end{pmatrix} \tag{8.14}$$

Starting with $\mathbf{L}\overline{\Delta\mathbf{q}}$, which is a vector with each component the dot-product of a left eigenvalue with $\overline{\Delta\mathbf{q}}$, we have

$$\mathbf{R\Lambda L}\overline{\Delta\mathbf{q}} = \left(\begin{array}{c|c|c} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{array}\right) \left(\begin{array}{ccc} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{array}\right) \left(\begin{array}{c} \mathbf{l}^{(-)} \cdot \overline{\Delta\mathbf{q}} \\ \mathbf{l}^{(\circ)} \cdot \overline{\Delta\mathbf{q}} \\ \mathbf{l}^{(+)} \cdot \overline{\Delta\mathbf{q}} \end{array}\right) \tag{8.15}$$

Next we see that multiplying this vector by $\mathbf{\Lambda}$ simply puts the eigenvalue with its respective eigenvector in the resulting column vector:

$$\mathbf{R\Lambda L}\overline{\Delta\mathbf{q}} = \left(\begin{array}{c|c|c} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{array}\right) \left(\begin{array}{c} \lambda^{(-)}\,\mathbf{l}^{(-)} \cdot \overline{\Delta\mathbf{q}} \\ \lambda^{(\circ)}\,\mathbf{l}^{(\circ)} \cdot \overline{\Delta\mathbf{q}} \\ \lambda^{(+)}\,\mathbf{l}^{(+)} \cdot \overline{\Delta\mathbf{q}} \end{array}\right) \tag{8.16}$$

Finally, the last multiply results in a column vector:

$$R\Lambda L\overline{\Delta\mathbf{q}} = \left(\begin{array}{c} r_1^{(-)}\,\lambda^{(-)}\,\mathbf{l}^{(-)} \cdot \overline{\Delta\mathbf{q}} + r_1^{(\circ)}\,\lambda^{(\circ)}\,\mathbf{l}^{(\circ)} \cdot \overline{\Delta\mathbf{q}} + r_1^{(+)}\,\lambda^{(+)}\,\mathbf{l}^{(+)} \cdot \overline{\Delta\mathbf{q}} \\ r_2^{(-)}\,\lambda^{(-)}\,\mathbf{l}^{(-)} \cdot \overline{\Delta\mathbf{q}} + r_2^{(\circ)}\,\lambda^{(\circ)}\,\mathbf{l}^{(\circ)} \cdot \overline{\Delta\mathbf{q}} + r_2^{(+)}\,\lambda^{(+)}\,\mathbf{l}^{(+)} \cdot \overline{\Delta\mathbf{q}} \\ r_3^{(-)}\,\lambda^{(-)}\,\mathbf{l}^{(-)} \cdot \overline{\Delta\mathbf{q}} + r_3^{(\circ)}\,\lambda^{(\circ)}\,\mathbf{l}^{(\circ)} \cdot \overline{\Delta\mathbf{q}} + r_3^{(+)}\,\lambda^{(+)}\,\mathbf{l}^{(+)} \cdot \overline{\Delta\mathbf{q}} \end{array}\right) \tag{8.17}$$

We can rewrite this compactly as:

$$\sum_\nu \lambda^{(\nu)}(\mathbf{l}^{(\nu)} \cdot \overline{\Delta\mathbf{q}})\mathbf{r}^{(\nu)} \tag{8.18}$$

where we use ν to indicate which wave we are summing over. A similar expansion is used for $\mathbf{RL}\overline{\Delta\mathbf{q}}$. In fact, any vector can be decomposed in this fashion:

$$\chi = \mathbf{I}\chi = \mathbf{RL}\chi = \sum_\nu (\mathbf{l}^{(\nu)} \cdot \chi)\mathbf{r}^{(\nu)} \tag{8.19}$$

And then it is easy to see that the above manipulations for $\mathbf{A}\Delta\mathbf{q}$ can be expressed as:

$$\mathbf{A}\Delta\mathbf{q} = \mathbf{A}\sum_\nu (\mathbf{l}^{(\nu)} \cdot \overline{\Delta\mathbf{q}})\mathbf{r}^{(\nu)} = \sum_\nu (\mathbf{l}^{(\nu)} \cdot \overline{\Delta\mathbf{q}})\mathbf{A}\mathbf{r}^{(\nu)} = \sum_\nu (\mathbf{l}^{(\nu)} \cdot \overline{\Delta\mathbf{q}})\lambda^{(\nu)}\mathbf{r}^{(\nu)} \tag{8.20}$$

where we used $\mathbf{A}\mathbf{r}^{(\nu)} = \lambda^{(\nu)}\mathbf{r}^{(\nu)}$. The quantity $(\mathbf{l}^{(\nu)} \cdot \overline{\Delta\mathbf{q}})$ that shows up here is the projection of the vector $\overline{\Delta\mathbf{q}}$ into the characteristic variable carried by wave ν. This sum shows, as discussed earlier, that each wave carries a jump in the characteristic variable, with the jump in the primitive variables proportion to the right eigenvector, $\mathbf{r}^{(\nu)}$.

The resulting vector for the left state is:

$$\mathbf{q}_{i+1/2,L}^{n+1/2} = \mathbf{q}_i^n + \frac{1}{2}\sum_{\nu;\lambda^{(\nu)}\geq 0}\left[1 - \frac{\Delta t}{\Delta x}\lambda_i^{(\nu)}\right](\mathbf{l}_i^{(\nu)} \cdot \overline{\Delta\mathbf{q}}_i)\mathbf{r}_i^{(\nu)} \tag{8.21}$$

Note that we make a slight change here, and only include a term in the sum if its wave is moving toward the interface ($\lambda^{(\nu)} \geq 0$). The quantity $\Delta t \lambda^{(\nu)}/\Delta x$ inside the brackets is simply the CFL number for the wave ν.

Starting with the data in the $i+1$ zone and expanding to the left, we can find the right state on the $i+1/2$ interface:

$$\mathbf{q}^{n+1/2}_{i+1/2,R} = \mathbf{q}^n_{i+1} - \frac{1}{2} \sum_{\nu;\lambda^{(\nu)}\leq 0} \left[1 + \frac{\Delta t}{\Delta x}\lambda^{(\nu)}_{i+1}\right] (\mathbf{l}^{(\nu)}_{i+1} \cdot \overline{\Delta \mathbf{q}}_{i+1})\mathbf{r}^{(\nu)}_{i+1} \tag{8.22}$$

A good discussion of this is in Miller & Colella [53] (Eq. 85). This expression is saying that each wave carries a jump in $\mathbf{r}^{(\nu)}$ and only those jumps moving toward the interface contribute to our interface state. This restriction of only summing up the waves moving toward the interface is sometimes called *characteristic tracing*. This decomposition in terms of the eigenvectors and eigenvalues is commonly called a *characteristic projection*. In terms of an operator, P, it can be expressed as:

$$P\chi = \sum_\nu (\mathbf{l}^{(\nu)} . \chi)\mathbf{r}^{(\nu)} \tag{8.23}$$

Exercise 8.1

Show that $P\mathbf{q} = \mathbf{q}$, using the eigenvectors corresponding to the primitive variable form of the Euler equations.

In the literature, sometimes a '$>$' or '$<$' subscript on P is used to indicate the characteristic tracing.

We could stop here, but Colella & Glaz [25] (p. 278) argue that the act of decomposing $\mathbf{A}$ in terms of the left and right eigenvectors is a linearization of the quasi-linear system, and we should minimize the size of the quantities that are subjected to this characteristic projection. To accomplish this, they suggest subtracting off a *reference state*. Saltzman (Eq. 8) further argues that since only jumps in the solution are used in constructing the interface state, and that the characteristic decomposition simply adds up all these jumps, we can subtract off the reference state and project the result. In other words, we can write:

$$\mathbf{q}^{n+1/2}_{i+1/2,L} - \mathbf{q}_{\rm ref} = \mathbf{q}^n_i - \mathbf{q}_{\rm ref} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\mathbf{A}_i\right]\overline{\Delta \mathbf{q}}_i \tag{8.24}$$

Then we subject the RHS to the characteristic projection—this tells us how much of the quantity $\mathbf{q}^{n+1/2}_{i+1/2,L} - \mathbf{q}_{\rm ref}$ reaches the interface. Colella & Glaz (p. 278) and Colella (Eq. 2.11) suggest

$$\mathbf{q}_{\rm ref} = \tilde{\mathbf{q}}_{i,L} \equiv \mathbf{q}^n_i + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\max(\lambda^{(+)}_i, 0)\right]\overline{\Delta \mathbf{q}}_i \tag{8.25}$$

where $\lambda^{(+)}$ is the fastest eigenvalue, and thus will see the largest portion of the linear profiles. Physically, this reference state represents the jump carried by the fastest wave moving toward the interface. Then,

$$\mathbf{q}^{n+1/2}_{i+1/2,L} - \tilde{\mathbf{q}}_{i,L} = \frac{1}{2}\frac{\Delta t}{\Delta x}\left[\max(\lambda_i^{(+)}, 0) - \mathbf{A}_i\right]\overline{\Delta \mathbf{q}}_i \tag{8.26}$$

and projecting this RHS (see Colella & Glaz Eq. 43; Miller & Colella Eq. 87), and isolating the interface state, we have

$$\mathbf{q}^{n+1/2}_{i+1/2,L} = \tilde{\mathbf{q}}_{i,L} + \frac{1}{2}\frac{\Delta t}{\Delta x}\sum_{\nu;\lambda^{(\nu)}\geq 0}\mathbf{l}_i^{(\nu)}\cdot\left[\max(\lambda_i^{(+)}, 0) - \mathbf{A}_i\right]\overline{\Delta \mathbf{q}}_i\,\mathbf{r}_i^{(\nu)} \tag{8.27}$$

$$= \tilde{\mathbf{q}}_{i,L} + \frac{1}{2}\frac{\Delta t}{\Delta x}\sum_{\nu;\lambda^{(\nu)}\geq 0}\left[\max(\lambda_i^{(+)}, 0) - \lambda_i^{(\nu)}\right](\mathbf{l}_i^{(\nu)}\cdot\overline{\Delta \mathbf{q}}_i)\,\mathbf{r}_i^{(\nu)} \tag{8.28}$$

This is equivalent to the expression in Saltzman [66] (p. 161, first column, second-to-last equation) and Colella [24] (p. 191, the group of expressions at the end)*. The corresponding state to the right of this interface is:

$$\mathbf{q}^{n+1/2}_{i+1/2,R} = \tilde{\mathbf{q}}_{i+1,R} + \frac{1}{2}\frac{\Delta t}{\Delta x}\sum_{\nu;\lambda^{(\nu)}\leq 0}\left[\min(\lambda_{i+1}^{(-)}, 0) - \lambda_{i+1}^{(\nu)}\right](\mathbf{l}_{i+1}^{(\nu)}\cdot\overline{\Delta \mathbf{q}}_{i+1})\,\mathbf{r}_{i+1}^{(\nu)} \tag{8.29}$$

where now the reference state captures the flow from the $i+1$ zone moving to the *left* to this interface (hence the appearance of $\lambda^{(-)}$, the leftmost eigenvalue):

$$\tilde{\mathbf{q}}_{i+1,R} = \mathbf{q}_{i+1} - \frac{1}{2}\left[1 + \frac{\Delta t}{\Delta x}\min(\lambda_{i+1}^{(-)}, 0)\right]\overline{\Delta \mathbf{q}}_{i+1} \tag{8.30}$$

Side note: the data in zone i will be used to construct the right state at $i - 1/2$ (the left interface) and the left state at $i + 1/2$ (the right interface) (see Figure 8.3). For this reason, codes usually compute the eigenvectors/eigenvalues for that zone and then compute $\mathbf{q}^{n+1/2}_{i-1/2,R}$ together with $\mathbf{q}^{n+1/2}_{i+1/2,L}$ in a loop over the zone centers.

8.2.3 Piecewise parabolic

The piecewise parabolic method uses a parabolic reconstruction in each cell. This is more accurate than the linear reconstruction. Figure 8.4 shows the reconstructed parabolic profiles within a few cells. Since the original PPM paper [28], there have been many discussions of the method, with many variations. Here we focus on the presentation by Miller & Colella [53], since that is the most straightforward. Note: even though a parabolic profile could be third-order accurate, the temporal discretization and prediction in this method is still only second-order.

*Since the combination $\mathbf{l}^{(\nu)}\cdot\Delta\mathbf{q}$ appears so frequently, some sources represent this as $\beta^{(\nu)}$ and then write the interface states as a linear combination of the β's. Some variation in the literation, depending on whether ρ or $\tau = 1/\rho$ is used as a primitive variable

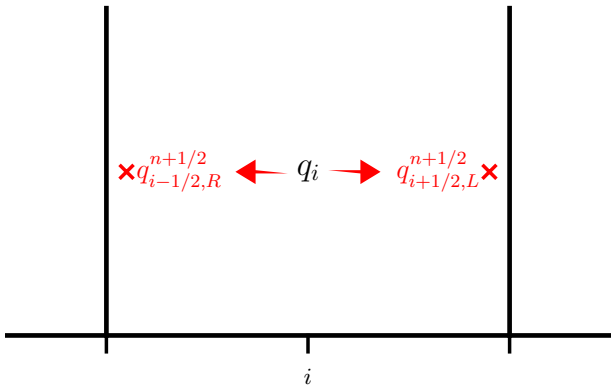

Figure 8.3: The two interface states that are constructed using $\mathbf{q}_i$ as the starting point.

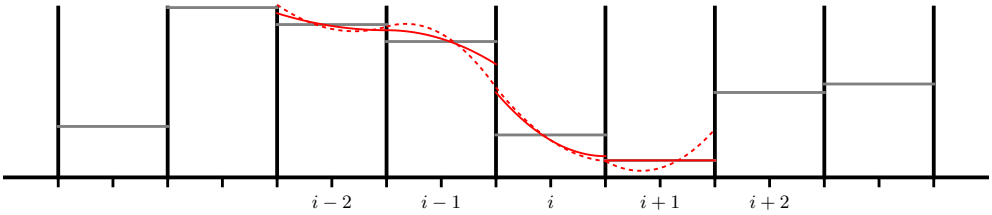

Figure 8.4: Piecewise parabolic reconstruction of the cell averages. The dotted line shows the unlimited parabolas—note how they touch at each interface, since the interface values come from the same interpolant initially. The solid line shows the limited parabolas.

Miller & Colella give an excellent description of how to take the results for piecewise linear reconstruction and generalize it to the case of PPM [28] (see Eqs. 88-90). Starting with Eq. 8.24, we can write this (after the characteristic projection) as

$$\mathbf{q}^{n+1/2}_{i+1/2,L} = \tilde{\mathbf{q}}_+ - \sum_{\nu;\lambda^{(\nu)}\geq 0} \mathbf{l}^{(\nu)}_i \cdot \left\{ \tilde{\mathbf{q}}_+ - \left[\mathbf{q}^n_i + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}\lambda^{(\nu)}_i\right)\overline{\Delta \mathbf{q}}_i \right] \right\} \mathbf{r}^{(\nu)}_i \tag{8.31}$$

Miller & Colella rewrite the portion inside the $[\ldots]$ recognizing that (similar to M&C Eq. 88, but for the $i+1/2, L$ interface):

$$\mathbf{q}^n_i + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}\lambda^{(\nu)}_i\right)\overline{\Delta \mathbf{q}}_i \approx \frac{1}{\lambda \Delta t}\int_{x_{i+1/2}-\lambda\Delta t}^{x_{i+1/2}} \mathbf{q}(x)dx \tag{8.32}$$

where $\mathbf{q}(x)$ is the reconstructed functional form of $\mathbf{q}$ in the zone.

Exercise 8.2

Show that this is exactly true for a linear reconstruction of $\mathbf{q}(x)$*, i.e.,* $\mathbf{q}(x) = \mathbf{q}_i + (\partial \mathbf{q}/\partial x)(x - x_i)$.

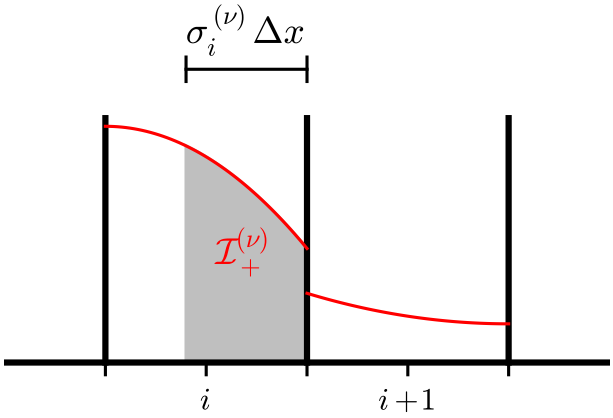

Figure 8.5: Integration under the parabolic profile. For each of the waves, $\sigma^{(\nu)}$ is the fraction of the cell that they cross in a timestep, and $\sigma^{(\nu)}\Delta x = \lambda^{(\nu)}\Delta t$ is the distance they can travel. Here we are integrating under the parabola to the right interface of cell i to define $\mathcal{I}_+^{(\nu)}$ (this is indicated by the shaded region). The $\mathcal{I}_+^{(\nu)}$ carried by this wave will be added to those carried by the other waves to form the left state at interface $i+1/2$.

The integral on the right represents the average of $\mathbf{q}$ that can reach the right interface of the cell i over timestep Δt, moving at the wavespeed λ. This suggests that we can replace the linear reconstruction of $\mathbf{q}$ with a parabolic one, and keep our expressions for the interface states.

In particular, we define

$$\mathcal{I}_+^{(\nu)}(\mathbf{q}_i) = \frac{1}{\sigma^{(\nu)}\Delta x}\int_{x_{i+1/2}-\sigma^{(\nu)}\Delta x}^{x_{i+1/2}} \mathbf{q}(x)dx \tag{8.33}$$

with $\sigma^{(\nu)} = |\lambda^{(\nu)}|\Delta t/\Delta x$ (see Almgren et al. Eq. 31) (see Figure 8.5). Then

$$\mathbf{q}_{i+1/2,L}^{n+1/2} = \tilde{\mathbf{q}}_+ - \sum_{\nu;\lambda^{(\nu)}\geq 0} \mathbf{l}_i^{(\nu)}\cdot\left(\tilde{\mathbf{q}}_+ - \mathcal{I}_+^{(\nu)}(\mathbf{q}_i)\right)\mathbf{r}_i^{(\nu)} \tag{8.34}$$

Miller & Colella choose the reference state as

$$\tilde{\mathbf{q}}_+ = \begin{cases} \mathcal{I}_+^{(+)}(\mathbf{q}_i) & \text{if } u+c>0 \\ \mathbf{q}_i & \text{otherwise} \end{cases} \tag{8.35}$$

where the superscript $(+)$ on $\mathcal{I}$ indicates that the fastest eigenvalue ($\lambda^{(+)} = u+c$) is used. This is similar in spirit to Eq. 8.25. Note: in the original PPM paper, if the wave is not approaching the interface, instead of using the cell-average, $\mathbf{q}_i$, they use the limit of the quadratic interpolant. In contrast to the above, the Castro paper [2] just uses $\mathbf{q}_i$ for the reference state regardless of whether the wave is moving toward or away from the interface. Note that if the system were linear, then the choice of reference state would not matter.

To finish the reconstruction, we need to know the parabolic form of $\mathbf{q}(x)$. Here, we do the reconstruction from the original PPM paper:

$$\mathbf{q}(x) = \mathbf{q}_- + \xi(x)\left(\Delta\mathbf{q} + \mathbf{q}_6(1-\xi(x))\right) \tag{8.36}$$

with $\Delta\mathbf{q} = \mathbf{q}_+ - \mathbf{q}_-$, and $\mathbf{q}_-$, $\mathbf{q}_+$ the values of the polynomial on the left and right edges, respectively, of the current cell, and

$$\mathbf{q}_6 \equiv 6\left[\mathbf{q}_i - \frac{1}{2}(\mathbf{q}_- + \mathbf{q}_+)\right] \tag{8.37}$$

and

$$\xi(x) = \frac{x - x_{i-1/2}}{\Delta x} \tag{8.38}$$

To complete the description, we need to determine the parameters of the parabola. The values of $\mathbf{q}_-$ and $\mathbf{q}_+$ are computed and limited as described in the original PPM paper. With this definition, we can do the integral $\mathcal{I}_+$:

$$\mathcal{I}_+^{(\nu)}(\mathbf{q}_i) = \mathbf{q}_{+,i} - \frac{\sigma_i^{(\nu)}}{2}\left[\Delta\mathbf{q}_i - \mathbf{q}_{6,i}\left(1 - \frac{2}{3}\sigma_i^{(\nu)}\right)\right] \tag{8.39}$$

Figure 8.5 illustrates the process of integrating under the parabolic profile.

Exercise 8.3

Show that $\mathbf{q}(x)$ is a conservative interpolant. That is

$$\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{q}(x)dx = \mathbf{q}_i \tag{8.40}$$

You can also see that the average over the left half of the zone is $\mathbf{q}_i - \frac{1}{4}\Delta\mathbf{q}$ and the average over the right half of the zone is $\mathbf{q}_i + \frac{1}{4}\Delta\mathbf{q}$. This means that there are equal areas between the integral and zone average on the left and right sides of the zone. This can be seen by looking at Figure 8.4.

Aside: Note that this characteristic projection of $\tilde{\mathbf{q}}_+ - \mathcal{I}_+^{(\nu)}$ is discussed in the original PPM paper in the paragraph following their Eq. 3.5. They do not keep things in this form however, and instead explicitly multiply out the $l \cdot [\ldots] r$ terms to arrive at their Eq. 3.6. For example, starting with Eq. 8.34, we can write the left velocity state as (leaving off the i subscripts on the vectors):

$$u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ - \sum_\nu \mathbf{l}^{(\nu)} \cdot (\tilde{\mathbf{q}}_+ - \mathcal{I}_+^{(\nu)}(\mathbf{q}))\ \underbrace{\mathbf{r}^{(\nu)}}_{\substack{\text{only the} \\ u \text{ 'slot'}}} \tag{8.41}$$

(where, as above, the $\sim$ indicates the reference state). Here the r eigenvector on the end is representative—we only pick the row corresponding to u in the $\mathbf{q}$ vector (in our case, the second row).

Putting in the eigenvectors and writing out the sum, we have:

$$\begin{aligned} u^{n+1/2}_{i+1/2,L} = \tilde{u}_+ &- \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}^{(-)}_+(\rho) \\ \tilde{u}_+ - \mathcal{I}^{(-)}_+(u) \\ \tilde{p}_+ - \mathcal{I}^{(-)}_+(p) \end{pmatrix} \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \\ &- \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}^{(\circ)}_+(\rho) \\ \tilde{u}_+ - \mathcal{I}^{(\circ)}_+(u) \\ \tilde{p}_+ - \mathcal{I}^{(\circ)}_+(p) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\ &- \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}^{(+)}_+(\rho) \\ \tilde{u}_+ - \mathcal{I}^{(+)}_+(u) \\ \tilde{p}_+ - \mathcal{I}^{(+)}_+(p) \end{pmatrix} \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix} \end{aligned} \tag{8.42}$$

Here again we show the entire right eigenvector for illustration, but only the element that comes into play is drawn in black. This shows that the second term is 0—the contact wave does not carry a jump in velocity. Multiplying out $\mathbf{l}^{(\nu)} \cdot (\tilde{\mathbf{q}}_+ - \mathcal{I}^{(\nu)}_+)$ we have:

$$\begin{aligned} u^{n+1/2}_{i+1/2,L} = \tilde{u}_+ &- \frac{1}{2}\left[(\tilde{u}_+ - \mathcal{I}^{(-)}_+(u)) - \frac{\tilde{p}_+ - \mathcal{I}^{(-)}_+(p)}{C}\right] \\ &- \frac{1}{2}\left[(\tilde{u}_+ - \mathcal{I}^{(+)}_+(u)) + \frac{\tilde{p}_+ - \mathcal{I}^{(+)}_+(p)}{C}\right] \end{aligned} \tag{8.43}$$

where C is the Lagrangian sound speed ($C = \sqrt{\gamma p \rho}$). Defining

$$\beta^{(+)} = -\frac{1}{2C}\left[(\tilde{u}_+ - \mathcal{I}^{(+)}_+(u)) + \frac{\tilde{p}_+ - \mathcal{I}^{(+)}_+(p)}{C}\right] \tag{8.44}$$

$$\beta^{(-)} = +\frac{1}{2C}\left[(\tilde{u}_+ - \mathcal{I}^{(-)}_+(u)) - \frac{\tilde{p}_+ - \mathcal{I}^{(-)}_+(p)}{C}\right] \tag{8.45}$$

we can write our left state as:

$$u^{n+1/2}_{i+1/2,L} = \tilde{u}_+ + C(\beta^{(+)} - \beta^{(-)}) \tag{8.46}$$

This is Eqs. 3.6 and 3.7 in the PPM paper. Note that in their construction appears to use the reference state in defining the Lagrangian sound speed (in their β expressions is written as $\tilde{C}$). This may follow from the comment before Eq. 3.6, "modified slightly for the present application". Similarly, the expressions for ρ_L and p_L can be written out.

Similar expressions can be derived for the right state at the left interface of the zone ($\mathbf{q}^{n+1/2}_{i-1/2,R}$). Here, the integral under the parabolic reconstruction is done over the region of each wave that can reach the left interface over our timestep:

$$\mathcal{I}^{(\nu)}_{-}(\mathbf{q}) = \frac{1}{\sigma^{(\nu)}\Delta x}\int_{x_{i-1/2}}^{x_{i-1/2}+\sigma^{(\nu)}\Delta x} \mathbf{q}(x)dx \tag{8.47}$$

The right state at $i-1/2$ using zone i data is:

$$\mathbf{q}^{n+1/2}_{i-1/2,R} = \tilde{\mathbf{q}}_{-} - \sum_{\nu;\lambda_\nu \le 0} \mathbf{l}^{(\nu)}_i \cdot \left(\tilde{\mathbf{q}}_{-} - \mathcal{I}^{(\nu)}_{-}(\mathbf{q}_i)\right)\mathbf{r}^{(\nu)}_i \tag{8.48}$$

where the reference state is now:

$$\tilde{\mathbf{q}}_{-} = \begin{cases} \mathcal{I}^{(-)}_{-}(\mathbf{q}_i) & \text{if } u-c<0 \\ \mathbf{q}_i & \text{otherwise} \end{cases} \tag{8.49}$$

where the $(-)$ superscript on $\mathcal{I}$ indicates that the most negative eigenvalue ($\lambda^- = u-c$) is used. The integral $\mathcal{I}^{(\nu)}_{-}(\mathbf{q})$ can be computed analytically by substituting in the parabolic interpolant, giving:

$$\mathcal{I}^{(\nu)}_{-}(\mathbf{q}_i) = \mathbf{q}_{-,i} + \frac{\sigma^{(\nu)}_i}{2}\left[\Delta\mathbf{q}_i + \mathbf{q}_{6,i}\left(1-\frac{2}{3}\sigma^{(\nu)}_i\right)\right] \tag{8.50}$$

This is equivalent to Eq. 31b in the Castro paper.

New PPM limiters

Recent work [27] has formulated improved limiters for PPM that do not clip the profiles at extrema. This only changes the limiting process in defining $\mathbf{q}_+$ and $\mathbf{q}_-$, and does not affect the subsequent parts of the algorithm.

8.2.4 Flattening and contact steepening

Shocks are self-steepening (this is how we detect them in the Riemann solver—we look for converging characteristics). This can cause trouble with the methods here, because the shocks may become too steep.

Flattening is a procedure to add additional dissipation at shocks, to ensure that they are smeared out over ~ 2 zones. The flattening procedure is a multi-dimensional operation that looks at the pressure and velocity profiles and returns a coefficient, $\chi \in [0,1]$ that multiplies the limited slopes. The convention most sources use is that $\chi = 1$ means no flattening (the slopes are unaltered), while $\chi = 0$ means complete flattening—the slopes are zeroed, dropping us to a first-order method. See for example in Saltzman [66]. Once the flattening coefficient is determined, the interface state is blended with the cell-centered value via:

$$\mathbf{q}^{n+1/2}_{i+1/2,\{L,R\}} \leftarrow (1-\chi)\mathbf{q}_i + \chi\mathbf{q}^{n+1/2}_{i+1/2,\{L,R\}} \tag{8.51}$$

Note that the flattening algorithm increases the stencil size of piecewise-linear and piecewise-parabolic reconstruction to 4 ghost cells on each side. This is because the flattening procedure itself looks at the pressure 2 zones away, and we need to construct the flattening coefficient in both the first ghost cell (since we need the interface values there) and the second ghost cell (since the flattening procedure looks at the coefficients in its immediate upwinded neighbor).

In contrast to shocks, contact waves do not steepen (they are associated with the middle characteristic wave, and the velocity does not change across that, meaning there cannot be any convergence). The original PPM paper advocates a contact steepening method to artificially steepen contact waves. While it shows good results in 1-d, it can be problematic in multi-dimensions.

Overall, the community seems split over whether this term should be used. Many people advocate that if you reach a situation where you think contact steepening may be necessary, it is more likely that the issue is that you do not have enough resolution.

8.2.5 Limiting on characteristic variables

Some authors (see for example, [76] Eqs. 37, 38) advocate limiting on the characteristic variables rather than the primitive variables. The characteristic slopes for the quantity carried by the wave ν can be found from the primitive variables as:

$$\Delta w^{(\nu)} = \mathbf{l}^{(\nu)} \cdot \Delta \mathbf{q} \tag{8.52}$$

any limiting would then be done to $\Delta w^{(\nu)}$ and the limited primitive variables would be recovered as:

$$\overline{\Delta \mathbf{q}} = \sum_{\nu} \overline{\Delta w}^{(\nu)} \mathbf{r}^{(\nu)} \tag{8.53}$$

(here we use an overline to indicate limiting).

This is attractive because it is more in the spirit of the linear advection equation and the formalism that was developed there. A potential downside is that when you limit on the characteristic variables and convert back to the primitive, the primitive variables may now fall outside of valid physical ranges (for example, negative density).

8.3 Riemann solvers

Once the interface states are created, the Riemann solver is called. This returns the solution at the interface:

$$\mathbf{q}_{i+1/2}^{n+1/2} = \mathcal{R}(\mathbf{q}_{i+1/2,L}^{n+1/2}, \mathbf{q}_{i+1/2,R}^{n+1/2}) \tag{8.54}$$

This is done for every interface, as illustrated in Figure 8.6.

As discussed in § 7.2.4, we need to determine which state is on our interface to compute the fluxes through the interface. The full solution of the Riemann problem

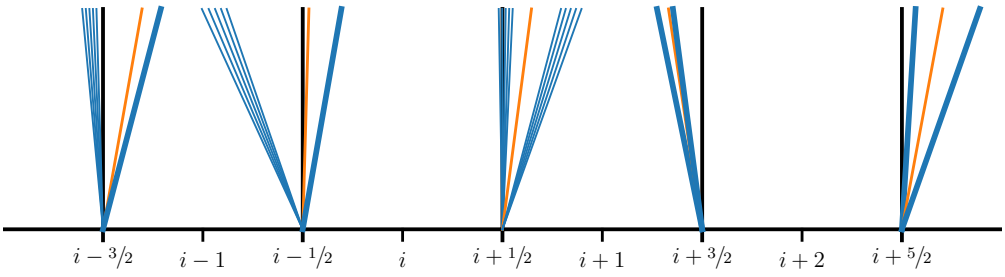

Figure 8.6: The Riemann wave structure resulting from the separate Riemann problems at each interface. For each interface, we find the state on the interface and use this to evaluate the flux through the interface.

can be quite expensive, so in practice, approximate Riemann solvers are used to speed the computation. Different Riemann solvers will have different approximations for finding the speeds of the left, center, and right wave, and evaluating the star state. In the 'star' region, only ρ jumps across the middle (contact) wave, the pressure and velocity are constant across that wave (see $\mathbf{r}^{(\circ)}$). We determine the state in the star region $(\rho_l^*, \rho_r^*, u^*, p^*)$ by using the jump conditions for the Euler equations or the Riemann invariants, as shown in § 7.2.3. Some approximate Riemann solvers assume that both waves are shocks or both are rarefactions, aimplying the form of the equation that must be solved to find $p_\star$.

Two-shock solvers, like the one in [25] are quite common in astrophyiscs. Figure 8.7 shows the Hugoniot curves for the Sod problem under the 2-shock assumption—they are quite close to those for the true solution. To further save on cost, approximate Riemann solvers for general equations of state often include additional thermodynamic information at the interfaces that overconstrains the system, but can remove the need to call an expensive equation of state routine in solving for the star state.

An additional approximation concerns rarefactions. Recall that a rarefaction involves diverging flow—it spreads out with time. Special consideration needs to be taken if the rarefaction wave spans the interface (a *transonic rarefaction*, § 7.2.4). In this case, most approximate Riemann solvers simply linearly interpolate between the left or right state and the appropriate star state instead of solving for the structure inside the rarefaction.

With this approximate state, the fluxes are computed as:

$$\mathbf{F}_{i+1/2}^{n+1/2} = \begin{pmatrix} \rho_{i+1/2}^{n+1/2} u_{i+1/2}^{n+1/2} \\ \rho_{i+1/2}^{n+1/2} (u_{i+1/2}^{n+1/2})^2 + p_{i+1/2}^{n+1/2} \\ u_{i+1/2}^{n+1/2} p_{i+1/2}^{n+1/2} / (\gamma - 1) + \frac{1}{2} \rho_{i+1/2}^{n+1/2} (u_{i+1/2}^{n+1/2})^3 + u_{i+1/2}^{n+1/2} p_{i+1/2}^{n+1/2} \end{pmatrix} \tag{8.55}$$

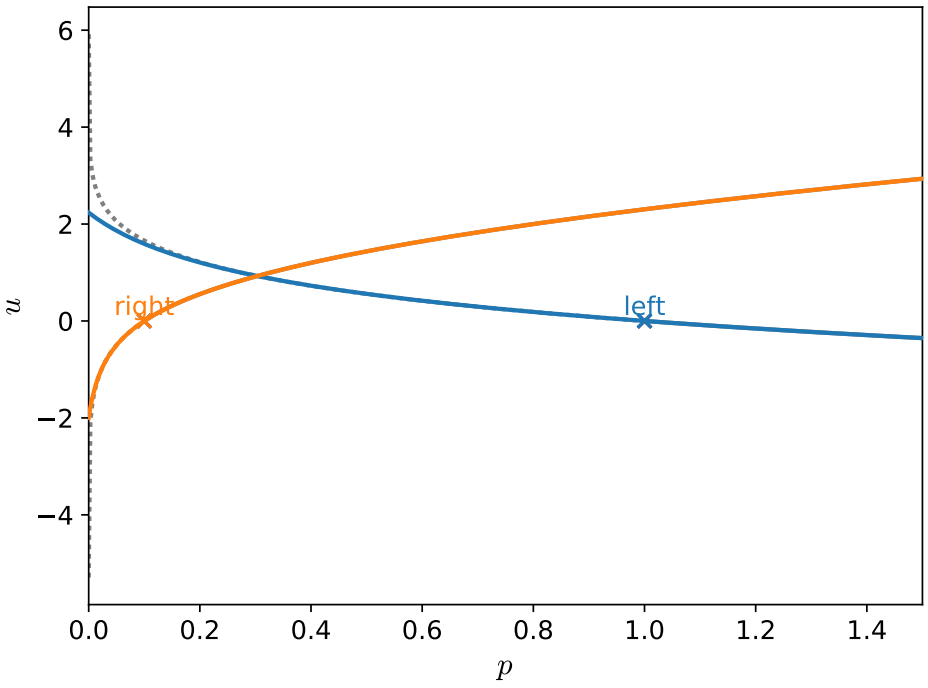

Figure 8.7: The approximate Hugoniot curves under the 2-shock approximation corresponding to the Sod problem. Also shown as the gray dotted line is the exact Hugoniot curves (compare to Figure 7.3). We see that the 2-shock approximation does a reasonable job near the intersection and only diverges significantly for small p (which is where the solution should really be a rarefaction).
hydro_examples: `riemann-2shock.py`

A different class of approximate Riemann solvers (the HLL family) approximate the fluxes directly instead of approximating the state first. These require estimates of the wave speeds, and care must be taken to ensure those estimates are valid for a general equation of state. These wave speeds are then used together with the Rankine-Hugoniot jump conditions to give the fluxes.

8.4 Conservative update

Once we have the fluxes, the conservative update is done as

$$\mathcal{U}_i^{n+1} = \mathcal{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2}\right) \tag{8.56}$$

The timestep, Δt is determined by the time it takes for the fastest wave to cross a single zone:

$$\Delta t < \min_d \left\{ \frac{\Delta x}{|\mathbf{U}\cdot\mathbf{e}_d| + c} \right\} \tag{8.57}$$

(This is the limit for the CTU unsplit scheme here, but see the discussion in § 5.4.3).

Often simulation codes will further restrict the timestep. Commonly used restrictions for pure hydrodynamics include a limit on the factor by which a timestep can grow from one step to the next (a typical value is 1.2), and an initial scaling, say of $1/10$, for the first timestep. Together, these will force the code to take a few initial steps before working up to the CFL limit.

8.4.1 Artificial viscosity

Colella and Woodward argue that behind slow-moving shocks these methods can have oscillations. The fix they propose is to use some artificial viscosity—this is additional dissipation that kicks in at shocks. (They argue that flattening alone is not enough).

We use a multidimensional analog of their artificial viscosity ([28], Eq. 4.5) which modifies the fluxes. By design, it only kicks in for converging flows, such that you would find around a shock.

8.5 Boundary conditions

Boundary conditions are implemented through ghost cells. The following are the most commonly used boundary conditions. For the expressions below, we use the subscript lo to denote the spatial index of the first valid zone in the domain (just inside the left boundary).

- *Outflow*: the idea here is that the flow should gracefully leave the domain. The simplest form is to simply give all variables a zero-gradient:

$$\begin{pmatrix} \rho_{\mathrm{lo}-1,j} \\ (\rho u)_{\mathrm{lo}-1,j} \\ (\rho v)_{\mathrm{lo}-1,j} \\ (\rho E)_{\mathrm{lo}-1,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo},j} \\ (\rho u)_{\mathrm{lo},j} \\ (\rho v)_{\mathrm{lo},j} \\ (\rho E)_{\mathrm{lo},j} \end{pmatrix} \tag{8.58}$$

  Note that these boundaries are not perfect. At the boundary, one (or more) of the waves from the Riemann problem can still enter the domain. Only for supersonic flow, do all waves point outward.

- *Reflect*: this is appropriate at a solid wall or symmetry plane. All variables are reflected across the boundary, with the normal velocity given the opposite sign. At the x-boundary, the first ghost cell is:

$$\begin{pmatrix} \rho_{\mathrm{lo}-1,j} \\ (\rho u)_{\mathrm{lo}-1,j} \\ (\rho v)_{\mathrm{lo}-1,j} \\ (\rho E)_{\mathrm{lo}-1,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo},j} \\ -(\rho u)_{\mathrm{lo},j} \\ (\rho v)_{\mathrm{lo},j} \\ (\rho E)_{\mathrm{lo},j} \end{pmatrix} \tag{8.59}$$

The next is:

$$\begin{pmatrix} \rho_{\mathrm{lo}-2,j} \\ (\rho u)_{\mathrm{lo}-2,j} \\ (\rho v)_{\mathrm{lo}-2,j} \\ (\rho E)_{\mathrm{lo}-2,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo}+1,j} \\ -(\rho u)_{\mathrm{lo}+1,j} \\ (\rho v)_{\mathrm{lo}+1,j} \\ (\rho E)_{\mathrm{lo}+1,j} \end{pmatrix} \tag{8.60}$$

and so on ...

- *Inflow*: inflow boundary conditions specify the state directly on the boundary. Technically, this state is on the boundary itself, not the cell-center. This can be accounted for by modifying the stencils used in the reconstruction near inflow boundaries.
- *Hydrostatic*: a hydrostatic boundary can be used at the base of an atmosphere to provide the pressure support necessary to hold up the atmosphere against gravity while still letting acoustic waves pass through. An example of this is described in [87].

8.6 Multidimensional problems

The multidimensional case is very similar to the multidimensional advection problem. Our system of equations is now:

$$\boldsymbol{\mathcal{U}}_t + [\mathbf{F}^{(x)}(\boldsymbol{\mathcal{U}})]_x + [\mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}})]_y = 0 \tag{8.61}$$

with

$$\boldsymbol{\mathcal{U}} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \qquad \mathbf{F}^{(x)}(\boldsymbol{\mathcal{U}}) = \begin{pmatrix} \rho u \\ \rho uu + p \\ \rho vu \\ \rho uE + up \end{pmatrix} \qquad \mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}) = \begin{pmatrix} \rho v \\ \rho vu \\ \rho vv + p \\ \rho vE + vp \end{pmatrix} \tag{8.62}$$

We note that there is no transformation that can convert the multidimensional system into characteristic variables, since we cannot simultaneously diagonalize the Jacobians corresponding to $\mathbf{F}^{(x)}$ and $\mathbf{F}^{(y)}$. Related to this is that when limiting, we limit one-dimensional slopes instead of doing a full multidimensional reconstruction and limiting (see [11] for a multidimensional limiting procedure for linear advection. For the Euler equations, since we cannot write the multidimensional system in a characteristic form, we cannot use this type of method).

For a directionally-unsplit discretization, we predict the cell-centered quantities to the edges by Taylor expanding the conservative state, $\boldsymbol{\mathcal{U}}$, in space and time. Now, when replacing the time derivative ($\partial \boldsymbol{\mathcal{U}}/\partial t$) with the divergence of the fluxes, we gain a transverse flux derivative term. For example, predicting to the upper x edge

of zone i, j, we have:

$$\mathcal{U}^{n+1/2}_{i+1/2,j,L} = \mathcal{U}^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \mathcal{U}}{\partial x} + \frac{\Delta t}{2}\frac{\partial \mathcal{U}}{\partial t} + \ldots \tag{8.63}$$

$$= \mathcal{U}^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \mathcal{U}}{\partial x} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(x)}}{\partial x} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(y)}}{\partial y} \tag{8.64}$$

$$= \mathcal{U}^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\mathbf{A}^{(x)}(\mathcal{U})\right]\Delta\mathcal{U} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(y)}}{\partial y} \tag{8.65}$$

where $\mathbf{A}^{(x)}(\mathcal{U}) \equiv \partial\mathbf{F}^{(x)}/\partial\mathcal{U}$. We decompose this into a *normal state* and a *transverse flux difference*. Adopting the notation from Colella (1990), we use $\hat{\mathcal{U}}$ to denote the normal state:

$$\hat{\mathcal{U}}^{n+1/2}_{i+1/2,j,L} \equiv \mathcal{U}^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\mathbf{A}^{(x)}(\mathcal{U})\right]\Delta\mathcal{U} \tag{8.66}$$

$$\mathcal{U}^{n+1/2}_{i+1/2,j,L} = \hat{\mathcal{U}}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(y)}}{\partial y} \tag{8.67}$$

The primitive variable form for this system is

$$\mathbf{q}_t + \mathbf{A}^{(x)}(\mathbf{q})\mathbf{q}_x + \mathbf{A}^{(y)}(\mathbf{q})\mathbf{q}_y = 0 \tag{8.68}$$

where

$$\mathbf{q} = \begin{pmatrix} \rho \\ u \\ v \\ p \end{pmatrix} \qquad \mathbf{A}^{(x)}(\mathbf{q}) = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & 1/\rho \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{pmatrix} \qquad \mathbf{A}^{(y)}(\mathbf{q}) = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & 1/\rho \\ 0 & 0 & \gamma p & v \end{pmatrix} \tag{8.69}$$

There are now 4 eigenvalues. For $\mathbf{A}^{(x)}(\mathbf{q})$, they are $u - c$, u, u, $u + c$. If we just look at the system for the x evolution, we see that the transverse velocity (in this case, v) just advects with velocity u, corresponding to the additional eigenvalue.

Exercise 8.4

Derive the form of $\mathbf{A}^{(x)}(\mathbf{q})$ and $\mathbf{A}^{(y)}(\mathbf{q})$ and find their left and right eigenvectors.

We note here that $\hat{\mathcal{U}}^{n+1/2}_{i+1/2,j,L}$ is essentially one-dimensional, since only the x-fluxes are involved (through $\mathbf{A}^{(x)}(\mathcal{U})$). This means that we can compute this term using the one-dimensional techniques developed in § 8.2. In particular, Colella (1990) suggest that we switch to primitive variables and compute this as:

$$\hat{\mathcal{U}}^{n+1/2}_{i+1/2,j,L} = \mathcal{U}(\hat{\mathbf{q}}^{n+1/2}_{i+1/2,j,L}) \tag{8.70}$$

Similarly, we consider the system projected along the y-direction to define the normal states on the y-edges, again using the one-dimensional reconstruction on the primitive variables from § 8.2:

$$\hat{\boldsymbol{\mathcal{U}}}^{n+1/2}_{i,j+1/2,L} = \boldsymbol{\mathcal{U}}(\hat{\mathbf{q}}^{n+1/2}_{i,j+1/2,L}) \tag{8.71}$$

To compute the full interface state (Eq. 8.67), we need to include the transverse term. Colella (1990) gives two different procedures for evaluating the transverse fluxes. The first is to simply use the cell-centered $\boldsymbol{\mathcal{U}}_{i,j}$ (Colella 1990, Eq. 2.13); the second is to use the reconstructed normal states (the $\hat{\boldsymbol{\mathcal{U}}}$'s) (Eq. 2.15). In both cases, we need to solve a *transverse Riemann problem* to find the true state on the transverse interface. This latter approach is what we prefer. In particular, for computing the full x-interface left state, $\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j,L}$, we need the transverse (y) states, which we define as

$$\boldsymbol{\mathcal{U}}^{T}_{i,j+1/2} = \mathcal{R}(\hat{\boldsymbol{\mathcal{U}}}^{n+1/2}_{i,j+1/2,L}, \hat{\boldsymbol{\mathcal{U}}}^{n+1/2}_{i,j+1/2,R}) \tag{8.72}$$

$$\boldsymbol{\mathcal{U}}^{T}_{i,j-1/2} = \mathcal{R}(\hat{\boldsymbol{\mathcal{U}}}^{n+1/2}_{i,j-1/2,L}, \hat{\boldsymbol{\mathcal{U}}}^{n+1/2}_{i,j-1/2,R}) \tag{8.73}$$

Taken together, the full interface state is now:

$$\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j,L} = \boldsymbol{\mathcal{U}}(\hat{\mathbf{q}}^{n+1/2}_{i+1/2,j,L}) - \frac{\Delta t}{2}\frac{\mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^{T}_{i,j+1/2}) - \mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^{T}_{i,j-1/2})}{\Delta y} \tag{8.74}$$

The right state at the $i+1/2$ interface can be similarly computed (starting with the data in zone $i+1, j$ and expanding to the left) as:

$$\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j,R} = \boldsymbol{\mathcal{U}}(\hat{\mathbf{q}}^{n+1/2}_{i+1/2,j,R}) - \frac{\Delta t}{2}\frac{\mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^{T}_{i+1,j+1/2}) - \mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^{T}_{i+1,j-1/2})}{\Delta y} \tag{8.75}$$

Note the indices on the transverse states—they are now to the right of the interface (since we are dealing with the right state).

