CALIFORNIA INSTITUTE OF TECHNOLOGY
Division of Physics, Mathematics, and Astronomy

Ay190 – Computational Astrophysics

Christian D. Ott and Andrew Benson
cott@tapir.caltech.edu, abenson@tapir.caltech.edu
February 17, 2012

Document under development.
2012-02-17 10:47:20 -0800 (Fri, 17 Feb 2012)
Revision 274
Last changed by cott

Table of Contents

Chapter I

Introduction

These lecture notes were developed as part of the Caltech Astrophysics course Ay190 – Computational Astrophysics. This course was first taught in the winter term 2010/2011 by Christian Ott and Andrew Benson.

The goal of this course is to briefly introduce the basics of numerical computation and then focus on various applications of numerical methods in astronomy and astrophysics.

The parts of these lecture notes dealing with the basics of numerical computation were heavily influenced by a set of lecture notes by Ian Hawke at the University of Southampton.

I.1 Literature

Unfortunately, there is no good single reference for