We then find the x-interface state by solving the Riemann problem normal to our interface:

$$\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j} = \mathcal{R}(\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j,L}, \boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j,R}) \tag{8.76}$$

Therefore, construction of the interface states now requires two Riemann solves: a transverse and normal one. The fluxes are then evaluated as:

$$\mathbf{F}^{(x),n+1/2}_{i+1/2,j} = \mathbf{F}^{(x)}(\boldsymbol{\mathcal{U}}^{n+1/2}_{i+1/2,j}) \tag{8.77}$$

Note, for multi-dimensional problems, in the Riemann solver, the transverse velocities are simply selected based on the speed of the contact, giving either the left or right state.

The final conservative update is done as:

$$\boldsymbol{\mathcal{U}}^{n+1}_{i,j} = \boldsymbol{\mathcal{U}}^{n}_{i,j} + \frac{\Delta t}{\Delta x}\left(\mathbf{F}^{(x),n+1/2}_{i-1/2,j} - \mathbf{F}^{(x),n+1/2}_{i+1/2,j}\right) + \frac{\Delta t}{\Delta y}\left(\mathbf{F}^{(y),n+1/2}_{i,j-1/2} - \mathbf{F}^{(y),n+1/2}_{i,j+1/2}\right) \tag{8.78}$$

8.6.1 3-d unsplit

The extension of the unsplit methodology to 3-d is described by Saltzman [66]. The basic idea is the same as in 2-d, except now additional transverse Riemann solve are needed to fully couple in the corners.

8.7 Source terms

Adding source terms is straightforward. For a system described by

$$\mathcal{U}_t + [\mathbf{F}^{(x)}(\mathcal{U})]_x + [\mathbf{F}^{(y)}(\mathcal{U})]_y = \mathbf{H} \tag{8.79}$$

we predict to the edges in the same fashion as described above, but now when we replace $\partial\mathcal{U}/\partial t$ with the divergence of the fluxes, we also pick up the source term. This appears as:

$$\mathcal{U}^{n+1/2}_{i+1/2,j,L} = \mathcal{U}^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \mathcal{U}}{\partial x} + \frac{\Delta t}{2}\frac{\partial \mathcal{U}}{\partial t} + \ldots \tag{8.80}$$

$$= \mathcal{U}^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \mathcal{U}}{\partial x} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(x)}}{\partial x} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(y)}}{\partial y} + \frac{\Delta t}{2}\mathbf{H}_{i,j} \tag{8.81}$$

$$= \mathcal{U}^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}\mathbf{A}^{(x)}(\mathcal{U})\right]\Delta\mathcal{U} - \frac{\Delta t}{2}\frac{\partial \mathbf{F}^{(y)}}{\partial y} + \frac{\Delta t}{2}\mathbf{H}_{i,j} \tag{8.82}$$

We can compute things as above, but simply add the source term to the $\hat{\mathcal{U}}$'s and carry it through.

Note that the source here is cell-centered. This expansion is second-order accurate. This is the approach outlined in Miller & Colella [53]. Also notice the similarity of this source term to a second-order Euler method for integrating ODEs.

Alternately, we can include the source terms in the characteristic tracing of the interface states. This is the approach taken in, e.g., the original PPM paper. To make things more concrete, let's consider just gravity as the source. Our primitive variable equations in this case are:

$$\mathbf{q}_t + \mathbf{A}(\mathbf{q})\mathbf{q}_x = \mathbf{G} \tag{8.83}$$

where $\mathbf{G} = (0, g, 0)^T$—i.e. the gravitational source only affects u, not ρ or p[†].

First we construct a parabolic profile of g in each zone and integrate under that profile to determine the average g carried by each wave to the interface, we'll denote these as $\mathcal{I}^{(\nu)}_{\pm}(g)$. Then we include the gravitational source term in the characteristic projection itself. Our projections are now:

$$\sum_{\nu;\lambda^{(\nu)}\geq 0} \mathbf{l}^{(\nu)} \cdot (\tilde{\mathbf{q}} - \mathcal{I}^{(\nu)}_{+}(\mathbf{q}) - \tfrac{\Delta t}{2}\mathbf{G})\mathbf{r}^{(\nu)} \tag{8.84}$$

[†]Note that in the PPM paper, they put $\mathbf{G}$ on the lefthand side of the primitive variable equation, so our signs are opposite.

for the left state, and

$$\sum_{\nu;\lambda^{(\nu)}\le 0} \mathbf{l}^{(\nu)} \cdot (\tilde{\mathbf{q}} - \mathcal{I}_-^{(\nu)}(\mathbf{q}) - \tfrac{\Delta t}{2}\mathbf{G})\mathbf{r}^{(\nu)} \tag{8.85}$$

for the right state. Since $\mathbf{G}$ is only non-zero for velocity, only the velocity changes. Writing out the sum (and performing the vector products), we get:

$$\begin{aligned} u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ &- \frac{1}{2}\left[\left(\tilde{u}_+ - \mathcal{I}_+^{(-)}(u) - \frac{\Delta t}{2}\mathcal{I}_+^{(-)}(g)\right) - \frac{\tilde{p}_+ - \mathcal{I}_+^{(-)}(p)}{C}\right] \\ &- \frac{1}{2}\left[\left(\tilde{u}_+ - \mathcal{I}_+^{(+)}(u) - \frac{\Delta t}{2}\mathcal{I}_+^{(+)}(g)\right) + \frac{\tilde{p}_+ - \mathcal{I}_+^{(+)}(p)}{C}\right] \end{aligned} \tag{8.86}$$

where the only change from Eq. 8.43 are the $\mathcal{I}_+^{(-)}(g)$ and $\mathcal{I}_+^{(+)}(g)$ terms. ‡

Regardless of how the source term information is included in the interface states, we also need to include it in the conservative update. To second-order, we need it to be time-centered, which usually means averaging the time-level n and $n+1$ sources:

$$\begin{aligned} \boldsymbol{\mathcal{U}}_{i,j}^{n+1} = \boldsymbol{\mathcal{U}}_{i,j}^{n} &+ \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2,j}^{(x),n+1/2} - \mathbf{F}_{i+1/2,j}^{(x),n+1/2}\right) \\ &+ \frac{\Delta t}{\Delta y}\left(\mathbf{F}_{i,j-1/2}^{(y),n+1/2} - \mathbf{F}_{i,j+1/2}^{(y),n+1/2}\right) + \frac{\Delta t}{2}\left(\mathbf{H}(\boldsymbol{\mathcal{U}}_{i,j}^{n}) + \mathbf{H}(\boldsymbol{\mathcal{U}}_{i,j}^{n+1})\right) \end{aligned} \tag{8.87}$$

As written, this appears to be an implicit update (since $\boldsymbol{\mathcal{U}}^{n+1}$ depends on $\mathbf{H}^{n+1}$), but often, the form of the source terms allows you to update the equations in sequence explicitly.

Again, using a constant gravitational acceleration as an example, and looking in 1-d for simplicity, $\boldsymbol{\mathcal{U}} = (\rho, \rho u, \rho E)^\intercal$ and $\mathbf{H} = (0, \rho g, \rho u g)^\intercal$, so our update sequence is:

$$\rho_i^{n+1} = \rho_i^n + \frac{\Delta t}{\Delta x}\left[\rho_{i-1/2}^{n+1/2} u_{i-1/2}^{n+1/2} - \rho_{i+1/2}^{n+1/2} u_{i+1/2}^{n+1/2}\right] \tag{8.88}$$

$$\begin{aligned} (\rho u)_i^{n+1} = (\rho u)_i^n &+ \frac{\Delta t}{\Delta x}\left[\rho_{i-1/2}^{n+1/2}(u_{i-1/2}^{n+1/2})^2 - \rho_{i+1/2}^{n+1/2}(u_{i+1/2}^{n+1/2})^2\right] + \frac{\Delta t}{\Delta x}\left(p_{i-1/2}^{n+1/2} - p_{i+1/2}^{n+1/2}\right) \\ &+ \frac{\Delta t}{2}(\rho_i^n + \rho_i^{n+1})g \end{aligned} \tag{8.89}$$

$$\begin{aligned} (\rho E)_i^{n+1} = (\rho E)_i^n + \frac{\Delta t}{\Delta x}\Big[&\left(\rho_{i-1/2}^{n+1/2} E_{i-1/2}^{n+1/2} + p_{i-1/2}^{n+1/2}\right) u_{i-1/2}^{n+1/2} - \\ &\left(\rho_{i+1/2}^{n+1/2} E_{i+1/2}^{n+1/2} + p_{i+1/2}^{n+1/2}\right) u_{i+1/2}^{n+1/2}\Big] + \frac{\Delta t}{2}\left[(\rho u)^n + (\rho u)^{n+1}\right] g \end{aligned} \tag{8.90}$$

‡These differ from the expression in the PPM paper, where $\Delta t\mathbf{G}$, not $(\Delta t/2)\mathbf{G}$ is used in the projection, however this appears to be a typo. To see this, notice that if both waves are moving toward the interface, then the source term that is added to the interface state is $(\Delta t/4)(\mathcal{I}_+^{(-)}(g) + \mathcal{I}_+^{(+)}(g))$ for the left state, which reduces to $(\Delta t/2)g$ for constant g—this matches the result from the Taylor expansion above (Eq. 8.82).

These updates can be done one after another without any implicit coupling. Other sources, like the Coriolis force, will involve an implicit update, but even then, it is local to a single zone and can be solved analytically.

8.8 Simple geometries

So far we have been working only with Cartesian geometries, but it is easy to extend these methods to simple non-Cartesian geometies, like spherical and cylindrical. These geometies allow us to capture 3-d volume expansion effects in lower dimensions.

For a 1-d solver, a spherical geometry means that our coordinate is the radius in the sphere, and as we move outward from the origin, the volume of a zone (which is actually now a spherical shell) grows. There are two places where we need to take the geometry into account: the reconstruction of the interface states and the conservative update.

Our 1-d system in spherical coordinates appears as:

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{1}{r^2}\frac{\partial r^2 \mathbf{F}}{\partial r} = 0 \tag{8.91}$$

The geometry factors that appear are from the spherical form of the divergence. Expanding out the radial derivative, we have:

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{\partial \mathbf{F}}{\partial r} = -\frac{2\mathbf{F}}{r} \tag{8.92}$$

Likewise, the primitive variable equations now have source terms:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + \rho\frac{\partial u}{\partial x} = -\frac{2\rho u}{r} \tag{8.93}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0 \tag{8.94}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \Gamma_1 p\frac{\partial u}{\partial x} = -\frac{2\Gamma_1 p u}{r} \tag{8.95}$$

Exercise 8.5

Derive the above system of 1-d spherical primitive variable equations starting from the conservative equations. Be sure to include the geometry factors everywhere there is a divergence.

For the prediction of the interface states, we now include these geometric source terms as sources to the interface state, following the same ideas as in § 8.7.

The conservative update now needs to include the geometry factors. However, it is complicated by the fact that in the momentum equation, the pressure term is a

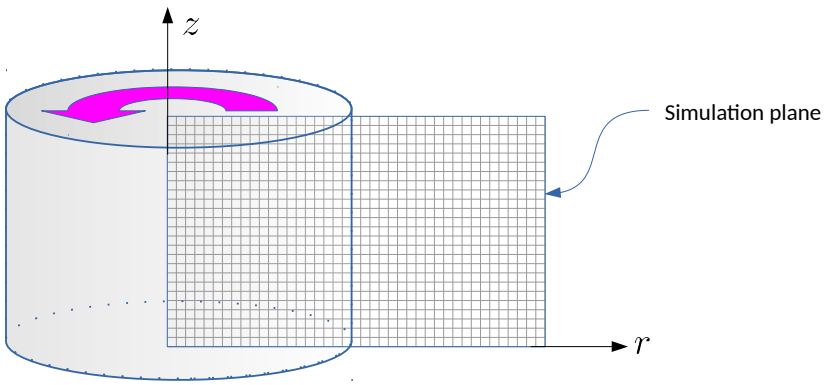

Figure 8.8: The axisymmetric computational domain. Here we imagine that the 2-d plane rotates through the cylindrical θ angle to create a volume.

gradient, not a divergence, and therefore has different geometic factors. The update of the system appears as:

$$\rho_i^{n+1} = \rho_i^n + \frac{\Delta t}{V_i}\left[A_{i-1/2}\rho_{i-1/2}^{n+1/2}u_{i-1/2}^{n+1/2} - A_{i+1/2}\rho_{i+1/2}^{n+1/2}u_{i+1/2}^{n+1/2}\right] \tag{8.96}$$

$$(\rho u)_i^{n+1} = (\rho u)_i^n + \frac{\Delta t}{V_i}\left[A_{i-1/2}\rho_{i-1/2}^{n+1/2}(u_{i-1/2}^{n+1/2})^2 - A_{i+1/2}\rho_{i+1/2}^{n+1/2}(u_{i+1/2}^{n+1/2})^2\right] + \frac{\Delta t}{\Delta r}\left(p_{i-1/2}^{n+1/2} - p_{i+1/2}^{n+1/2}\right) \tag{8.97}$$

$$(\rho E)_i^{n+1} = (\rho E)_i^n + \frac{\Delta t}{V_i}\left[A_{i-1/2}\left(\rho_{i-1/2}^{n+1/2}E_{i-1/2}^{n+1/2} + p_{i-1/2}^{n+1/2}\right)u_{i-1/2}^{n+1/2} - A_{i+1/2}\left(\rho_{i+1/2}^{n+1/2}E_{i+1/2}^{n+1/2} + p_{i+1/2}^{n+1/2}\right)u_{i+1/2}^{n+1/2}\right] \tag{8.98}$$

Here, the geometry factors are:

$$A_{i-1/2} = (r_{i-1/2})^2 \tag{8.99}$$

$$V_i = (r_i)^2\Delta r \tag{8.100}$$

It is also common to do 2-d axisymmetric models—here the r and z coordinates from a cylindrical geometry are modeled. Again, this appears Cartesian, except there is a volume factor implicit in the divergence that must be accounted for. Our system in axisymmetric coordinates[§] is:

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{1}{r}\frac{\partial r\mathbf{F}^{(r)}}{\partial r} + \frac{\partial \mathbf{F}^{(z)}}{\partial z} = 0 \tag{8.101}$$

Expanding out the r derivative, we can write this as:

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{\partial \mathbf{F}^{(r)}}{\partial r} + \frac{\partial \mathbf{F}^{(z)}}{\partial z} = -\frac{\mathbf{F}^{(r)}}{r} \tag{8.102}$$

[§]Some sources will call this cylindrical coordinates, but note that the grid is not a polar grid—it is still Cartesian

Again, the primitive variable version of this expanded form is used for the interface state prediction. The conservative update follows the same idea as the 1-d spherical version. The area and volume factors only differ from their Cartesian counterparts in the radial direction, and take the form:

$$A_{i-1/2,j} = r_{i-1/2}\Delta z \tag{8.103}$$

$$A_{i,j-1/2} = r_i \Delta z \tag{8.104}$$

$$V_{i,j} = r_i \Delta r \Delta z \tag{8.105}$$

These choices of geometric factors reproduce a discretized form of the cylindrical divergence:

$$\nabla \cdot \boldsymbol{\phi} = \frac{1}{r}\frac{\partial (r\phi^{(r)})}{\partial r} + \frac{\partial \phi^{(z)}}{\partial z} \tag{8.106}$$

Just as with the 1-d spherical case, the pressure term in the momentum equation needs to be treated separately from the flux, since it enters as a gradient and not a divergence.[¶]

8.9 Some Test problems

There are a large number of standard test problems that are used to test out our methods. Like we say with advection, it is best to have a problem with an analytic solution. Here we show just a few of the most popular test problems.

8.9.1 Shock tubes

Shock tubes are Riemann problems—consider the evolution of an initial discontinuity in the domain. The evolution with time will just be the solution to the Riemann problem that we described § 7.2. The initial conditions can be varied to produce any combination of shocks and rarefactions as the left and right waves. A popular initial condition is the Sod problem [74] which results is a right moving shock and contact and a left moving rarefaction.

Note: it is a good test of your code's ability to preserve symmetry to flip the initial conditions left/right and rerun. The results should be the same to machine precision, but atleast to roundoff error. A common reason for breaking symmetry is using inequalities in your code that are biases in a direction, e.g.,

```
if u > 0:
```

[¶]It is common to see the divergence term expressed as

$$\nabla \cdot \boldsymbol{\phi} = \frac{1}{r^\alpha}\frac{\partial (r^\alpha \phi^{(r)})}{\partial r} + \ldots \tag{8.107}$$

where $\alpha = 1$ for axisymmetric and $\alpha = 2$ for 1-d spherical. This allows for the geometry correction to be expressed more concisely, as the source term takes the form $-\alpha F/r$.

```
   # positive velocity test case
else:
   # negative or zero velocity test case
```

This has a left-right bias, since we don't handle the case where $u = 0$ separately. A better construction would test on $u < 0$ alone, and then have a final *else* clause to catch $u = 0$.

Since these tests start out with a discontinuity, they are not the best tests to use for convergence testing. Wherever there is an initiali discontinuity, the limiters will kick in and drop your method to first-order accurate.

For a general equation of state, you can still solve the Riemann problem exactly and define analogous test problems to those commonly used for a gamma-law gas. Some shock tube test problems for a stellar equation of state are shown in [88].

For the tests shown below, we use the hydro1d code described in Appendix C.

Sod problem

The initial conditions for the Sod problem [74] are:

$$\begin{aligned} \rho_l &= 1 & \rho_r &= 1/8 \\ u_l &= 0 & u_r &= 0 \\ p_l &= 1 & p_r &= 1/10 \end{aligned} \tag{8.108}$$

usually with $\gamma = 1.4$

These result in a left wave moving contact and rightward moving contact and shock. The shock is not particularly strong, but this problem is a nice demonstration of the types of hydrodynamic waves.

Double rarefaction

The double rarefaction problem starts with initially diverging flow that creates a vacuum state in the middle. The initial conditions (as given in [82] are

$$\begin{aligned} \rho_l &= 1 & \rho_r &= 1 \\ u_l &= -2.0 & u_r &= 2.0 \\ p_l &= 0.4 & p_r &= 0.4 \end{aligned} \tag{8.109}$$

8.9.2 Sedov blast wave

The Sedov (or Sedov-Taylor) blast wave [70] is a point explosion—energy, $\mathcal{E}_{\rm expl}$, is placed at a point in a uniform domain. A spherical shockwave propagates outward,

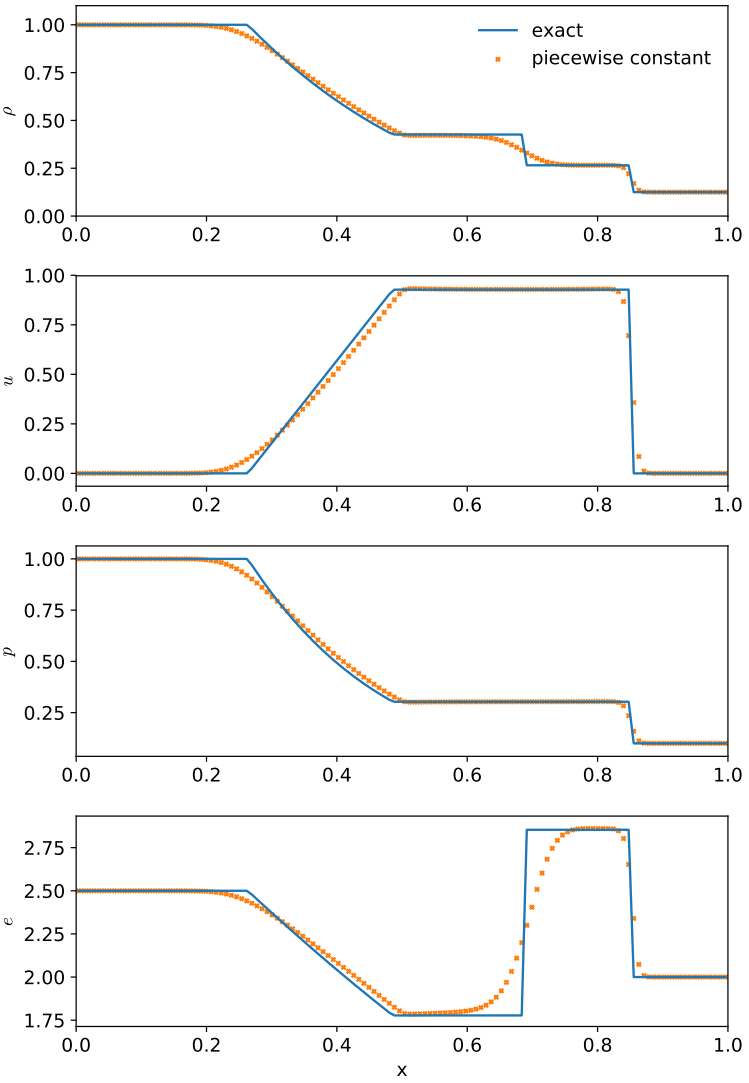

Figure 8.9: Piecewise constant reconstruction with the Sod problem, using 128 zones, $C = 0.8$, and the CGF Riemann solver. This was run with hydro1d using the sod problem setup, setting `godunov_type=0` and visualized with the `sod_compare.py` script there.

evacuating the region in the center. The Sedov problem also has an analytic solution[||], and this problem is a good way of testing out the geometric factors for 1-d spherical and 2-d axisymmetric geometries.

The major difficulty with initializing the Sedov problem is representing a point, where all the energy desposited, on the grid. If you just initialize a singe zone, then the size of the point changes as you change resolution. Additionally, in 2- or 3-d

[||] See [42] for a nice code to general the solutions

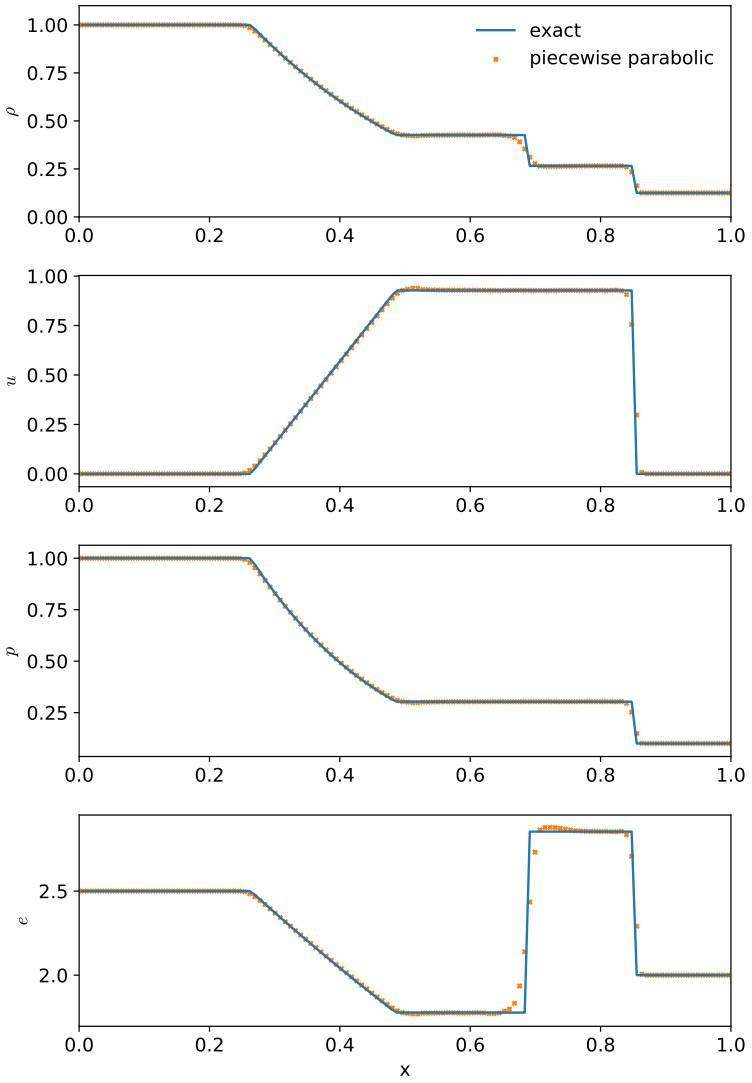

Figure 8.10: Piecewise parabolic reconstruction with the Sod problem, using 128 zones, $C = 0.8$, and the CGF Riemann solver. This was run with hydro1d using the sod problem setup, setting `godunov_type=2` and visualized with the `sod_compare.py` script there.

Cartesian coordinates, the point will be squared off. A standard way of initializing this (see, e.g., [58]) is to imagine the energy deposition inflating a region of radius $r_{\rm init}$ like a balloon, resulting in an energy density, $E = \mathcal{E}_{\rm expl}/V_{\rm init}$, where $V_{\rm init} = 4\pi r_{\rm init}^3/3$ for a spherical blast wave and $V_{\rm init} = \pi r_{\rm init}^2$ for a cylindrical blast wave. Then the pressure is

$$p = \frac{\mathcal{E}_{\rm expl}}{V_{\rm init}}(\gamma - 1) \tag{8.110}$$

Figure 8.13 shows the solution to the Sedov problem in 1-d in a spherical geometry.

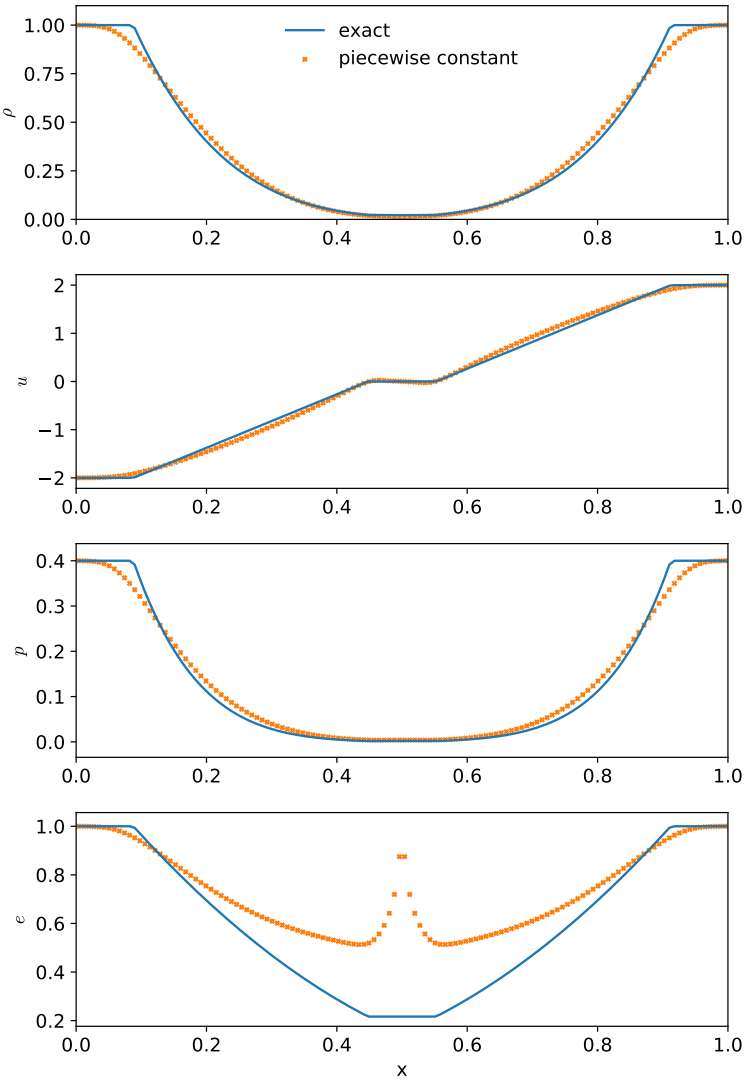

Figure 8.11: Piecewise constant reconstruction with the double rarefaction problem, using 128 zones, $C = 0.8$, and the CGF Riemann solver. This was run with hydro1d using the sod problem setup, setting `godunov_type=0` and visualized with the `sod_compare.py` script there.

This is compared to the spherical Sedov solution (solid line). There is good agreement in the position of the shock and density and velocity profiles. The pressure behind the shock is a little low—this is likely an artifact of the initialization process.

In 2-d, we run the problem in Cartesian coordinates—this produces a cylindrical blast wave. Figure 8.14 shows the solution in 2-d. The density and pressure look very symmetric. In the velocity magnitude plot, we see an imprint of the grid along coordinate axes, and likewise in the internal energy. The bump in internal energy near the origin arises because of the error in defining e from E and U. We can produce

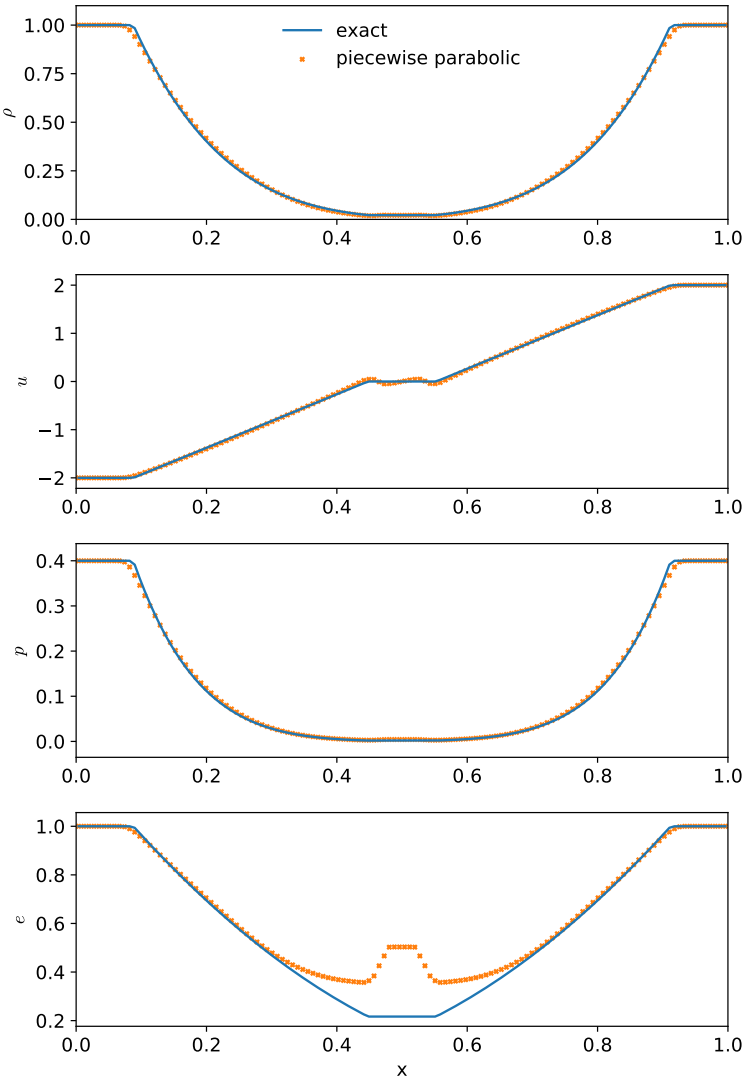

Figure 8.12: Piecewise parabolic reconstruction with the double rarefaction problem, using 128 zones, $C = 0.8$, and the CGF Riemann solver. This was run with hydro1d using the sod problem setup, setting `godunov_type=2` and visualized with the `sod_compare.py` script there.

an angle-averaged profile of this and compare to the analytic solution, shown in Figure 8.15.

8.9.3 Advection

We can run a simple advection test analogous to the tests we used in Ch. 4. However, because we are now doing hydrodynamics, we need to suppress the dynamics. This is accomplished by putting the profile we want to advect in the density field and then put it in pressure equilibrium by adjusting the internal energy. For example,

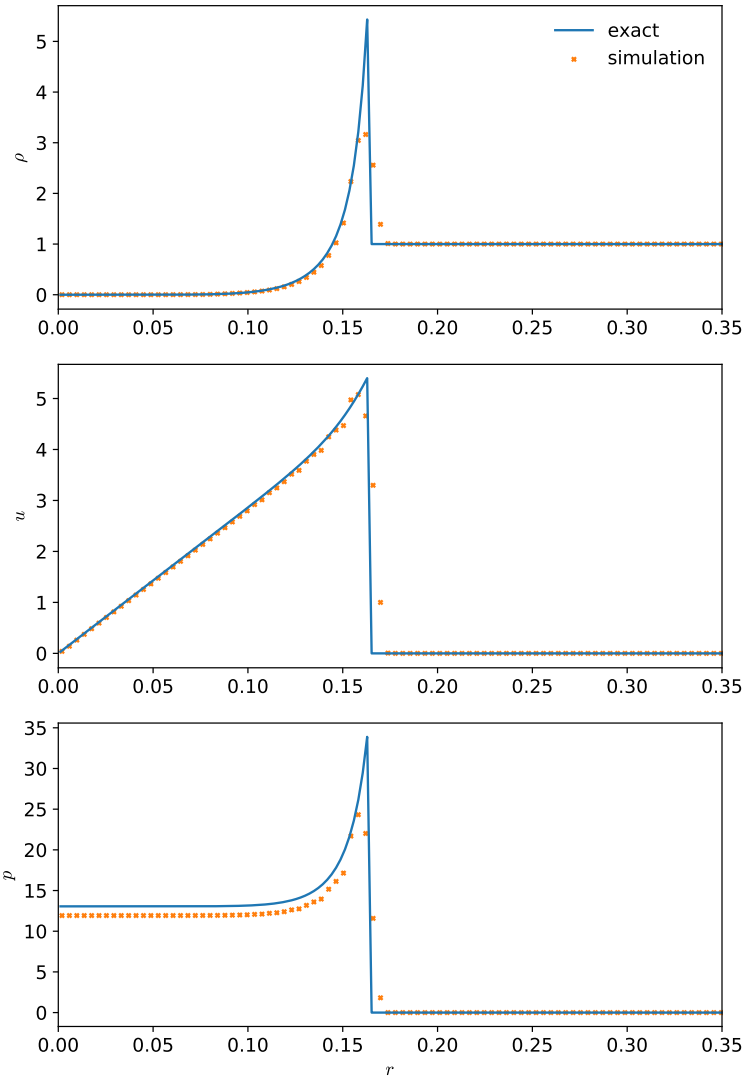

Figure 8.13: 1-d Sedov explosion with piecewise parabolic reconstruction, using 128 zones (on $r \in [0, 1]$), $\mathcal{C} = 0.8$, and the CGF Riemann solver. This was run with hydro1d using the `sedov` problem setup and visualized with the `sedov_compare.py` script there. The 1-d spherical geometry used makes this act as if it were a sphere.

consider a Gaussian profile. We initialize the density as:

$$\rho = (\rho_1 - \rho_0) e^{-(x-x_c)^2/\sigma^2} + \rho_0 \tag{8.111}$$

To advect the profile to the right, we choose a constant velocity,

$$u = u_0 \tag{8.112}$$

and to suppress dynamics, we make the pressure constant,

$$p = p_0 \tag{8.113}$$

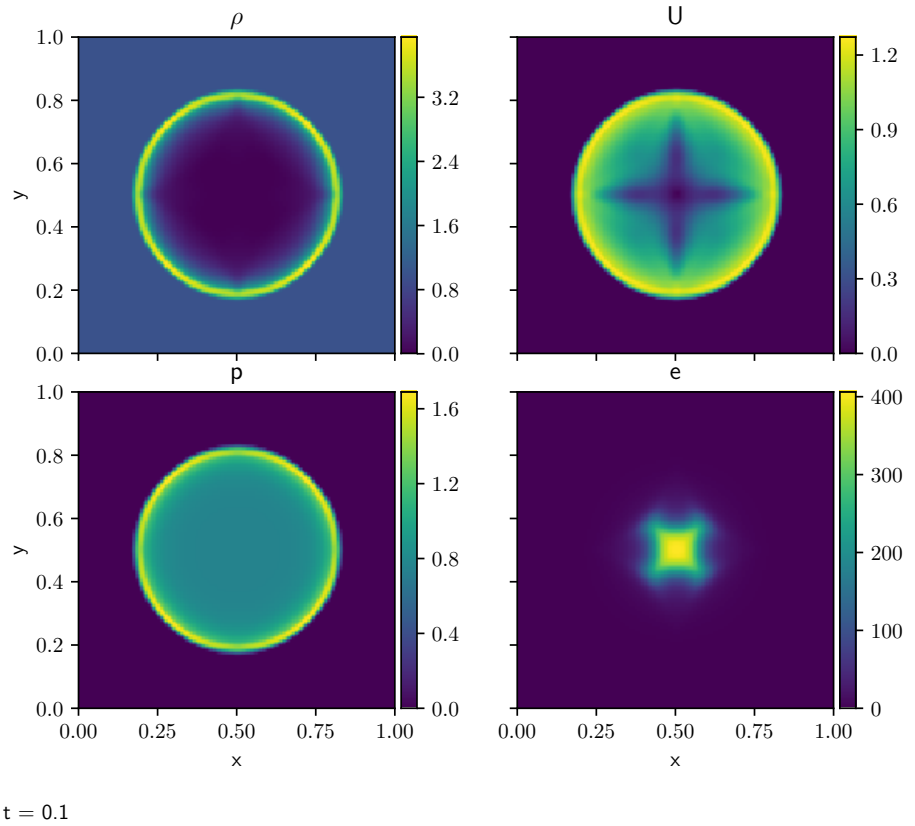

Figure 8.14: 2-d Sedov explosion with piecewise linear reconstruction, using 128^2 zones (on $r \in [0,1] \times [0,1]$), $C = 0.8$, and the HLLC Riemann solver. This was run with pyro as `./pyro.py compressible sedov inputs.sedov`. An initial perturbation size of $r_{\rm init} = 0.01$ was used.

Finally, we compute the total energy as

$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho u^2 \tag{8.114}$$

Figure 8.16 shows an example of a Gaussian profile advected for 10 periods. The initial conditions used $\rho_0 = 10^{-3}$, $\rho_1 = 1$, $p_0 = 10^{-6}$, $u_0 = 1$, and $\sigma = 0.1$ (these coincide with the choices used in [34]). The non-zero value for the ambient density, ρ_0, ensures that any quantities that are derived by dividing by density remain well-defined. The result is similar to what we saw when we considered pure advection—the shape is mostly preserved, but the peak of the Gaussian is clipped.

8.9.4 Slow moving shock

Slow moving (or stationary) shocks can be difficult to model, as oscillations can be setup behind the shock (this is discussed a little in [28, 46]). We can produce a slow

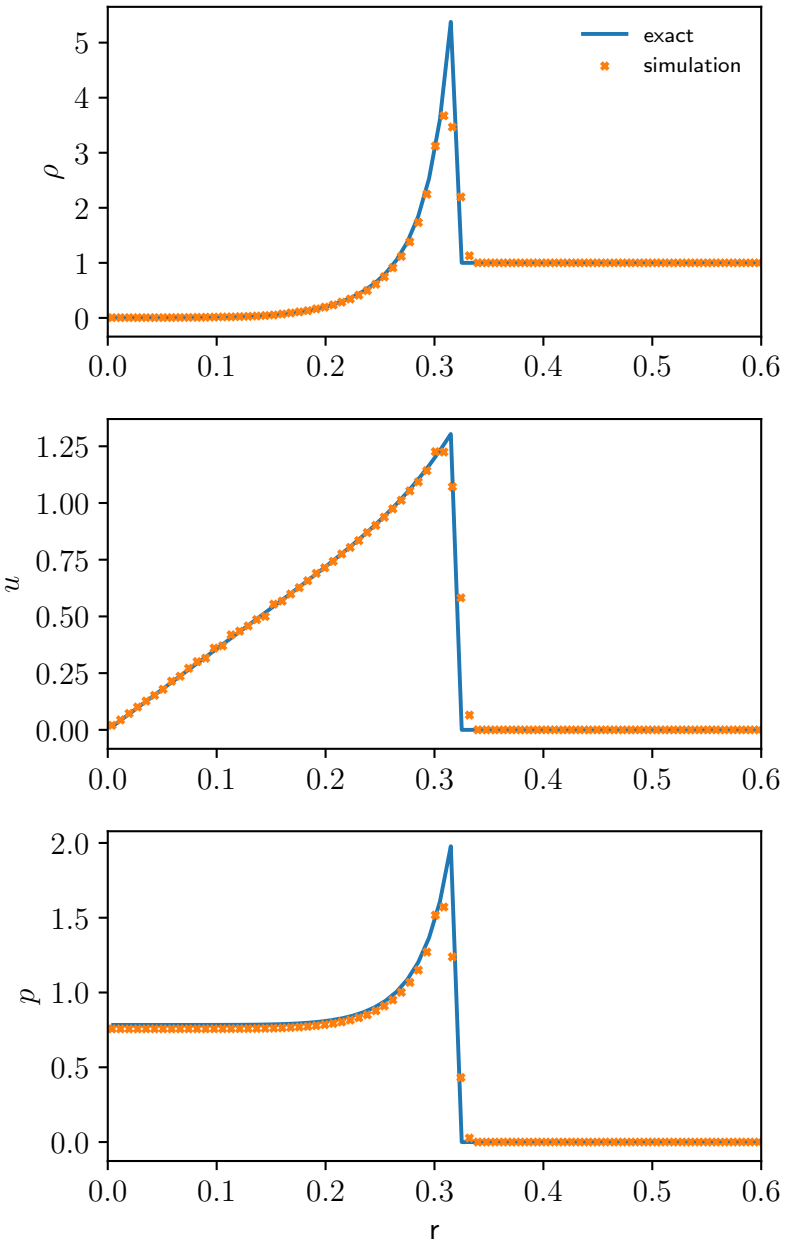

Figure 8.15: Angle-average profile for the 2-d Sedov explosion from Figure 8.14 shown with the analytic solution. This was constructed using the `sedov_compare.py` script in pyro.

moving shock as a shock tube, and we can use the jump conditions across a shock that were derived for the Riemann problem to find the conditions to setup a stationary (or slow-moving) shock.

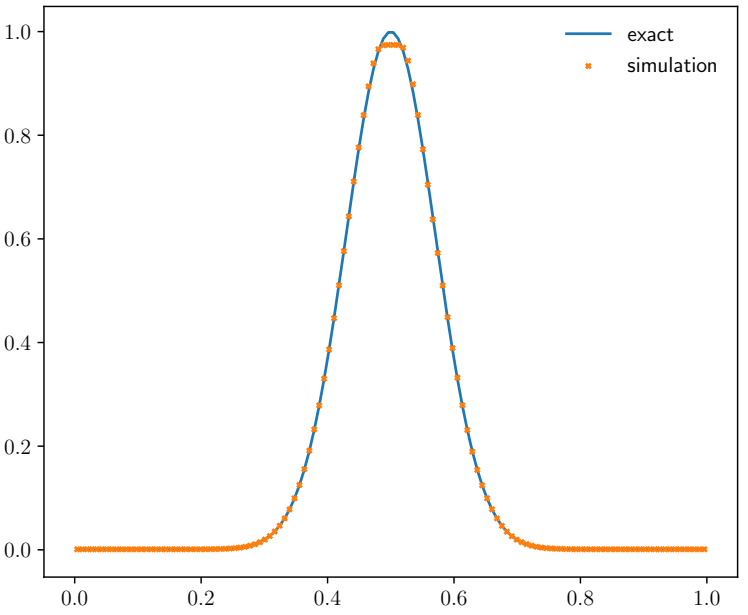

Figure 8.16: Piecewise parabolic reconstruction with an advection test, using 128 zones, $C = 0.8$, and the CGF Riemann solver. A Gaussian profile was advected for 10 periods. This was run with hydro1d using the `advect` problem setup and visualized with the `advect_compare.py` script there.

The speed of a right-moving shock was found (see Eq. 7.90) as:

$$S = u_r + c_r \left[\left(\frac{p_\star}{p_r} \right) \frac{\gamma + 1}{2\gamma} + \frac{\gamma - 1}{2\gamma} \right]^{1/2} \tag{8.115}$$

We want $S = 0$, which allows us to express the pre-shock velocity, u_r, as:

$$u_r = -c_r \left[\left(\frac{p_\star}{p_r} \right) \frac{\gamma + 1}{2\gamma} + \frac{\gamma - 1}{2\gamma} \right]^{1/2} \tag{8.116}$$

We have the freedom to choose the density and pressure ahead of the shock, ρ_r and p_r, which in turn gives us c_r. Next, we can pick the strength of the shock by choosing the jump in pressure, $p_\star / p_r$. Together, this allows us to compute u_r, and thus we know the entire pre-shock state. We can compute the post-shock state (which was the star state when we discussed the Riemann problem) using the jump conditions, Eqs. 7.86 and 7.89.**

For a pressure jump of 100 across a shock, the following conditions will generate a

**The script `slow_shock.py` will find the initial conditions to generate a stationary shock.

stationary right-facing shock (with $\gamma = 1.4$):

$$\begin{aligned} \rho_l &= 5.6698 & \rho_r &= 1 \\ u_l &= -1.9336 & u_r &= -10.9636 \\ p_l &= 100 & p_r &= 1 \end{aligned} \tag{8.117}$$

By adjusting the velocity of both the left and right state, we can produce a strong shock that moves slowly across the grid. For instance, a shock with $S = 0.4$ results from

$$\begin{aligned} \rho_l &= 5.6698 & \rho_r &= 1 \\ u_l &= -1.5336 & u_r &= -10.5636 \\ p_l &= 100 & p_r &= 1 \end{aligned} \tag{8.118}$$

8.9.5 Two-dimensional Riemann problems

Several different 2-d Riemann problems were introduced in [69], to explore the multi-dimensional interactive of the different hydrodynamic waves. These problems initialize the 4 quadrants of the domain with different states, and watch the ensuing evolution. There are some analytic estimates that can be compared to, but also these tests can provide a means of assessing the symmetry of a code in the presence of complex flows. We use the setup corresponding to *configuration 3* in that paper (this same setup is used in [45]).

8.10 Method of lines integration and higher order

Just like we explored with linear advection (§ 5.3), instead of doing the characteristic tracing, we could rely on the integrator to do the work for us.

Discretizing our system in space leads to the following system:

$$\frac{d\mathcal{U}_{i,j}}{dt} = -\frac{\mathbf{F}^{(x)}(\mathcal{U}_{i+1/2,j}) - \mathbf{F}^{(x)}(\mathcal{U}_{i-1/2,j})}{\Delta x} - \frac{\mathbf{F}^{(y)}(\mathcal{U}_{i,j+1/2}) - \mathbf{F}^{(y)}(\mathcal{U}_{i,j-1/2})}{\Delta y} \tag{8.119}$$

Note that there is no time superscript in the $\mathcal{U}$ used to evaluate the fluxes on the righthand side—we have not done any time discretization yet. Now we can use an ODE integrator to solve this system.

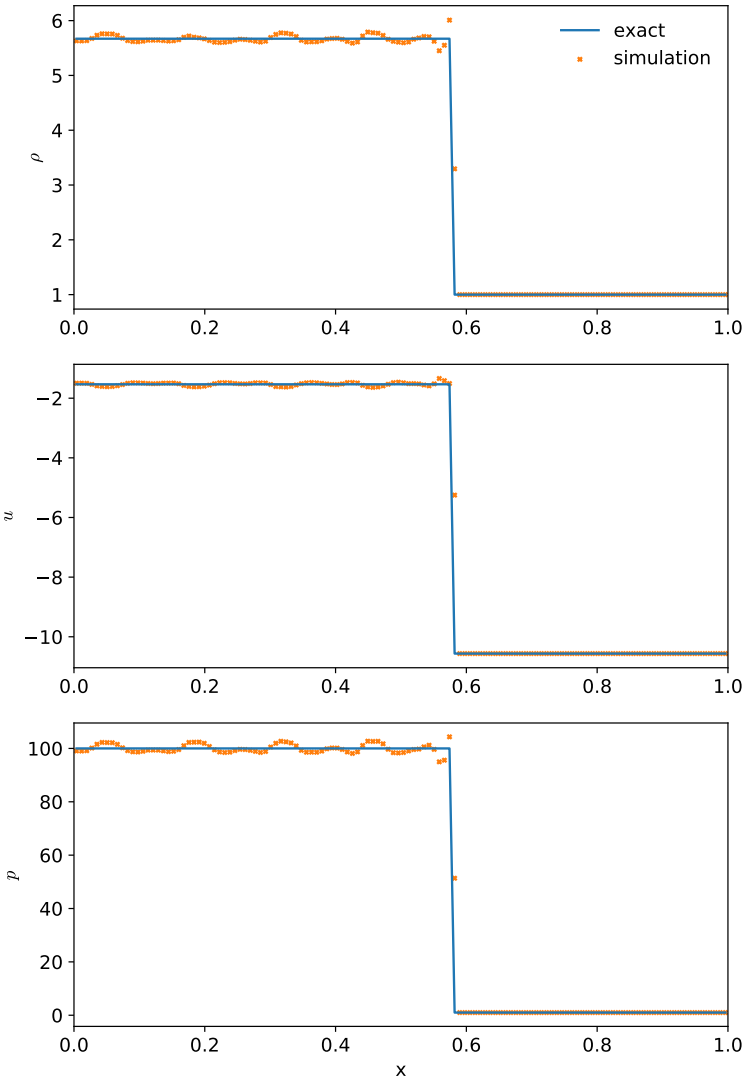

Figure 8.17: 1-d slow moving shock problem with piecewise parabolic reconstruction, using 128 zones (on $r \in [0, 1]$), $C = 0.8$, and the CGF Riemann solver. This was run with `hydro1d` using the `sod` problem setup and visualized with the `sod_compare.py` script there.

Consider second-order Runge-Kutta. We evaluate two slopes,

$$\mathbf{k}_1 = \Delta t \left[-\frac{\mathbf{F}^{(x)}(\boldsymbol{\mathcal{U}}^n_{i+1/2,j}) - \mathbf{F}^{(x)}(\boldsymbol{\mathcal{U}}^n_{i-1/2,j})}{\Delta x} - \frac{\mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^n_{i,j+1/2}) - \mathbf{F}^{(y)}(\boldsymbol{\mathcal{U}}^n_{i,j-1/2})}{\Delta y} \right] \tag{8.120}$$

$$\mathbf{k}_2 = \Delta t \left[-\frac{\mathbf{F}^{(x)}([\boldsymbol{\mathcal{U}}^n + \mathbf{k}_{1/2}]_{i+1/2,j}) - \mathbf{F}^{(x)}([\boldsymbol{\mathcal{U}}^n + \mathbf{k}_{1/2}]_{i-1/2,j})}{\Delta x} - \frac{\mathbf{F}^{(y)}([\boldsymbol{\mathcal{U}}^n + \mathbf{k}_{1/2}]_{i,j+1/2}) - \mathbf{F}^{(y)}([\boldsymbol{\mathcal{U}}^n + \mathbf{k}_{1/2}]_{i,j-1/2})}{\Delta y} \right] \tag{8.121}$$

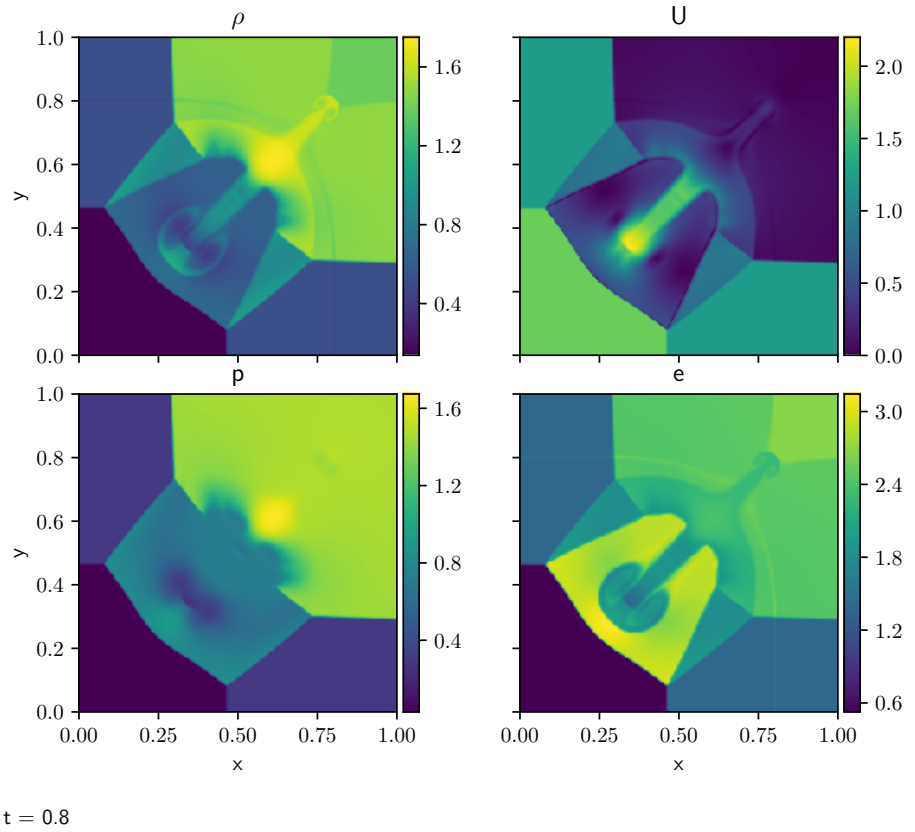

Figure 8.18: Two-dimensional Riemann problem from [69].

and then

$$\mathcal{U}_{i,j}^{n+1} = \mathcal{U}_{i,j}^{n} + \mathbf{k}_2 \tag{8.122}$$

In the construction of the interface states, $\mathcal{U}_{i+1/2,j}^{n}$ or $[\mathcal{U}^n + \mathbf{k}_{1/2}]_{i+1/2,j}$, there is no explicit transverse term, since that arose from Taylor expanding $\mathcal{U}_{i,j}^{n}$ in time through $\Delta t/2$. Instead, we simply construct these interface states using a one-dimensional reconstruction and solve a Riemann problem at each interface. The evaluation of the second slope, $\mathbf{k}_2$, implicitly includes the transverse information since we add $\mathbf{k}_{1/2}$ to $\mathcal{U}_{i,j}^{n}$ before doing the prediction to the interfaces. Also note, however, that in this construction of the interface states, there is no characteristic projection, since that arises from predicting the interface states forward in time. Again, these interface states are at a constant time, not predicted into the future.

Generally speaking we want the order of accuracy in time to match that in space. The fourth-order Runge-Kutta method is a popular method for integrating ODEs, so it makes sense to couple this with a fourth-order-in-space method. However, going higher-order than second-order is more challenging. The key issue is that we can no longer simply approximate the cell average as the cell-center value, i.e., $\langle \phi \rangle_i \neq \phi_i$. This comes into play, for instance, in translating between the conserved and primitive

variables. A fully fourth-order method is presented in [52]

Note that when using a Runge-Kutta method-of-lines integrator for the time-discretization of a multidimensional system, the timestep is actually more restrictive than the cases presented above that predicted the interface states to the half-time and performed characteristic tracing. Titarev & Toro [81] claim that you need $0 < C < 1/2$ for 2-d flows and $0 < C < 1/3$ for 3-d flows.

An additional complexity arises when doing multiphysics. Often we split the different physical processes up and treat them in turn. There are standard methods to do this with second-order accuracy in time, but higher-order is more tricky.

8.11 Thermodynamic issues

8.11.1 Defining temperature

Although not needed for the pure Euler equations, it is sometimes desirable to define the temperature for source terms (like reactions) or complex equations of state. The temperature can typically be found from the equation of state given the internal energy:

$$e = E - \frac{1}{2}u^2 \tag{8.123}$$

$$T = T(e, \rho) \tag{8.124}$$

Trouble can arise when you are in a region of flow where the kinetic energy dominates (high Mach number flow). In this case, the e defined via subtraction can become negative due to truncation error in the evolution of u compared to E. In this instance, one must either impose a floor value for e or find an alternate method of deriving it.

In [19], an alternate formulation of the Euler equations is proposed. Both the total energy equation *and* the internal energy equation are evolved in each zone. When the flow is dominated by kinetic energy, then the internal energy from the internal energy evolution equation is used. The cost of this is conservation—the internal energy is not a conserved quantity, and switching to it introduces conservation of energy errors.

8.11.2 General equation of state

The above methods were formulated with a constant gamma equation of state. A general equation of state (such as degenerate electrons) requires a more complex method. Most methods are designed to reduce the need to call a complex equation of state frequently, and work by augmenting the vector of primitive variables with additional thermodynamic information. There are two parts of the adaption to a general equation of state: the interface states and the Riemann problem.

Carrying γ_e

The classic prescription for extending this methodology is presented by Colella and Glaz [25]. They construct a thermodynamic index,

$$\gamma_e = \frac{p}{\rho e} + 1 \tag{8.125}$$

and derive an evolution equation for γ_e (C&G, Eq. 26). We can derive a similar expression as

$$\begin{aligned} \frac{D\gamma_e}{Dt} &= \frac{D}{Dt}\left(\frac{p}{\rho e} + 1\right) = -\frac{p}{(\rho e)^2}\frac{D(\rho e)}{Dt} + \frac{1}{\rho e}\frac{Dp}{Dt} \\ &= (\gamma_e - 1)(\gamma_e - \Gamma_1)\nabla \cdot \mathbf{U} \end{aligned} \tag{8.126}$$

where we used Eqs. 7.22 and 7.99, and the definition of the sound speed.

This evolution equation is used to predict γ_e to interfaces, and these interface values of γ_e are used in the Riemann solver presented there to find the fluxes through the interface. A different adiabatic index (they call Γ, we call Γ_1) appears in the definition of the sound speed. They argue that this can be brought to interfaces in a piecewise constant fashion while still making the overall method second order, since Γ_1 does not explicitly appear in the fluxes (see the discussion at the top of page 277).

We can derive the characteristic structure of this system for use in the tracing in the construction of interface states[††]. If we write our system as

$$\mathbf{q} = \begin{pmatrix} \tau \\ u \\ p \\ \gamma_e \end{pmatrix} \tag{8.127}$$

(we use $\tau = 1/\rho$ here for consistency with CG), we have

$$\mathbf{A} = \begin{pmatrix} u & -\tau & 0 & 0 \\ 0 & u & \tau & 0 \\ 0 & c^2/\tau & u & 0 \\ 0 & -\alpha & 0 & u \end{pmatrix} \tag{8.128}$$

where we define $\alpha = (\gamma_e - 1)(\gamma_e - \Gamma_1)$ for convenience. The right eigenvectors are:

$$\mathbf{r}^{(-)} = \begin{pmatrix} 1 \\ c/\tau \\ -c^2/\tau^2 \\ \alpha/\tau \end{pmatrix} \qquad \mathbf{r}^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{r}^{(\circ,\gamma_e)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathbf{r}^{(+)} = \begin{pmatrix} 1 \\ -c/\tau \\ -c^2/\tau^2 \\ \alpha/\tau \end{pmatrix} \tag{8.129}$$

[††]A Jupyter notebook using SymPy that derives these eigenvectors is available here: hydro_examples: `euler-generaleos.ipynb`

and corresponding left eigenvectors are:

$$\mathbf{l}^{(-)} = \begin{pmatrix} 0 & \frac{\tau}{2c} & -\frac{\tau^2}{2c^2} & 0 \end{pmatrix} \tag{8.130}$$

$$\mathbf{l}^{(\circ)} = \begin{pmatrix} 0 & \frac{\tau}{2c} & -\frac{\tau^2}{2c^2} & 0 \end{pmatrix} \tag{8.131}$$

$$\mathbf{l}^{(\circ,\gamma_e)} = \begin{pmatrix} 0 & 0 & \frac{\alpha\tau}{c^2} & 1 \end{pmatrix} \tag{8.132}$$

$$\mathbf{l}^{(+)} = \begin{pmatrix} 0 & -\frac{\tau}{2c} & -\frac{\tau^2}{2c^2} & 0 \end{pmatrix} \tag{8.133}$$

Carrying (ρe)

Alternately, the Castro paper [2] relies on an idea from an unpublished manuscript by Colella, Glaz, and Ferguson that predicts ρe to edges in addition to ρ, u, and p. Since ρe comes from a conservation-like equation (Eq. 7.99), predicting it to the interface in the unsplit formulation is straightforward. This over-specifies the thermodynamics, but eliminates the need for γ_e.

With the addition of ρe, our system becomes:

$$\mathbf{q} = \begin{pmatrix} \rho \\ u \\ p \\ \rho e \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 1/\rho & 0 \\ 0 & \rho c^2 & u & 0 \\ 0 & \rho h & 0 & u \end{pmatrix} \tag{8.134}$$

where $h = e + p/\rho$ is the specific enthalpy. The eigenvalues of this system are:

$$\lambda^{(-)} = u - c \qquad \lambda^{(\circ)} = u \qquad \lambda^{(\circ,\rho e)} = u \qquad \lambda^{(+)} = u + c \tag{8.135}$$

and the eigenvectors are:

$$\mathbf{r}^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \\ h \end{pmatrix} \qquad \mathbf{r}^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{r}^{(\circ,\rho e)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathbf{r}^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \\ h \end{pmatrix} \tag{8.136}$$

and

$$\begin{aligned} \mathbf{l}^{(-)} &= \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} & 0 \end{pmatrix} \\ \mathbf{l}^{(\circ)} &= \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} & 0 \end{pmatrix} \\ \mathbf{l}^{(\circ,\rho e)} &= \begin{pmatrix} 0 & 0 & -\frac{h}{c^2} & 1 \end{pmatrix} \\ \mathbf{l}^{(+)} &= \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} & 0 \end{pmatrix} \end{aligned} \tag{8.137}$$

Remember that the state variables in the $\mathbf{q}$ vector are mixed into the other states by $\mathbf{l} \cdot \mathbf{q}$. Since all $\mathbf{l}^{(\nu)}$'s have 0 in the ρe 'slot' (the last position) except for $\mathbf{l}^{(\circ,\rho e)}$, and the corresponding $\mathbf{r}^{(\circ,\rho e)}$ is only non-zero in the ρe slot, this means that ρe is not mixed into the other state variables. This is as expected, since ρe is not needed in the system.

Also recall that the jump carried by the wave ν is proportional to $\mathbf{r}^{(\nu)}$—since $\mathbf{r}^{(-)}$, $\mathbf{r}^{(\circ,\rho e)}$, and $\mathbf{r}^{(+)}$ have non-zero ρe elements, this means that ρe jumps across these three waves.

Working through the sum for the (ρe) state, and using a $\sim$ to denote the reference states, we arrive at:

$$\begin{aligned}(\rho e)^{n+1/2}_{i+1/2,L} = \widetilde{(\rho e)} &- \frac{1}{2}\left[-\frac{\rho}{c}\left(\tilde{u} - \mathcal{I}^{(1)}_{+}(u)\right) + \frac{1}{c^2}\left(\tilde{p} - \mathcal{I}^{(1)}_{+}(p)\right)\right] h \\ &- \left[-\frac{h}{c^2}\left(\tilde{p} - \mathcal{I}^{(3)}_{+}(p)\right) + \left(\widetilde{(\rho e)} - \mathcal{I}^{(3)}_{+}(\rho e)\right)\right] \\ &- \frac{1}{2}\left[\frac{\rho}{c}\left(\tilde{u} - \mathcal{I}^{(4)}_{+}(u)\right) + \frac{1}{c^2}\left(\tilde{p} - \mathcal{I}^{(4)}_{+}(p)\right)\right] h \end{aligned} \tag{8.138}$$

This is the expression that is found in the Castro code. If you are dealing with a simple geometry, then the divergences have metric terms. In 1-d, we then have:

$$\frac{\partial(\rho e)}{\partial t} + \frac{1}{r^\alpha}\frac{\partial(r^\alpha \rho u e)}{\partial r} + p\frac{1}{r^\alpha}\frac{\partial(r^\alpha u)}{\partial r} \tag{8.139}$$

Expanding out the derivatives gives use the equation in Cartesian form with a geometric source term:

$$\frac{\partial(\rho e)}{\partial t} + \frac{\partial(\rho u e)}{\partial r} + p\frac{\partial u}{\partial r} = -\frac{\alpha \rho h u}{r} \tag{8.140}$$

where h is the specific enthalpy. This can be accommodated using the procedure described in § 8.8.

All of these methods are designed to avoid EOS calls where possible, since general equations of state can be expensive.

Extending these to an unsplit formulation requires carrying an additional auxiliary variable from the primitive state back to the conserved state and adding the transverse gradients to its interface state. Castro deals with a conserved state of $\boldsymbol{\mathcal{U}} = (\rho, \rho\mathbf{U}, \rho E, p)$, and explicitly adds the transverse terms found in the multidimensional form of Eq. 7.22 to the normal states of p.

8.12 WENO methods for the Euler equations

When dealing with nonlinear systems the flux split method introduced in section 6.4 generalizes directly. The simplest, and most diffusive flux splitting uses the maximum characteristic speed α. For a system, such as the Euler equations, the simplest method is to compute α by maximizing over all characteristics. The directional fluxes $\mathbf{F}^{(\pm)}(u)$ can then be reconstructed using a high order WENO scheme component by component.

However, this approach can lead to significant oscillations, even in moderate tests. Rather than doing the reconstruction of the components of the directional fluxes, a

safer approach is to work with characteristic variables. The discussion in chapter 7 suggests that, by computing the left and right eigenvectors of the appropriate Jacobian matrix $\mathbf{A} = \partial\mathbf{F}/\partial\boldsymbol{\mathcal{U}}$, we can project the directional fluxes into the directional *characteristic* fluxes (using the left eigenvectors), reconstruct these characteristic fluxes using a high order method, and then compute the reconstructed directional fluxes required (using the right eigenvectors). As the characteristic fluxes should contain information about only a single wave at a time, this *should* give similar results to the scalar case and reduce problems from close waves from different families.

The Jacobian matrix needs to be computed separately for each point at which the flux needs computing. The best choice of state from which the Jacobian is computed is not clear: often the arithmetic average of neighbouring states is used, but more complex choices can give better results.

A direct comparison of component-wise and characteristic-wise flux-vector split WENO methods, applied to the Sod test, is given in figure 8.19 for $r = 3$ and in figure 8.20 for $r = 5$. The comparison to the PPM method in figure 8.10 is particularly instructive. The PPM method is specifically designed for hydrodynamic problems, and is much better at cleanly capturing the discontinuities, especially the contact. The WENO approach will have a higher order of accuracy, which will be clearest on tests with more smooth variation. The oscillations introduced by the WENO methods are more pronounced as the order is increased, but are reduced by the use of characteristic-wise reconstruction. This is most clearly seen in the results for the internal energy in the $r = 5$ case shown in figure 8.20.

Another illustration of the advantages of a numerical method, such as PPM, that is specifically designed for the Euler equations, is shown in the double rarefaction test in figure 8.21. The advantages of the higher order methods are shown by how well the WENO schemes capture the edges of the rarefaction waves, even with so few points and the diffusive Lax-Friedrichs flux splitting. However, the artificial heating effect seen at the trivial contact at the center of the domain is about as bad as that from piecewise constant reconstruction, and nowhere near as good as PPM. Whilst increasing the reconstruction order has a small impact, a less diffusive flux splitting would be needed to approach PPM's performance.

8.12.1 Extensions

The main advantage of the WENO method is that it retains its high order when extending to multiple dimensions using dimensional splitting. From the finite-difference form there are no transverse Riemann problems to solve. From using the Method of Lines there is no issue about ordering the dimensional sweeps: the updates in each direction are computed separately but applied together in the time integrator. This formally retains the high-order accuracy but can lose significant absolute accuracy – compare the advection of a top-hat function with lower order methods using transverse Riemann problem solutions.

The global Lax-Friedrichs flux splitting above can be excessively diffusive. A local Lax-Friedrichs flux splitting, where α is computed at each point by maximising over the characteristic speed within the stencil at that point, is less diffusive but slightly more expensive. Roe-style flux splittings are possible but not always stable.

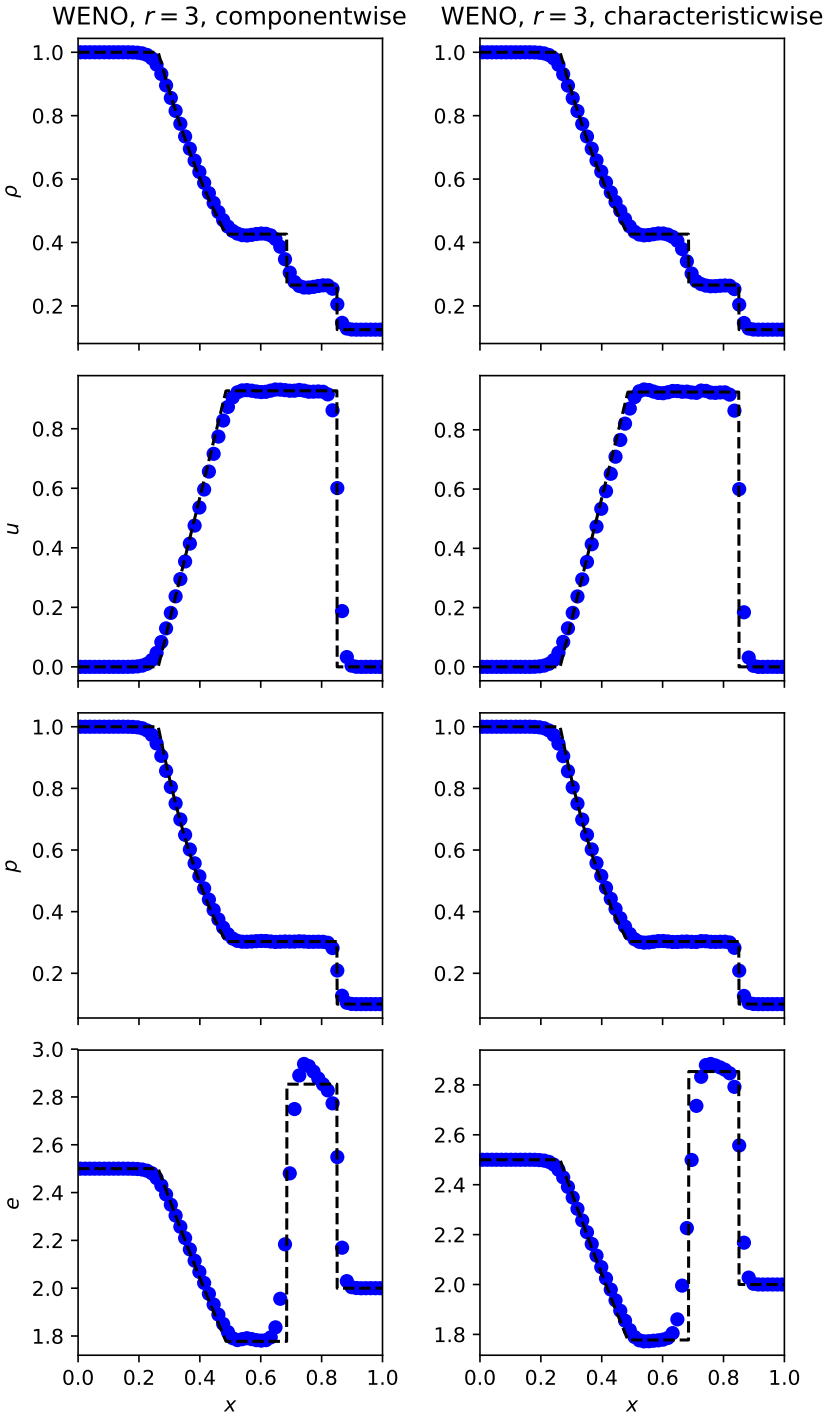

Figure 8.19: The Sod test solved with WENO methods, $r = 3$, using characteristic-wise and component-wise reconstruction, using 64 zones and a cfl of 0.5. To compare with Godunov's method see figure 8.9 and with PPM see figure 8.10. The WENO method does not capture discontinuities, especially the contact, as well as the PPM method which is specifically designed for the Euler equations. The small oscillations visible in the component-wise reconstruction are damped by using the more complex and expensive characteristic-wise approach.

hydro_examples: `weno_euler.py`

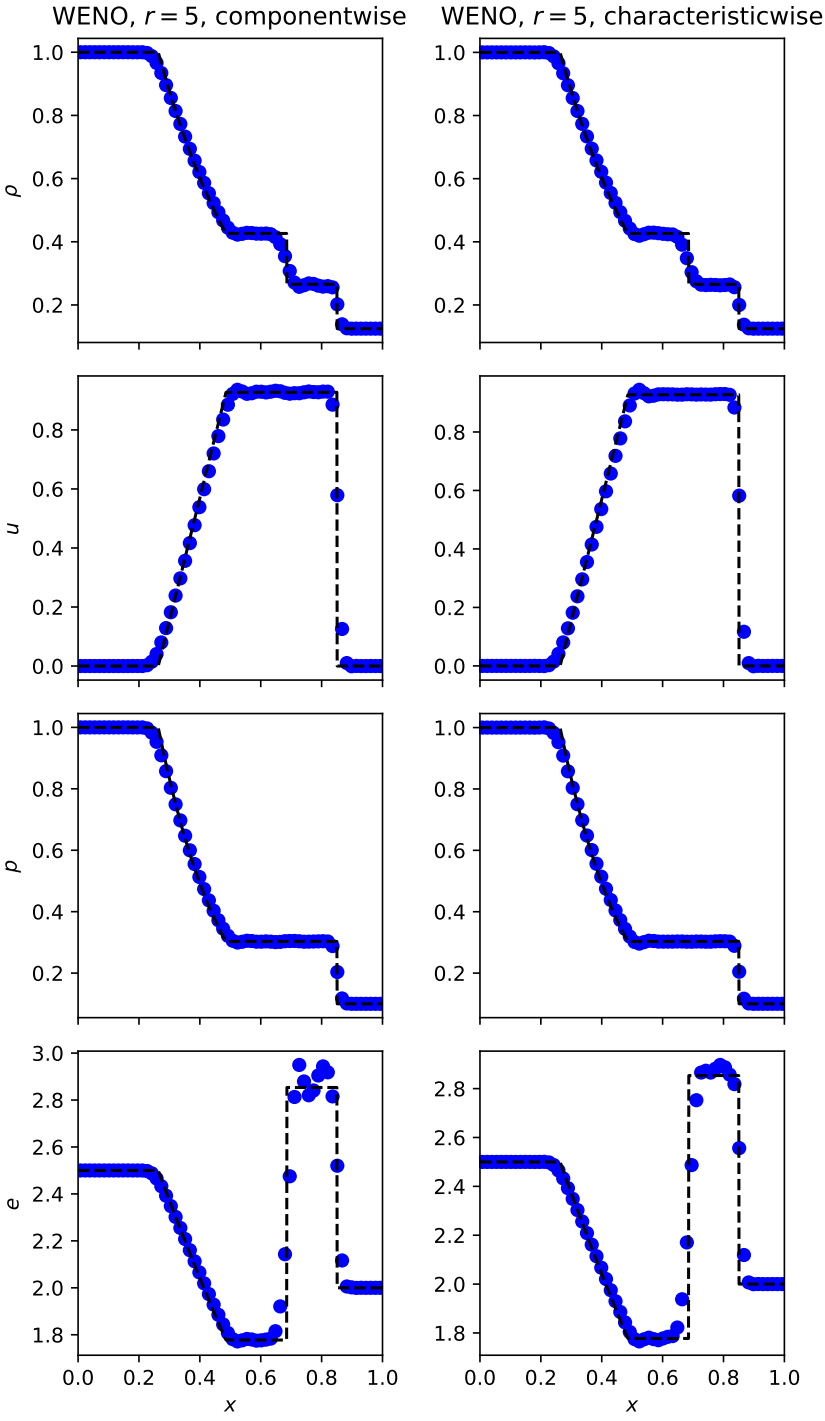

Figure 8.20: The Sod test solved with WENO methods, $r = 5$, using characteristic-wise and component-wise reconstruction, using 64 zones and a cfl of 0.5. Compare with figure 8.19 to see the effect of the higher order of WENO scheme. The oscillations in the component-wise approach are much more pronounced as the order of the reconstruction is increased, and the characteristic-wise approach continues to help with this.
hydro_examples: `weno_euler.py`

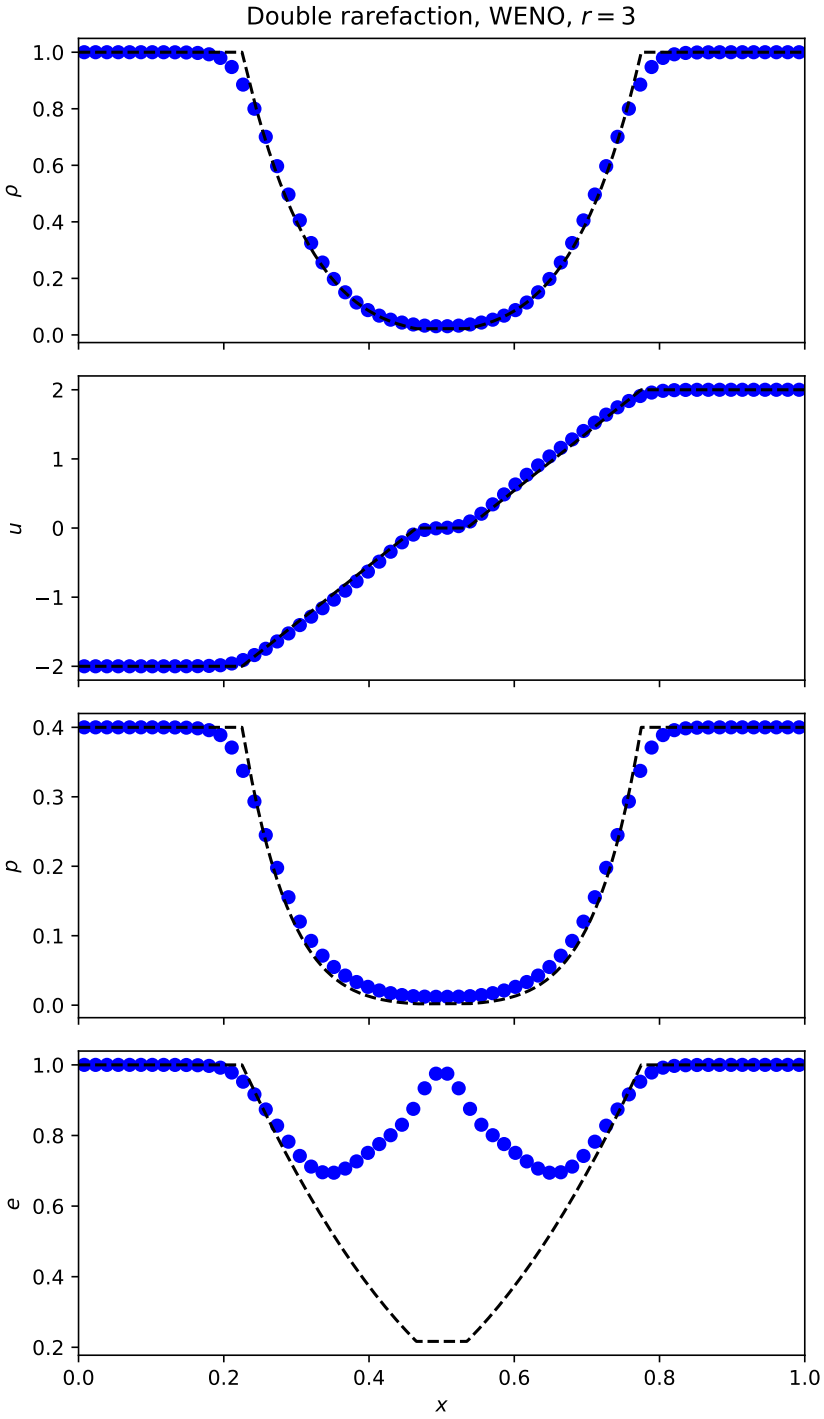

Figure 8.21: The double rarefaction test solved with WENO methods, $r = 3$, using characteristic-wise and component-wise reconstruction, using 64 zones and a cfl of 0.5. To compare with Godunov's method see figure 8.11 and with PPM see figure 8.12. The WENO method captures the edges of the rarefactions almost as well as the PPM method, but is as almost as poor as Godunov's method at the trivial contact at the center. A less diffusive flux-splitting may help here. Increasing the reconstruction order has little effect.

hydro_examples: `weno_euler.py`

Part III

Elliptic and Parabolic Problems

Chapter 9

Elliptic Equations and Multigrid

9.1 Elliptic equations

The simplest elliptic PDE is *Laplace's equation*:

$$\nabla^2 \phi = 0 \tag{9.1}$$

Only slightly more complex is *Poisson's equation* (Laplace + a source term):

$$\nabla^2 \phi = f \tag{9.2}$$

These equations can arise in electrostatics (for the electric potential), solving for the gravitational potential from a mass distribution, or enforcing a divergence constraint on a vector field (we'll see this when we consider incompressible flow).

Another common elliptic equation is the *Helmholtz equation*:

$$(\alpha - \nabla \cdot \beta \nabla)\phi = f \tag{9.3}$$

A Helmholtz equation can arise, for example, from a time-dependent equation (like diffusion) by discretizing in time.

Notice that there is no time-dependence in any of these equations. The quantity ϕ is specified instantaneously in the domain subject to boundary conditions. This makes the solution methods very different then what we saw for hyperbolic problems.

9.2 Fourier Method

A direct way of solving a constant-coefficient elliptic equation is using Fourier transforms. Using a general Fourier transform (which we consider here) works only for

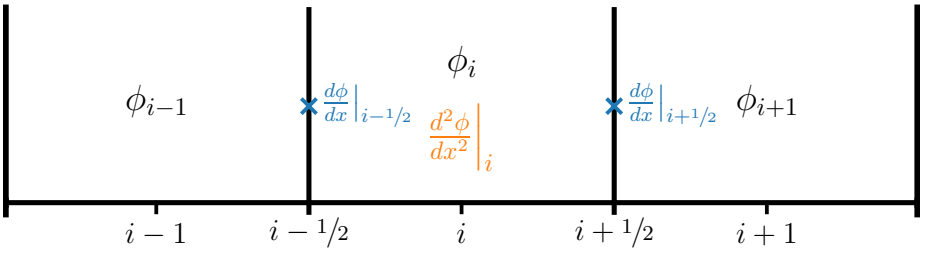

Figure 9.1: The centerings of the first and second derivatives for a standard Laplacian discretization. Our data, ϕ, is cell-centered. The first-derivatives, $d\phi/dx$, are edge-centered, and the second-derivative, $d^2\phi/dx^2$, is cell-centered.

periodic boundary conditions, but other basis functions (e.g., all sines or all cosines) can be used for other boundary conditions.

Consider the Poisson equation:

$$\nabla^2\phi = f \tag{9.4}$$

We will difference this in a second-order accurate fashion—see Figure 9.1. In 1-d, the Laplacian is just the second-derivative. If our solution is defined at cell-centers, then we first compute the first-derivative on cell edges:

$$\left.\frac{d\phi}{dx}\right|_{i-1/2} = \frac{\phi_i - \phi_{i-1}}{\Delta x} \tag{9.5}$$

$$\left.\frac{d\phi}{dx}\right|_{i+1/2} = \frac{\phi_{i+1} - \phi_i}{\Delta x} \tag{9.6}$$

These are second-order accurate on the interface. We can then compute the second-derivative at the cell-center by differencing these edge values:

$$\left.\frac{d^2\phi}{dx^2}\right|_i = \frac{d\phi/dx|_{i+1/2} - d\phi/dx|_{i-1/2}}{\Delta x} \tag{9.7}$$

The extension to 2-d is straightforward. Thinking of the Laplacian as $\nabla^2\phi = \nabla\cdot\nabla\phi$, we first compute the gradient of ϕ on edges:

$$[\nabla\phi\cdot\hat{x}]_{i+1/2,j} = \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{9.8}$$

$$[\nabla\phi\cdot\hat{y}]_{i,j+1/2} = \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{9.9}$$

Again, since this is defined on edges, this represents a centered difference, and is therefore second-order accurate. We then difference the edge-centered gradients to

the center to get the Laplacian at cell-centers:

$$[\nabla^2\phi]_{i,j} = \frac{[\nabla\phi\cdot\hat{x}]_{i+1/2,j} - [\nabla\phi\cdot\hat{x}]_{i-1/2,j}}{\Delta x} + \frac{[\nabla\phi\cdot\hat{y}]_{i,j+1/2} - [\nabla\phi\cdot\hat{y}]_{i,j-1/2}}{\Delta y}$$
$$= \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{9.10}$$

Again, since we used a centered-difference of the edge values, this expression is second-order accurate. This is the standard *5-point stencil* for the 2-d Laplacian*.

We now assume that we have an FFT subroutine (see § 1.2.6) that can take our discrete real-space data, $\phi_{i,j}$ and return the discrete Fourier coefficients, Φ_{k_x,k_y}, and likewise for the source term:

$$\Phi_{k_x,k_y} = \mathcal{F}(\phi_{i,j}) \quad F_{k_x,k_y} = \mathcal{F}(f_{i,j}) \tag{9.11}$$

The power of the Fourier method is that derivatives in real space are multiplications in Fourier space, which makes the solution process in Fourier space straightforward.

We now express $\phi_{i,j}$ and $f_{i,j}$ as sums over their Fourier components. Here we define M as the number of grid points in the x-direction and N as the number of grid points in the y-direction. As before, we are using i as the grid index, we will use I as the imaginary unit:

$$\phi_{i,j} = \frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}\Phi_{k_x,k_y}e^{2\pi I i k_x/M}e^{2\pi I j k_y/N} \tag{9.12}$$

$$f_{i,j} = \frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}F_{k_x,k_y}e^{2\pi I i k_x/M}e^{2\pi I j k_y/N} \tag{9.13}$$

Inserting these into the differenced equation, we have:

$$\frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}\left\{\frac{\Phi_{k_x,k_y}}{\Delta x^2}e^{2\pi I j k_y/N}\left[e^{2\pi I(i+1)k_x/M} - 2e^{2\pi I i k_x/M} + e^{2\pi I(i-1)k_x/M}\right] + \right.$$
$$\left.\frac{\Phi_{k_x,k_y}}{\Delta y^2}e^{2\pi I i k_x/M}\left[e^{2\pi I(j+1)k_y/N} - 2e^{2\pi I j k_y/N} + e^{2\pi I(j-1)k_y/N}\right]\right\} =$$
$$\frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}F_{k_x,k_y}e^{2\pi I i k_x/M}e^{2\pi I j k_y/N} \tag{9.14}$$

We can bring the righthand side into the sums on the left, and we can then look at

*There are other possible second-order accurate stencils, including a 9-point stencil in 2-d, that are less commonly used.

just a single (k_x, k_y) term in the series:

$$e^{2\pi I i k_x/M} e^{2\pi I j k_y/N} \left\{ \frac{\Phi_{k_x,k_x}}{\Delta x^2} \left[e^{2\pi I k_x/M} + e^{-2\pi I k_x/M} - 2 \right] + \frac{\Phi_{k_x,k_x}}{\Delta y^2} \left[e^{2\pi I k_y/N} + e^{-2\pi I k_y/N} - 2 \right] - F_{k_x,k_y} \right\} = 0 \tag{9.15}$$

Simplifying, we have:

$$\Phi_{k_x,k_y} = \frac{1}{2} \frac{F_{k_x,k_y}}{\left[\cos(2\pi k_x/M) - 1\right] \Delta x^{-2} + \left[\cos(2\pi k_y/N) - 1\right] \Delta y^{-2}} \tag{9.16}$$

This is the algebraic solution to the Poisson equation in Fourier (frequency) space. Once we evaluate this, we can get the real-space solution by doing the inverse transform:

$$\phi_{i,j} = \mathcal{F}^{-1}(\Phi_{k_x,k_y}) \tag{9.17}$$

We can test this technique with the source term:

$$f = 8\pi^2 \cos(4\pi y) \left[\cos(4\pi x) - \sin(4\pi x)\right] - 16\pi^2 \left[\sin(4\pi x) \cos(2\pi y)^2 + \sin(2\pi x)^2 \cos(4\pi y)\right] \tag{9.18}$$

which has the analytic solution[†]:

$$\phi = \sin(2\pi x)^2 \cos(4\pi y) + \sin(4\pi x) \cos(2\pi y)^2 \tag{9.19}$$

Note that this solution has the required periodic behavior. Figure 9.2 shows the solution.

The main downside of this approach is that, because we solve for a single component independently (Eq. 9.16), this only works for linear problems with constant coefficients. This makes it an excellent choice for cosmological problems solving the gravitational Poisson equation with periodic boundaries on all sides of the domain (see, e.g., [41]). However, for a problem like:

$$\nabla \cdot (\beta \nabla \phi) = f \tag{9.20}$$

there would be "cross-talk" between the Fourier modes of β and ϕ, and we would not be able to solve for a single mode of Φ_{k_x,k_y} independently. We discuss more general methods that work for these forms next.

[†]Note: throughout this chapter, we devise test problems by picking a function that meets the desired boundary conditions and then inserting it into the analytic equation we are solving to find the righthand side

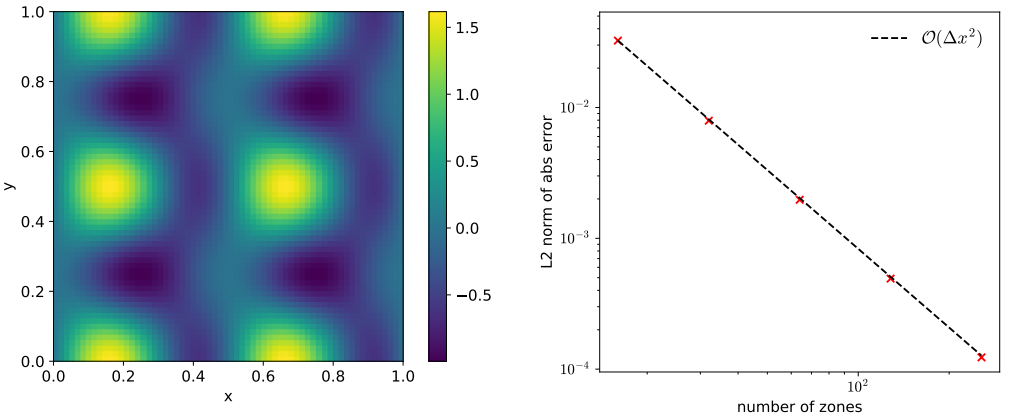

Figure 9.2: (left) Solution to the Poisson equation on a 64^2 grid with source from Eq. 9.18. (right) Error vs. the true solution as a function of resolution for the Fourier method, showing second-order convergence. hydro_examples: `poisson_fft.py`

9.3 Relaxation

Relaxation is an iterative technique, and as we will see shortly, it provides the basis for the multigrid technique.

Consider Poisson's equation, again differenced as:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{9.21}$$

For each zone (i, j), we couple in the zones ± 1 in x and ± 1 in y. For the moment, consider the case where $\Delta x = \Delta y$. If we solve this discretized equation for $\phi_{i,j}$, then we have:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{9.22}$$

A similar expression exists for every zone in our domain, coupling all the zones together. We can't separate the solution of $\phi_{i,j}$ for the neighboring zones, but instead can apply an iterative technique called *relaxation* (also sometimes called *smoothing* because generally speaking the solution to elliptic equations is a smooth function) to find the solution for ϕ everywhere.

Imagine an initial guess to ϕ: $\phi^{(0)}_{i,j}$. We can improve that guess by using our difference equation to define a new value of ϕ, $\phi^{(1)}_{i,j}$:

$$\phi^{(1)}_{i,j} = \frac{1}{4}(\phi^{(0)}_{i+1,j} + \phi^{(0)}_{i-1,j} + \phi^{(0)}_{i,j+1} + \phi^{(0)}_{i,j-1} - \Delta x^2 f_{i,j}) \tag{9.23}$$

or generally, the $k+1$ iteration will see:

$$\phi^{(k+1)}_{i,j} = \frac{1}{4}(\phi^{(k)}_{i+1,j} + \phi^{(k)}_{i-1,j} + \phi^{(k)}_{i,j+1} + \phi^{(k)}_{i,j-1} - \Delta x^2 f_{i,j}) \tag{9.24}$$

This will (slowly) converge to the true solution[‡], since each zone is coupled to each other zone (and to the boundary values that we need to specify—more on that in a moment). This form of relaxation is called *Jacobi iteration*. To implement this, you need two copies of ϕ—the old iteration value and the new iteration value.

An alternate way to do the relaxation is to update $\phi_{i,j}$ in place, as soon as the new value is known. Thus the neighboring cells will see a mix of the old and new solutions. We can express this in-place updating as:

$$\phi_{i,j} \leftarrow \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{9.25}$$

This only requires a single copy of ϕ to be stored. This technique is called *Gauss-Seidel iteration*. A host of other relaxation methods exist, including linear combinations of these two. An excellent discussion of these approaches, and their strengths and weaknesses is given in [17].

Next consider the Helmholz equation with constant coefficients:

$$(\alpha - \beta\nabla^2)\phi = f \tag{9.26}$$

We can discretize this as:

$$\alpha\phi_{i,j} - \beta\left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}\right) = f_{i,j} \tag{9.27}$$

and the update of $\phi_{i,j}$ through relaxation is:

$$\phi_{i,j} \leftarrow \left(f_{i,j} + \frac{\beta}{\Delta x^2}\phi_{i+1,j} + \frac{\beta}{\Delta x^2}\phi_{i-1,j} + \frac{\beta}{\Delta y^2}\phi_{i,j+1} + \frac{\beta}{\Delta y^2}\phi_{i,j-1}\right) \Big/ \left(\alpha + \frac{2\beta}{\Delta x^2} + \frac{2\beta}{\Delta y^2}\right) \tag{9.28}$$

Notice that if $\alpha = 0$, $\beta = -1$, and $\Delta x = \Delta y$, we recover the relaxation expression for Poisson's equation from above.

9.3.1 Boundary conditions

When using a cell-centered grid, no points fall exactly on the boundary, so we need to use ghost cells to specify boundary conditions. A single ghost cell is sufficient for the 5-point stencil used here. The common types of boundary conditions are *Dirichlet* (specified value on the boundary), *Neumann* (specified first derivative on the boundary), and periodic. Some restrictions apply (see discuss this later, in § 9.5).

Consider Dirichlet boundary conditions, specifying values ϕ_l on the left and ϕ_r on the right boundaries.[§] We'll label the first zone inside the domain, at the left boundary,

[‡]Formally, convergence is only guaranteed if the matrix in our linear system is *diagonally dominant*. The Laplacian used here is not quite diagonally domainant, but these methods still converge.

[§]If the value, ϕ_l or ϕ_r is zero, we call this a *homogeneous boundary condition*. Otherwise we call it an *inhomogeneous boundary condition*

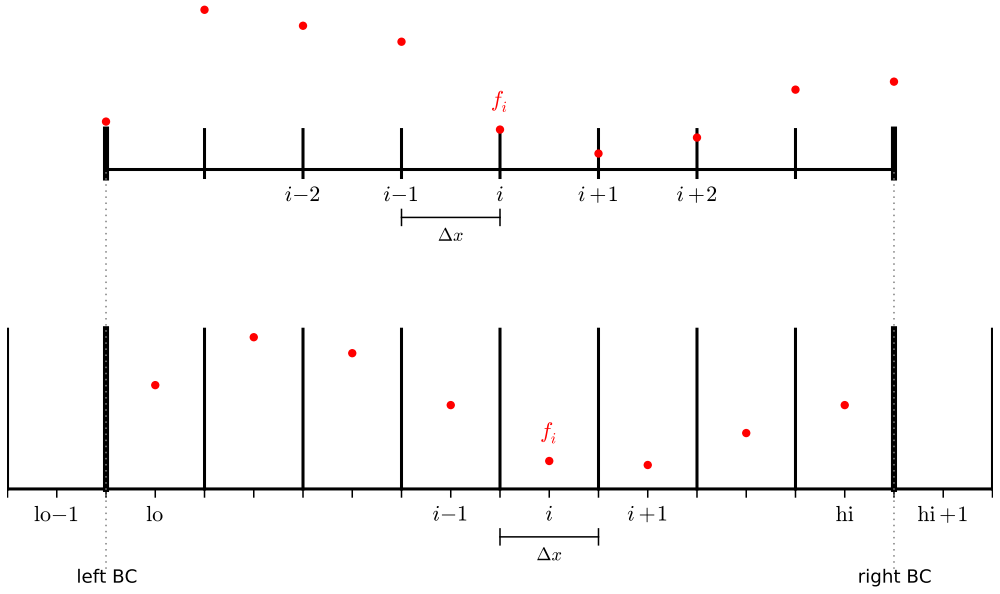

Figure 9.3: A node-centered (top) and cell-centered (bottom) finite difference grid showing the data and domain boundaries. Notice that for the cell-centered grid, there is no data point precisely at the boundary.

lo, and the last zone inside the domain, at the right boundary, hi—see Figure 9.3. To second order, we can average the zone values on either side of the interface to get the boundary condition:

$$\phi_l = \frac{1}{2}(\phi_{\rm lo} + \phi_{\rm lo-1}) \tag{9.29}$$

$$\phi_r = \frac{1}{2}(\phi_{\rm hi} + \phi_{\rm hi+1}) \tag{9.30}$$

This then tells us that the values we need to assign to the ghost cells are:

$$\phi_{\rm lo-1} = 2\phi_l - \phi_{\rm lo} \tag{9.31}$$

$$\phi_{\rm hi+1} = 2\phi_r - \phi_{\rm hi} \tag{9.32}$$

If we instead consider Neumann boundary conditions, we specify values of the derivative on the boundaries: $\phi_x|_l$ on the left and $\phi_x|_r$ on the right. We note that a single difference across the boundary is second-order accurate on the boundary (it is a centered-difference there), so to second-order:

$$\phi_x|_l = \frac{\phi_{\rm lo} - \phi_{\rm lo-1}}{\Delta x} \tag{9.33}$$

$$\phi_x|_r = \frac{\phi_{\rm hi+1} - \phi_{\rm hi}}{\Delta x} \tag{9.34}$$

This then tells us that the ghost cells are filled as:

$$\phi_{\mathrm{lo}-1} = \phi_{\mathrm{lo}} - \Delta x \, \phi_x|_l \tag{9.35}$$

$$\phi_{\mathrm{hi}+1} = \phi_{\mathrm{hi}} + \Delta x \, \phi_x|_r \tag{9.36}$$

9.3.2 Residual and true error

The *residual error* is a measure of how well our discrete solution satisfies the discretized equation. For the Poisson equation, we can the residual as:

$$r_{i,j} = f_{i,j} - (L\phi)_{i,j} \tag{9.37}$$

and the residual error as:

$$\epsilon^{(r)} = \|r\| \tag{9.38}$$

where L represents our discretized Laplacian. Note that r is the error with respect to the discrete form of the equation. The true error is the measure of how well our discrete solution approximates the true solution. If ϕ^{true} satisfies $\nabla^2 \phi^{\mathrm{true}} = f$, then the true error in each zone is

$$e_{i,j} = \phi^{\mathrm{true}}(x_i, y_j) - \phi_{i,j} \tag{9.39}$$

and

$$\epsilon^{\mathrm{true}} = \|e_{i,j}\| \tag{9.40}$$

We can make $\epsilon^{(r)}$ approach machine precision by performing more and more relaxation iterations, but after some point, this will no longer improve ϵ^{true}. The only way to improve ϵ^{true} is to make Δx and Δy smaller. In practice we do not know the true solution so we cannot compute ϵ^{true} and will instead have to rely on $\epsilon^{(r)}$ to monitor our error.

Note that since our operator is linear,

$$Le = L\phi^{\mathrm{true}} - L\phi = f - L\phi = r \tag{9.41}$$

so the error in our solution obeys a Poisson equation with the residual as the source—we'll see this in the next section.

9.3.3 Norms

There are several different norms that are typically used in defining errors on the grid. The L_∞ norm (or 'inf'-norm) is just the maximum error on the grid:

$$\|e\|_\infty = \max_{i,j}\{|e_{i,j}|\} \tag{9.42}$$

This will pick up on local errors.

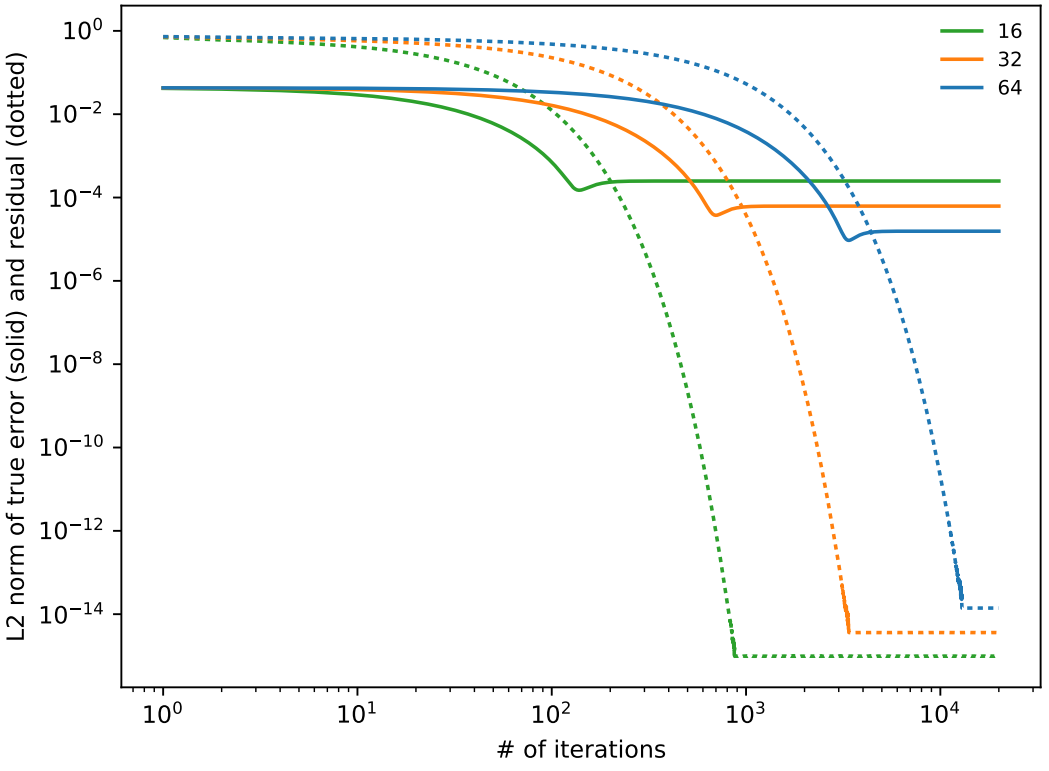

Figure 9.4: Gauss-Seidel relaxation applied to $\phi_{xx} = \sin(x)$ with $\phi(0) = \phi(1) = 0$. Shown are the L2 norm of the error compared with the true solution (solid lines) and the L2 norm of the residual (dotted lines) for 3 different resolutions (16, 32, and 64 zones).
hydro_examples: `smooth.py`

The L_1 norm and L_2 norms are more global.

$$\|e\|_1 = \frac{1}{N}\sum_{i,j}|e_{i,j}| \tag{9.43}$$

$$\|e\|_2 = \left(\frac{1}{N}\sum_{i,j}|e_{i,j}|^2\right)^{1/2} \tag{9.44}$$

Generally, the measure in L_2 falls between L_∞ and L_1. Regardless of the norm used, if the problem converges, it should converge in all norms. For reference, the AMReX library[¶] uses L_∞ in its multigrid solvers.

9.3.4 Performance

Consider the simple Poisson problem[||] on $x \in [0, 1]$:

$$\phi_{xx} = \sin(x), \qquad \phi(0) = \phi(1) = 0 \tag{9.45}$$

The analytic solution to this is simply

$$\phi^a(x) = -\sin(x) + x\sin(1) \tag{9.46}$$

We can perform smoothing and compute both the error against the analytic solution (the 'true' error), $e \equiv \|\phi^a(x_i) - \phi_i\|_2$ and the residual error, $\|r_i\|_2$. Figure 9.4 shows these errors as a function of the number of smoothing iterations for 3 different resolutions.

Notice that the true error stalls at a relatively high value—this is the truncation error of the method. From one resolution to the next, the true error changes as Δx^2, indicating that we are converging as our method should. No additional amount of smoothing will change this—we are getting the best answer to the problem we can with our choice of discretization. In contrast, the residual error decreases to machine precision levels—this is indicating that our solution is an exact solution to the discrete equation (to roundoff-error). In practice, we can only monitor the residual error, not the true error, and we hope that small residual error implies a small true error.

Figure 9.5 shows the error with respect to the true solution and of the residual for pure smoothing in three different norms. The overall behavior is qualitative similar regardless of the choice of norm.

To demonstrate the influence of the boundary conditions, Figure 9.6 shows the norm of the true error for the same problem, but this time with a naive implementation of the boundary conditions—simply initializing the ghost cell to the boundary value, instead of averaging to the interface:

$$\phi_{\mathrm{lo}-1} = \phi_{\mathrm{lo}} \tag{9.47}$$

$$\phi_{\mathrm{hi}+1} = \phi_{\mathrm{hi}} \tag{9.48}$$

We see that with this mistake at the boundaries, the error of the entire solution is affected, and we get only first-order convergence with resolution (this can be seen by looking at the spacing of the curves). The previous solution, with the correct BCs is shown for reference, and shows second-order convergence. It is important to note that because every zone is linked to every other zone (and the boundary) in elliptic problems, an error at the boundary can pollute the global solution.

[¶] `https://github.com/amrex-codes`

[||] This is the test problem used throughout *A Multigrid Tutorial* [17]. We use the same problem here to allow for easy comparison to the discussions in that text.

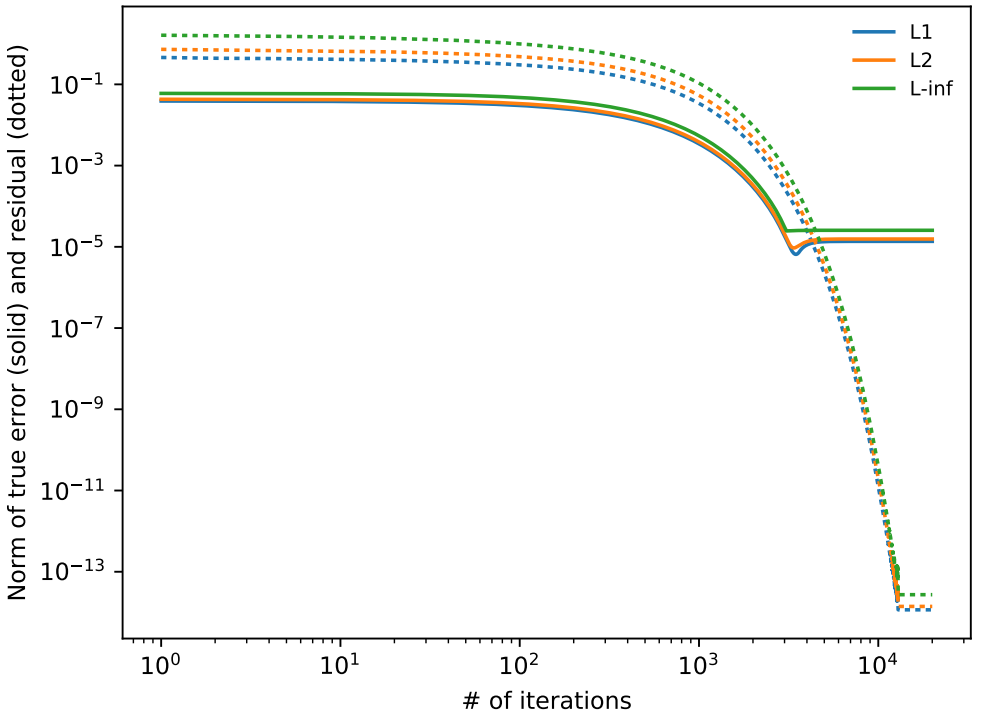

Figure 9.5: Gauss-Seidel relaxation applied to $\phi_{xx} = \sin(x)$ with $\phi(0) = \phi(1) = 0$. This is like figure 9.4, but now we show the error in 3 different norms: L_1, L_2, and L_∞.
hydro_examples: `smooth-norms.py`

9.3.5 Frequency/wavelength-dependent error

We can think of the error in the solution as a superposition of high (short) and low (long) frequency (wavelength) modes. Smoothing works really well to eliminate the short wavelength noise quickly, but many iterations are needed to remove the long wavelength noise (see Figure **??**). A very important concept to understand here is that when we talk about long wavelength error, we are expressing it in terms of the number of zones across the feature, not as physical length. We can get an intuitive feel for this behavior by thinking about how smoothing works. Every zone is coupled to every other zone, and we can think about each zone seeing one more zone away for each iteration. When the error is short wavelength, that means that there are only a few zones across it, and after a few iterations, all of the zones have seen the short wavelength error, and can eliminate it. For a very long wavelength error, many iterations will be needed until the smoothing couples one zone to another that is a wavelength away.

This behavior suggests that if we could represent our problem on a coarser grid, the error will now be of shorter wavelength, and smoothing will be more efficient. This

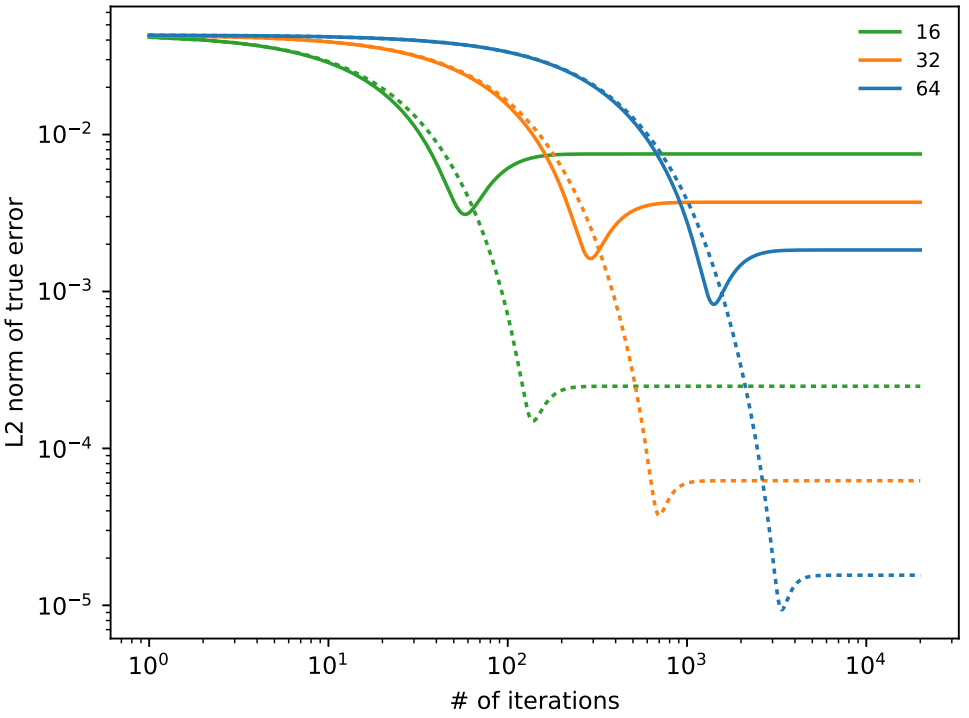

Figure 9.6: The same problem as in figure 9.4, but now we done with a naive first-order boundary conditions—just initializing the ghost cell to the boundary value (solid lines). We see that this achieves only first-order convergence in the true error. The correct second-order implmentation is shown as the dotted lines.
hydro_examples: `smooth-badbcs.py`

is the core idea behind multigrid, which we see next.

Exercise 9.1

Implement 1-d smoothing for the Laplace equation on cc-grid. Use an initial guess for the solution:

$$\phi_0(x) = \frac{1}{3}(\sin(2\pi x) + \sin(2\pi\, 8x) + \sin(2\pi\, 16x)) \tag{9.49}$$

on a 128 zone grid with Dirichlet boundary conditions. This initial guess has both high-frequency and low-frequency noise. Observe that the high-frequency stuff goes after only a few smoothing iterations, but many iterations are needed to remove the low-frequency noise. You should see something like Figure **??**.

Figure 9.7 illustrates this behavior. We are solving the Laplace equation, $\phi'' = 0$ in 1-

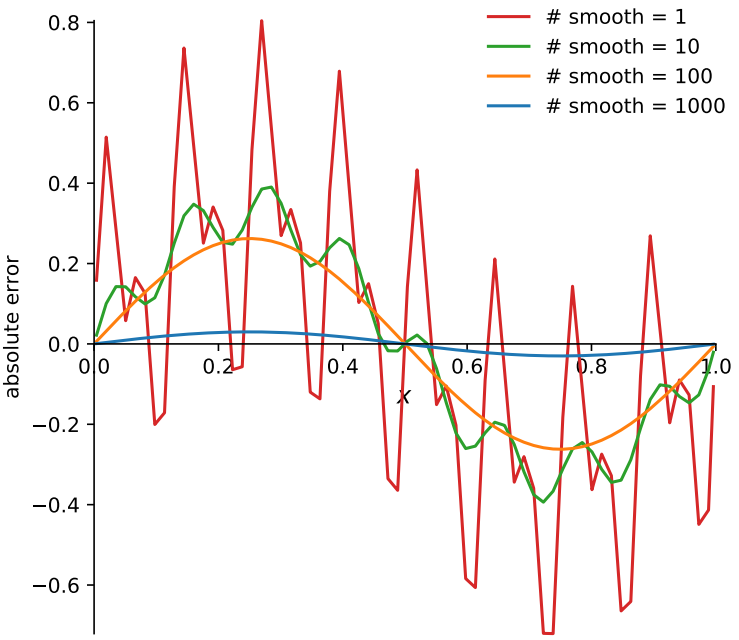

Figure 9.7: Error in the solution to $\phi'' = 0$ given an initial guess with 3 different wavenumbers of noise. A 128 zone grid was used. The different curves are different numbers of smoothing iterations.
hydro_examples: `smooth-modes.py`

d on $[0, 1]$ with homogeneous Dirichlet boundary conditions. The solution is simply $\phi = 0$. We use Eq. 9.49 as an initial guess—this is a superposition of 3 different modes. We see that the after just a few iterations, the shortest wavelength mode, $\sin(2\pi 16x)$ is no longer apparent in the error, and just the two longer wavelength modes dominate. By 100 iterations, the error appears to be only due to the longest wavelength mode, $\sin(2\pi x)$. Even after 1000 iterations, we still see this mode in the error. This demonstrates that the longest wavelength modes in the error take the longest to smooth away.

9.4 Multigrid

The text *A Multigrid Tutorial* [17] provides an excellent introduction to the mechanics of multigrid. The discussion here differs mainly in that we are dealing with cell-centered/finite-volume data. We already saw that the treatment of boundary condi-

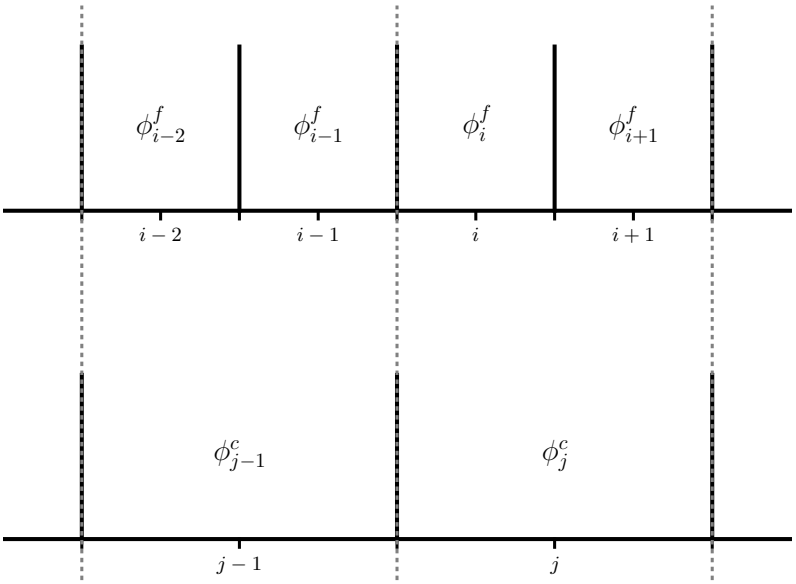

Figure 9.8: A fine grid and corresponding underlying coarse grid.

tions is more complicated because we do not have a point directly on the boundary. The other complication comes in transferring the data from a fine grid to a coarser grid, and back. Before we discuss the multigrid technique, we'll understand how the operations between grids work.

9.4.1 Prolongation and restriction on cell-centered grids

Multigrid relies on transferring the problem up and down a hierarchy of grids. The process of moving the data from the fine grid to the coarse grid is called *restriction*. The reverse process, moving data from the coarse grid to the fine grid is called *prolongation*. If the data on our grid is a conserved quantity, we want restriction and prolongation to conserve the amount of stuff when transitioning data between the fine and coarse grids.

1-d

Consider a 1-d finite-volume/cell-centered finite-difference grid shown in Figure 9.8—we see a fine grid and the corresponding coarser grid. If ϕ represents a density, then conservation requires:

$$\phi^c_j = \frac{1}{\Delta x^c}(\Delta x^f \phi^f_i + \Delta x^f \phi^f_{i+1}) \tag{9.50}$$

or, for a jump in 2 in resolution ($\Delta x^c = 2\Delta x^f$),

$$\phi_j^c = \frac{1}{2}(\phi_i^f + \phi_{i+1}^f) \tag{9.51}$$

This latter form appears as a simple average to the interface of the two fine cells / center of the corresponding coarse cell.

The simplest type of prolongation is simply *direct injection*:

$$\phi_i^f = \phi_j^c \tag{9.52}$$

$$\phi_{i+1}^f = \phi_j^c \tag{9.53}$$

A higher-order method is to do linear reconstruction of the coarse data and then average under the profile, e.g.,

$$\phi(x) = \frac{\phi_{j+1}^c - \phi_{j-1}^c}{2\Delta x^c}(x - x_j^c) + \phi_j^c \tag{9.54}$$

To second-order, we can find the values of ϕ_i^f and ϕ_{i+1}^f by evaluating $\phi(x)$ at the x-coordinate corresponding to their cell-centers,

$$x_i^f = x_j^c - \frac{\Delta x^c}{4} \tag{9.55}$$

$$x_{i+1}^f = x_j^c + \frac{\Delta x^c}{4} \tag{9.56}$$

giving

$$\phi_i^f = \phi_j^c - \frac{1}{8}(\phi_{j+1}^c - \phi_{j-1}^c) \tag{9.57}$$

$$\phi_{i+1}^f = \phi_j^c + \frac{1}{8}(\phi_{j+1}^c - \phi_{j-1}^c) \tag{9.58}$$

Notice that this is conservative, since $\Delta x^f(\phi_i^f + \phi_{i+1}^f) = \Delta x^c \phi_j^c$.

2-d

Restriction from the fine grid to the coarse grid is straightforward. Since the fine cells are perfectly enclosed by a single coarse cell, we simply average:

$$\phi_{i,j}^c = \frac{1}{4}(\phi_{--}^f + \phi_{+-}^f + \phi_{-+}^f + \phi_{++}^f) \tag{9.59}$$

Prolongation requires us to reconstruct the coarse data and use this reconstruction to determine what the fine cell values are. For instance, a linear reconstruction of the coarse data in x and y is:

$$\phi(x,y) = \frac{m_x}{\Delta x^c}(x - x_i^c) + \frac{m_y}{\Delta y^c}(y - y_j^c) + \phi_{i,j}^c \tag{9.60}$$

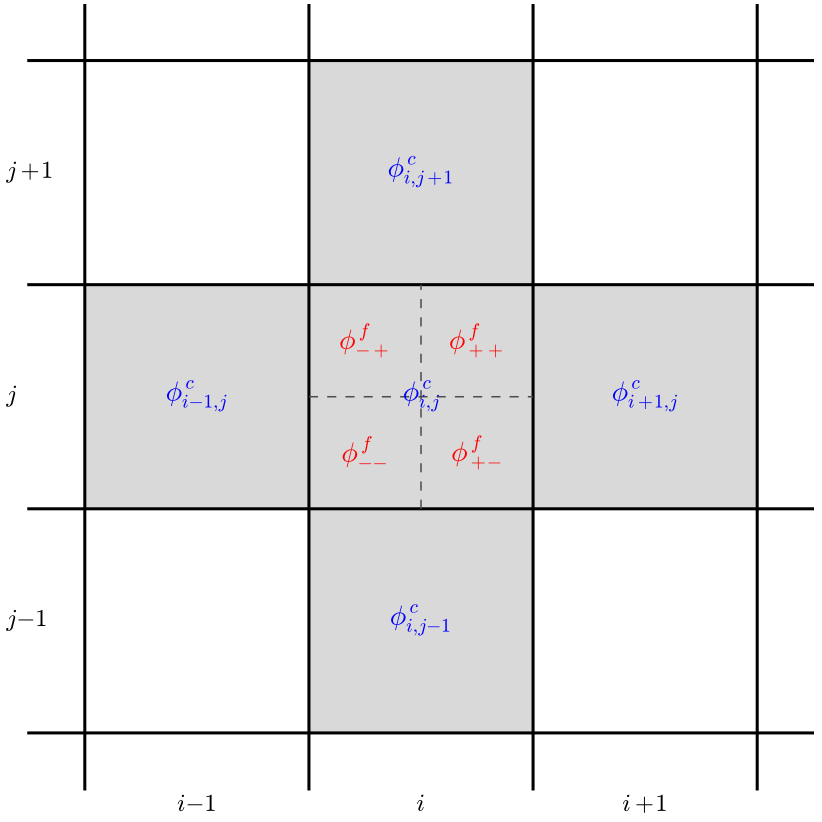

Figure 9.9: Four fine cells and the underlying coarse grid. For prolongation, the fine cells in red are initialized from a coarse parent. The gray coarse cells are used in the reconstruction of the coarse data. For restriction, the fine cells are averaged to the underlying coarse cell.

with slopes:

$$m_x = \frac{1}{2}(\phi^c_{i+1,j} - \phi^c_{i-1,j}) \tag{9.61}$$

$$m_y = \frac{1}{2}(\phi^c_{i,j+1} - \phi^c_{i,j-1}) \tag{9.62}$$

When averaged over the coarse cell, $\phi(x,y)$ recovers the average, $\phi^c_{i,j}$ in that cell (this means that our interpolant is conservative). We can evaluate the value in the fine cells by evaluating $\phi(x,y)$ at the center of the fine cells,

$$x^f_\pm = x^c_i \pm \frac{\Delta x^c}{4} \tag{9.63}$$

$$y^f_\pm = y^c_j \pm \frac{\Delta y^c}{4} \tag{9.64}$$

$$\tag{9.65}$$

This gives

$$\phi^f_{\pm\pm} = \phi^c_{i,j} \pm \frac{1}{4} m_x \pm \frac{1}{4} m_y \tag{9.66}$$

(Note: you would get the same expression if you averaged $\phi(x, y)$ over the fine cell.)

There are other options for prolongation and restriction, both of higher and lower order accuracy. However, the methods above seem to work well.

9.4.2 Multigrid cycles

The basic idea of multigrid** is to smooth a little on the current grid solving $L\phi = f$, compute the residual, r, then *restrict* r to a coarser grid and smooth on that grid solving $Le = r$, restrict again, ... Once you reach a sufficiently coarse grid, the problem solved exactly. Then the data is moved up to the finer grids, a process called *prolongation*. The error on the coarse grid, e, is prolonged to the finer grid. This error is then used to correct the solution on the finer grid, some smoothing is done, and then the data is prolonged up again.

Note: on the coarse grids, you are not solving the original system, but rather an error equation. If the boundary conditions in the original system are inhomogeneous, the boundary conditions for the error equations are now homogeneous. This must be understood by any ghost cell filling routines.

There are many different forms of the multigrid process. The simplest is called the *V-cycle*. Here you start of the fine grid, restrict down to the coarsest, solve, and then prolong back up to the finest. The flow looks like a 'V'. You continue with additional V-cycles until the residual error is smaller than your tolerance.

9.4.3 Bottom solver

Once the grid is sufficiently coarse, the linear system is small enough to be solved directly. This is the bottom solver operation. In the most ideal case, where the finest grid is some power of 2, $N_x = N_y = 2^n$, then the multigrid procedure can continue down until a 2×2 grid is created (Figure 9.10 illustrates this idea for a one-dimensional grid). This is the coarsest grid upon which one can still impose boundary conditions. With this small grid, just doing additional smoothing is sufficient enough to 'solve' the problem. No fancy bottom solver is needed.

For a general rectangular grid or one that is not a power of 2, the coarsest grid will likely be larger. For the general case, a linear system solver like conjugate gradient (or a variant) is used on the coarsest grid.

**In these discussions, we use *multigrid* to mean *geometric multigrid*, where the coarsening is done to the grid geometry directly. The alternative is *algebraic multigrid*, where it is the structure of the matrix in the linear system itself that is coarsened.

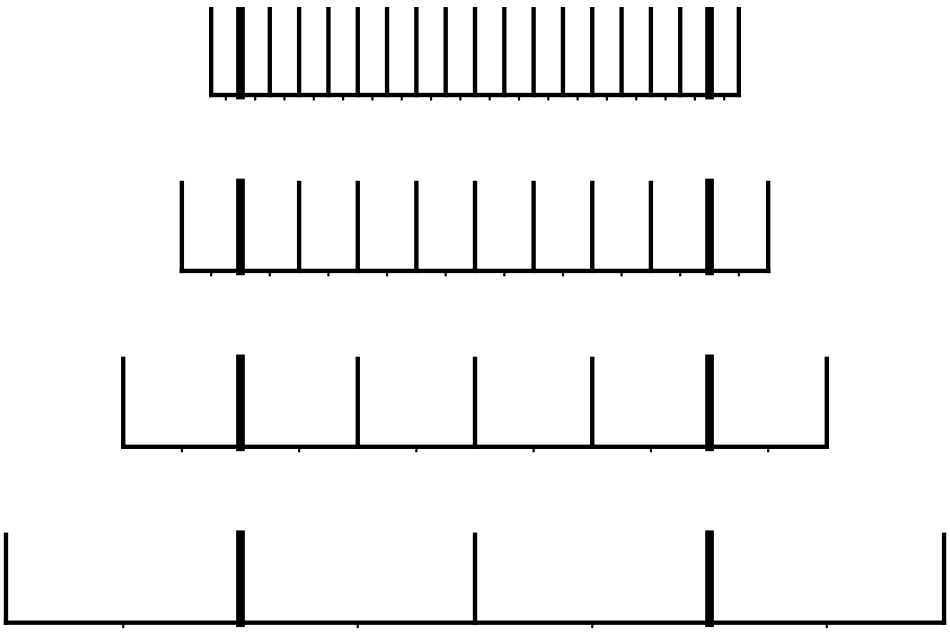

Figure 9.10: Illustration of the hierarchy of grids leading to the coarsest 2-zone grid (in one-dimension). Each grid has a single ghost cell to accommodate boundary conditions.

9.4.4 Boundary conditions throughout the hierarchy

The general inhomogeneous boundary conditions from Eqs. 9.31 and 9.35 apply to the finest level. But because we are solving the residual equation of the coarsest levels in the multigrid hierarchy, the boundary conditions on $Le = r$ are all homogeneous (but of the same type, Dirichlet, Neumann, or periodic, as the fine level).

Implementing these boundary conditions in your multigrid solver means that you will have separate actions for the fine level (where inhomogeneous boundaries may apply) and the coarser levels (where you will always be homogeneous).

An alternate way to enforce boundary conditions is via *boundary charges*. For inhomogeneous boundary conditions, boundary charges can be used to convert the BCs to homogeneous BCs. This has the advantage of allowing the ghost cell filling routines only deal with the homogeneous case.

Consider the one-dimensional Poisson equation, near the left boundary our discretized equation appears as:

$$\frac{\phi_{\mathrm{lo}-1} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{9.67}$$

Inhomogeneous BCs at the left boundary would give the condition:

$$\phi_{\mathrm{lo}-1} = 2\phi_l - \phi_{\mathrm{lo}} \tag{9.68}$$

Substituting this into the discrete equation, we have:

$$\frac{2\phi_l - \phi_{\mathrm{lo}} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{9.69}$$

Bringing the boundary condition value over to the RHS, we see

$$\frac{-3\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} - \frac{2\phi_l}{\Delta x^2} \tag{9.70}$$

Now the left side looks precisely like the differenced Poisson equation with homogeneous Dirichlet BCs. The RHS has an additional 'charge' that captures the boundary value. By modifying the source term, f, in the multigrid solver to include this charge, we can use the homogeneous ghost cell filling routines throughout the multigrid algorithm. This technique is discussed a bit in [26].

Note that the form of the boundary charge will depend on the form of the elliptic equation—the expressions derived above apply only for $\nabla^2\phi = f$.

9.4.5 Stopping criteria

Repeated V-cycles are done until:

$$\|r\| < \epsilon\|f\| \tag{9.71}$$

on the finest grid, for some user-input tolerance, ϵ. Here, $\|f\|$ is called the *source norm*. If $\|f\| = 0$, then we stop when

$$\|r\| < \epsilon \tag{9.72}$$

Picking the tolerance ϵ is sometimes problem-dependent, and generally speaking, a problem with a large number of zones will require a looser tolerance.

The general rule-of-thumb is that each V-cycle should reduce your residual by about 1 order of magnitude. It is important that your bottom solver solves the coarse problem to a tolerance of 10^{-3} or 10^{-4} in order for the solver to converge. Figure 9.11 shows the true and residual errors for $\phi_{xx} = \sin(x)$ as a function of V-cycle number, illustrating the expected performance.

The overall convergence of the multigrid algorithm is limited by the discretization of the Laplacian operator used and the implementation of the boundary conditions. Figure 9.12 shows the error in the solution as the number of zones is increased—demonstrating second-order convergence for our implementation.

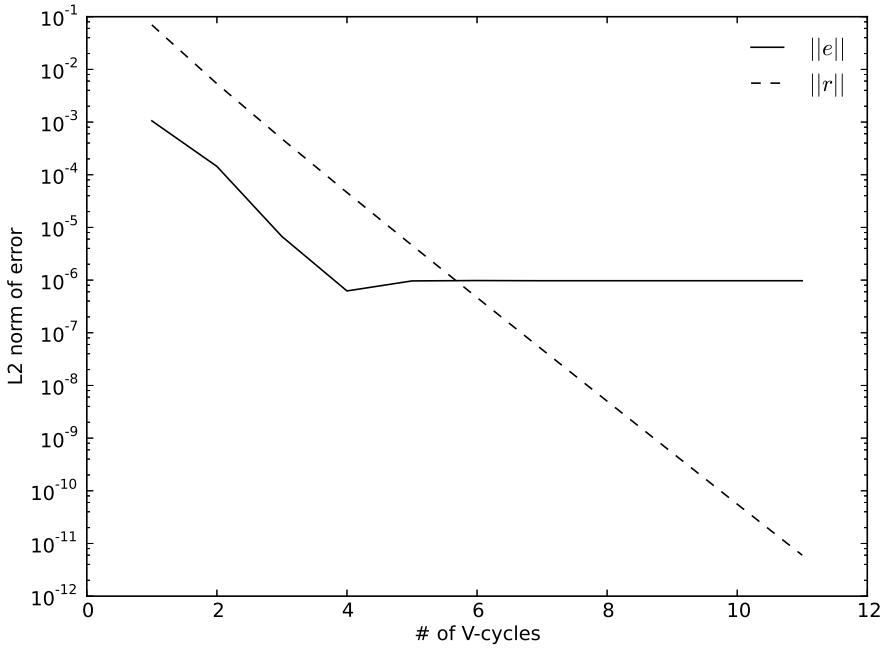

Figure 9.11: Error in the multigrid solution to our model problem ($\phi_{xx} = \sin(x)$) as a function of V-cycle. We see that the true error, $\|e\|$ stalls at truncation error while the residual error, $\|r\|$ reaches roundoff error, the same behavior as seen with smoothing alone (as expected).
hydro_examples: `mg_test.py`

9.5 Solvability

For $\nabla^2\phi = f$ with periodic or Neumann boundaries all around, the sum of f must equal 0 otherwise the solution will not converge. Instead, we will simply find the solution increase each V-cycle. This is seen as follows:

$$\int_\Omega f d\Omega = \int_\Omega \nabla^2\phi d\Omega = \int_{\partial\Omega} \nabla\phi \cdot n dS = 0 \tag{9.73}$$

For all homogeneous Neumann boundaries, we have $\nabla\phi \cdot dS = 0$ by construction, so that integral is zero, requiring that the source integrate to zero. If the Neumann boundaries are inhomogeneous, there is still a solvability condition on f based on the sum on the boundary values.

A simple example of solvability is:

$$\phi'' = 0 \tag{9.74}$$

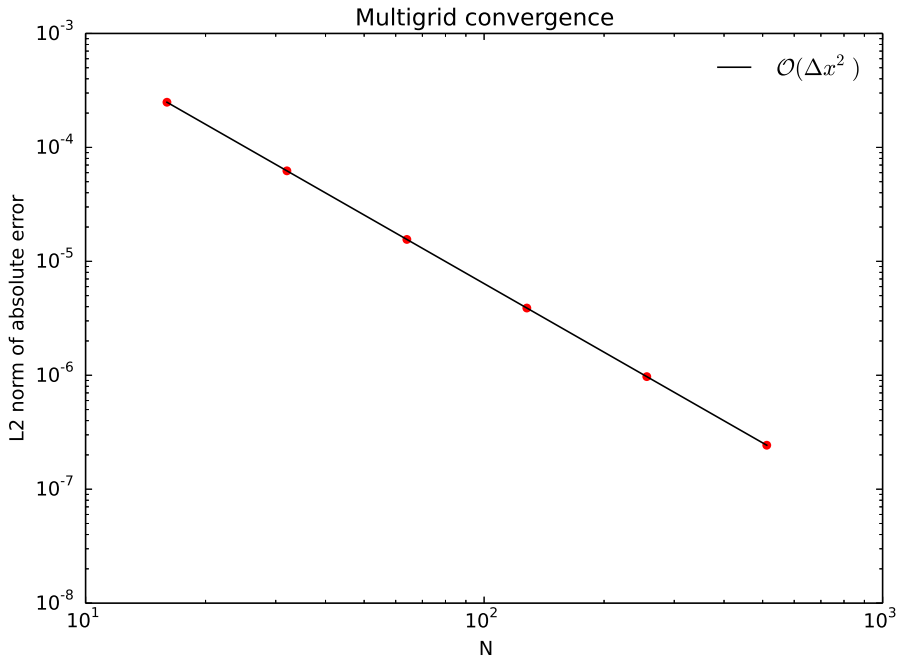

Figure 9.12: Convergence of the multigrid algorithm.
hydro_examples: `mg_converge.py`

on $[a, b]$ with

$$\phi'(a) = A \tag{9.75}$$
$$\phi'(b) = B \tag{9.76}$$

We can integrate $\phi'' = 0$ to get $\phi(x) = \alpha x + \beta$ where α and β are integration constants. But note that this is just a straight line, with a single slope, α, so it is not possible to specify two unique slopes, A and B at the boundary unless there is a non-zero source term.

For all periodic boundaries, we have $\nabla\phi|_{\text{left}} = -\nabla\phi|_{\text{right}}$ on the left and right boundaries by definition of the periodicity (and similarly for the top and bottom). Again this implies that f must integrate to zero.

Sometimes, with periodic boundary conditions all around, you need to enforce that f integrate to zero numerically to test convergence. This is discussed in § 15.2.1.

9.6 Going Further

9.6.1 Red-black Ordering

When using domain decomposition to spread the problem across parallel processors, the smoothing is often done as *red-black Gauss-Seidel*. In this ordering, you imagine

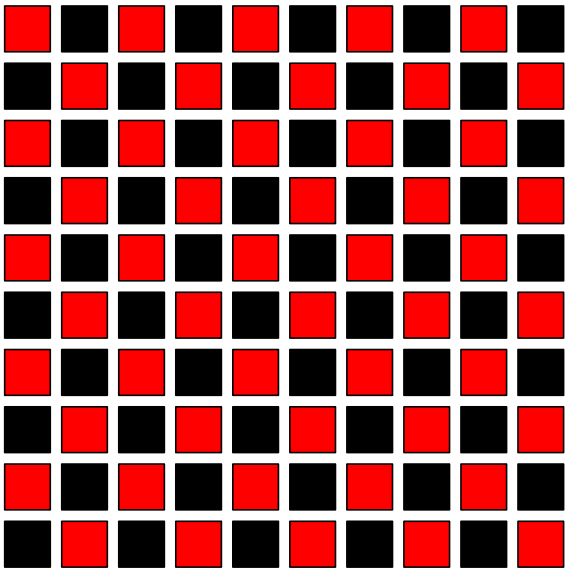

Figure 9.13: The red-black ordering of zones.

the grid to be a checkerboard (see Figure 9.13). In the first Gauss-Seidel pass you update the red squares and in the second, the black squares. The advantage is that when updating the red, you can be sure that none of the zones you depend on (the neighboring black zones) will change. This makes the decomposition parallel. Note: this works for the standard 5-point Laplacian. If you are doing some other operator with a different stencil, then this decomposition may no longer hold.

9.6.2 More General Elliptic Equations

The most general *second-order* elliptic equation takes the form:

$$\alpha\phi + \nabla \cdot (\beta \nabla \phi) + \gamma \cdot \nabla \phi + \nabla \cdot (\zeta \phi) = f \tag{9.77}$$

Here, γ and ζ are vectors. Solving a general elliptic equation of this form can be accomplished with multigrid using the same basic ideas here. The main change is that the smoothing algorithm and the construction of the residual will need to discretize the more general operator, and these coefficients will need to be restricted to the coarser grids (some on edges). This is explored in § 15.2 for a variable-coefficient Poisson equation:

$$\nabla \cdot (\beta \nabla \phi) = f \tag{9.78}$$

Chapter 10

Diffusion

10.1 Diffusion

Physically, a diffusive process obeys Fick's law—the quantity that is diffusing, ϕ, moves from higher to lower concentration at a rate proportional to the gradient,

$$F = -k\nabla\phi \tag{10.1}$$

If we think of this as a diffusive flux, then we can write the time-rate-of-change of ϕ as a conservation law:

$$\phi_t + \nabla \cdot F(\phi) = 0 \tag{10.2}$$

This gives rise to the diffusion equation:

$$\phi_t = \nabla \cdot (k\nabla\phi) \tag{10.3}$$

Diffusion has a time-dependence like the advection equations we already saw, but also has a global character like an elliptic problem. Unlike advection, there is no sense of upwinding in diffusion. As we will see, we can cast our differenced equations in a form reminiscent of an elliptic problem.

In one-dimension, and assuming that the diffusion coefficient, k, is constant, we have

$$\frac{\partial\phi}{\partial t} = \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) \tag{10.4}$$

The diffusion equation can describe thermal diffusion (for example, as part of the energy equation in compressible flow), species/mass diffusion for multi-species flows, or the viscous terms in incompressible flows. In this form, the diffusion coefficient (or conductivity), k, can be a function of x, or even ϕ.

We will consider a constant diffusion coefficient as our model problem:

$$\frac{\partial \phi}{\partial t} = k \frac{\partial^2 \phi}{\partial x^2} \tag{10.5}$$

The diffusion equation is the prototypical parabolic PDE. The basic behavior of the diffusion equation is to take strongly peaked concentrations of ϕ and smooth them out with time.

10.2 Explicit differencing

A nice overview of the discretizations for diffusion and their stability is given in [64]. The simplest way to difference this equation is *explicit* in time (i.e., the righthand side depends only on the old state):

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} \tag{10.6}$$

This is second-order accurate in space, but only first order accurate in time (since the righthand side is not centered in time).

As with the advection equation, when differenced explicitly, there is a constraint on the timestep required for stability. Looking at the growth of a single Fourier mode, $\phi_i^n = A^n e^{iI\theta}$ with $I = \sqrt{-1}$, we find:

$$\frac{A^{n+1}}{A^n} = 1 + 2\frac{k\Delta t}{\Delta x^2}(\cos\theta - 1) \tag{10.7}$$

Stability requires that $|A^{n+1}/A^n| \leq 1$, which can only be true if $2k\Delta t/\Delta x^2 \leq 1$. Therefore, our timestep constraint in this case is

$$\Delta t < \frac{1}{2}\frac{\Delta x^2}{k} \tag{10.8}$$

As with advection, we often write the timestep as

$$\Delta t = \frac{C}{2}\frac{\Delta x^2}{k} \tag{10.9}$$

where C is a constant, $C < 1$. Note the Δx^2 dependence—this constraint can become really restrictive.

Exercise 10.1

Derive Eq. 10.7 by inserting $\phi_i^n = A^n e^{ij\theta}$ into Eq. 10.6

To complete the solution, we need boundary conditions at the left (x_l) and right (x_r) boundaries. These are typically either Dirichlet:

$$\phi|_{x=x_l} = \phi_l \tag{10.10}$$
$$\phi|_{x=x_r} = \phi_r \tag{10.11}$$

or Neumann:

$$\phi_x|_{x=x_l} = \phi_x|_l \tag{10.12}$$
$$\phi_x|_{x=x_r} = \phi_x|_r \tag{10.13}$$

Physically, a Dirichlet BC means fixing the value of ϕ (e.g., temperature) on the boundary. A Neumann BC means fixing the flux on the boundary.

Like with advection, it helps to have a good test problem to evaluate our methods for diffusion. We can use the fact that the diffusion of a Gaussian is a Gaussian with a smaller amplitude and greaterwidth.

Exercise 10.2

Write a one-dimensional explicit diffusion solver for the domain $[0, 1]$ *with Neumann boundary conditions at each end and* $k = 1$*, using the discretization of Eq. 10.6.*
If we begin with a Gaussian, the resulting solution is also a Gaussian, giving a solution[a]*:*

$$\phi(x,t) = (\phi_2 - \phi_1)\sqrt{\frac{t_0}{t+t_0}}e^{-\frac{1}{4}(x-x_c)^2/k(t+t_0)} + \phi_1 \tag{10.14}$$

Initialize our problem with $t = 0$*, and take* $t_0 = 0.001$*,* $\phi_1 = 1$*, and* $\phi_2 = 2$*, and* x_c *is the coordinate of the center of the domain. Run until* $t = 0.01$ *and compare to the analytic solution above.*

[a] Note: the 2- and 3-d solutions are slightly different than this 1-d solution

Figure 10.1 shows the solution for 64 zones and a value of $C = 0.8$. We see good agreement with the analytic solution. Note that if the initial conditions are not well resolve initially, then the solution will be bad (see Figure 10.2).

As with advection, if we exceed the timestep limit ($C > 1$), then the solution is unstable. This is shown in Figure 10.3.

Our spatial order-of-accuracy is second-order. Figure 10.4 shows the error as a function of number of zones, using the L_2 norm of the solution with respect to the analytic solution. Notice that at the coarsest resolution the error is very high—we are not resolving the initial conditions well enough to have a meaningful solution. At higher resolutions, we converge as $O(\Delta x^2)$. Recall though that our method has a truncation

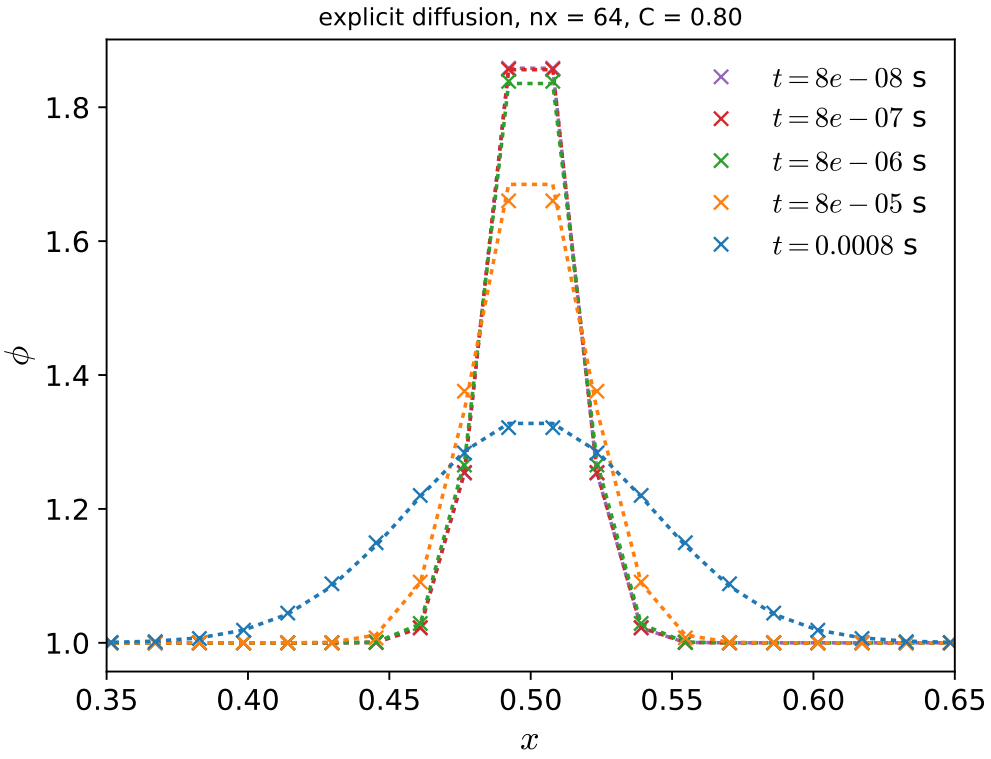

Figure 10.1: Diffusion of a Gaussian using the explicit differencing of Eq. 10.6 with 64 zones and C = 0.8, shown at several times. The dotted line is the analytic solution.
hydro_examples: `diffusion_explicit.py`

error that is $O(\Delta t) + O(\Delta x^2)$, but we don't see the first-order in time scaling. This is because of the timestep restriction. Since $\Delta t \sim \Delta x^2/k$, as we cut Δx by 2, Δt drops by 4, so the timestep choice makes our $O(\Delta t)$ truncation error go as $O(\Delta x^2)$.

Note, this is not the case for advection, where $\Delta t \sim \Delta x$, so for our advection discretizations we will want the same order of accuracy in our spatial and temporal discretizations.

10.3 Implicit with direct solve

Recall that an implicit discretization of advection did not have a timestep restriction for stability. The same holds true for diffusion. A backward-Euler implicit discretization would be:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k \frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{\Delta x^2} \tag{10.15}$$

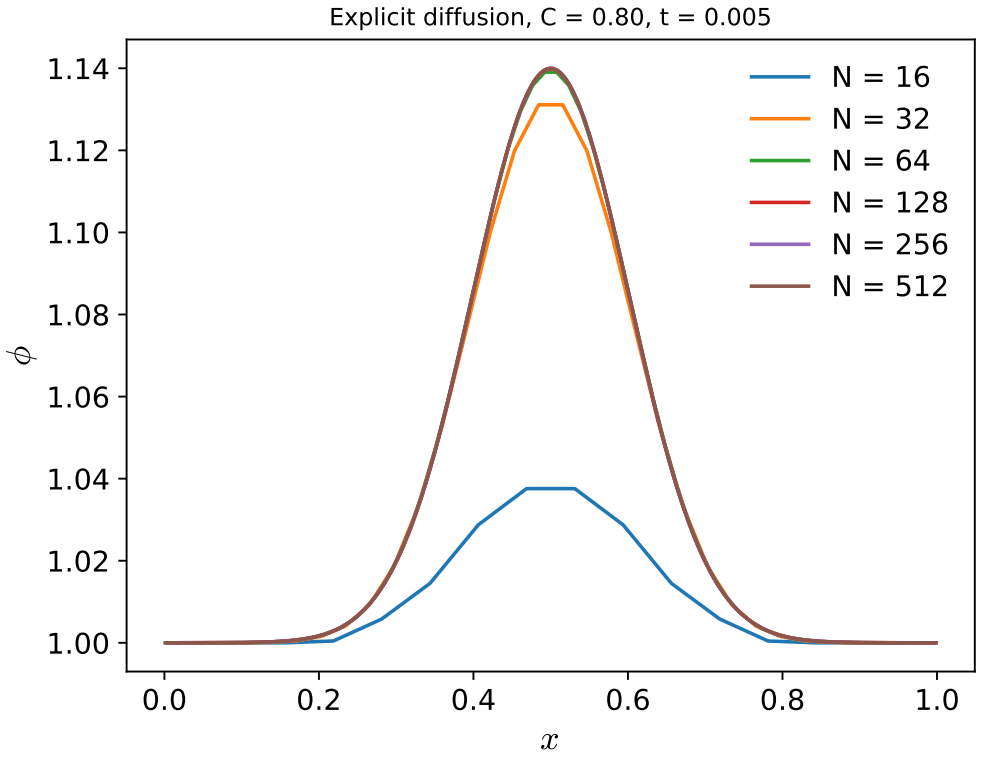

Figure 10.2: Diffusion of a Gaussian using the explicit differencing of Eq. 10.6 with different resolutions. several times. The dotted line is the analytic solution.
hydro_examples: `diffusion_explicit.py`

The only difference with Eq. 10.6 is the time-level of ϕ on the righthand side. Defining:

$$\alpha \equiv k\frac{\Delta t}{\Delta x^2} \tag{10.16}$$

This is still first-order in time. We can do the stability analysis to see the growth of a mode, giving,

$$\frac{A^{n+1}}{A^n} = \frac{1}{1 - 2\alpha(\cos\theta - 1)} \tag{10.17}$$

We see that $|A^{n+1}/A^n| \leq 1$ for all θ, α, so this discretization is unconditionally stable. However, the timestep will still determine the accuracy.

We can write Eq. 10.15 as:

$$-\alpha\phi_{i+1}^{n+1} + (1 + 2\alpha)\phi_i^{n+1} - \alpha\phi_{i-1}^{n+1} = \phi_i^n \tag{10.18}$$

This is a set of coupled algebraic equations. We can write this in matrix form. Using a cell-centered grid, we will solve for the values [lo, hi]. The implicit method can use any $C > 0$.

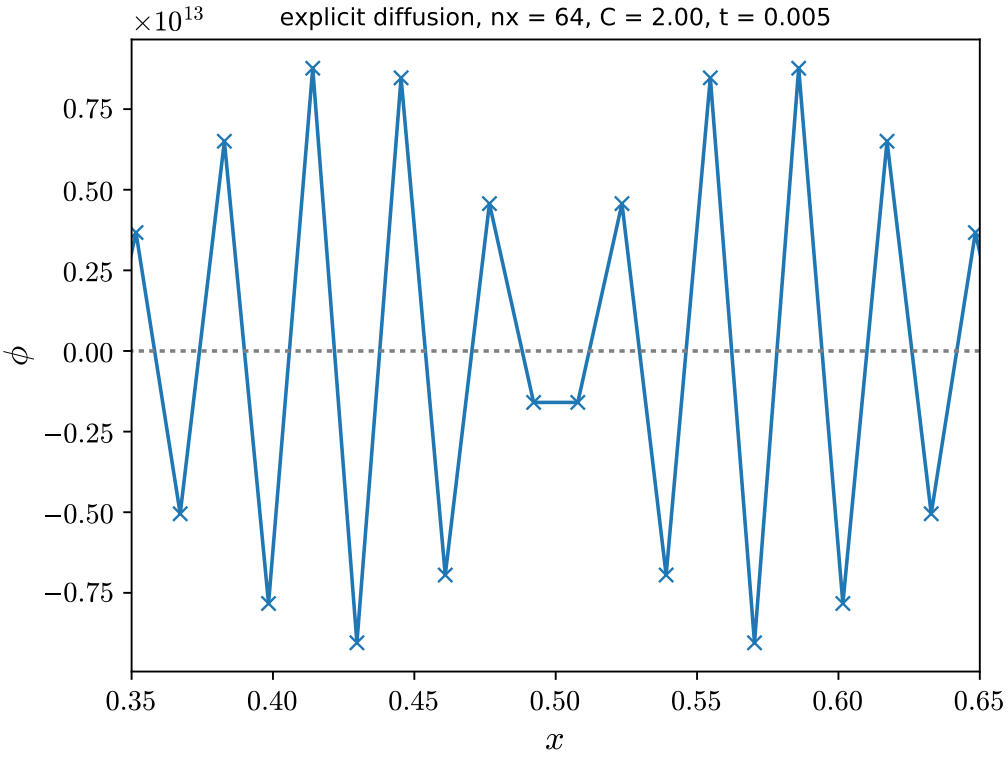

Figure 10.3: Diffusion of a Gaussian using the explicit differencing of Eq. 10.6 with 64 zones, but a timestep with $C > 1$, showing that the solution is unstable.
hydro_examples: `diffusion_explicit.py`

We specify boundary conditions by modifying the stencil (Eq. 10.18) for the updates to lo and hi*. For example, homogeneous Neumann BCs on the left mean:

$$\phi_{\mathrm{lo}-1} = \phi_{\mathrm{lo}} \tag{10.19}$$

and substituting this into Eq 10.18, the update for the leftmost cell is:

$$(1+\alpha)\phi_{\mathrm{lo}}^{n+1} - \alpha\phi_{\mathrm{lo}+1}^{n+1} = \phi_{\mathrm{lo}}^{n} \tag{10.20}$$

If we choose Dirichlet BCs on the right ($\phi|_{x=x_l} = A$), then:

$$\phi_{\mathrm{hi}+1} = 2A - \phi_{\mathrm{hi}} \tag{10.21}$$

Substituting this into Eq 10.18 the update for the rightmost cell is:

$$-\alpha\phi_{\mathrm{hi}-1}^{n+1} + (1+3\alpha)\phi_{\mathrm{hi}}^{n+1} = \phi_{\mathrm{hi}}^{n} + \alpha 2A \tag{10.22}$$

*Here, we use the lo, hi notation for grid indices from § 3.3.3

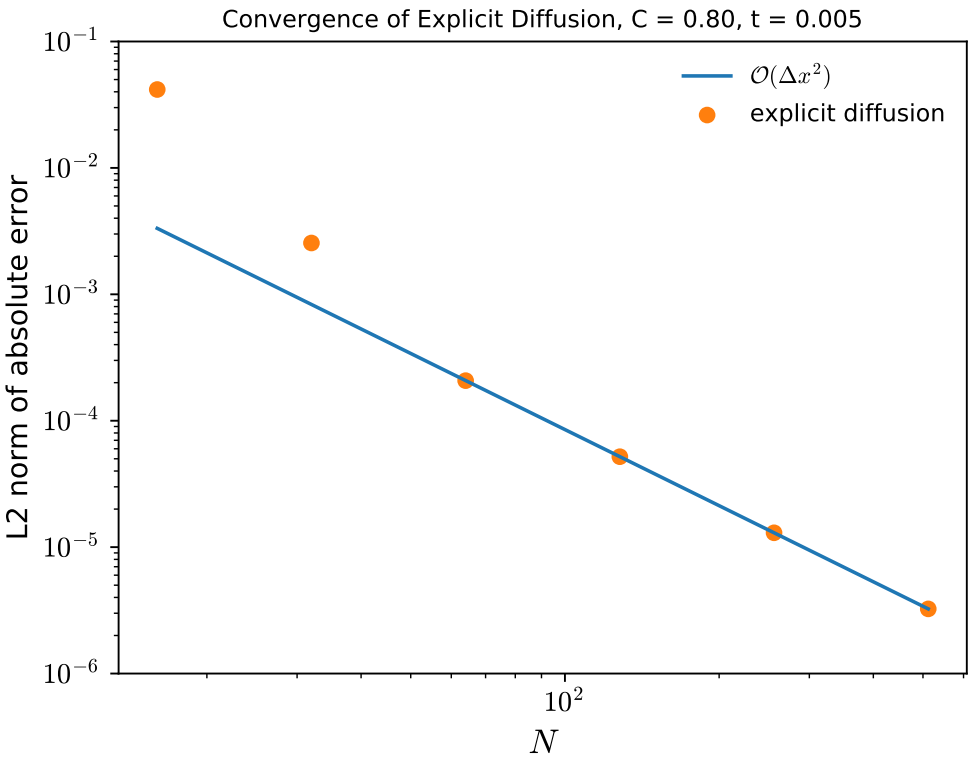

Figure 10.4: Error convergence with resolution of the explicit diffusion using the differencing of Eq. 10.6
hydro_examples: `diffusion_explicit.py`

For all other interior cells, the stencil is unchanged. The resulting system appears as a *tridiagonal* matrix.

$$\begin{pmatrix} 1+\alpha & -\alpha & & & & & & \\ -\alpha & 1+2\alpha & -\alpha & & & & & \\ & -\alpha & 1+2\alpha & -\alpha & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -\alpha & 1+2\alpha & -\alpha & \\ & & & & & -\alpha & 1+3\alpha & \end{pmatrix} \begin{pmatrix} \phi^{n+1}_{\rm lo} \\ \phi^{n+1}_{\rm lo+1} \\ \phi^{n+1}_{\rm lo+2} \\ \vdots \\ \vdots \\ \phi^{n+1}_{\rm hi-1} \\ \phi^{n+1}_{\rm hi} \end{pmatrix} = \begin{pmatrix} \phi^{n}_{\rm lo} \\ \phi^{n}_{\rm lo+1} \\ \phi^{n}_{\rm lo+2} \\ \vdots \\ \vdots \\ \phi^{n}_{\rm hi-1} \\ \phi^{n}_{\rm hi} + \alpha 2A \end{pmatrix} \tag{10.23}$$

This can be solved by standard matrix operations, using a tridiagonal solvers (for example). Notice that the ghost cells do not appear in this linear system—we are only updating the interior points.

Crank-Nicolson time discretization

A second-order in time discretization requires us to center the righthand side in time. We do this as:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{k}{2}\left(\frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} + \frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{\Delta x^2}\right) \tag{10.24}$$

This looks like the average of the explicit and implicit systems we just saw. This time-discretization is called *Crank-Nicolson*. Again, using $\alpha \equiv k\Delta t/\Delta x^2$, and grouping all the $n+1$ terms on the left we have:

$$\phi_i^{n+1} - \frac{\alpha}{2}\left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right) = \phi_i^n + \frac{\alpha}{2}\left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right) \tag{10.25}$$

and grouping together the the $n+1$ terms by zone, we have:

$$-\frac{\alpha}{2}\phi_{i+1}^{n+1} + (1+\alpha)\phi_i^{n+1} - \frac{\alpha}{2}\phi_{i-1}^{n+1} = \phi_i^n + \frac{\alpha}{2}\left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right) \tag{10.26}$$

Considering homogeneous Neumann boundary conditions on the left and right, we again have $\phi_{\mathrm{lo}-1}^{n+1} = \phi_{\mathrm{lo}}^{n+1}$ and $\phi_{\mathrm{hi}+1}^{n+1} = \phi_{\mathrm{hi}}^{n+1}$, and our stencil at the boundary becomes

$$-\frac{\alpha}{2}\phi_{\mathrm{lo}+1}^{n+1} + \left(1+\frac{\alpha}{2}\right)\phi_{\mathrm{lo}}^{n+1} = \phi_{\mathrm{lo}}^n + \frac{\alpha}{2}\left(\phi_{\mathrm{lo}+1}^n - 2\phi_{\mathrm{lo}}^n + \phi_{\mathrm{lo}-1}^n\right) \tag{10.27}$$

The matrix form of this system is:

$$\begin{pmatrix}
1+\frac{\alpha}{2} & -\frac{\alpha}{2} & & & & & \\
-\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & & & & \\
 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & \ddots & \ddots & \ddots & \\
 & & & & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} \\
 & & & & & -\frac{\alpha}{2} & 1+\frac{\alpha}{2}
\end{pmatrix}
\begin{pmatrix}
\phi_{\mathrm{lo}}^{n+1} \\ \phi_{\mathrm{lo}+1}^{n+1} \\ \phi_{\mathrm{lo}+2}^{n+1} \\ \vdots \\ \vdots \\ \phi_{\mathrm{hi}-1}^{n+1} \\ \phi_{\mathrm{hi}}^{n+1}
\end{pmatrix}
=
\begin{pmatrix}
\phi_{\mathrm{lo}}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}}^n \\
\phi_{\mathrm{lo}+1}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}+1}^n \\
\phi_{\mathrm{lo}+2}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}+2}^n \\
\vdots \\
\vdots \\
\phi_{\mathrm{hi}-1}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{hi}-1}^n \\
\phi_{\mathrm{hi}}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{hi}}^n
\end{pmatrix} \tag{10.28}$$

Figure 10.5 shows the result of using $\alpha = 0.8$ and $\alpha = 8.0$. We see that they are both stable, but that the smaller timestep is closer to the analytic solution (especially at early times).

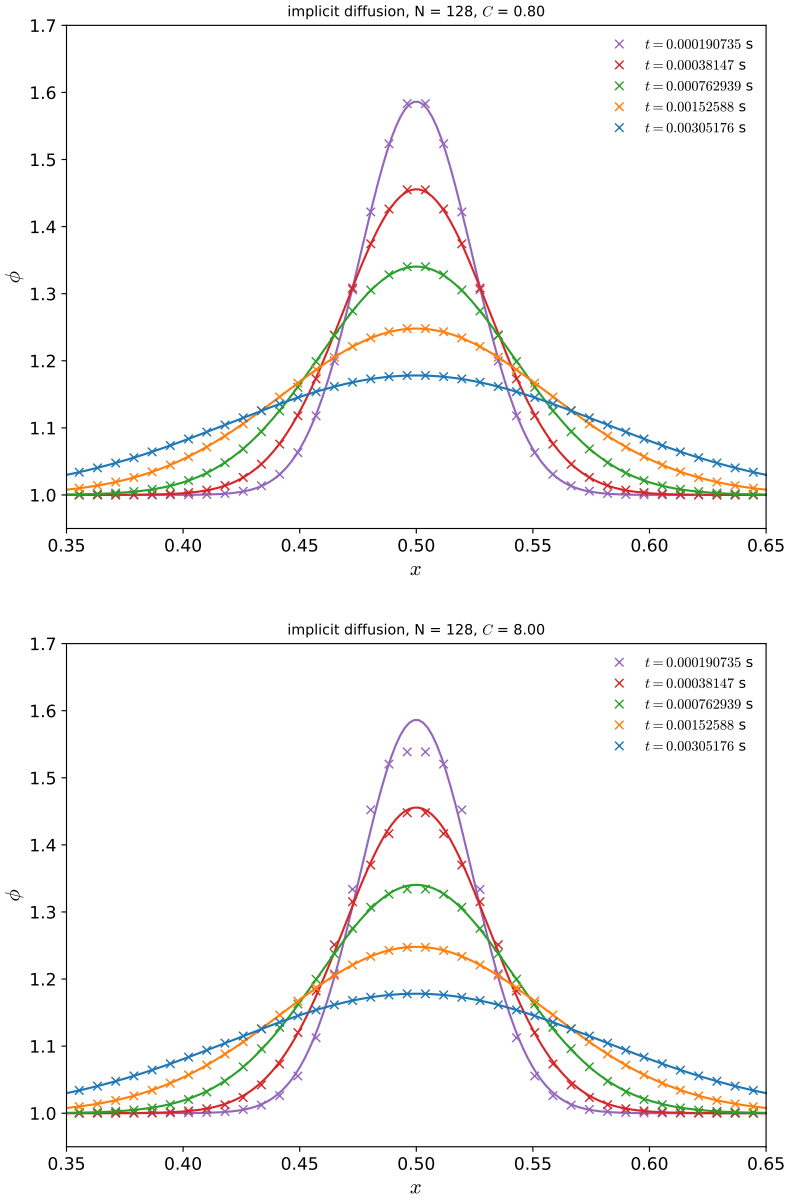

Figure 10.5: Implicit diffusion of a Gaussian (with Crank-Nicolson discretization) with $C = 0.8$ and $C = 8.0$. The exact solution at each time is shown as the dotted line.
hydro_examples: `diffusion_implicit.py`

Exercise 10.3

Write a one-dimensional implicit diffusion solver for the domain $[0, 1]$ *with Neumann boundary conditions at each end and* $k = 1$. *Your solver should use a tridiagonal solver and initialize a matrix like that above. Use*

a timestep close to the explicit step, a grid with N = 128 zones.
Use a Gaussian for your initial conditions (as you did for the explicit problem.

10.4 Implicit multi-dimensional diffusion via multigrid

Instead of doing a direct solve of the matrix form of the system, we can use multigrid techniques. Consider the Crank-Nicolson system we just looked at:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left(k\nabla^2\phi_i^n + k\nabla^2\phi_i^{n+1}\right) \tag{10.29}$$

Grouping all the $n+1$ terms on the left, we find:

$$\phi_i^{n+1} - \frac{\Delta t}{2}k\nabla^2\phi_i^{n+1} = \phi_i^n + \frac{\Delta t}{2}k\nabla^2\phi_i^n \tag{10.30}$$

This is in the form of a constant-coefficient Helmholtz equation,

$$(\alpha - \beta\nabla^2)\phi^{n+1} = f \tag{10.31}$$

with

$$\alpha = 1 \tag{10.32}$$

$$\beta = \frac{\Delta t}{2}k \tag{10.33}$$

$$f = \phi^n + \frac{\Delta t}{2}k\nabla^2\phi^n \tag{10.34}$$

This can be solved using multigrid techniques with a Helmholtz operator. The same boundary conditions discussed in Chapter 9 apply here. The main difference between the multigrid technique for the Poisson problem and the Helmholtz problem is the form of the smoother. In 1-d, we discretize Eq. 10.31 with a second-order difference expression for the second derivative, and isolate ϕ_i, giving a smoothing operation of the form:

$$\phi_i \leftarrow \left(f_i + \frac{\beta}{\Delta x^2}[\phi_{i+1} + \phi_{i-1}]\right) \Big/ \left(\alpha + \frac{2\beta}{\Delta x^2}\right) \tag{10.35}$$

Note that when you take $\alpha = 0$ and $\beta = -1$ in Eq. 10.35, the smoothing operation reduces to the form that we saw in Chapter 9 for just the Poisson equation.

Recall, when using multigrid, you do not need to actually construct the matrix. This is usually the most efficient way to implement diffusion in a multi-dimensional simulation code, especially when distributing the grid across parallel processors. Since the discretization is the same as the direct matrix solve, Eq. 10.28, the result will be exactly the same (to the tolerance of the multigrid solver).

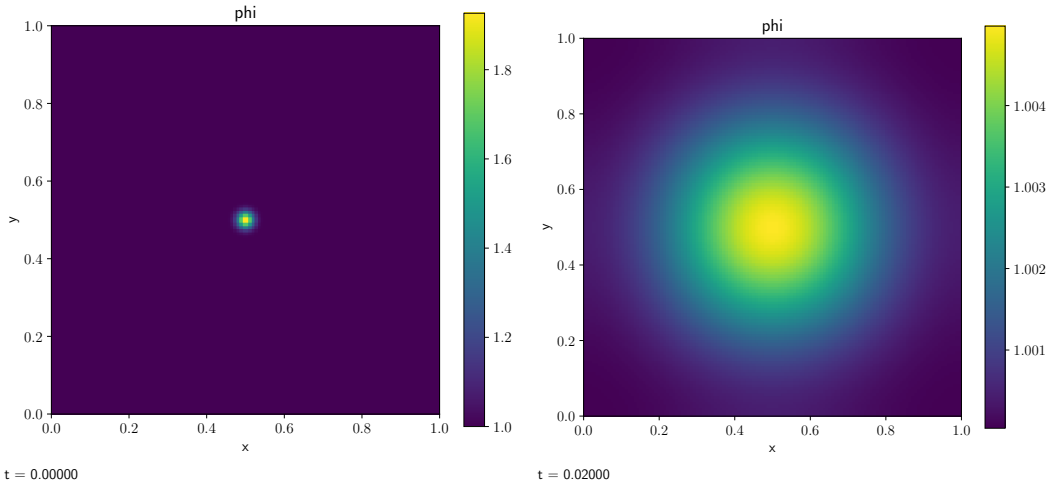

Figure 10.6: Diffusion of a Gaussian in 2-d with 128^2 zones using $k = 1.0$ and $\mathcal{C} = 2.0$. This used Crank-Nicolson time-discretization and multigrid. This can be run in pyro as `./pyro diffusion gaussian inputs.diffusion`.

Figure 10.7 shows the result of diffusing a Gaussian profile on a 2-d grid using Crank-Nicolson time-discretization, a constant coefficient, and multigrid to solve the discretized system, using pyro. There is a good agreement between the analytic and numerical solution, showing that this scheme models the diffusion of an initially resolved Gaussian well.

Exercise 10.4

The diffusion solver in pyro *uses Crank-Nicolson differencing in time. Modify the solver to do first-order backward Euler. This will change the form of the linear system (coefficients and righthand side), but should not require any changes to the multigrid solver itself.*
Compare the solution with the Crank-Nicolson solution for a very coarsely-resolved Gaussian and a finely-resolved Gaussian.

10.4.1 Convergence

One needs to be careful with the Crank-Nicolson discretization. If the initial data is under-resolved and you are taking a big timestep ($C \gg 1$), then the method can be unstable. Figure 10.8 shows such a case for the Gaussian diffusion with Crank-Nicolson discretization and 64 zones with $C = 10$. For this reason, simulation codes often drop down to the simple backwards difference time-discretization for implicit diffusion of under-resolved flows. A good discussion of the different types of stability can be found in [47].

Figure 10.9 shows the convergence of several of the methods discussed here on the

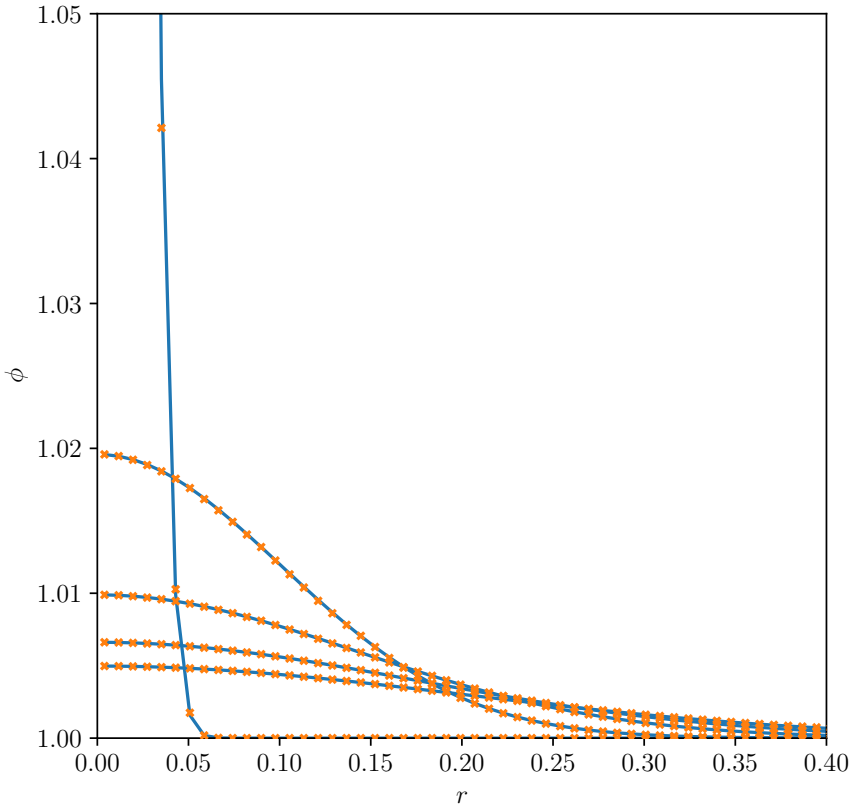

Figure 10.7: A comparison of 2-d implicit diffusion from Figure 10.6 with the analytic solution at several times. The 2-d solution was averaged over angles to yield a profile as a function of radius, using the `gauss_diffusion_compare.py` in pyro.

diffusion of a Gaussian, for two different Courant numbers, 0.8 and 8.0. For the lower case, we can do the diffusion explicitly as well as implicitly, and we see that all methods show the same trends, with the Crank-Nicolson method being the most accurate (since it is the only second-order-in-time method shown). For the larger timestep, we only run the implicit methods (the explicit case is unstable). We see that at coarse resolution, the errors in the methods are similar—this is because the solution is unresolved. At higher resolution, the error for the Crank-Nicolson method is an order of magnitude lower.

10.5 Non-constant Conductivity

For a non-constant conductivity, our equation has the form:

$$\frac{\partial \phi}{\partial t} = \frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) \tag{10.36}$$

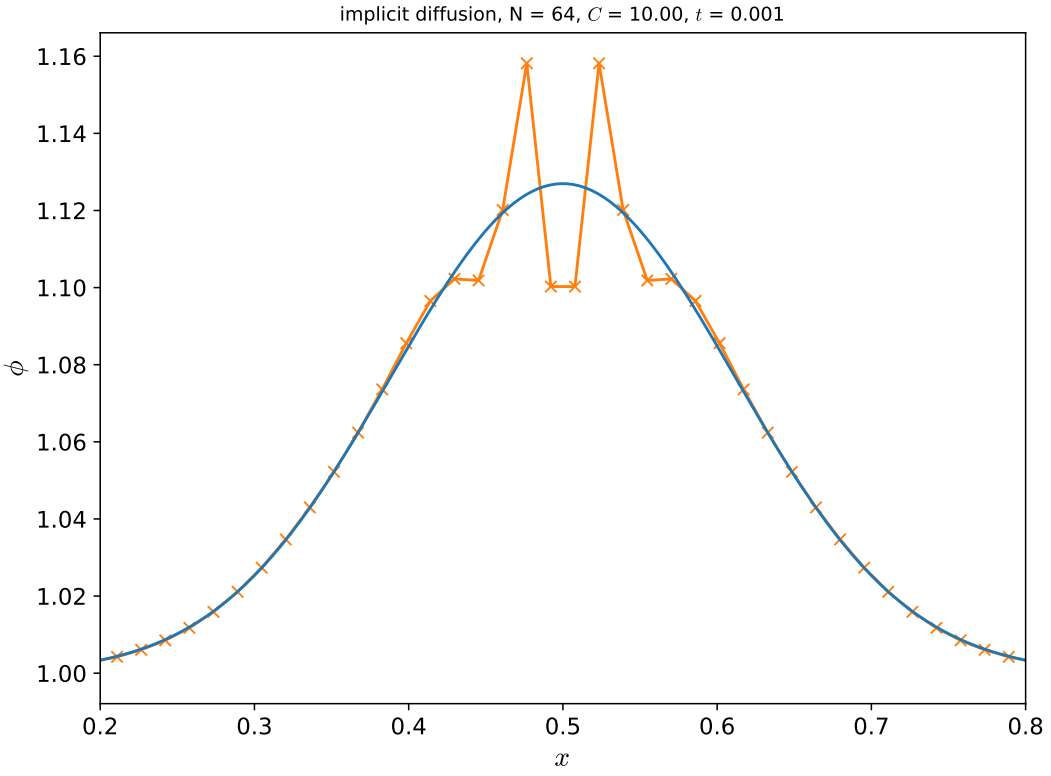

Figure 10.8: Crank-Nicolson diffusion of a Gaussian on under-resolved initial data with a large timestep. Here we use 64 zones and $C = 10$.
hydro_examples: `diffusion_implicit.py`

For the case where $k = k(x)$, we discretize as:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{\{k\nabla\phi\}_{i+1/2} - \{k\nabla\phi\}_{i-1/2}}{\Delta x} \tag{10.37}$$

Here we need the values of k at the interfaces, $k_{i-1/2}$ and $k_{i+1/2}$. We can get these from the cell-centered values in a variety of ways including straight-averaging:

$$k_{i+1/2} = \frac{1}{2}(k_i + k_{i+1}) \tag{10.38}$$

or averaging the inverses:

$$\frac{1}{k_{i+1/2}} = \frac{1}{2}\left(\frac{1}{k_i} + \frac{1}{k_{i+1}}\right) \tag{10.39}$$

The actual form should be motivated by the physics

Slightly more complicated are state-dependent transport coefficients—the transport coefficients themselves depend on the quantity being diffused:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla\cdot[k(\phi^n)\nabla\phi^n]_i + \nabla\cdot[k(\phi^{n+1})\nabla\phi^{n+1}]_i\right\} \tag{10.40}$$

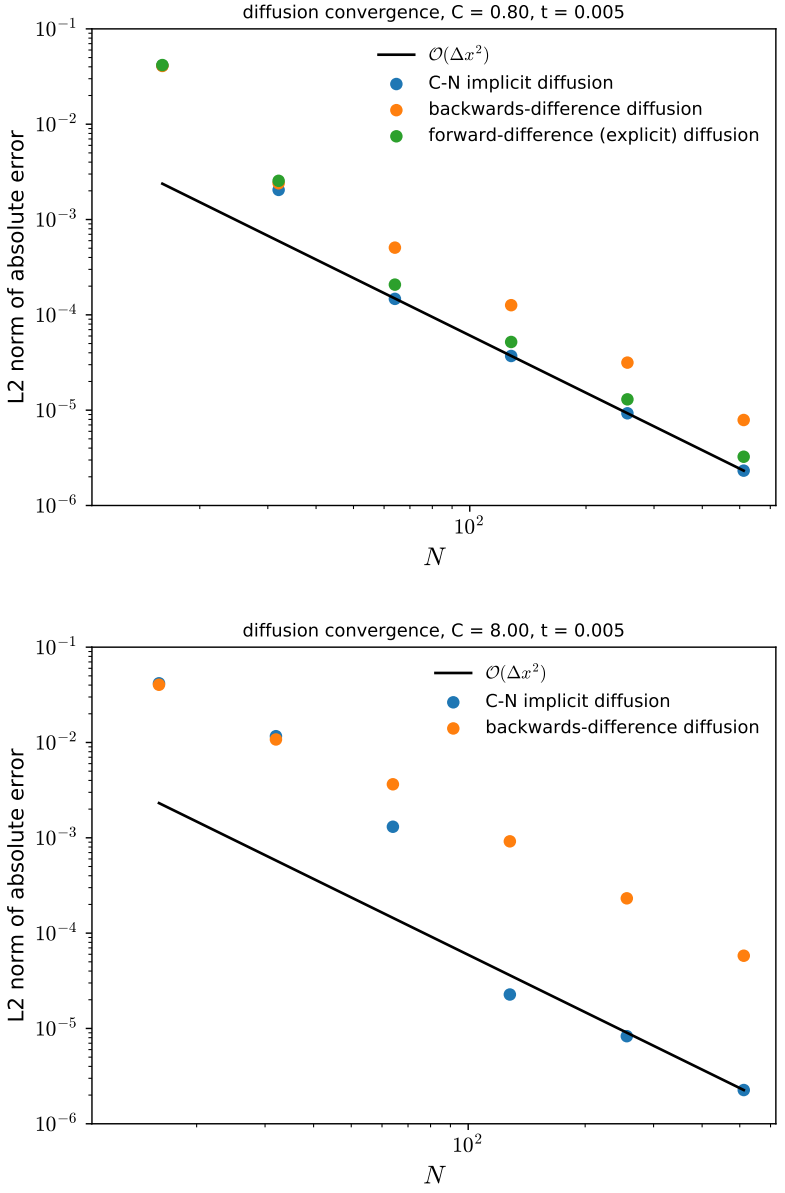

Figure 10.9: Convergence of the explicit, backward-difference, and Crank-Nicolson diffusion methods for $C = 0.8$ (top) and $C = 8.0$ (bottom). For the latter case, the explicit method is not valid and is not shown. We see that the Crank-Nicolson method has the lowest error, and when resolving the data, has second-order convergence.
hydro_examples: `diff_converge.py`

(for example, with thermal diffusion, the conductivity can be temperature dependent). In this case, we can achieve second-order accuracy by doing a predictor-corrector. First we diffuse with the transport coefficients evaluated at the old time,

giving a provisional state, $\phi^\star$:

$$\frac{\phi_i^\star - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla\cdot[k(\phi^n)\nabla\phi^n] + \nabla\cdot[k(\phi^n)\nabla\phi^\star]\right\} \tag{10.41}$$

Then we redo the diffusion, evaluating k with $\phi^\star$ to center the righthand side in time, giving the new state, ϕ^{n+1}:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla\cdot[k(\phi^n)\nabla\phi^n] + \nabla\cdot[k(\phi^\star)\nabla\phi^{n+1}]\right\} \tag{10.42}$$

This is the approach used, for example, in [13].

10.6 Diffusion in Hydrodynamics

Often we find diffusion represented as one of many physical processes in a single equation. For example, consider the internal energy equation with both reactions and diffusion:

$$\rho\frac{\partial e}{\partial t} + \rho U\cdot\nabla e + p\nabla\cdot U = \nabla\cdot k\nabla T + \rho S \tag{10.43}$$

This can be solved via an explicit-implicit discretization. First the advection terms are computed as:

$$A = \rho U\cdot\nabla e + p\nabla\cdot U \tag{10.44}$$

Then the advective-diffusive part is solved implicitly. Expressing $e = e(\rho, T)$, and using the chain rule,

$$\nabla e = e_T\nabla T + e_\rho\nabla\rho \tag{10.45}$$

where $e_T = \partial e/\partial T|_\rho \equiv c_v$ is the specific heat at constant volume and $e_\rho = \partial e/\partial\rho|_T$. Rewriting, we have:

$$\nabla T = (\nabla e - e_\rho\nabla\rho)/c_v \tag{10.46}$$

and then

$$\rho\frac{\partial e}{\partial t} = \nabla\cdot(k/c_v)\nabla e - \nabla\cdot(ke_\rho/c_v)\nabla\rho - A + \rho S \tag{10.47}$$

This is now a diffusion equation for e, which can be solved by the techniques described above. Note: if the transport coefficients (including c_v, e_ρ) are dependent on e, then we still need to do a predictor-corrector method here. This is discussed, for example, in [13, 48]. A simple case for this type of advection-diffusion is also shown in § 11.2.

Part IV

Multiphysics applications

Chapter 11

Model Multiphysics Problems

11.1 Integrating Multiphysics

Consider a system whose evolution depends on several different physical processes, represented by the operators A, D, R (advection, diffusion, and reactions, respectively).

$$\phi_t = -A(\phi) + D(\phi) + R(\phi) \tag{11.1}$$

One way to solve this system is to discretize each of the operators in space. For instance the discrete advection operator, $[A(\phi)]_i$ might use the ideas on piecewise linear reconstruction techniques discussed in chapter 4, $[D(\phi)]_i$ can use the discrete form of the Laplacian from chapter 10, and $[R(\phi)]_i$ may be an algebraic relation. This leaves us with an ordinary differential equation for the time-evolution of ϕ,

$$\frac{d\phi_i}{dt} = -[A(\phi)]_i + [D(\phi)]_i + [R(\phi)]_i \tag{11.2}$$

which can be solve using standard ODE techniques. This is the *method of lines* technique we saw with advection (§ 5.3) and compressible hydrodynamics (§ 8.10), and can be a powerful technique to solve PDEs or systems of PDEs with multiple physics operators

A difficulty arises if these processes each have different timescales associated with them. For instance, reactions may be vigorous and require a small timestep to accurately capture the changes, but the advection is slow. Or, recall that the timestep limiter for explicit diffusion scales as Δx^2 while explicit advection scales as Δx, so these processes could demand very different timescale for evolution. Therefore, we don't want to use the same timestep for all the processes, since that will needlessly make things computationally expensive. *Operator splitting* solves for the effects of each operator separately, using whichever timestep (and time-discretization, e.g., explicit or implicit) is most suited to the operation. The result of one operation is used

as the input to the next*. The downside of this approach is that the operations may not be well coupled.

11.2 Ex: diffusion-reaction

Consider a diffusion-reaction equation:

$$\phi_t = \kappa \phi_{xx} + \frac{1}{\tau} R(\phi) \tag{11.3}$$

This can be thought of as a simple model for a combustion flame, and can propagate a front. It is often the case that the reactions are stiff, and require a smaller timestep then the diffusion part. In fact, we may want to use an implicit integration method designed for stiff ODEs for the reaction part, but use a standard explicit method for the diffusion part. This requires operator splitting.

We can use Strang splitting [77] to make the integration second-order accurate overall:

$$\phi^{n+1} = R_{\Delta t/2} D_{\Delta t} R_{\Delta t/2} \phi^n \tag{11.4}$$

where $R_{\Delta t/2}$ represents reacting for a step of $\Delta t/2$ and $D_{\Delta t}$ represents diffusing for a step of Δt. In each case, these operators act as if the other were not present, but they see the effect of the previous operation on the input ϕ. Note that no explicit source terms describing one process appear in the other process's update. The procedure for updating appears as:

1. *Evolve reaction ODE system for $\Delta t/2$*

   Define $\phi^\star$ as the the solution to the ODE:

   $$\frac{d\phi}{dt} = \frac{1}{\tau} R(\phi), \quad \phi(t^n) = \phi^n, \quad t \in [t^n, t^{n+1/2}] \tag{11.5}$$

2. *Solve the diffusion equation for Δt with an implicit Crank-Nicolson discretization*

   $$\frac{\phi^{\star\star} - \phi^\star}{\Delta t} = \frac{1}{2}(D(\phi^\star) + D(\phi^{\star\star})) \tag{11.6}$$

   Note that the starting point is $\phi^\star$.

3. *Evolve reaction ODE system for $\Delta t/2$*

   Define ϕ^{n+1} as the the solution to the ODE:

   $$\text{define } \phi^{n+1} : \frac{d\phi}{dt} = \frac{1}{\tau} R(\phi), \quad \phi(t^{n+1/2}) = \phi^{\star\star}, \quad t \in [t^{n+1/2}, t^{n+1}] \tag{11.7}$$

*This directly parallels the dimensional splitting approach we saw for advection in § 5.4.1

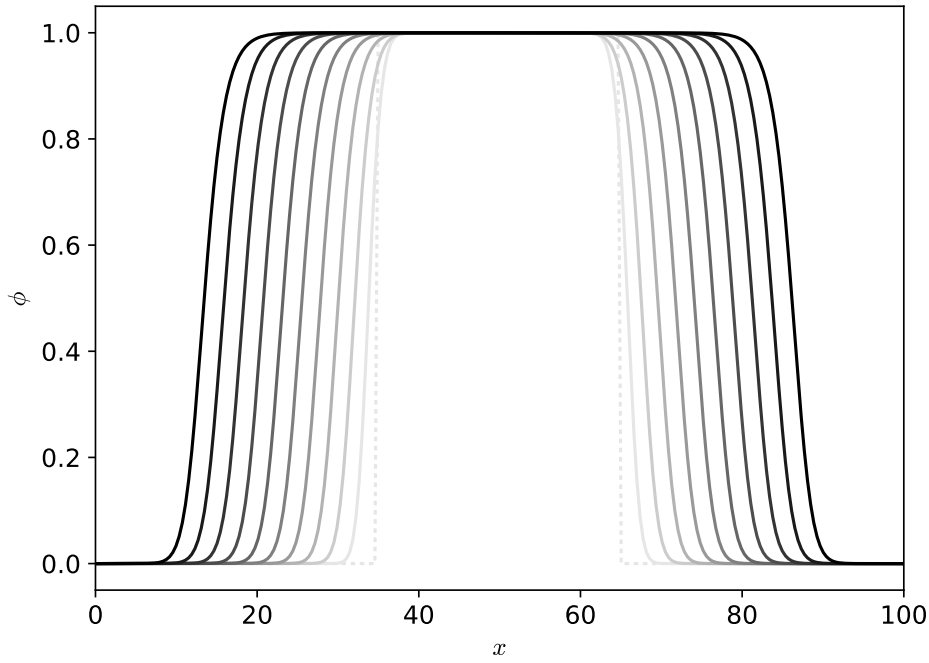

Figure 11.1: Solution to the diffusion-reaction equation with 256 zones, and $\kappa = 0.1$, $\tau = 1.0$. The lines shown are spaced 8.0 time-units apart. We see the initial smoothed tophat profile giving rise to a traveling front.
hydro_examples: `diffusion-reaction.py`

Consider a simple reaction source

$$R(\phi) = \frac{1}{4}\phi(1-\phi) \tag{11.8}$$

This is called a KPP reaction source. Here ϕ can be thought of as a progress variable that varies between pure ash ($\phi = 0$) and pure fuel ($\phi = 1$). Figure 11.1 shows the solution to our diffusion-reaction equation with 256 zones, $\kappa = 0.1$, $\tau = 1.0$ at several times.

The solution in this case is a wave with speed $S = \sqrt{\kappa/\tau}$ and thickness $\delta = \sqrt{\kappa\tau}$ (see [84] for some details of this system).

Exercise 11.1

Solve the the difusion reaction equation with the source given in Eq. **??**. *Note that you should begin with some smooth initial conditions—if they are too sharp than the C-N discretization will cause jagged features to appear. Compare your results to Figure 11.1.*

11.3 Ex: advection-diffusion

The viscous Burgers' equation appears as:

$$u_t + uu_x = \epsilon u_{xx} \tag{11.9}$$

This admits shocks and rarefactions just like the inviscid form we studied in Chapter 6, but now the viscosity can act to smooth out the shock—instead of being infinitely thin, it will have a physical width.

As we saw earlier, there are efficient, accurate methods for handling the advective parts explicitly, but for diffusion, we often want to solve it implicitly. We can split the solution up, but couple the two processes together to make a method that is overall second-order accurate in time. We write our equation as:

$$u_t + A(u) = D(u) \tag{11.10}$$

with $A(u) = [\frac{1}{2}u^2]_x$ and $D(u) = \epsilon u_{xx}$. Then our update appears in two steps.

1. *Find the advective update over the timestep*: We use an approximation of the diffusion term at time-level n, $D(u^n)$ as a source in the construction of the interface states for the advective part. Once the interface states, $u_{i+1/2}^{n+1/2}$ are known, we construct the advective update term as:

$$A_i^{n+1/2} = \frac{\left[\frac{1}{2}\left(u_{i+1/2}^{n+1/2}\right)^2\right] - \left[\frac{1}{2}\left(u_{i-1/2}^{n+1/2}\right)^2\right]}{\Delta x} \tag{11.11}$$

2. *Solve the diffusion equation with the advective source*: We use a Crank-Nicolson discretization of the diffusion part of our equation, with the advective update term appearing as a source.

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}D(u^n) + \frac{1}{2}D(u^{n+1}) - A^{n+1/2} \tag{11.12}$$

   This is a linear system that can be solved as a tridiagonal matrix or with multigrid. The result of this step is that u^{n+1} is updated with both the advection and diffusion terms.

Because the diffusion is done implicitly, the timestep constraint (for stability) for this solve is due to the advective portion only.

For step 1, the addition of the explicit diffusion source requires a small change to the method we used to predict the interface states.

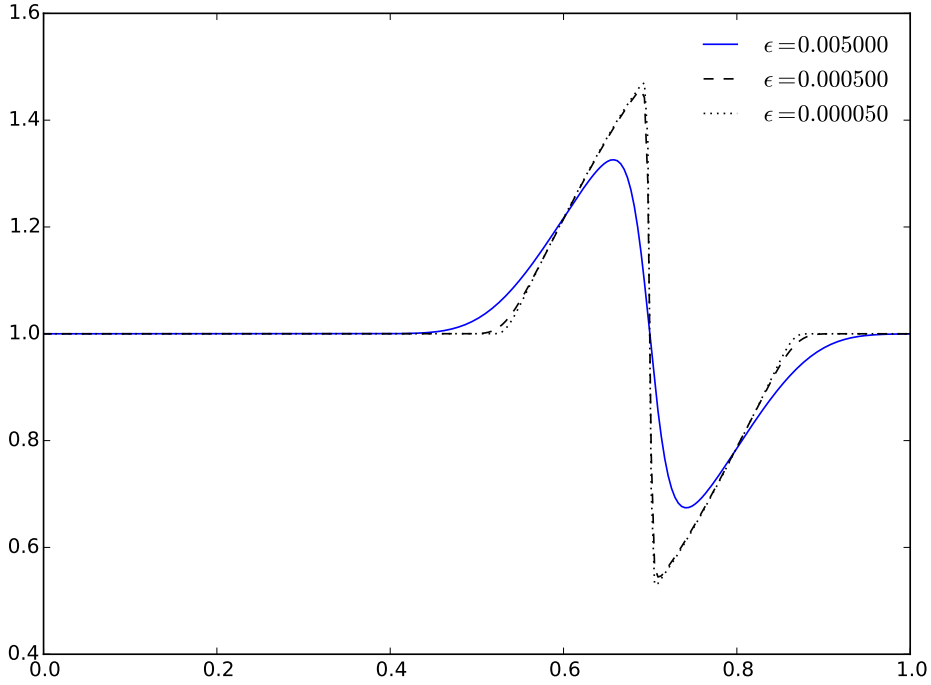

Figure 11.2: Solution to the viscous Burgers' equation with a variety of different viscosities. The initial conditions was a single wavelength of a sine wave for $x \in [1/3, 2/3]$, and $u = 1$ otherwise.
hydro_examples: `burgersvisc.py`

$$u^{n+1}_{i+1/2,L} = u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\frac{\partial u}{\partial t} + \ldots \tag{11.13}$$

$$= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left(-u_i\frac{\partial u}{\partial x} + D(u^n_i)\right) + \ldots \tag{11.14}$$

$$= u^n_i + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_i\right)\frac{\partial u}{\partial x} + \frac{\Delta t}{2}D(u^n) + \ldots \tag{11.15}$$

here the source term (shown in red) incorporates the effects of the diffusion on the prediction of the states for advection. This entered into our states when we replaced $\partial u/\partial t$ with our PDE equation. The spatial derivative, $\partial u/\partial x$ is replaced by a monotonized difference, and the method then proceeds as with the regular Burgers' equation. The Riemann problem is unchanged from the inviscid case.

Figure 11.2 shows the solution of the viscous Burgers' equation for shock initial conditions with different amounts of viscosity. Notice that the effect of the viscosity is to smooth the shock profile, but the shock position itself agrees between the cases.

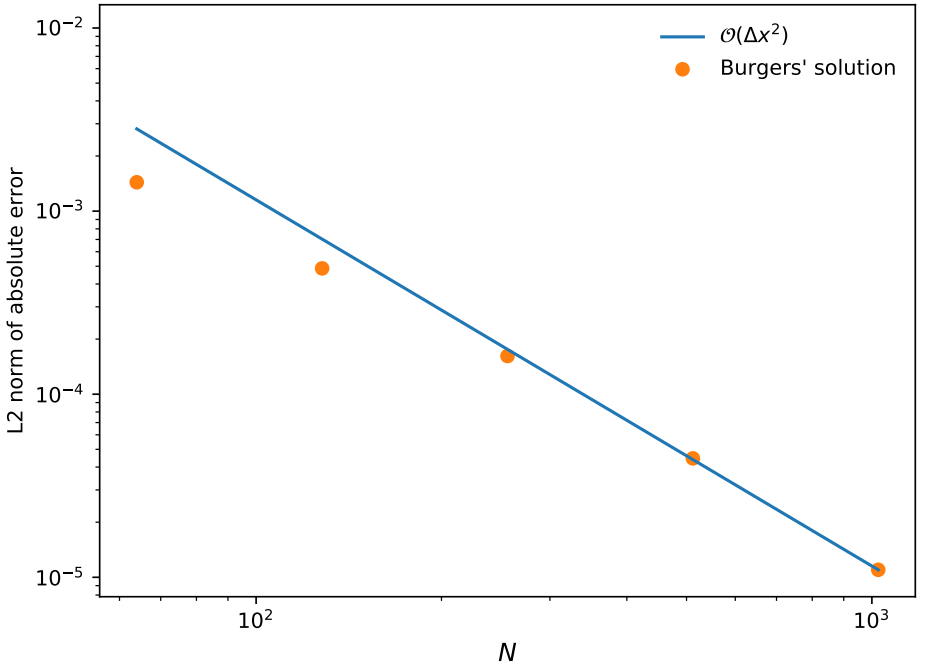

Figure 11.3: Convergence of the Burgers' solution with $\epsilon = 0.005$. To test convergence a run with 4096 zones was done as a benchmark solution. We see second-order convergence for the higher-resolution runs in this sample.
hydro_examples: `burgersvisc_converge.py`

11.3.1 Convergence without an analytic solution

Assessing the convergence of this is more difficult than the tests we looked at before, since there is no analytic solution to compare to. In this case, we can compare a very high-resolution solution and take this to be the *right* answer / benchmark solution, and then compare coarser simulations to a coarsened version of the benchmark solution. This is done in Figure 11.3. The fine zones of the benchmark are averaged into a corresponding coarse grid zone to produce a coarsened representation of the benchmark, and the norm of the error with the coarse simulation is computed.

Chapter 12

Reactive Flow

12.1 Introduction

Many astrophysical problems involve modeling nuclear (or chemical) reactions coupled with hydrodynamics. Astrophysical reacting flows share a lot of similarities with terrestrial combustion, including both subsonic (deflagrations) and supersonic (detonations) burning fronts. Nice discussions of the physics of reacting flows can be found in, e.g., [59, 62]

For multifluid flows, the Euler equations are augmented with continuity equations for each of the nuclear (or chemical) species. If we denote the density of species k as ρ_k, then conservation of mass for each species individually implies:

$$\frac{\partial \rho_k}{\partial t} + \nabla \cdot (\rho_k \mathbf{U}) = 0 \tag{12.1}$$

Summing Eq. 12.1 over k gives us back the mass contunity equation, since $\sum_k \rho_k = \rho$. The ρ_k are sometimes called *partial densities*. Note that using both Eq. 12.1 and the normal mass continuity equation for ρ overspecifies the system.

It is often easier to define a mass fraction of species k, X_k, as

$$X_k = \frac{\rho_k}{\rho} \tag{12.2}$$

and it is easy to see that $\sum_k X_k = 1$, and

$$\frac{\partial (\rho X_k)}{\partial t} + \nabla \cdot (\rho X_k \mathbf{U}) = 0 \tag{12.3}$$

Exercise 12.1

Using the continuity equation, show that we can write Eq. 12.3 as an advection equation:

$$\frac{\partial X_k}{\partial t} + \mathbf{U} \cdot \nabla X_k = 0 \tag{12.4}$$

or

$$\frac{DX_k}{Dt} = 0 \tag{12.5}$$

This latter form says that as a fluid element advects with the flow, its composition does not change. However, reactions can turn one species into another, so for reacting flow, $DX_k/Dt \neq 0$. If we define the creation rate for species k as $\dot{\omega}_k$, then we have:

$$\frac{DX_k}{Dt} = \dot{\omega}_k \tag{12.6}$$

Since $\sum_k X_k = 1$, it must be that $\sum_k \dot{\omega}_k = 0$.

Reactions will also release (or consume) energy, due to the change in nuclear or chemical binding energy. This energy will be a source to the internal energy. In particular, our first law of thermodynamics now has a source:

$$Tds = de + pd\left(\frac{1}{\rho}\right) = q_{\text{react}} \tag{12.7}$$

where q_{react} is the specific energy release due to reactions (energy / mass). Following a fluid element, this takes the form:

$$T\frac{Ds}{Dt} = \frac{De}{Dt} + p\frac{D(1/\rho)}{Dt} = H_{\text{nuc}} \tag{12.8}$$

Where now H_{nuc} is the time-rate of energy release per unit mass (dq_{react}/dt).

An additional energy source that we often consider along with reactions is thermal diffusion. Fick's law says that the heat flux is proportional to ∇T, so we write:

$$F_{\text{cond}} = -k_{\text{th}} \nabla T \tag{12.9}$$

as the conductive heat flux. Here the '$-$' indicates that heat flows from hot to cold and k_{th} is the thermal conductivity. See § 10.1 for more discussion.

The conservative Euler equations with reactive and diffusive source terms appear as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{12.10}$$

$$\frac{\partial \rho X_k}{\partial t} + \nabla \cdot (\rho \mathbf{U} X_k) = \rho \dot{\omega}_k \tag{12.11}$$

$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p = 0 \tag{12.12}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{U} + p\mathbf{U}) = \nabla \cdot (k_{\text{th}} \nabla T) + \rho H_{\text{nuc}} \tag{12.13}$$

We'll write the heat sources simply as H,

$$H = H_{\rm nuc} + \frac{1}{\rho}\nabla \cdot k_{\rm th}\nabla T \tag{12.14}$$

In primitive form, the derivation of the pressure equation is a complex. Previously, we found the pressure evolution in the absence of sources by differentiating pressure along streamlines (Eq. 7.21):

$$\frac{Dp}{Dt} = \left.\frac{\partial p}{\partial \rho}\right|_s \frac{D\rho}{Dt} + \left.\frac{\partial p}{\partial s}\right|_\rho \frac{Ds}{Dt} \tag{12.15}$$

where, as before, we recognize that $\Gamma_1 \equiv \partial \log p/\partial \log \rho|_s$. Now, however, we have an entropy source, so $Ds/Dt \neq 0$. The Maxwell relations* tell us that

$$\partial p/\partial s|_\rho = \rho^2 \partial T/\partial \rho|_s \tag{12.16}$$

This gives us the form:

$$\frac{Dp}{Dt} = \frac{p}{\rho}\Gamma_1 \frac{D\rho}{Dt} + \rho^2 \left.\frac{\partial T}{\partial \rho}\right|_s \frac{H}{T} \tag{12.17}$$

We need an expression for $\partial T/\partial \rho|_s$ in terms of derivatives of ρ and T (since that is what our equations of state typically provide). Consider Γ_1, with $p = p(\rho, T)$:

$$\Gamma_1 = \frac{\rho}{p}\left.\frac{dp}{d\rho}\right|_s = \frac{\rho}{p}\left[p_\rho + p_T \left.\frac{dT}{d\rho}\right|_s\right] \tag{12.18}$$

where we use the shorthand $p_\rho \equiv \partial p/\partial \rho|_T$ and $p_T \equiv \partial p/\partial T|_\rho$. This tells us that

$$\left.\frac{\partial T}{\partial \rho}\right|_s = \frac{p}{\rho p_T}(\Gamma_1 - \chi_\rho) \tag{12.19}$$

where $\chi_\rho \equiv \partial \log p/\partial \log \rho|_T$. To further simplify this, we use some standard relations for general equations of state

$$\frac{\Gamma_1}{\chi_\rho} = \frac{c_p}{c_v} \tag{12.20}$$

and

$$c_p - c_v = \frac{p}{\rho T}\frac{\chi_T^2}{\chi_\rho} \tag{12.21}$$

with $\chi_T \equiv \partial \log p/\partial \log T|_\rho$ (see, e.g., [40] for both of these relations). These allow us to write

$$\left.\frac{\partial T}{\partial \rho}\right|_s = \frac{p\chi_T}{\rho^2 c_v} \tag{12.22}$$

*see, for instance, [73], for some discussion on these relations

Putting this all together, we have:

$$\frac{Dp}{Dt} = \frac{p}{\rho}\Gamma_1 \frac{D\rho}{Dt} + \frac{p\chi_T}{c_v T} H \tag{12.23}$$

and using the continuity equation, $D\rho/Dt = -\rho\nabla \cdot \mathbf{U}$, we finally have:

$$\frac{\partial p}{\partial t} + \mathbf{U} \cdot \nabla p + \Gamma_1 p \nabla \cdot \mathbf{U} = \Gamma_1 p \sigma H \tag{12.24}$$

where we defined

$$\sigma = \frac{\partial p/\partial T|_\rho}{\rho c_p \partial p/\partial \rho|_T} \tag{12.25}$$

The σ notation follows from [6].

The primitive variable is then

$$\frac{\partial \rho}{\partial t} + \mathbf{U} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{U} = 0 \tag{12.26}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \frac{1}{\rho}\nabla p = 0 \tag{12.27}$$

$$\frac{\partial p}{\partial t} + \mathbf{U} \cdot \nabla p + \Gamma_1 p \nabla \cdot \mathbf{U} = \Gamma_1 p \sigma H \tag{12.28}$$

$$\frac{\partial X_k}{\partial t} + \mathbf{U} \cdot \nabla X_k = \dot{\omega}_k \tag{12.29}$$

Finally, we can derive a temperature evolution equation following the method from § 7.3. We find

$$c_v \frac{DT}{Dt} = \left(\frac{\partial e}{\partial \rho}\bigg|_T - \frac{p}{\rho} \right) \nabla \cdot \mathbf{U} + H \tag{12.30}$$

or

$$c_p \frac{DT}{Dt} = \left(\frac{1}{\rho} - \frac{\partial h}{\partial p}\bigg|_T \right) \frac{Dp}{Dt} + H \tag{12.31}$$

depending on whether we start with e or h.

12.2 Operator splitting approach

Using operator splitting, we separate the equations into hydrodynamics and reactive parts and we do these operations in turn. Following the ideas from § 11.2, we perform the update in stages. If we denote the reaction update operator as $R_{\Delta t}$ and the hydrodynamics operator as $A_{\Delta t}$, then our update appears as:

$$\boldsymbol{\mathcal{U}}^{n+1} = R_{\Delta t/2} A_{\Delta t} R_{\Delta t/2} \boldsymbol{\mathcal{U}}^n \tag{12.32}$$

To make it more explicit, we write out system as:

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \nabla \cdot \mathbf{F}(\boldsymbol{\mathcal{U}}) = \boldsymbol{\mathcal{R}}(\boldsymbol{\mathcal{U}}) \tag{12.33}$$

where $\mathcal{R} = (0, \dot{\omega}_k, 0, \rho H_{\rm nuc})^\intercal$ is the vector of reactive sources. The advance through Δt in time is:

1. *Evolve reaction part for $\Delta t/2$.*

   We define $\mathcal{U}_i^\star$ as the result of integrating

   $$\frac{d\mathcal{U}_i}{dt} = \mathcal{R}(\mathcal{U}_i) \tag{12.34}$$

   over $t \in [t^n, t^{n+1/2}]$ with initial conditions, $\mathcal{U}_i(t^n) = \mathcal{U}_i^n$.

2. *Solve the hydrodynamics portion for Δt, beginning with the state $\mathcal{U}^\star$*

   $$\frac{\mathcal{U}_i^{\star\star} - \mathcal{U}_i^\star}{\Delta t} = \frac{\mathbf{F}^{(x)}(\mathcal{U}_{i-1/2}^{\star,n+1/2}) - \mathbf{F}^{(x)}(\mathcal{U}_{i+1/2}^{\star,n+1/2})}{\Delta x} \tag{12.35}$$

   where $\mathcal{U}_{i-1/2}^{\star,n+1/2}$ is the predict interface state (using the ideas from § 8.2), beginning with the state $\mathcal{U}^\star$. Note that $\mathcal{R}$ does not appear here explicitly.

   Alternately, we could use the MOL hydrodynamics update here, but the basic idea will be the same—the hydrodynamics does not explicitly see the reaction terms.

3. *Evolve reaction part for $\Delta t/2$*

   We reach the final update, $\mathcal{U}^{n+1}$, by integrating

   $$\frac{d\mathcal{U}}{dt} = \mathcal{R}(\mathcal{U}) \tag{12.36}$$

   over $t \in [t^{1/2}, t^{n+1}]$ with initial conditions $\mathcal{U}(t^{n+1/2}) = \mathcal{U}^{\star\star}$.

12.2.1 Adding species to hydrodynamics

In the hydrodynamics algorithm, we neglect the diffusion and reaction terms. Their effects are incorporated implicitly by performing the hydrodynamics update starting with the state that has already experienced burning.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{12.37}$$

$$\frac{\partial (\rho X_k)}{\partial t} + \nabla \cdot (\rho \mathbf{U} X_k) = 0 \tag{12.38}$$

$$\frac{\partial (\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p = 0 \tag{12.39}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{U} + p\mathbf{U}) = 0 \tag{12.40}$$

When we now consider our primitive variables: $\mathbf{q} = (\rho, u, p, X_k)$, we find

$$\mathbf{A}(\mathbf{q}) = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 1/\rho & 0 \\ 0 & \Gamma_1 p & u & 0 \\ 0 & 0 & 0 & u \end{pmatrix} \tag{12.41}$$

There are now 4 eigenvalues, with the new one also being simply u. This says that the species simply advect with the flow. The right eigenvectors are now:

$$\mathbf{r}^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \\ 0 \end{pmatrix} \qquad \mathbf{r}^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{r}^{(\circ,X)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathbf{r}^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \\ 0 \end{pmatrix} \tag{12.42}$$

corresponding to $\lambda^{(-)} = u - c$, $\lambda^{(\circ)} = u$, $\lambda^{(\circ,X)} = u$, and $\lambda^{(+)} = u + c$. We see that for the species, the only non-zero element is for one of the u eigenvectors ($\mathbf{r}^{(\circ,X)}$). This means that X_k only jumps over this middle wave. In the Riemann solver then, there is no 'star' state for the species, it just jumps across the contact wave.

To add species into the solver, you simply need to reconstruct X_k as described in Chapter 8 to find the interface values. If we are doing characteristic tracing, then we use this new $\mathbf{A}(\mathbf{q})$ and associated eigenvectors. We then solve the Riemann problem, with X_k on the interface taken as simply the left or right state depending on the sign of the contact wave speed, and do the conservative update for ρX_k using the species flux.

One issue that can arise with species is that even if $\sum_k X_k = 1$ initially, after the update, that may no longer be true. There are a variety of ways to handle this:

- You can update the species, (ρX_k) to the new time and then define the density to be $\rho = \sum_k (\rho X_k)$—this means that you are not relying on the value of the density from the mass continuity equation itself.

- You can force the interface states of X_k to sum to 1. Because the limiting is non-linear, this is where problems can arise. If the interface values of X_k are forced to sum to 1 (by renormalizing), then the updated cell-centered value of X_k will as well. This is the approach discussed in [63].

- You can design the limiting procedure to preserve the summation property. This approach is sometimes taken in the combustion field. For piecewise linear reconstruction, this can be obtained by computing the limited slopes of all the species, and taking the most restrictive slope and applying this same slope to all the species.

12.2.2 Integrating the reaction network

The reactive portion of our operator split system is:

$$\frac{dX_k}{dt} = \dot{\omega}_k \tag{12.43}$$

$$\frac{d(\rho e)}{dt} = \rho H_{\rm nuc} \tag{12.44}$$

As written, this system is not closed without the equation of state, since $\dot{\omega}_k = \dot{\omega}_k(\rho, T, X_k)$, and likewise for $H_{\rm nuc}$. Our temperature evolution equation reduces to

$$c_x \frac{dT}{dt} = H_{\rm nuc} \tag{12.45}$$

where c_x is c_v or c_p, depending on the physical effects we are trying to capture. This difference only arises because we are neglecting the hydrodynamic portions of the temperature evolution during the reaction step.

Many simulation codes neglect the evolution of temperature during the reaction step (see, e.g., [34]).

An excellent introduction to integrating astrophysical nuclear reaction networks is found in [79]. The main issue with reaction networks is that they tend to be stiff, which means that implicit integration techniques are needed. There are a number of well-tested, freely available implicit ODE integrators (e.g. [18]).

For stiff systems, you need to supply both a righthand side routine and a Jacobian routine[†].

12.2.3 Incorporating explicit diffusion

12.3 Burning modes

12.3.1 Convective burning

12.3.2 Deflagrations

A deflagration or flame is a subsonic burning front that propagates via a balance of diffusion and reactions. Thermal diffusion raises the temperature of fuel ahead of the flame to ignition and the energy release heats the ash, keeping the thermal gradient needed for diffusion. In terrestrial flames, species diffusion can be important as well, but this is usually negligible in astrophysical flames.

Because the burning front is subsonic, sound waves can communicate across the flame and therefore the fuel and ash are at the same pressure. The temperature jump

[†]although this could be calculated via finite-differences if needed

across the flame means that the density will drop behind the flame to preserve the constant pressure. Our temperature evolution equation, taking $Dp/Dt = 0$, is

$$\rho c_p \frac{DT}{Dt} = \nabla \cdot k_{\text{th}} \nabla T + \rho H_{\text{nuc}} \tag{12.46}$$

We can estimate the timescale for diffusion by neglecting the reactions and writing

$$\frac{DT}{Dt} = \frac{1}{\rho c_p} \nabla \cdot k_{\text{th}} \nabla T \approx \mathcal{D} \nabla^2 T \tag{12.47}$$

where $\mathcal{D} = k_{\text{th}}/(\rho c_p)$ is the diffusion coefficient (assuming ρc_p is spatially constant). Dimensional analysis suggests that the diffusion timescale is then

$$t_{\text{diff}} \approx \frac{\delta^2}{\mathcal{D}} \tag{12.48}$$

where δ is the width of the region where diffusion takes place—we'll identify this as the flame thickness.

Similarly, we can estimate the reaction timescale by considering

$$\rho c_p \frac{DT}{Dt} = \rho H_{\text{nuc}} \tag{12.49}$$

and writing

$$t_{\text{burn}} \approx \frac{c_p T}{H_{\text{nuc}}} \tag{12.50}$$

We can alternately expand H_{nuc} in terms of temperature if we wish to capture some of the temperature sensitivity.

Equating these timescales gives

$$\delta \sim \left(\frac{k_{\text{th}} T}{\rho H_{\text{nuc}}} \right)^{1/2} \tag{12.51}$$

which is an estimate of the flame thickness. We can define the flame speed as the time it takes to burn through this thickness:

$$v_f \sim \frac{\delta}{t_{\text{burn}}} \sim \left(\frac{k_{\text{th}} H_{\text{nuc}}}{\rho c_p T} \right)^{1/2} \tag{12.52}$$

We note the main dependences are:

$$v_f \sim \sqrt{k_{\text{th}} H_{\text{nuc}}} \qquad \delta \sim \sqrt{\frac{k_{\text{th}}}{H_{\text{nuc}}}} \tag{12.53}$$

12.3.3 Detonations

Chapter 13

Planning a Simulation

13.1 How to setup a simulation?

When you perform a simulation, you are making a series of approximations. The first comes with the choice of equation set. As discussed earlier, for pure hydrodynamics, this often means solving the Euler rather than the full Navier-Stokes equations. The next approximation is made when discretizing the equations, using the methods we described in the earlier chapters.

Astrophysical flows involve a range of length and timescales. Often it is not possible to capture all these scales in a single simulation, so some comprimises must be made. If you are interested in the dynamics on the full scale of the system, L, then you will resolve down to some cutoff scale l, where $L/l \sim O(10^3\text{–}10^4)$. At scales smaller than l, you neglect the flow features. The smallest scale of importance is the disipation scale $\lambda \ll L$.

If you are not resolving down to the dissipation scales in our problem, then you are working in the realm of *large eddy simulations* (LES; see [67]). Proper LES requires a subgrid model to treat the unresolve scales and their feedback onto your grid. If you instead rely on numerical diffusion of your method to do the dissipation, you are doing *implicit large eddy simulation* or ILES—for many flows, this be sufficient to capture the dynamics (including turbulence) that are important [8, 49].

13.2 Dimensionality and picking your resolution

Nature is three-dimensional, so realistic simulaitons generally require full 3-d modeling. Nevertheless, 2-d simulations can be useful to scope out the problem space, and in particular, to get a feel for the timescales and resolution needed to resolve

smallscale features. The major place where dimensionality affects results is for instabilities and turbulence. The trend in 2-d is for small scale vortices to merge and build larger and larger structures (see, e.g., [16]), while in 3-d motions at the largest scale cascade down to smaller scales, driving turbulent flow down to the smallest scales.

Turbulence can be very important in astrophysical flows, and properly capturing it requires modeling a large range of lengthscales (physically, the objective is to get a large *Reynolds number*: the ratio of the advective force to the dissipative force). Generally, you'll just start to get a resolved inertial range over a decade in wavenumber in your power spectrum at around 512 zones on a side of your domain (using the methods that we discussed here). With fewer points than this, you are unlikely to capture turbulence.

If nuclear burning is modeled, the steepness of the reaction rate will set a lengthscale. Burning can also wash out instabilities, a process sometimes called *fire polishing* [12, 80]. Here, if the timescale for reactions is faster than the growth rate of an instability, the instability is suppressed. This can set a small scale cutoff that is well above the viscous dissipation scale.

Hydrostatic equilibrium also puts demands on the resolution. Recall that the momentum equation is:

$$\rho \frac{D\mathbf{U}}{Dt} + \nabla p = \rho \mathbf{g} \tag{13.1}$$

If hydrostatic equilibrium, $\nabla p = \rho \mathbf{g}$, is not exactly satisfied, then the residual will appear as a force that generates an acceleration and non-zero velocities will build up. A good rule of thumb is that you need 10 points per pressure scale height to get a reasonable hydrostatic balance (this is discussed at bit in [87]).

Spatial resolution is often in contention with improved physics—both can add to the computational cost of a simulation, so for fixed cost, you have often have to choose between more physics (e.g., larger reaction network) or more resolution.

An important part of the simulation process is to ensure that your results are converged. In a *convergence study*, you run the same simulation at various resolutions and look to see that the major results (e.g., integral quantities) do not change significantly with resolution. Ideally you would show that any results you are interested are moving toward an asymptotic value. Even if you are not converged however, you could use the trends observed with resolution to help understand the possible error in your simulation.

13.3 Boundary conditions

Another approximation you make is how to treat the boundaries. Usually one chooses symmetry / reflecting, outflow / zero-gradient, inflow (specifying the value on the boundary), or periodic. In the case of symmetry, the fluid state is reflected

and the normal velocity is reflected with a sign change. The transverse velocity can be left as is (a slip wall) or set to zero at the boundary (no-slip wall).

Generally speaking, you should put your boundaries as far away from the place where the action is occuring as possible. For example, if you are modeling a convective region bounded by stably stratified flow, then you should have a large buffer of stable atmosphere bounding it.

Periodic boundary conditions don't magically fix everything either. With a triply-periodic box (periodic on all boundaries), then you are confining the flow. If energy is injected into the box, then there is no place for it to expand, and this can lead to artifical heating or compression.

Symmetry / reflecting boundaries provide a simple way to reduce a problem size, for example, modeling only an octant of a star in 3-d instead of the full star. But they also introduce artifacts, since they force the normal velocity to zero right on the boundary. This means that flow through the center of a star (e.g., if we model only an octant) can be restricted, resulting in artifical flow / convective features. Wall heating can also be an issue—forcing the normal velocity to zero at the symmetry boundary can lead to stagnation of the flow there, allowing hot spots to develop and not be transported away.

13.4 Timestep considerations

The timestep constraint we discussed (the CFL condition) is required for stability, but that is different than accuracy. In particular, accurately capturing some physical processes (that may appear as source terms) might require more restrictive timesteps to ensure that they are properly coupled with the hydrodynamics.

An example where this can commonly arise is reactive flows—combining hydrodynamics with nuclear or chemical reactions. If the reactions are very vigorous, then you may run into a situation where you exaust your fuel in a zone in a single step, without giving the hydrodynamics a chance to advect new fuel in. This represents a breakdown of the coupling of the hydrodynamics and reactions. A common way to improve the accuracy here is to monitor the change in mass fractions of a fuel and set a timestep limit such that:

$$\Delta t_{\rm react} = C_{\rm react} \frac{X_{\rm fuel}}{|\dot{\omega}_{\rm fuel}|} \tag{13.2}$$

where $C_{\rm react} < 1$. The species creation rate, $\dot{X}_{\rm react}$, is typically evaluated using the value from the previous step.

13.5 Convergence and multiphysics

For a full simulation, with hydrodynamics coupled to other physics, you should test your code to ensure that it is converging at the rate that you expect. Usually in this case there is no analytic solution, so you need to measure the convergence numerically. A standard way to do this is to run pairs of simulations that differ by a factor of two in resolution, we'll denote the coarser simulation with the subscript c and the finer simulation with the subscript f. The error is computed for a field ϕ by coarsening the finer simulation by a factor of two, and computing the norm of the difference:

$$\epsilon \equiv \|\phi_c - \mathcal{C}(\phi_f)\| \tag{13.3}$$

Here, we denote the coarsening operator as $\mathcal{C}(.)$. In this case, two simulations give us a single error. We then repeat this, taking the fine simulation from this comparison as the coarse for the next, and using an even finer simulation to generate a new error. We can then measure the convergence of a multiphysics simulation by comparing the errors from the two pairs of the simulations. This procedure was used in, e.g., [4, 52].

Ideally, a smooth problem should be used, since discontinuities by definition do not converge with resolution (and limiters, if used, would kick in there as well).

13.6 Computational cost

The main limitation to your choice of resolution and timestep is computational cost. For a three-dimensional simulation in a box with N zones on a side, the work scales as $O(N^4)$. The cost to advance a single timestep is based on the volume, N^3, and the number of timesteps to get you to a fixed simulation time, $t_{\rm max}$, is $t_{\rm max}/\Delta t$, but $\Delta t \sim \Delta x/s \sim L/(Ns)$, where s is the fastest information speed. So the number of timesteps increases with N, making the total cost N^4. Think about this for a minute—if you double the number of zones, then the amount of work you need to do increases by $16\times$.

A common conflict that arises is: Do you do one "hero" calculation or many smaller calculations to understand the sensitivity of your results to input parameters? Computing centers that grant allocations usually setup their queues to favor big jobs that use as much of the machine as possible (as that's what their funders use as a metric for their success).

It is important to understand that no single algorithm will offer everything you need.

13.7 I/O

Another challenge with simulations is data storage. For a calculation on a 1024^3 grid, using double precision, each number you store takes 8 bytes / zone, so a single

variable on the grid will require 8 GB. For compressible hydro, you will have atleast 5 variables (in 3-d), ρ, $(\rho \mathbf{U})$, and (ρE), so this is a minimum of 40 GB per file. Often you will want some derived quantities (like temperature) and composition quantities in your output, which will greatly increase your storage requirements.

For this reason, simulation codes often have different types of output. Checkpoint files store all the variables that are needed to restart the calculation from where you left off—these should be stored in a (portable) binary format at machine precision. You don't need to keep these around forever, perhaps only saving the last few as necessary to restart your calculation as it works through a computing center's queue. Plotfiles store a lot of variables—basically anything that will be needed for the analysis of the simulation. Since you don't need these to restart, you can probably afford to store the variables at single precision. Sometimes you might have multiple levels of plotfiles, storing a few variables (like density) very frequently to allow for the production of smooth movies, and storing a lot of variables and derived quantities at a much longer cadence.

A perpetual challenge with the analysis and visualization of simulations is that sometimes you don't know what you are looking for until the simulation is complete, so you cannot simply do runtime visualization and not save the raw data as the simulation progresses. This leads to collections of plotfiles that can easily top 100 TB per simulation. Runtime diagnostics can help a little—if you know any global quantities that you will be interested in over the course of the simulation (like peak temperature, or total energy) you can compute these on the fly and simply store a single number per step (or every N steps) into a file.

Part V

Low Speed Hydrodynamics

Chapter 14

Incompressible Flow and Projection Methods

14.1 Incompressible flow

As a fluid parcel advects through a domain, it compresses and expands due to a variety of effects (stratification, local heat release, acoustic/shock waves). The Lagrangian derivative of the density captures the changes in the fluid, and is a measure of its compressibility. From the continuity equation, we see:

$$-\frac{1}{\rho}\frac{D\rho}{Dt} = \nabla \cdot \mathbf{U} \tag{14.1}$$

Note that for $\nabla \cdot \mathbf{U} > 0$, we have $-(1/\rho)(D\rho/Dt) > 0$, which means that ρ gets smaller—this is expansion of the Lagrangian fluid element.

A fluid in which the density (and therefore volume) of a fluid element is not allowed to change is called *incompressible*. An incompressible fluid obeys the velocity constraint:

$$\nabla \cdot \mathbf{U} = 0 \tag{14.2}$$

(since $D\rho/Dt = 0$). The incompressible fluid approximation is a reasonable approximation when the Mach number of the fluid is small ($\ll 1$). To complete the system, we add the momentum equation. If we take the density to be constant everywhere in the domain (not just in our fluid element), then we have:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \nabla p = 0 \tag{14.3}$$

Note that p here looks like a pressure, but it is not subject to any equation of state. This system is closed as written. The value of p is determined such that the velocity constraint is satisfied.

We can gain more insight into the applicability of the incompressible equations by doing an asymptotic expansion. Starting with the momentum equation from the Euler equations:

$$\frac{\partial(\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p = 0 \tag{14.4}$$

we can introduce dimensionless variables:

$$\bar{\mathbf{U}} = \frac{\mathbf{U}}{|\mathbf{U}_0|}; \quad \bar{t} = \frac{t}{t_0}; \quad \bar{\rho} = \frac{\rho}{\rho_0}; \quad \bar{x} = \frac{x}{L_0}; \quad \bar{p} = \frac{p}{\rho_0 c_0^2} \tag{14.5}$$

and we find the dimensionless evolution equation:

$$\frac{\partial(\bar{\rho}\bar{\mathbf{U}})}{\partial \bar{t}} + \bar{\nabla} \cdot (\bar{\rho}\bar{\mathbf{U}}\bar{\mathbf{U}}) + \frac{1}{M^2}\bar{\nabla}\bar{p} = 0 \tag{14.6}$$

Now we introduce an expansion of pressure in terms of Mach number:

$$p = p_0 + p_1 M + p_2 M^2 \tag{14.7}$$

Grouping terms by power of Mach number, we find:

$$\nabla p_0 = \nabla p_1 = 0 \tag{14.8}$$

and

$$\frac{\partial(\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p_2 = 0 \tag{14.9}$$

Only p_2, which is sometimes called the *dynamical pressure* appears in the dynamics. All of the thermodynamic content is in p_0, consistent with our argument previously that no equation of state is needed.

Recalling that pressure evolves according to

$$\frac{\partial p}{\partial t} + \gamma p \nabla \cdot \mathbf{U} + \mathbf{U} \cdot \nabla p = 0 \tag{14.10}$$

The general expression for the velocity divergence is:

$$\nabla \cdot U = -\frac{1}{\gamma p}\frac{Dp}{Dt} \tag{14.11}$$

from our pressure expansion, we see that

$$\nabla \cdot \mathbf{U} \sim \mathcal{O}(M^2) \tag{14.12}$$

This justifies our argument that the incompressible equations apply at low Mach number.

14.2 Projection methods

The basic idea behind a projection method is that any vector field can be decomposed into a divergence free part and the gradient of a scalar (this is sometimes called a *Hodge decomposition*). Given a velocity field $\mathbf{U}^\star$, we can express it in terms of the divergence free part $\mathbf{U}^d$ and a scalar, ϕ as:

$$\mathbf{U}^\star = \mathbf{U}^d + \nabla\phi \tag{14.13}$$

Taking the divergence of each side, and noting that $\nabla \cdot \mathbf{U}^d = 0$, we have

$$\nabla \cdot \mathbf{U}^\star = \nabla^2\phi \tag{14.14}$$

This is an elliptic equation. Given suitable boundary conditions, we can solve for ϕ (for instance, using multigrid) and recover the divergence free part of $\mathbf{U}^\star$ as:

$$\mathbf{U}^d = \mathbf{U}^\star - \nabla\phi \tag{14.15}$$

We call this operation of extracting the divergence free part of a velocity field a *projection*. This can be expressed by defining an operator P, such that $P\mathbf{U}^\star = \mathbf{U}^d$, and $(I - P)\mathbf{U}^\star = \nabla\phi$. From the momentum equation, we see that $\partial\mathbf{U}/\partial t + \nabla p$ is in the form of a divergence free term + the gradient of a scalar. This means that advancing the velocity field subject to the constraint involves solving:

$$\mathbf{U}_t = P(\mathbf{U}_t + \nabla p) = P(-\mathbf{U} \cdot \nabla\mathbf{U}) \tag{14.16}$$

See Bell, Colella, and Howell [10] for a nice discussion of this.

The original projection method for incompressible flows goes back to Chorin [21]. Instead of evolving Eq. 14.16 directly, we break the update into pieces. The basic idea is to evolve the velocity advection equation without regard to the constraint, yielding a *provisional velocity* field which is then subjected to a projection to enforce the divergence-free constraint. Bell, Colella, and Glaz (BCG) [9] introduced a projection method that uses standard Godunov methods for the advection terms (much like is done with compressible flow) and then solves an elliptic equation to enforce the constraint. This division of the operations in the algorithm (advect then project) is a type of *fractional step* method.

There are different ways to discretize the operators that make up the projection. We denote the discretized divergence as D and the discretized gradient operation as G. For an exact projection, the discretized Laplacian, L, would be the same as applying G and D in succession (i.e. $L = DG$). Depending on our data centerings, we may prefer to discretize the Laplacian independently to D and G, such that $L \neq DG$. This is called an *approximate projection*. Note that for an approximate projection, P is not idempotent: $P^2\mathbf{U} \neq P\mathbf{U}$.

Many variations on this basic idea exist, using alternate forms of the projection, different grid centerings of the ϕ variable, and additional physics.

Figure 14.1 shows an example of a projection. The initial velocity field is specified as:

$$u = -\sin(\pi x)^2 \sin(2\pi y) \tag{14.17}$$
$$v = \sin(\pi y)^2 \sin(2\pi x) \tag{14.18}$$

which is divergence free and doubly-periodic. This is then modified by adding the gradient of a scalar, ϕ, of the form:

$$\phi = \frac{1}{10} \cos(2\pi y) \cos(2\pi x) \tag{14.19}$$

yielding a new velocity, $\mathbf{U}_p = \mathbf{U} + \nabla\phi$. The result is the middle panel in the figure, and is not divergence free. A projection is then done, to recover ϕ by solving

$$\nabla^2 \phi = \nabla \cdot \mathbf{U}_p \tag{14.20}$$

For this figure, we use the standard 5-point Laplacian and a divergence built via centered-differences:

$$\nabla \cdot \mathbf{U} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} \tag{14.21}$$

Together, these represent an approximation projection.

Exercise 14.1

Implement an approximate projection, using pure smoothing for solving the elliptic equation. Start with the velocity field described by Eqs. 14.17 and 14.18 and the scalar from Eq. 14.19. Compute a poluted velocity field, $\mathbf{U}_p = \mathbf{U} + \nabla\phi$ and then project it to recover the original velocity field.

14.3 Cell-centered approximate projection solver

Here we describe an incompressible algorithm that uses cell-centered data throughout—$\mathbf{U}$ and p are both cell-centered. The projection at the end is an approximate projection. The basic algorithm flow is

- Create the time-centered advective velocities through the faces of the zones.
- Project the advective velocities such that they obey the velocity constraint
- Construct the time-centered interface states of all quantities on the faces of the zones using the advective velocity.
- Update the velocity to the new time. This is defines the provisional velocity field—it does not yet satisfy the constraint.
- Enforce the velocity constraint by projecting the velocity.

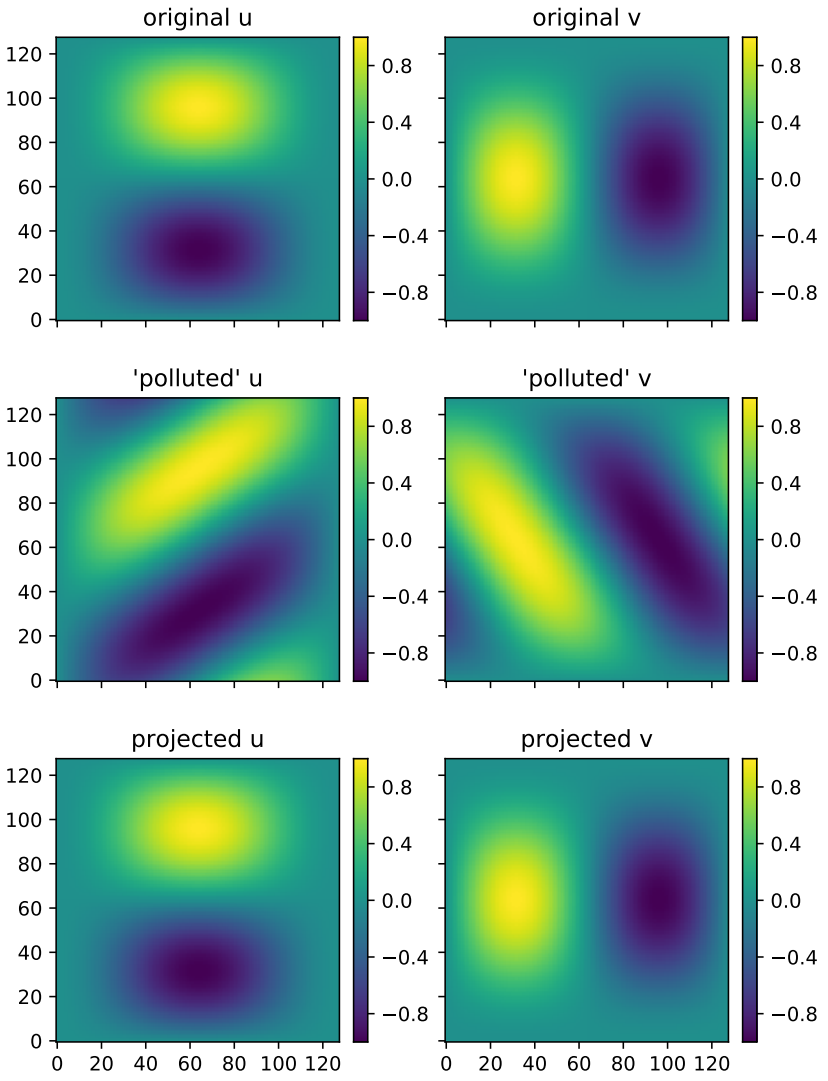

Figure 14.1: An example of an approximate projection, on a 128^2 grid. The top shows the original velocity field, Eqs. 14.17 and 14.18. This was poluted by adding the gradient of a scalar, Eq. 14.19 (middle) and then a projection was done to recover the original velocity field (bottom). In this case, the L_2 norm of the error on the velocity (comparing before and after projection) is 0.000189. hydro_examples: `project.py`

The description below is pieced together from a variety of sources. BCH describes a cell-centered method, but with an exact projection (with a larger, decoupled stencil). Almgren, Bell, and Szymczak (ABS) [7] describes an approximate projection method, but with a node-centered final projection. We follow this paper closely up until the projection. Martin and Colella [50] (and Martin's PhD thesis) method uses a cell-centered projection, as is described here. They go into additional effort to describe this for a refined grid. All of these method are largely alike, aside from how the

discretization of the final projection is handled.

14.3.1 Advective velocity

In predicting the interface states, we first seek to construct the velocities through the interfaces. A key concept throughout the advection step is that once we have the normal velocities on the interfaces, we can use these to upwind left and right states of any quantity to get their interface value. The advective velocities we construct here, $u^{\rm adv}$ and $v^{\rm adv}$, will later be used to upwind the various states we predict to the interfaces. We only need the velocity through the interfaces, as shown in the figure 14.2. This staggered grid arrangement is sometimes called a MAC grid.

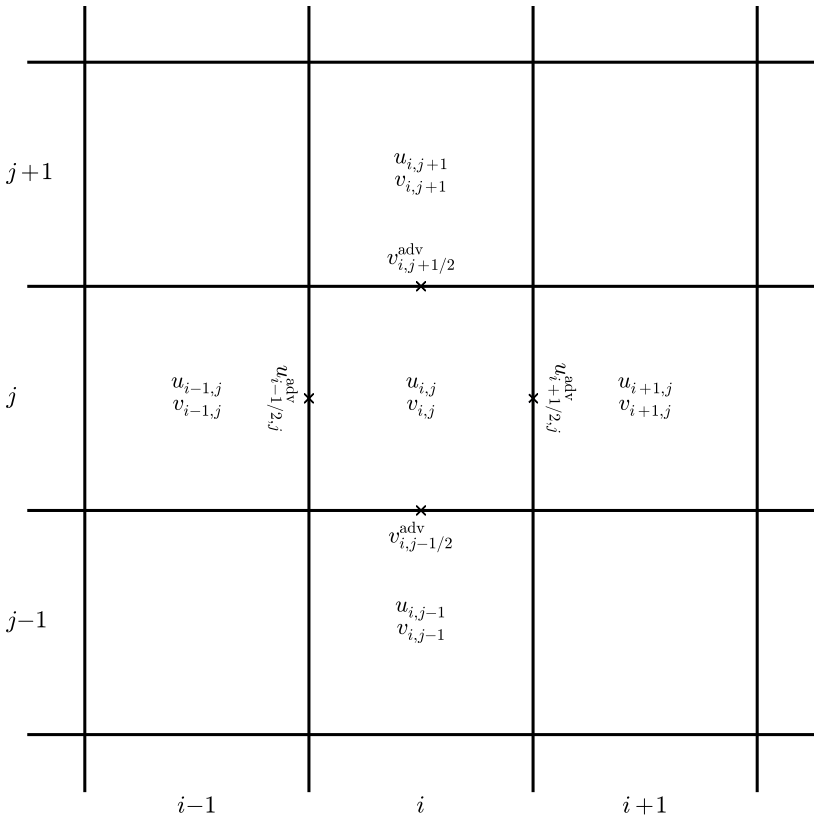

Figure 14.2: The staggered 'MAC' grid for the advective velocities.

We follow ABS. Our velocity evolution system (writing out the individual components of **U**: u and v) is

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x} = 0 \tag{14.22}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \frac{\partial p}{\partial y} = 0 \tag{14.23}$$

Our goal in this step is to predict time-centered interface values of the normal velocity (u on x-edges and v on y-edges). The prediction follows from Taylor expanding the state to the interface (through $\Delta x/2$ or $\Delta y/2$) and to the half-time (through $\Delta t/2$). As with the regular advection, we can have left and right states which we will resolve by solving a Riemann problem. The left interface state of u at $i+1/2, j$ is found as:

$$\begin{aligned} u^{n+1/2}_{i+1/2,j,L} &= u_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial u}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\frac{\partial u}{\partial t}\right|_{i,j} && (14.24)\\ &= u_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial u}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\left(-u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x}\right)\right|_{i,j} && (14.25)\\ &= u_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\left.\frac{\partial u}{\partial x}\right|_{i,j} - \frac{\Delta t}{2}\left(v\frac{\partial u}{\partial y}\right)_{i,j} - \frac{\Delta t}{2}\left.\frac{\partial p}{\partial x}\right|_{i,j} && (14.26)\\ & && (14.27) \end{aligned}$$

We express $\partial u/\partial x|_{i,j}$ as $\overline{\Delta u}^{(x)}_{i,j}/\Delta x$, where $\overline{\Delta u}^{(x)}_{i,j}$ is the limited slope of u in the x-direction in zone i, j. Our interface state is then:

$$u^{n+1/2}_{i+1/2,j,L} = \underbrace{u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\overline{\Delta u}^{(x)}_{i,j}}_{\equiv \hat{u}^{n+1/2}_{i+1/2,j,L}} - \underbrace{\frac{\Delta t}{2}\left(v\frac{\partial u}{\partial y}\right)_{i,j}}_{\text{transverse term}} - \frac{\Delta t}{2}\left.\frac{\partial p}{\partial x}\right|_{i,j} \tag{14.28}$$

Similarly, for v through the y faces, we find:

$$v^{n+1/2}_{i,j+1/2,L} = \underbrace{v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}v\right)\overline{\Delta v}^{(y)}_{i,j}}_{\equiv \hat{v}^{n+1/2}_{i,j+1/2,L}} - \underbrace{\frac{\Delta t}{2}\left(u\frac{\partial v}{\partial x}\right)_{i,j}}_{\text{transverse term}} - \frac{\Delta t}{2}\left.\frac{\partial p}{\partial y}\right|_{i,j} \tag{14.29}$$

As indicated above (and following ABS and the similar notation used by Colella [24]), we denote the quantities that will be used to evaluate the transverse states (consisting only of the normal predictor) with a '^'. These will be used to evaluate the transverse terms labeled above.

We predict u and v to both the x and y interfaces, using only the normal part of the predictor. This gives us the left and right 'hat' states on each interface.

u on x-interfaces:

$$\begin{aligned} \hat{u}^{n+1/2}_{i+1/2,j,L} &= u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\overline{\Delta u}^{(x)}_{i,j} && (14.30)\\ \hat{u}^{n+1/2}_{i+1/2,j,R} &= u_{i+1,j} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta x}u_{i+1,j}\right)\overline{\Delta u}^{(x)}_{i+1,j} && (14.31) \end{aligned}$$

v on x-interfaces:

$$\hat{v}^{n+1/2}_{i+1/2,j,L} = v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x} u_{i,j}\right) \overline{\Delta v}^{(x)}_{i,j} \tag{14.32}$$

$$\hat{v}^{n+1/2}_{i,j+1/2,R} = v_{i+1,j} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta x} u_{i+1,j}\right) \overline{\Delta v}^{(x)}_{i+1,j} \tag{14.33}$$

u on y-interfaces:

$$\hat{u}^{n+1/2}_{i,j+1/2,L} = u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta y} v_{i,j}\right) \overline{\Delta u}^{(y)}_{i,j} \tag{14.34}$$

$$\hat{u}^{n+1/2}_{i,j+1/2,R} = u_{i,j+1} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta y} v_{i,j+1}\right) \overline{\Delta u}^{(y)}_{i,j+1} \tag{14.35}$$

v on y-interfaces:

$$\hat{v}^{n+1/2}_{i,j+1/2,L} = v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta y} v_{i,j}\right) \overline{\Delta v}^{(y)}_{i,j} \tag{14.36}$$

$$\hat{v}^{n+1/2}_{i,j+1/2,R} = v_{i,j+1} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta y} v_{i,j+1}\right) \overline{\Delta v}^{(y)}_{i,j+1} \tag{14.37}$$

Note that the 'right' state is constructed using the data to the right of the interface. Also note that in constructing these transverse velocities, we do not include the p term.

Next we find the advective velocity through each interface. The incompressible velocity equation looks like the inviscid Burger's equation, and the Riemann solver follows that construction. BCG provide the implementation used here (and throughout the incompressible literature). Also see Toro [82]. We denote the resulting velocities with the 'adv' superscript, as these are the normal velocities used to advect the hat states. The Riemann problem solution is:

$$\mathcal{R}(q_L, q_R) = \begin{cases} q_L & \text{if } q_L > 0, \quad q_L + q_R > 0 \\ 0 & \text{if } q_L \leq 0, \quad q_R \geq 0 \\ q_R & \text{otherwise} \end{cases} \tag{14.38}$$

We solve this for each of the normal velocities, giving:

$$\hat{u}^{\text{adv}}_{i+1/2,j} = \mathcal{R}(\hat{u}^{n+1/2}_{i+1/2,j,L}, \hat{u}^{n+1/2}_{i+1/2,j,R}) \tag{14.39}$$

$$\hat{v}^{\text{adv}}_{i,j+1/2} = \mathcal{R}(\hat{v}^{n+1/2}_{i,j+1/2,L}, \hat{v}^{n+1/2}_{i,j+1/2,R}) \tag{14.40}$$

These advective velocities (sometimes called the *transverse velocities*) are used to resolve the left and right states of all the hat quantities by simple upwinding. For a u

or v state on the x-interface, we upwind based on $\hat{u}^{\rm adv}$; and for a u or v state on the y-interface, we upwind based on $\hat{v}^{\rm adv}$. If we write the upwinding as:

$$\mathcal{U}[s^{\rm adv}](q_L, q_R) = \begin{cases} q_L & \text{if } s^{\rm adv} > 0 \\ \frac{1}{2}(q_L + q_R) & \text{if } s^{\rm adv} = 0 \\ q_R & \text{if } s^{\rm adv} < 0 \end{cases} \tag{14.41}$$

Then the interface states are:

$$\hat{u}_{i+1/2,j} = \mathcal{U}[\hat{u}^{\rm adv}_{i+1/2,j}](\hat{u}^{n+1/2}_{i+1/2,j,L}, \hat{u}^{n+1/2}_{i+1/2,j,R}) \tag{14.42}$$

$$\hat{v}_{i+1/2,j} = \mathcal{U}[\hat{u}^{\rm adv}_{i+1/2,j}](\hat{v}^{n+1/2}_{i+1/2,j,L}, \hat{v}^{n+1/2}_{i+1/2,j,R}) \tag{14.43}$$

$$\hat{u}_{i,j+1/2} = \mathcal{U}[\hat{v}^{\rm adv}_{i,j+1/2}](\hat{u}^{n+1/2}_{i,j+1/2,L}, \hat{u}^{n+1/2}_{i,j+1/2,R}) \tag{14.44}$$

$$\hat{v}_{i,j+1/2} = \mathcal{U}[\hat{v}^{\rm adv}_{i,j+1/2}](\hat{v}^{n+1/2}_{i,j+1/2,L}, \hat{v}^{n+1/2}_{i,j+1/2,R}) \tag{14.45}$$

Now we can construct the full left and right predictions for the normal velocities on each interface (Eqs. 14.28 and 14.29). This involves simply adding the transverse term to the hat quantities and adding the pressure gradient.

$$u^{n+1/2}_{i+1/2,j,L} = \hat{u}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}\left[\frac{1}{2}\left(\hat{v}^{\rm adv}_{i,j-1/2} + \hat{v}^{\rm adv}_{i,j+1/2}\right)\right]\left(\frac{\hat{u}^{n+1/2}_{i,j+1/2} - \hat{u}^{n+1/2}_{i,j-1/2}}{\Delta y}\right) - \frac{\Delta t}{2}(Gp)^{(x),n-1/2}_{i,j} \tag{14.46}$$

and

$$v^{n+1/2}_{i,j+1/2,L} = \hat{v}^{n+1/2}_{i,j+1/2,L} - \frac{\Delta t}{2}\left[\frac{1}{2}\left(\hat{u}^{\rm adv}_{i-1/2,j} + \hat{u}^{\rm adv}_{i+1/2,j}\right)\right]\left(\frac{\hat{v}^{n+1/2}_{i+1/2,j} - \hat{v}^{n+1/2}_{i-1/2,j}}{\Delta x}\right) - \frac{\Delta t}{2}(Gp)^{(y),n-1/2}_{i,j} \tag{14.47}$$

Here $(Gp)^{(x),n-1/2}_{i,j}$ and $(Gp)^{(y),n-1/2}_{i,j}$ are difference-approximations to ∇p in the x and y directions respectively. Note that they are lagged—these come from the projection at the end of the previous timestep. See BCG for a discussion. A similar construction is done for the right states at the interface.

Finally, we do a Riemann solve (again, using the Burger's form of the Riemann problem) followed by upwinding to get the normal advective velocities. This is basically the $\mathcal{R}(.,.)$ operation followed by $\mathcal{U}(.,.)$. Together, it is:

$$u^{\rm adv}_{i+1/2,j} = \begin{cases} u^{n+1/2}_{i+1/2,j,L} & \text{if } u^{n+1/2}_{i+1/2,j,L} > 0, \quad u^{n+1/2}_{i+1/2,j,L} + u^{n+1/2}_{i+1/2,j,R} > 0 \\ \frac{1}{2}\left(u^{n+1/2}_{i+1/2,j,L} + u^{n+1/2}_{i+1/2,j,R}\right) & \text{if } u^{n+1/2}_{i+1/2,j,L} \le 0, \quad u^{n+1/2}_{i+1/2,j,R} \ge 0 \\ u^{n+1/2}_{i+1/2,j,R} & \text{otherwise} \end{cases} \tag{14.48}$$

and similar for $v^{\rm adv}_{i,j+1/2}$. These velocities are sometimes referred to as the MAC velocities.

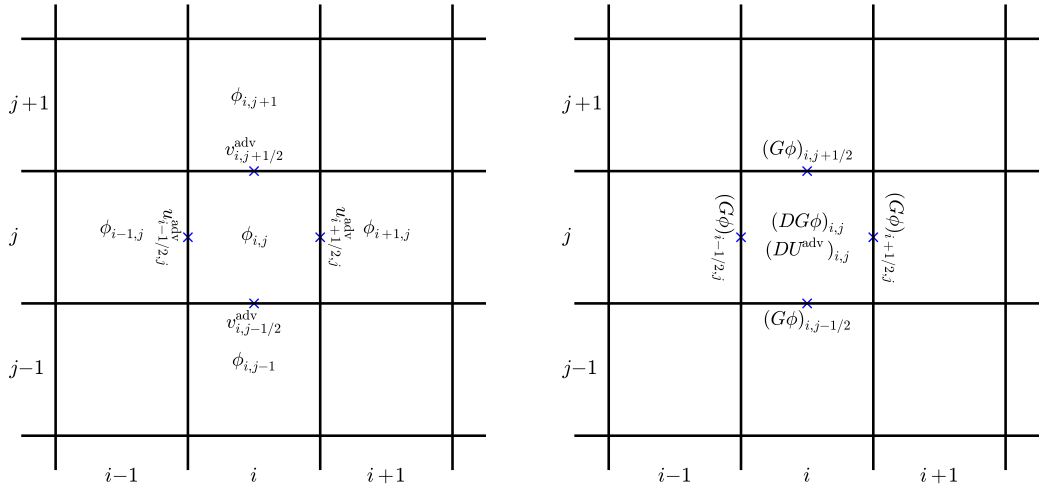

Figure 14.3: The centerings of the various components that make up the MAC projection.

14.3.2 MAC projection

We could simply use these time-centered advective velocities to construct the fluxes through the interfaces and update to the new time level. However BCH showed that such a method is unstable for CFL > 0.5. The fix is to enforce the velocity constraint on these advective velocities. This involves projecting the velocity field onto the space that is divergence free. This projection is usually called the MAC projection. Once the MAC-projected advective velocities are computed, we can reconstruct the interface states using this divergence-free velocity field. Figure 14.3 shows the location of the various quantities that participate it the MAC projection.

The divergence of the MAC velocities is cell-centered and constructed as:

$$(D\mathbf{U})_{i,j} = \frac{u^{\rm adv}_{i+1/2,j} - u^{\rm adv}_{i-1/2,j}}{\Delta x} + \frac{v^{\rm adv}_{i,j+1/2} - v^{\rm adv}_{i,j-1/2}}{\Delta y} \tag{14.49}$$

We define a cell-centered ϕ. $G\phi$ will then be edge-centered on a MAC grid, and $L\phi = DG\phi$ is again cell-centered. Since $L = DG$, this makes the MAC projection an exact projection.

We solve

$$L\phi = D\mathbf{U} \tag{14.50}$$

using multigrid V-cycles and then update the MAC velocities as:

$$u^{\rm adv}_{i+1/2,j} = u^{\rm adv}_{i+1/2,j} - \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{14.51}$$

$$v^{\rm adv}_{i,j+1/2} = v^{\rm adv}_{i,j+1/2} - \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{14.52}$$

14.3.3 Reconstruct interface states

Next we redo the construction of the interface states. This procedure is identical to that above—construct the interface states $\hat{u}_{L,R}$, $\hat{v}_{L,R}$ on all edges, upwind based on $\hat{u}^{\rm adv}$, $\hat{v}^{\rm adv}$, and use these to construct the full states (including transverse terms). Now however, we construct the interface states of u and v on both the x and y-interfaces (not just the normal component at each interface). Finally, instead of solving a Riemann problem to resolve the left and right states, we simply upwind using the MAC-projected $u^{\rm adv}$ and $v^{\rm adv}$. This results in the interface state $u^{n+1/2}_{i+1/2,j}$, $v^{n+1/2}_{i+1/2,j}$, $u^{n+1/2}_{i,j+1/2}$, $v^{n+1/2}_{i,j+1/2}$.

The only reason we need to do this step over, instead of using the interface states that we predicted previously is we want to ensure that they are consistent with the MAC-projected advective velocities (and therefore, consistent with the constraint).

14.3.4 Provisional update

Once we have the time-centered interface states that are consistent with the MAC-projected advective velocities, we can update the velocities to the new time by discretizing the advective terms $(\mathbf{U}\cdot\nabla\mathbf{U})$. We express the advective terms for u as $A^{(u),n+1/2}_{i,j}$ and those for v as $A^{(v),n+1/2}_{i,j}$. These have the form:

$$\begin{aligned} A^{(u),n+1/2}_{i,j} = \quad & \frac{1}{2}\left(u^{\rm adv}_{i-1/2,j} + u^{\rm adv}_{i+1/2,j}\right)\frac{u^{n+1/2}_{i+1/2,j} - u^{n+1/2}_{i-1/2,j}}{\Delta x} + \\ & \frac{1}{2}\left(v^{\rm adv}_{i,j-1/2} + v^{\rm adv}_{i,j+1/2}\right)\frac{u^{n+1/2}_{i,j+1/2} - u^{n+1/2}_{i,j-1/2}}{\Delta y} \end{aligned} \tag{14.53}$$

$$\begin{aligned} A^{(v),n+1/2}_{i,j} = \quad & \frac{1}{2}\left(u^{\rm adv}_{i-1/2,j} + u^{\rm adv}_{i+1/2,j}\right)\frac{v^{n+1/2}_{i+1/2,j} - v^{n+1/2}_{i-1/2,j}}{\Delta x} + \\ & \frac{1}{2}\left(v^{\rm adv}_{i,j-1/2} + v^{\rm adv}_{i,j+1/2}\right)\frac{v^{n+1/2}_{i,j+1/2} - v^{n+1/2}_{i,j-1/2}}{\Delta y} \end{aligned} \tag{14.54}$$

The normal update for u, v would include the Gp term and appear as:

$$u^\star_{i,j} = u^n_{i,j} - \Delta t A^{(u),n+1/2}_{i,j} - \Delta t (Gp)^{(x),n-1/2}_{i,j} \tag{14.55}$$

$$v^\star_{i,j} = v^n_{i,j} - \Delta t A^{(v),n+1/2}_{i,j} - \Delta t (Gp)^{(y),n-1/2}_{i,j} \tag{14.56}$$

Note that at this point, we don't have an updated p, so we use a lagged value from the previous step's projection.

Alternately, we can note that for an exact projection, Gp, is the gradient of a scalar and would be removed by the projection, so we can omit it in this update, giving an

alternate provisional update:

$$u^{\star\star}_{i,j} = u^n_{i,j} - \Delta t A^{(u),n+1/2}_{i,j} \tag{14.57}$$

$$v^{\star\star}_{i,j} = v^n_{i,j} - \Delta t A^{(v),n+1/2}_{i,j} \tag{14.58}$$

Following the notation in Martin, we distinguish between these with an '$\star$' vs. '$\star\star$' *.

14.3.5 Approximate projection

This provisional velocity field does not yet obey the constraint. To enforce the constraint, we need to do a projection. Here is where we have the flexibility on whether to include the $Gp^{n-1/2}$ term. If we were doing an exact projection, then adding the gradient of a scalar would not change the divergence-free velocity field, so there would be no need to add it.

BCH did an exact projection on a cell-centered grid. There, the divergence operator is:

$$(D\mathbf{U})_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} \tag{14.59}$$

This gives a cell-centered $D\mathbf{U}$. If we want ϕ cell-centered, then the gradient, $G\phi$ must also be cell centered so $L\phi = DG\phi$ is cell-centered. This means that we must have

$$(G\phi)^{(x)}_{i,j} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \tag{14.60}$$

$$(G\phi)^{(y)}_{i,j} = \frac{\phi_{i,j+1} - \phi_{i,j+1}}{2\Delta y} \tag{14.61}$$

The resulting Laplacian would then be a 5-point stencil that skips over the immediate neighbors:

$$(L\phi)_{i,j} = \frac{\phi_{i+2,j} - 2\phi_{i,j} + \phi_{i-2,j}}{4\Delta x^2} + \frac{\phi_{i,j+2} - 2\phi_{i,j} + \phi_{i,j-2}}{4\Delta y^2} \tag{14.62}$$

This decomposes the domain into 4 distinct grids that are only linked together at the boundaries. While an exact projection, this decoupling can be undesirable.

Approximate projections relax the idea that $L = DG$. In an exact projection, when you apply the projection operator, P, in succession, the result is unchanged ($P^2 = P$). This is not the case for an approximate projection. As a result, exactly what form you project matters. For an approximate projection, we can use the standard 5-point stencil for the Laplacian,

$$(L\phi)_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} \tag{14.63}$$

*Note that these are identical to ABC [3] approximation projections (1) and (2) (a quick look at ABC might suggest the opposite, but note that their definition of $\mathbf{U}^\star$ already includes a $-Gp^{n-1/2}$ term, so by explicitly adding it back in, you are dealing with the case where $\mathbf{U}^\star$ was updated without any $Gp^{n-1/2}$ term, like the '$\star\star$' case above.)

together with the cell-centered divergence above (Eq. 14.59).

Rider [65] and Almgren, Bell, and Crutchfield (ABC) [3] explore various forms of what to project when doing the approximate projection. For instance, do we include the $Gp^{n-1/2}$ term in the provisional velocity or not? Chorin noted that if viscosity is being modeled, then it is necessary to include it here to get second-order accuracy. Also, one can project the update to the velocity, $(\mathbf{U}^\star - \mathbf{U}^n)/\Delta t$ instead of just the new velocity, since $\mathbf{U}^n$ should already be divergence free. Rider argues that projecting the update is not desirable with approximate projections, since any error in $\mathbf{U}^n$ being divergence-free is carried forward to the new $\mathbf{U}^{n+1}$. One issue is that a cell-centered approximate projection cannot remove all sources of divergence (see Rider and Martin's PhD thesis).

When projecting the new velocity, we scale by Δt to get a quantity that has dimensions of pressure. The procedure for the projection differs slightly depending on whether we project $\mathbf{U}^\star$ or $\mathbf{U}^{\star\star}$:

- case I: projecting $\mathbf{U}^\star/\Delta t$.

  From the expression above, this looks like:

  $$\frac{\mathbf{U}^\star}{\Delta t} = \frac{\mathbf{U}^n}{\Delta t} - A^{(u),n+1/2} - (Gp)^{(x),n-1/2} \tag{14.64}$$

  Ideally, $\mathbf{U}^n$ is already divergence free, and $Gp^{n-1/2}$ is the gradient of a scalar, which will be removed, so the projection should pick out the divergence free portion of $A^{(u)}$. We solve:

  $$L\phi = D(\mathbf{U}^\star/\Delta t) \tag{14.65}$$

  using multigrid V-cycles. We then find the new, divergence free velocity field as:

  $$\mathbf{U}^{n+1} = \mathbf{U}^\star - \Delta t G\phi \tag{14.66}$$

  Since we already included $Gp^{n-1/2}$ in what we projected, $G\phi$ will be the correction,

  $$G\phi = Gp^{n+1/2} - Gp^{n-1/2} \tag{14.67}$$

  or

  $$Gp^{n+1/2} = Gp^{n-1/2} + G\phi \tag{14.68}$$

  (see Martin 2.5 or ABC). We store this for the next timestep.

- case II: projecting $\mathbf{U}^{\star\star}/\Delta t$.

  From the expression above, this looks like:

  $$\frac{\mathbf{U}^{\star\star}}{\Delta t} = \frac{\mathbf{U}^n}{\Delta t} - A^{(u),n+1/2} \tag{14.69}$$

There is no explicit $Gp^{n-1/2}$ term. We solve:

$$L\phi = D(\mathbf{U}^{\star\star}/\Delta t) \tag{14.70}$$

using multigrid V-cycles. We then find the new, divergence free velocity field as:

$$\mathbf{U}^{n+1} = \mathbf{U}^{\star\star} - \Delta t G\phi \tag{14.71}$$

Since there was no $Gp^{n-1/2}$ in what we projected, $p^{n+1/2} = \phi$, and

$$Gp^{n+1/2} = G\phi \tag{14.72}$$

We store this for the next timestep.

One pathology of this form of the projection is that $(D\mathbf{U})_{i,j}$ does not actually make use of the velocity field in zone (i, j). This decoupling from the local zone can result in a checkerboarding pattern in the projected velocity field.

14.4 Boundary conditions

For the advection portion of the algorithm, the boundary conditions on u and v are implemented in the usual way, using ghost cells. For the projection,

For a periodic domain, the boundary conditions on ϕ are likewise periodic. At a solid wall or inflow boundary, we already predicted the velocity that we want at the wall (in the advection step), and we do not want this value to change in the projection step. Since the correction is:

$$\mathbf{U}^{n+1} = \mathbf{U}^{\star} - \nabla\phi \tag{14.73}$$

we want $\nabla\phi \cdot n = 0$.

At outflow boundaries, we do not want to introduce any shear as we go through the boundary. This means that we do not want any tangential acceleration. Setting $\phi = 0$ on the boundary enforces $\nabla\phi \cdot t = 0$, where t is the unit vector tangential to the boundary.

See ABS for a discussion of the boundary conditions.

14.5 Bootstrapping

At step 0, we do not have a value of $Gp^{-1/2}$. To get an initial value for Gp, we run through the entire evolution algorithm starting with the initial data. At the end of a step, we reset u and v to the initial values and store the Gp at the end of this step as $Gp^{-1/2}$.

It is also common to precede this initialization by first projecting the velocity field to ensure it is divergence free. This way, we do not have to rely on the initial conditions to always set a divergence free velocity field.

14.6 Test problems

14.6.1 Convergence test

Minion [54] introduced a simple test problem with an analytic solution. The velocity field is initialized as:

$$u(x,y) = 1 - 2\cos(2\pi x)\sin(2\pi y) \tag{14.74}$$
$$v(x,y) = 1 + 2\sin(2\pi x)\cos(2\pi y) \tag{14.75}$$

The exact solution at some time t is:

$$u(x,y,t) = 1 - 2\cos(2\pi(x-t))\sin(2\pi(y-t)) \tag{14.76}$$
$$v(x,y,t) = 1 + 2\sin(2\pi(x-t))\cos(2\pi(y-t)) \tag{14.77}$$

Minion also gives the pressure, but this is not needed for the solution. This is run on a doubly-periodic unit square domain. The main utility of this set of initial conditions is that we can use the analytic solution to measure the convergence behavior of the algorithm.

14.7 Extensions

- *Variable density incompressible*: Bell & Marcus [14] describe how to extend these methods to variable density flows. This means that the density in the domain may not be constant, but within a fluid element, the density does not change. This can arise, for instance, in modeling the Rayleigh-Taylor instability.

  The basic idea follows the method described above. Now the mass continuity equation is also evolved:

  $$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{14.78}$$

  The density is predicted to the interfaces follow the same procedure above and upwinded using the MAC velocities. For the projection, the decomposition is written as:

  $$\mathbf{U} = \mathbf{U}^d + \frac{1}{\rho}\nabla\phi \tag{14.79}$$

  and the elliptic equation is now a variable-coefficient equation:

  $$\nabla \cdot \frac{1}{\rho}\nabla\phi = \nabla \cdot \mathbf{U} \tag{14.80}$$

- *Viscosity*: When viscosity is included in the system, our momentum equation becomes:

  $$\mathbf{U}_t + \mathbf{U} \cdot \nabla \mathbf{U} + \nabla p = \epsilon \nabla^2 \mathbf{U} \tag{14.81}$$

The solution process for this equation is largely the same. Following BCH, first the advective term is computed by predicting the velocities to the interfaces, doing the MAC projection, and then forming $A^{n+1/2}$. Now there is an explicit viscosity term (at time-level n) present in the prediction, as a source term. The provision velocity update is no longer a simple flux update, but instead requires solving two decoupled diffusion-like equations (one for u and one for v). These are differenced using Crank-Nicolson centering:

$$\frac{u^\star - u^n}{\Delta t} = -A^{(u),n+1/2} - \nabla p^{(x),n-1/2} + \frac{\epsilon}{2}\nabla^2(u^n + u^\star) \tag{14.82}$$

$$\frac{v^\star - v^n}{\Delta t} = -A^{(v),n+1/2} - \nabla p^{(y),n-1/2} + \frac{\epsilon}{2}\nabla^2(v^n + v^\star) \tag{14.83}$$

This involves two separate multigrid solves. Once $\mathbf{U}^\star$ is found, the final projection is done as usual.

- *Low Mach number combustion*: In low-Mach number combustion flows, the fluid is allowed to respond to local heat release by expanding. The velocity constraint is derived by differentiating the equation of state along particle paths, leading to the appearance of a source term:

$$\nabla \cdot \mathbf{U} = S \tag{14.84}$$

Here, S, incorporates the compressibility effects due to the heat release and diffusion. This system is used when modeling burning fronts (flames). This type of flow can be thought of as linking two incompressible states (the fuel and the ash) by the expansion across the interface.

The solution technique largely follows that of the incompressible flow. One caveat, because the constraint now has a local heat source, S, doing the cell-centered divergence described above leads to a decoupling of $D\mathbf{U}$ from the local source, since the stencil of $D\mathbf{U}$ does not include the zone upon which it is centered.

This system is described in detail in [13, 29, 61].

- *Stratified flows*: When the background is stratified, a different velocity constraint can arise, capturing the expansion of the fluid element due to the pressure change with altitude. For example, with an ideal gas, the equation of state can be recast as:

$$\nabla \cdot (p_0^{1/\gamma}) = 0 \tag{14.85}$$

where $p_0(z)$ is the pressure as a function of height, representing the hydrostatic background, and γ is the ratio of specific heats. For a general equation of state, $p_0^(1/\gamma)$ is replaced with something more complicated (see [4–6] for the development of a low Mach hydrodynamics method for stratified astrophysical flows).

- *Nodal projection*: instead of discretizing the final projection using a cell-centered ϕ, ABS use a node-centered ϕ. While this is still an approximate projection, this discretization couples in the zone we are centered on, and is said to be able to do a better job removing pathological divergent velocity fields that cell-centered projections stumble on.

Chapter 15

Low Mach Number Methods

Incompressible flow represents the zero-Mach number limit of fluid flow—no compressibility effects are modeled. We can extend the ideas of incompressibe flow to allow us to model some compressibility effects, giving rise to low Mach number methods.

15.1 Low Mach divergence constraints

The key idea in solvers for low Mach number flows is that, as a fluid element advects, its pressure remains the same as the background state. For an atmosphere, this background state is a hydrostatic profile. For a smallscale combustion flow, this background state is a spatially constant pressure (and constant-in-time if the domain is open). We'll denote this background pressure as $p_0(r,t)$, where r is a radial coordinate.

Previously, we derived the pressure evolution with reactive sources (see Eq. 12.24) by following the evolution of pressure as a fluid element moves (e.g., the Lagrangian derivative). For the low Mach flows, we want to see the evolution p_0 in the fluid element, which is expressed as:

$$\frac{\partial p_0}{\partial t} + \mathbf{U}\cdot\nabla p_0 + \Gamma_1 p_0 \nabla\cdot\mathbf{U} = \Gamma_1 p_0 \sigma H_{\rm nuc} \tag{15.1}$$

We can rewrite this as a constraint on the velocity field by solving for $\nabla\cdot\mathbf{U}$:

$$\nabla\cdot\mathbf{U} + \frac{1}{\Gamma_1 p_0}\frac{Dp_0}{Dt} = \sigma H_{\rm nuc} \equiv S \tag{15.2}$$

with

$$\sigma = \frac{\partial p_0/\partial T|_\rho}{\rho c_p \partial p_0/\partial\rho|_T} \tag{15.3}$$

This is the general constraint equation for low Mach flow. Note that the only approximation we made is $p \rightarrow p_0$. This form of the constraint, for a general equation of state, was originally derived in [5].

Combustion limit

A useful limit is smallscale combustion. In an open domain, we can take p_0 as constant, so $Dp_0/Dt = 0$, and we are left with

$$\nabla \cdot \mathbf{U} = S \tag{15.4}$$

This looks like the constraint for incompressible flow, but with a source to the divergence. This source captures the compressible effects due to local heat release—as a fluid parcel moves, the only changes to its density will come through local heat release. Methods for this type of constraint are discussed in [13, 29, 61].

Atmospheric case

Another interesting case is that of an atmosphere. If we consider an ideal gas, then $\Gamma_1 = \gamma =$ constant. A second appromation we take is that $p_0 = p_0(r)$—i.e., no time dependence to the atmosphere. Finally, we'll consider the case with no local heat sources ($S = 0$). Then we have

$$\nabla \cdot \mathbf{U} + \frac{1}{\gamma p_0} \mathbf{U} \cdot \nabla p_0 = 0 \tag{15.5}$$

which is equivalent to

$$\nabla \cdot \left(p_0^{1/\gamma} \mathbf{U} \right) = 0 \tag{15.6}$$

This constraint captures the changes in compressibilty due to the background stratification of an atmosphere. This form was originally derived for atmospheric flows by [32] and generalized to stellar flows in [5]. If the structure of the atmosphere is isentropic, then we know that $d \log p_0 = \gamma d \log \rho_0$, where we use ρ_0 to represent the density corresponding to p_0, and we can write this constraint as:

$$\nabla \cdot (\rho_0 \mathbf{U}) = 0 \tag{15.7}$$

This is the traditional anelastic constraint.

Extensions involving external heating sources or reactions are discussed in [4, 6], and dealing with the non-constant Γ_1 is discussed in [5, 43]. The time-dependence of the background atmosphere is explored in [6] for plane-parallel atmospheres, following the ideas from [1], and in [57] for spherical, self-gravitating bodies.

15.2 Multigrid for Variable-Density Flows

The solution methodology for these low Mach number systems follows that of the incompressible flow, but with two additions. First, we need to incorporate a density (mass continuity) evolution equation—this will follow the same techniques we already saw for advection, as we'll see later. Next, we need to be able to enforce more general forms of the divergence constraint, which as we'll see in a moment, require us to solve a variable-coefficient elliptic equation. Our multigrid technique will need to be suitably modified.

We now need to solve an elliptic equation of the form:

$$\nabla \cdot (\eta \nabla \phi) = f \tag{15.8}$$

If we denote the discrete divergence and gradient operators as D and G, then our operator will be $L_\eta \equiv D\eta G$. If we wish to use a cell-centered discretization for ϕ, then using a standard centered-difference for D and G will result in a stencil that reaches two zones on either side of the current zone. This can lead to an odd-even decoupling.

Instead, we again use an approximate projection. We discretize the variable-coefficient Laplacian as:

$$\begin{aligned}(L_\eta \phi)_{i,j} = &\frac{\eta_{i+1/2,j}(\phi_{i+1,j} - \phi_{i,j}) - \eta_{i-1/2,j}(\phi_{i,j} - \phi_{i-1,j})}{\Delta x^2} + \\ &\frac{\eta_{i,j+1/2}(\phi_{i,j+1} - \phi_{i,j}) - \eta_{i,j-1/2}(\phi_{i,j} - \phi_{i,j-1})}{\Delta y^2}\end{aligned} \tag{15.9}$$

We can define the interface values of η as the averages of the cell-centered values, e.g.,

$$\eta_{i,j+1/2} = \frac{1}{2}(\eta_{i,j} + \eta_{i,j+1}) \tag{15.10}$$

Our elliptic equation is then

$$(L_\eta \phi)_{i,j} = f_{i,j} \tag{15.11}$$

The relaxation method for this operator again relies on isolating $\phi_{i,j}$, yielding:

$$\phi_{i,j} = \frac{\tilde{\eta}_{i+1/2,j}\phi_{i+1,j} + \tilde{\eta}_{i-1/2,j}\phi_{i-1,j} + \tilde{\eta}_{i,j+1/2}\phi_{i,j+1} + \tilde{\eta}_{i,j-1/2}\phi_{i,j-1} - f_{i,j}}{\tilde{\eta}_{i+1/2,j} + \tilde{\eta}_{i-1/2,j} + \tilde{\eta}_{i,j+1/2} + \tilde{\eta}_{i,j-1/2}} \tag{15.12}$$

with the shorthand that $\tilde{\eta}_{i\pm1/2,j} = \eta_{i\pm1/2,j}/\Delta x^2$ and $\tilde{\eta}_{i,j\pm1/2} = \eta_{i,j\pm1/2}/\Delta y^2$.

To put this into our multigrid framework, there are three changes we need to make:

- The smoothing function needs to implement the more general smoothing described by Eq. 15.12.

- The residual function needs to compute $(L_\eta \phi)_{i,j}$ according to Eq. 15.9, and then $r_{i,j} = f_{i,j} - (L_\eta \phi)_{i,j}$.
- The coefficients, η should be averaged to the edges on the fine grid and then restricted down the multigrid hierarchy as edge-based quantities.

15.2.1 Test problem

Periodic

To test the solver, we need to devise a problem with a known analytic solution. The easiest way to do this is to pick an $\eta(x)$ and ϕ and then do the divergence and gradients to find the required righthand side, f. We'll use periodic BCs, and for our equation $\nabla \cdot (\eta \nabla \phi) = f$, the following provide a well-posed test problem:

$$\begin{aligned} \eta &= 2 + \cos(2\pi x)\cos(2\pi y) \\ f &= -16.0\pi^2 \left[\cos(2\pi x)\cos(2\pi y) + 1\right] \sin(2\pi x)\sin(2\pi y) \end{aligned} \tag{15.13}$$

with the solution:

$$\phi^{\text{true}} = \sin(2\pi x)\sin(2\pi y)$$

There is an important caveat when dealing with a purely-periodic problem. Since there is no boundary values to "anchor" the solution, it is free to float. Solving the elliptic problem will give use the correct $\nabla\phi$, but the average of ϕ over the domain is unconstrained. For our algorithms, it is $\nabla\phi$ that matters (that is the forcing term that enters into the momentum equation).

When for checking convergence, we want to compare to the exact solution. We therefore normalize ϕ by subtracting off its average value before computing the norm of the error with respect to the exact solution:

$$\epsilon = \| \phi_{i,j} - \bar{\phi} - \phi^{\text{true}}_{i,j} \| \ , \tag{15.14}$$

where

$$\bar{\phi} = \frac{1}{N_x N_y} \sum_{i,j} \phi_{i,j} \tag{15.15}$$

As discussed in § 9.6, this can arise if the discrete form the righthand side, $f_{i,j}$ does not sum exactly to zero. Figure 15.1 shows the solution to this problem with a 512^2 grid. and the convergence of the solver described here is shown in Figure 15.2.

Dirichlet

We can run the same problem with Dirichlet boundary conditions on ϕ, and we are free to pick different boundary conditions for η, since it represents a different physical quantity. Since we only have homogeneous Dirichlet or Neumann BCs implemented, we'll run with Neumann BCs on η.

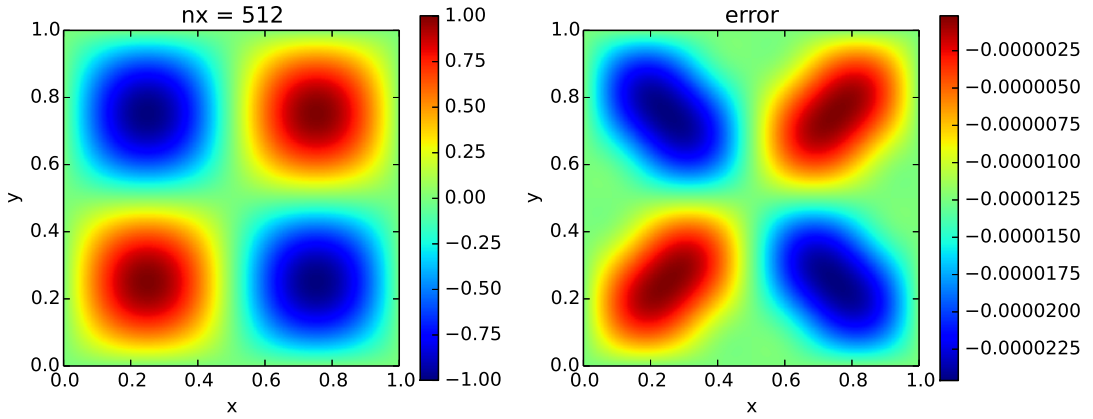

Figure 15.1: Solution and error to the variable-coefficient Poisson problem defined in Eq. 15.13. This test can be run with pyro `multigrid/test_mg_vc_periodic.py`.

15.3 Atmospheric flows

15.3.1 Equation Set

For atmospheric flows, we define a one-dimensional base state that is in hydrostatic equilibrium. In general, this base state can be time-dependent, expanding in response to heating in the atmosphere (see e.g. [1, 6]. Here we'll consider only a time-independent state.

We'll follow the procedure defined in [57]: we define ρ_0 as the lateral average of ρ:

$$\rho_{0_j} = \frac{1}{N_x} \sum_i \rho_{i,j} \tag{15.16}$$

and then we define the base state pressure, p_0, by integrating the equation of hydrostatic equilibrium, $dp_0/dy = \rho g$, as:

$$p_{0_{j+1}} = p_{0_j} + \frac{\Delta y}{2}(\rho_{0_j} + \rho_{0_{j+1}})g \tag{15.17}$$

with an initial condition of

$$p_{0_{\rm jlo}} = \frac{1}{N_x} \sum_i p^{\rm initial}_{i,{\rm jlo}} \tag{15.18}$$

The compressible momentum equation (written in terms of velocity is):

$$\rho \frac{\partial \mathbf{U}}{\partial t} + \rho \mathbf{U} \cdot \nabla \mathbf{U} + \nabla p = \rho \mathbf{g} \tag{15.19}$$

Subtracting off the base state, and defining the perturbational pressure (sometimes called the dynamic pressure) as $p' = p - p_0$, and perturbational density as $\rho' = \rho - \rho_0$, we have:

$$\rho \frac{\partial \mathbf{U}}{\partial t} + \rho \mathbf{U} \cdot \nabla \mathbf{U} + \nabla p' = \rho' \mathbf{g} \tag{15.20}$$

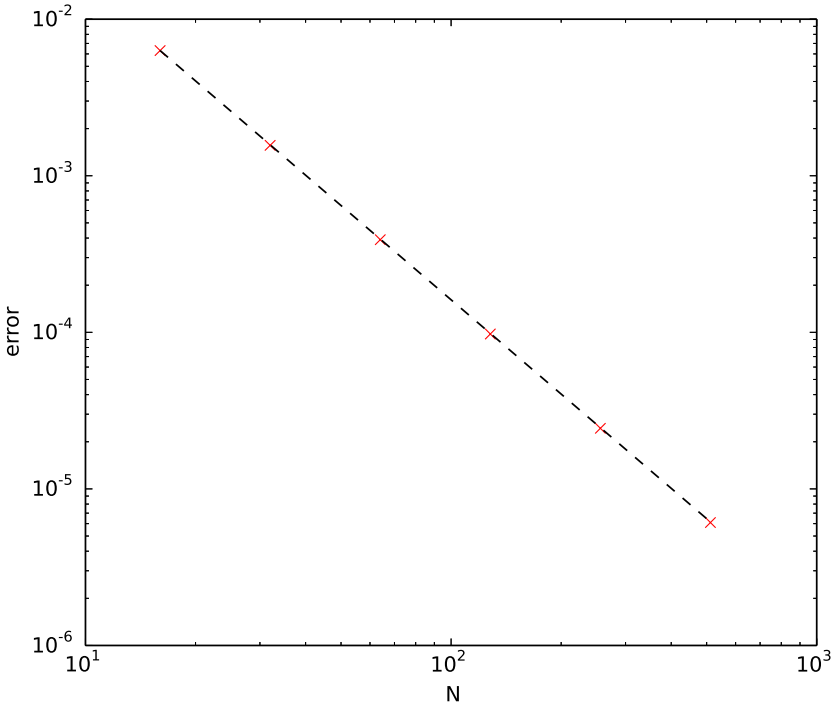

Figure 15.2: Convergence of the variable-coefficient multigrid solver for the test problem defined in Eq. 15.13. This test can be run with pyro `multigrid/test_mg_vc_periodic.py`.

or

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \frac{1}{\rho} \nabla p' = \frac{\rho'}{\rho} \mathbf{g} \tag{15.21}$$

Several authors [43, 83] explored the idea of energy conservation in a low Mach number system and found that an additional term (which can look like a buoyancy) is needed in the low Mach number formulation, yielding:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} + \frac{\beta_0}{\rho} \nabla \left(\frac{p'}{\beta_0} \right) = \frac{\rho'}{\rho} \mathbf{g} \tag{15.22}$$

Completing the system are the continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{15.23}$$

and the constraint,

$$\nabla \cdot (\beta_0 \mathbf{U}) = 0 \tag{15.24}$$

with $\beta_0 = p_0^{1/\gamma}$.

15.3.2 Solution Procedure

The general solution procedure is for a single step is:

I. *Predict* $\mathbf{U}$ *to the interfaces*

II. *Enforce the divergence constraint on the interface* $\mathbf{U}$*'s (the MAC projection) to get* $\mathbf{U}^{\text{adv}}$.

Decomposing the velocity field as

$$\mathbf{U}^\star = \mathbf{U}^d + \frac{\beta_0}{\rho}\nabla\phi \tag{15.25}$$

as suggested from the form of the momentum equation, our Poisson equation can be defined by multiplying by β_0 and taking the divergence, and using $\nabla \cdot (\beta_0 \mathbf{U}^d) = 0$, giving

$$\nabla \cdot \frac{\beta_0^2}{\rho}\nabla\phi = \nabla \cdot (\beta_0 \mathbf{U}^\star) \tag{15.26}$$

Note that when written like this, ϕ has units of p'/β_0.

For the MAC projection, we have edge-centered velocities (in their respective coordinate direction). We define an edge-centered β_0 as

$$\beta_{0_{j+1/2}} = \frac{1}{2}(\beta_{0_j} + \beta_{0_{j+1}}) \tag{15.27}$$

(note that, since β_0 is one-dimensional, we only average in the vertical direction). The divergence term is then:

$$[\nabla \cdot (\beta_0 \mathbf{U})]^{\text{adv}}_{i,j} = \beta_{0_j}\frac{u^{\text{adv}}_{i+1/2,j} - u^{\text{adv}}_{i-1/2,j}}{\Delta x} + \frac{\beta_{0_{j+1/2}} v^{\text{adv}}_{i,j+1/2} - \beta_{0_{j-1/2}} v^{\text{adv}}_{i,j-1/2}}{\Delta y} \tag{15.28}$$

We define the coefficient, η, in the Poisson equation as:

$$\eta_{i,j} = \frac{{\beta_0}_j^2}{\rho_{i,j}} \tag{15.29}$$

and bring it to edges simply through averaging. Multigrid is used to solve for $\phi_{i,j}$.

We then solve

$$(L_\eta \phi)_{i,j} = [\nabla \cdot (\beta_0 \mathbf{U})]^{\text{adv}}_{i,j} \tag{15.30}$$

Once we solve for ϕ, we correct the velocity as:

$$\mathbf{U}^{\text{new}} = \mathbf{U}^\star - \frac{\beta_0}{\rho}\nabla\phi \tag{15.31}$$

Since the MAC velocities are edge-centered, our correction appears as:

$$u^{\rm adv}_{i+1/2,j} = u^{\rm adv}_{i+1/2,j} - \left(\frac{\beta_0}{\rho}\right)_{i+1/2,j} \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{15.32}$$

$$v^{\rm adv}_{i,j+1/2} = v^{\rm adv}_{i,j+1/2} - \left(\frac{\beta_0}{\rho}\right)_{i,j+1/2} \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{15.33}$$

III. *Predict ρ to the interfaces*

We need to solve the continuity equation. We use the same techniques that were used for advection. Our equation is:

$$\rho_t + (\rho u)_x + (\rho v)_y = 0 \tag{15.34}$$

The x-interface left state would be:

$$\begin{aligned}
\rho^{n+1/2}_{i+1/2,j,L} &= \rho^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \rho}{\partial x} + \frac{\Delta t}{2}\frac{\partial \rho}{\partial t} + \ldots \\
&= \rho^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \rho}{\partial x} + \frac{\Delta t}{2}\left[-(\rho u)_x - (\rho v)_y\right]_{i,j} \\
&= \underbrace{\rho^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\frac{\partial \rho}{\partial x}}_{\hat{\rho}^{n+1/2}_{i+1/2,j,L}} - \frac{\Delta t}{2}\left[\rho u_x\right]_{i,j} - \frac{\Delta t}{2}\left[(\rho v)_y\right]_{i,j}
\end{aligned} \tag{15.35}$$

A similar construction would yield the right state at that interface, and the y-interface states.

Since we already have the MAC-projected advected velocities at this point, we can use them in the construction of the interface states and the upwinding. As before, we split this into a normal part (the $\hat{\rho}$ term) and "transverse" part (which here includes the non-advective part of the divergence). We first construct the normal parts as

$$\hat{\rho}^{n+1/2}_{i+1/2,j,L} = \rho^n_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u^{\rm adv}_{i+1/2,j}\right)\overline{\Delta\rho}^{(x)}_{i,j} \tag{15.36}$$

and then solve the Riemann problem (upwinding) for these:

$$\hat{\rho}^{n+1/2}_{i+1/2,j} = \mathcal{U}[u^{\rm adv}_{i+1/2,j}](\hat{\rho}^{n+1/2}_{i+1/2,j,L}, \hat{\rho}^{n+1/2}_{i+1/2,j,R}) = \begin{cases} \hat{\rho}^{n+1/2}_{i+1/2,j,L} & u^{\rm adv}_{i+1/2,j} > 0 \\ \hat{\rho}^{n+1/2}_{i+1/2,j,R} & u^{\rm adv}_{i+1/2,j} < 0 \end{cases} \tag{15.37}$$

The same procedure is done for the y-interfaces.

The full states are then constructed using these "hat" states. For example,

$$\begin{aligned}
\rho^{n+1/2}_{i+1/2,j,L} = \hat{\rho}^{n+1/2}_{i+1/2,j,L} &- \frac{\Delta t}{2}\rho_{i,j}\frac{u^{\rm adv}_{i+1/2,j} - u^{\rm adv}_{i-1/2,j}}{\Delta x} \\
&- \frac{\Delta t}{2}\frac{\hat{\rho}^{n+1/2}_{i,j+1/2}v^{\rm adv}_{i,j+1/2} - \hat{\rho}^{n+1/2}_{i,j-1/2}v^{\rm adv}_{i,j-1/2}}{\Delta y}
\end{aligned} \tag{15.38}$$

Once the new states on both the x- and y-interfaces are constructed, we again upwind to find the final ρ state on each interface:

$$\rho^{n+1/2}_{i+1/2,j} = \mathcal{U}[u^{\text{adv}}_{i+1/2,j}](\rho^{n+1/2}_{i+1/2,j,L}, \rho^{n+1/2}_{i+1/2,j,R}) = \begin{cases} \rho^{n+1/2}_{i+1/2,j,L} & u^{\text{adv}}_{i+1/2,j} > 0 \\ \rho^{n+1/2}_{i+1/2,j,R} & u^{\text{adv}}_{i+1/2,j} < 0 \end{cases} \tag{15.39}$$

IV. *Do the conservative update of ρ*

Once the interface states are found, we can conservatively-difference the continuity equation:

$$\begin{aligned} \rho^{n+1}_{i,j} = \rho^{n}_{i,j} &- \frac{\Delta t}{\Delta x}\left(\rho^{n+1/2}_{i+1/2,j} u^{\text{adv}}_{i+1/2,j} - \rho^{n+1/2}_{i-1/2,j} u^{\text{adv}}_{i-1/2,j}\right) \\ &- \frac{\Delta t}{\Delta y}\left(\rho^{n+1/2}_{i,j+1/2} v^{\text{adv}}_{i,j+1/2} - \rho^{n+1/2}_{i,j-1/2} v^{\text{adv}}_{i,j-1/2}\right) \end{aligned} \tag{15.40}$$

V. *Update $\mathbf{U}^n$ to $\mathbf{U}^{n+1,\star}$*

VI. *Enforce the divergence constraint on $\mathbf{U}^{n+1}$*

For the final projection, we have cell-centered velocties. We define the divergence term as:

$$[\nabla \cdot (\beta_0 \mathbf{U})]^{\star}_{i,j} = \beta_{0_j} \frac{u^{\star}_{i+1,j} - u^{\star}_{i-1,j}}{2\Delta x} + \frac{\beta_{0_{j+1}} v^{\star}_{i,j+1} - \beta_{0_{j-1}} v^{\star}_{i,j-1}}{2\Delta y} \tag{15.41}$$

15.3.3 Timestep constraint

In addition to the advective timestep constraint, an additional constraint is needed if the velocity field is initialially zero. In [4], a constraint based on the buoyancy forcing is used. First, we compute the maxiumum buoyancy acceleration,

$$a_{\max} = \max\left\{\left|\frac{\rho' \mathbf{g}}{\rho}\right|\right\} \tag{15.42}$$

and then

$$\Delta t_{\text{force}} = \left(\frac{2\Delta x}{a_{\max}}\right)^{1/2} \tag{15.43}$$

This is based simply on $\Delta x = (1/2)at^2$—the time it takes for the buoyant force to accelerate a fluid element across a zone width.

15.3.4 Bootstrapping

First we need to make sure that the initial velocity field satisfies our constraint

Next we need to find $\nabla p'^{n-1/2}$ for the first step.

15.4 Combustion

Taking $p = p_0 + p'$, with $p_0 =$ constant, the system becomes:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \tag{15.44}$$

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U}\mathbf{U}) + \nabla p' = 0 \tag{15.45}$$

$$\nabla \cdot \mathbf{U} = S \tag{15.46}$$

15.4.1 Species

15.4.2 Constraint

Our constraint equation is $\nabla \cdot \mathbf{U} = S$. Decomposing the velocity field as

$$\mathbf{U}^\star = \mathbf{U}^d + \frac{1}{\rho}\nabla\phi \tag{15.47}$$

our Poisson equation can be defined by taking the divergence, and using $\nabla \cdot \mathbf{U}^d = S$, giving

$$\nabla \cdot \frac{1}{\rho}\nabla\phi = \nabla \cdot \mathbf{U}^\star - S \tag{15.48}$$

15.4.3 Solution Procedure

The general solution procedure is for a single step is:

- React for $\Delta t/2$
- Do the hydrodynamcis for Δt
  - Predict $\mathbf{U}$ to the interfaces
  - Enforce the divergence constraint on the interface $\mathbf{U}$'s (the MAC projection) to get $\mathbf{U}^{\mathrm{adv}}$.
  - Predict ρ to the interfaces
  - Do the conservative update of ρ
  - Update $\mathbf{U}^n$ to $\mathbf{U}^{n+1}$
  - Enforce the divergence constraint on $\mathbf{U}^{n+1}$
- React for $\Delta t/2$

Part VI

Code Descriptions

Appendix A

Using hydro_examples

A.1 Introduction

The hydro_examples codes are simple 1-d solvers written in python that illustrate many of the ideas in these nodes. They are used for making many of the figures found throughout (wherever the "⌨ hydro_examples: " note is found). Most are written for python 3.x.

A.2 Getting hydro_examples

The hydro_examples codes are hosted on github:
`https://github.com/zingale/hydro_examples`

There are a few ways to get them. The simplest way is to just clone from github on the commandline:

```
git clone https://github.com/zingale/hydro_examples
```

This will create a local git repo on your machine with all of the code. You can then run the scripts, "as is" without needing to go to github anymore. Periodically, there may be updates to the scripts, which you can obtain by issuing "`git pull`" in the `hydro_examples/` directory. Note that if you have made any local changes to the scripts, `git` may complain. The process of merging changes is descibed online in various places, e.g. "Resolving a merge conflict from the command line" from github.

Alternately, you can use github's web interface to fork it. Logon to github (or create an account) and click on the "Fork" icon on the hydro_examples page. You can then interact with your version of the repo as needed. This method will allow you to

push changes back to github, and, if you think they should be included in the main hydro_examples repo, issue a pull-request.

A.3 hydro_examples codes

The codes in hydro_examples are organized into directories named after the chapters in these notes. Each directory is self-contained. The following scripts are available:

- `advection/`
  - `advection.py`: a 1-d second-order linear advection solver with a wide range of limiters.
  - `fdadvect_implicit.py`: a 1-d first-order implicit finite-difference advection solver using periodic boundary conditions.
  - `fdadvect.py`: a 1-d first-order explicit finite-difference linear advection solver using upwinded differencing.
- `burgers/`
  - `burgers.py`: a 1-d second-order solver for the inviscid Burgers' equation, with initial conditions corresponding to a shock and a rarefaction.
- `compressible/`
  - `cfl.py`: a simple brute force script used to find the stability limit of a multidimensional advection discretization.
  - `euler.ipynb`: an Jupyter notebook using SymPy that derives the eigensystem for the primitive-variable form of the Euler equations.
  - `euler-generaleos.ipynb`: an Jupyter notebook using SymPy that derives the eigensystem for the primitive-variable form of various forms of Euler equations with a general EOS (with either (ρe) or γ_e augmenting the system) and for gray radiation hydrodynamics. Additionally, this notebook computes the full form of the interface states.
  - `riemann.py`: the main Riemann class for gamma-law Euler equations. The `RiemannProblem` class has methods to find the star state, plot the Hugoniot curves, and sample the solution.
  - `riemann-2shock.py`: draw the Hugoniot curves in the u-p plane for a pair of states that comprise a Riemann problem, *under the 2-shock approximation*.
  - `riemann-phase.py`: draw the Hugoniot curves in the u-p plane for a pair of states that comprise a Riemann problem.

  - `riemann-sod.py`: solve the Riemann problem for the Sod initial conditions and plot the exact solution.

- `diffusion/`

  - `diff_converge.py`: make a convergence plot (error vs. resolution) of different explicit and implicit diffusion methods.

  - `diffusion_explicit.py`: solve the constant-diffusivity diffusion equation explicitly. The method is first-order accurate in time, but second-order in space. A Gaussian profile is diffused—the analytic solution is also a Gaussian.

  - `diffusion_fo_implicit.py`: solve the constant-diffusivity diffusion equation implicitly. Backwards-difference time-discretization is used, resulting in a first-order-in-time method. Spatial discretization is second order. A Gaussian profile is diffused.

  - `diffusion_implicit.py`: solve the constant-diffusivity diffusion equation implicitly. Crank-Nicolson time-discretization is used, resulting in a second-order method. A Gaussian profile is diffused.

- `elliptic/`

  - `poisson_fft.py`: an FFT solver for a 2-d Poisson problem with periodic boundaries.

- `finite-volume/`

  - `conservative-interpolation.ipynb`: an Jupyter notebook using SymPy that derives conservative interpolants.

- `incompressible/`

  - `project.py`: a simple example of using a projection to recover a divergence-free velocity field.

- `multigrid/`

  - `mg_converge.py`: a convergence test of the multigrid solver. A Poisson problem is solved at various resolutions and compared to the exact solution. This demonstrates second-order accuracy.

  - `mg_test.py`: a simple driver for the multigrid solver. This sets up and solves a Poisson problem and plots the behavior of the solution as a function of V-cycle number.

  - `multigrid.py`: a multigrid class for cell-centered data. This implements pure V-cycles. A square domain with 2^N zones (N a positive integer) is required.

  - `patch1d.py`: a class for 1-d cell-centered data that lives on a grid. This manages the data, handles boundary conditions, and provides routines for prolongation and restriction to other grids.
  - `smooth.py`: solve a Poisson equation with smoothing only (but we still use the interface through the multigrid object).

- `multiphysics/`
  - `burgersvisc.py`: solve the viscous Burgers equation. The advective terms are treated explicitly with a second-order accurate method. The diffusive term is solved using an implicit Crank-Nicolson discretization. The overall coupling is second-order accurate.
  - `diffusion-reaction.py`: solve a diffusion-reaction equation that propagates a diffusive reacting front (flame). A simple reaction term is modeled. The diffusion is solved using a second-order Crank-Nicolson discretization. The reactions are evolved using the VODE ODE solver (via SciPy). The two processes are coupled together using Strang-splitting to be second-order accurate in time.

Appendix B

Using pyro

B.1 Introduction

pyro is a simple two-dimensional code that implements the core solvers described in these notes. It is written primarily in python, with some kernels in Fortran. Clarity of the methods is emphasized over performance.

B.2 Getting pyro

pyro can be downloaded from its github repository, `https://github.com/python-hydro/pyro2` as:

```
git clone https://github.com/python-hydro/pyro2
```

pyro uses the `h5py`, matplotlib, and numpy libraries. Several routines are written in Fortran and need to be compiled. The script `mk.sh` will build the packages. This requires that `f2py` is installed. By default, pyro will use python 3. The following sequence will setup pyro:

- Set the `PYRO_HOME` environment variable to point to the main `pyro2/` directory, e.g.,

  ```
  export PYRO_HOME=/path/to/pyro2
  ```

- Set the `PYTHONPATH` environment variable to point to the main `pyro2/` directory, e.g.,

  ```
  export PYTHONPATH="$PYTHONPATH:$PYRO_HOME"
  ```

- Build the Fortran source using the `mk.sh` script. In the `pyro2/` directory, simply do

```
./mk.sh
```

B.3 pyro solvers

pyro offers the following 2-d solvers:

- *advection*: an unsplit, second-order method for linear advection, following the ideas from Chapter 4.
- *advection_rk*: an alternate advection algorithm using method-of-lines time integration, again following the ideas from Chapter 4.
- *compressible*: an unsplit, second-order compressible hydrodynamics solver using the piecewise linear reconstruction discussed in Chapter 8.
- *compressible_rk*: an alternate compressible algorithm using method-of-lines time integration, using the piecewise linear reconstruction discussed in Chapter 8.
- *diffusion*: a second-order implicit diffusion solver, based on the ideas from Chapter 10.
- *incompressible*: a second-order incompressible hydrodynamics solver using a cell-centered approximate projection, as discussed in Chapter 14.
- *lm_atm*: a low-Mach number hydrodynamics solver for atmospheric flows, as discussed in S 15.3
- *multigrid*: a multigrid solver for constant-coefficient Helmholtz elliptic equations. This follows the ideas from Chapter 9, and is used by the diffusion and incompressible solvers.

B.4 pyro's structure

The structure of the code and descriptions of the various runtime parameters is found on the pyro webpage, `http://python-hydro.github.io/pyro2/`, and described in [86]. Here we provide a summary.

The grid structure is managed by the `patch.Grid2d` class. Data is that lives on the grid is contained in a `patch.CellCenterData2d` object. Methods are available to provide access to the data and fill the ghost cells. When accessing the data on the grid, a `array_indexer.ArrayIndexer` object is returned. This is a subclass of the normal NumPy `ndarray` that provides additional methods that are useful for interacting with finite-volume data. In particular, for an object `a`, you can do `a.ip(1)` to mimic

$a_{i+1,j}$. The Jupyter notebook mesh-examples.ipynb walks through these classes to demonstrate how they work.

Each pyro solver is its own python module. All but the multigrid solver represent time-dependent problems. Each of these provide a `Simulation` class that provides the routines necessary to initialize the solver, determine the timestep, and evolve the solution in time. Each solver has one or more sets of initial conditions defined in the solver's `problems/` directory.

All time-dependent problems are run through the `pyro.py` script. The general form is:

```
./pyro.py solver problem inputs
```

where *solver* is one of {`advection`, `advection_rk`, `compressible`, `compressible_rk`, `diffusion`, `incompressible`, `lm_atm`}, *problem* is one of the problems defined in the solver's `problems/` directory, and *inputs* is the name of an input file that defines the values of runtime parameter.

The possible runtime parameters and their defaults are defined in the `_defaults` files in the main directory and each solver and problem directory. Note that the inputs file need not be in the `pyro2/` directory. The solver's `problems/` directory will also be checked.

B.5 Running pyro

A simple Gaussian advection simulation is provided by the advection *smooth* problem. This is run as:

```
./pyro.py advection smooth inputs.smooth
```

As this is run, the solution will be visualized at each step, showing the progression of the simulation.

A list of the problems available for each solver is given in Table B.1. For the multigrid solver, there are scripts available in the `multigrid/` directory that illustrate its use.

B.6 Output and visualization

pyro uses HDF5 for I/O* The simple script `plot.py` in the `pyro/` directory can read in the data and plot the output. It will determine which solver was used for the run and do the visualization using that solver's methods.

*Previously a simple python `pickle` was done, but this was not very portable.

B.7 Testing

The script `test.py` provides an interface to pyro's unit and regression tests. Simply running it was `./test.py` will executate all of the tests. The unit tests are enabled through the pytest framework.

solver	section	problem	problem description
advection	§ 5.4.2	smooth	advect a smooth Gaussian profile
		tophat	advect a discontinuous tophat profile
advection_rk	§ 5.3	same as advection	
compressible	Ch. 8	advect	a Gaussian profile in the density field advected diagonally (in constant pressure background)
		bubble	a buoyant bubble in a stratified atmosphere
		kh	setup a shear layer to drive Kelvin-Helmholtz instabilities
		quad	2-d Riemann problem based on [68]
		rt	a simple Rayleigh-Taylor instability
		sedov	the classic Sedov-Taylor blast wave
		sod	the Sod shock tube problem
compressible_rk	Ch. 8	same as compressible	
diffusion	§ 10.4	gaussian	diffuse an initial Gaussian profile
incompressible	Ch. 14	converge	A simple incompressible problem with known analytic solution.
		shear	a doubly-periodic shear layer
lm_atm	Ch. 15.3	bubble	a buoyant bubble in a stratified atmosphere

Table B.1: pyro solvers and their distributed problems. The relevant section in these notes that describes the implementation is also given.

Appendix C

Using hydro1d

C.1 Introduction

hydro1d is a one-dimensional compressible hydrodynamics code written in modern Fortran. In particular, it implements the piecewise parabolic method described in § 8.2.3, in both Cartesian and spherical geometries. It assumes a gamma-law equation of state and supports gravity.

C.2 Getting hydro1d

hydro1d can be downloaded from its github repository, `https://github.com/zingale/hydro1d` as:

```
git clone https://github.com/zingale/hydro1d
```

Some details on the code can be found on its webpage: `http://zingale.github.io/hydro1d/` . As with the other codes detailed here, if you wish to contribute, fix bugs, etc., you can create issues or pull requests through the code's github page.

C.3 hydro1d's structure

There a few major data structures in hydro1d, defined in the `grid.f90` file:

- `grid_t` : this holds the grid information, with fields for the low and high indicies of the valid domain, and coordinate information (including interface areas and volumes, to support spherical geometry).
- `gridvar_t` : data that lives on a grid. This holds both a `data` array and the `grid_t` that it is defined on.

- `gridedgevar_t` : like `gridvar_t`, but for data defined at the interfaces between zones.

The integer keys defined in the `variables_module` allow you to index the difference state fields in the data arrays of the grid variables.

C.4 Running hydro1d

Each problem is in it's own directory, and the code is built and run there. For example, to run the Sod shock tube problem, do:

```
cd hydro1d/sod
make
./hydro1d inputs-sod-xp
```

As the code is built, object files and modules will be output into the `_build` subdirectory. `make realclean` will clean up the objects. Things are setup for `gfortran` by default—you will need to edit the `Ghydro.mak` with different options for different compilers. Some bits of Fortran 2003 and 2008 are used, so an up-to-date compiler is needed.

A number of runtime options can be set—these are listed in `params.f90` and `probparams.f90` (the latter is problem-specific parameters).

C.5 Problem setups

The following problem setups are provided:

- `advect` : a simple advection test where a density profile is advected through periodic boundaries, in a uniform pressure medium (to suppress dynamics)
- `hse` : a simple 1-d atmosphere in hydrostatic equilibrium. This test works to see if the atmosphere stays in HSE.
- `sedov` : a 1-d spherical Sedov explosion.
- `sod` : a shock tube setup for testing against exact Riemann solvers.

Bibliography

[1] A. Almgren. A new look at the pseudo-incompressible solution to Lamb's problem of hydrostatic adjustment. *J Atmos Sci*, 57:995–998, April 2000. (Cited on pages 244 and 247)

[2] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, and M. Zingale. CASTRO: A New Compressible Astrophysical Solver. I. Hydrodynamics and Self-gravity. *Astrophys J*, 715:1221–1238, June 2010. (Cited on pages 123 and 153)

[3] A. S. Almgren, J. B. Bell, and W. Y. Crutchfield. Approximate projection methods: Part I. Inviscid analysis. *SIAM J Sci Comput*, 22(4):1139–59, 2000. (Cited on pages 236 and 237)

[4] A. S. Almgren, J. B. Bell, A. Nonaka, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. III. Reactions. *Astrophys J*, 684:449–470, 2008. (Cited on pages 220, 240, 244, and 251)

[5] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. I. Hydrodynamics. *Astrophys J*, 637:922–936, February 2006. (Cited on page 244)

[6] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. II. Energy Evolution. *Astrophys J*, 649:927–938, October 2006. (Cited on pages 212, 240, 244, and 247)

[7] A. S. Almgren, J. B. Bell, and W. G. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM J Sci Comput*, 17(2):358–369, March 1996. (Cited on page 229)

[8] A. Aspden, N. Nikiforakis, S. Dalziel, and J. B. Bell. Analysis of implicit LES methods. *Communications in Applied Mathematics and Computational Science*, 3(1):103–126, 2008. (Cited on page 217)

[9] J. B. Bell, P. Colella, and H. M. Glaz. A Second Order Projection Method for the Incompressible Navier-Stokes Equations. *J Comput Phys*, 85:257, December 1989. (Cited on page 227)

[10] J. B. Bell, P. Colella, and L. H. Howell. An efficient second-order projection method for viscous incompressible flow. In *Proceedings of the Tenth AIAA Computational Fluid Dynamics Conference*, pages 360–367. AIAA, June 1991. see also: https://seesar.lbl.gov/anag/publications/colella/A_2_10.pdf. (Cited on page 227)

[11] J. B. Bell, C. N. Dawson, and G. R. Shubin. An unsplit, higher order Godunov method for scalar conservation l aws in multiple dimensions. *J Comput Phys*, 74:1–24, 1988. (Cited on pages 78 and 131)

[12] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. Zingale. Direct Numerical Simulations of Type Ia Supernovae Flames. II. The Rayleigh-Taylor Instability. *Astrophys J*, 608:883–906, June 2004. (Cited on page 218)

[13] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. A. Zingale. Adaptive low Mach number simulations of nuclear flame microphysics. *J Comput Phys*, 195(2):677–694, 2004. (Cited on pages 112, 199, 240, and 244)

[14] J. B. Bell and D. L. Marcus. A second-order projection method for variable-density flows. *J Comput Phys*, 101(2):334 – 348, 1992. (Cited on page 239)

[15] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J Comput Phys*, 82(1):64–84, May 1989. (Cited on page 33)

[16] G. Boffetta and R. E. Ecke. Two-Dimensional Turbulence. *Annual Review of Fluid Mechanics*, 44:427–451, January 2012. (Cited on page 218)

[17] W. L. Briggs, V-E. Henson, and S. F. McCormick. *A Mutigrid Tutorial, 2nd Ed.* SIAM, 2000. (Cited on pages 168, 172, and 175)

[18] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. VODE: A variable coefficient ode solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989. (Cited on page 215)

[19] G. L. Bryan, M. L. Norman, J. M. Stone, R. Cen, and J. P. Ostriker. A piecewise parabolic method for cosmological hydrodynamics. *Computer Physics Communications*, 89:149–168, August 1995. (Cited on page 151)

[20] G. D. Byrne and A. C. Hindmarsh. Stiff ODE Solvers: A Review of Current and Coming Attractions. *UCRL-94297 Preprint*, 1986. (Cited on page 18)

[21] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968. (Cited on page 227)

[22] A. R. Choudhuri. *The physics of fluids and plasmas : an introduction for astrophysicists /*. November 1998. (Cited on page 94)

[23] P. Colella. A direct Eulerian MUSCL scheme for gas dynamics. *SIAM J Sci Stat Comput*, 6(1):104–117, 1985. (Cited on page 55)

[24] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J Comput Phys*, 87:171–200, March 1990. (Cited on pages 50, 55, 66, 67, 69, 121, and 231)

[25] P. Colella and H. M. Glaz. Efficient solution algorithms for the Riemann problem for real gases. *J Comput Phys*, 59:264–289, June 1985. (Cited on pages 104, 120, 128, and 152)

[26] P. Colella and E. G. Puckett. *Modern Numerical Methods for Fluid Flow*. unpublished manuscript. obtained from http://www.amath.unc.edu/Faculty/minion/class/puckett/. (Cited on page 181)

[27] P. Colella and M. D. Sekora. A limiter for PPM that preserves accuracy at smooth extrema. *J Comput Phys*, 227:7069–7076, July 2008. (Cited on page 126)

[28] P. Colella and P. R. Woodward. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *J Comput Phys*, 54:174–201, September 1984. (Cited on pages 121, 122, 130, and 145)

[29] M. S. Day and J. B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4(4):535–556, 2000. (Cited on pages 240 and 244)

[30] M. de Val-Borro, R. G. Edgar, P. Artymowicz, P. Ciecielag, P. Cresswell, G. D'Angelo, E. J. Delgado-Donate, G. Dirksen, S. Fromang, A. Gawryszczak, H. Klahr, W. Kley, W. Lyra, F. Masset, G. Mellema, R. P. Nelson, S.-J. Paardekooper, A. Peplinski, A. Pierens, T. Plewa, K. Rice, C. Schäfer, and R. Speith. A comparative study of disc-planet interaction. *Mon Not R Astron Soc*, 370:529–558, August 2006. (Cited on page 4)

[31] G. Dimonte, D. L. Youngs, A. Dimits, S. Weber, M. Marinak, S. Wunsch, C. Garasi, A. Robinson, M. J. Andrews, P. Ramaprabhu, A. C. Calder, B. Fryxell, J. Biello, L. Dursi, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. Timmes, H. Tufo, Y.-N. Young, and M. Zingale. A comparative study of the turbulent Rayleigh-Taylor instability using high-resolution three-dimensional numerical simulations: The Alpha-Group collaboration. *Physics of Fluids*, 16:1668–1693, May 2004. (Cited on page 4)

[32] D. R. Durran. Improving the anelastic approximation. *J Atmos Sci*, 46(11):1453–1461, 1989. (Cited on page 244)

[33] C. S. Frenk, S. D. M. White, P. Bode, J. R. Bond, G. L. Bryan, R. Cen, H. M. P. Couchman, A. E. Evrard, N. Gnedin, A. Jenkins, A. M. Khokhlov, A. Klypin, J. F. Navarro, M. L. Norman, J. P. Ostriker, J. M. Owen, F. R. Pearce, U.-L. Pen, M. Steinmetz, P. A. Thomas, J. V. Villumsen, J. W. Wadsley, M. S. Warren, G. Xu, and G. Yepes. The Santa Barbara Cluster Comparison Project: A Comparison of Cosmological Hydrodynamics Solutions. *Astrophys J*, 525:554–582, November 1999. (Cited on page 4)

[34] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *Astrophys J Supplement*, 131:273–334, November 2000. (Cited on pages 67, 145, and 215)

[35] A. Garcia. *Numerical Methods for Physics, 2nd Edition*. Addison-Wesley, 1999. (Cited on pages 5, 10, and 16)

[36] G.A. Gerolymos, D. Sénéchal, and I. Vallet. Very-high-order WENO schemes. *Journal of Computational Physics*, 228(23):8481–8524, dec 2009. (Cited on page 75)

[37] S. K. Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Mat. Sb. (N.S.)*, 47(89):271–306, 1959. (Cited on page 117)

[38] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. (Cited on page 5)

[39] S. Gottlieb and C.-W. Shu. Total Variation Diminishing Runge-Kutta Schemes. *ICASE Report No. 96-50, NASA Langley Research Center*, July 1996. (Cited on page 71)

[40] C. J. Hansen, S. D. Kawaler, and V. Trimble. *Stellar interiors : physical principles, structure, and evolution*. 2004. (Cited on pages 96 and 211)

[41] K. Heitmann, Z. Lukić, P. Fasel, S. Habib, M. S. Warren, M. White, J. Ahrens, L. Ankeny, R. Armstrong, B. O'Shea, P. M. Ricker, V. Springel, J. Stadel, and H. Trac. The cosmic code comparison project. *Computational Science and Discovery*, 1(1):015003, October 2008. (Cited on page 166)

[42] J. R. Kamm and F. X. Timmes. On Efficient Generation of Numerically Robust Sedov Solutions. 2007. Los Alamos preprint la-ur-07-2849, `http://cococubed.asu.edu/papers/la-ur-07-2849.pdf`. (Cited on page 140)

[43] Rupert Klein and Olivier Pauluis. Thermodynamic consistency of a pseudoincompressible approximation for general equations of state. *J Atmos Sci*, March 2012. (Cited on pages 244 and 248)

[44] C. B. Laney. *Computational Gasdynamics*. Cambridge, 1998. (Cited on page 53)

[45] R. J. LeVeque. Wave Propagation Algorithms for Multidimensional Hyperbolic Systems. *Journal of Computational Physics*, 131:327–353, March 1997. (Cited on page 148)

[46] Randall J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. (Cited on pages 22, 54, 55, 78, 84, 89, 94, 98, and 145)

[47] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Classics in Applied Mathematics. SIAM, Society of Industrial and Applied Mathematics, 2007. (Cited on page 195)

[48] C. M. Malone, A. Nonaka, A. S. Almgren, J. B. Bell, and M. Zingale. Multidimensional Modeling of Type I X-ray Bursts. I. Two-dimensional Convection Prior to the Outburst of a Pure ^{4}He Accretor. *Astrophys J*, 728:118, February 2011. (Cited on page 199)

[49] Len G. Margolin and William J. Rider. A rationale for implicit turbulence modelling. *International Journal for Numerical Methods in Fluids*, 39(9):821–841, 2002. (Cited on page 217)

[50] D. F. Martin and P. Colella. A Cell-Centered Adaptive Projection Method for the Incompressible Euler Equations. *Journal of Computational Physics*, 163:271–312, September 2000. (Cited on page 229)

[51] S. May, A. J. Nonaka, A. S. Almgren, and J. B. Bell. An unsplit, higher order Godunov method using quadratic reconstruction for advection in multiple dimensions. *Communications in Applied Mathematics and Computational Science*, 6(1), 2011. (Cited on page 78)

[52] P. McCorquodale and P. Colella. A high-order finite-volume method for conservation laws on locally refined grids. *Communication in Applied Mathematics and Computational Science*, 6(1):1–25, 2011. (Cited on pages 71, 151, and 220)

[53] G. H. Miller and P. Colella. A Conservative Three-Dimensional Eulerian Method for Coupled Solid-Fluid Shock Capturing. *J Comput Phys*, 183:26–82, November 2002. (Cited on pages 120, 121, and 134)

[54] M. L. Minion. A Projection Method for Locally Refined Grids. *J Comput Phys*, 127:158–177, 1996. (Cited on page 239)

[55] J. J. Monaghan. An introduction to SPH. *Computer Physics Communications*, 48:89–96, January 1988. (Cited on page 25)

[56] M. Newman. *Computational Physics*. CreateSpace Independent Publishing Platform, 2012. (Cited on page 5)

[57] A. Nonaka, A. S. Almgren, J. B. Bell, M. J. Lijewski, C. M. Malone, and M. Zingale. MAESTRO: An adaptive low mach number hydrodynamics algorithm for stellar flows. *Astrophys J Supplement*, 188:358–383, 2010. (Cited on pages 244 and 247)

[58] M. Omang, S. Børve, and J. Trulsen. SPH in spherical and cylindrical coordinates. *Journal of Computational Physics*, 213:391–412, March 2006. (Cited on page 141)

[59] E. S. Oran and J. B. Boris. *Numerical Simulation of Reactive Flow*. Cambridge University Press, 2nd edition, 2005. (Cited on page 209)

[60] T. Pang. *An Introduction to Computational Physics, 2nd Edition*. Cambridge, 2006. (Cited on page 5)

[61] R. B. Pember, L. H. Howell, J. B. Bell, P. Colella, W. Y. Crutchfield, W. A. Fiveland, and J. P. Jessee. An adaptive projection method for unsteady low-Mach number combustion. *Comb. Sci. Tech.*, 140:123–168, 1998. (Cited on pages 240 and 244)

[62] Norbert Peters. *Turbulent Combustion*. Cambridge University Press, 2000. (Cited on page 209)

[63] T. Plewa and E. Müller. The consistent multi-fluid advection method. *Astron Astrophys*, 342:179–191, February 1999. (Cited on page 214)

[64] R. D. Richtmyer and K. W. Morton. *Different Methods for Initial-Value Problems*. Krieger Publishing Company, 2nd edition, 1994. (Cited on page 186)

[65] W. J. Rider. Approximate projection methods for incompressible flow: Implementation, variants and robustness. Technical report, LANL UNCLASSIFIED REPORT LA-UR-94-2000, LOS ALAMOS NATIONAL LABORATORY, 1995. (Cited on page 237)

[66] J. Saltzman. An Unsplit 3D Upwind Method for Hyperbolic Conservation Laws. *J Comput Phys*, 115:153–168, November 1994. (Cited on pages 67, 121, 126, and 134)

[67] W. Schmidt. Large Eddy Simulations in Astrophysics. *Living Reviews in Computational Astrophysics*, 1:2, December 2015. (Cited on page 217)

[68] C. W. Schulz-Rinne, J. P. Collins, and H. M. Glaz. Numerical Solution of the Riemann Problem for Two-Dimensional Gas Dynamics. *SIAM J Sci Comput*, 14(6):1394–1414, 1993. (Cited on page 263)

[69] Carsten W. Schulz-Rinne, James P. Collins, and Harland M. Glaz. Numerical solution of the riemann problem for two-dimensional gas dynamics. *SIAM Journal on Scientific Computing*, 14(6):1394–1414, 1993. (Cited on pages xii, 148, and 150)

[70] L. I. Sedov. *Similarity and Dimensional Methods in Mechanics*. Academic Press, 1959. translated from the 4th Russian Ed. (Cited on page 139)

[71] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes II. *J Comput Phys*, 83:32–78, 1989. (Cited on pages 67 and 71)

[72] Chi-wang Shu. Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws. Technical Report 97, 1997. (Cited on page 77)

[73] F. H. Shu. *Physics of Astrophysics, Vol. II*. University Science Books, 1992. (Cited on pages 94 and 211)

[74] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation la ws. *J Comput Phys*, 27:1–31, April 1978. (Cited on pages 98, 138, and 139)

[75] V. Springel. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *Mon Not R Astron Soc*, 401:791–851, January 2010. (Cited on page 33)

[76] J. M. Stone, T. A. Gardiner, P. Teuben, J. F. Hawley, and J. B. Simon. Athena: A New Code for Astrophysical MHD. *Astrophys J Suppl S*, 178:137–177, September 2008. (Cited on pages 95 and 127)

[77] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):pp. 506–517, 1968. (Cited on pages 64 and 204)

[78] F. X. Timmes. Integration of Nuclear Reaction Networks for Stellar Hydrodynamics. *Astrophys J Supplement*, 124:241–263, September 1999. (Cited on page 18)

[79] F. X. Timmes. Integration of Nuclear Reaction Networks for Stellar Hydrodynamics. *APJ:sup*, 124:241–263, September 1999. (Cited on page 215)

[80] F. X. Timmes and S. E. Woosley. The conductive propagation of nuclear flames. I - Degenerate C + O and O + NE + MG white dwarfs. *Astrophys J*, 396:649–667, September 1992. (Cited on page 218)

[81] V.A. Titarev and E.F. Toro. Finite-volume {WENO} schemes for three-dimensional conservation laws. *Journal of Computational Physics*, 201(1):238 – 260, 2004. (Cited on pages 71 and 151)

[82] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, 1997. (Cited on pages 55, 78, 95, 98, 100, 139, and 232)

[83] Geoffrey M. Vasil, Daniel Lecoanet, Benjamin P. Brown, Toby S. Wood, and Ellen G. Zweibel. Energy conservation and gravity waves in sound-proof treatments of stellar interiors. ii. lagrangian constrained analysis. *Astrophys J*, 773:169–, 2013. (Cited on page 248)

[84] Natalia Vladimirova, V. Gregory Weirs, and Lenya Ryzhik. Flame capturing with an advection-reaction-diffusion model. *Combustion Theory and Modelling*, 10(5):727–747, 2006. (Cited on page 205)

[85] S. Yakowitz and F. Szidarovszky. *An Introduction to Numerical Computations, 2nd edition*. Prentice Hall, 1989. (Cited on page 8)

[86] M. Zingale. pyro: A teaching code for computational astrophysical hydrodynamics. *Astronomy and Computing*, 2014. accepted for publication. (Cited on pages xviii and 260)

[87] M. Zingale, L. J. Dursi, J. ZuHone, A. C. Calder, B. Fryxell, T. Plewa, J. W. Truran, A. Caceres, K. Olson, P. M. Ricker, K. Riley, R. Rosner, A. Siegel, F. X. Timmes, and N. Vladimirova. Mapping Initial Hydrostatic Models in Godunov Codes. *Astrophys J Supplement*, 143:539–565, December 2002. (Cited on pages 131 and 218)

[88] M. Zingale and M. P. Katz. On the Piecewise Parabolic Method for Compressible Flow With Stellar Equations of State. *Astrophys J Supplement*, 216:31, February 2015. (Cited on page 139)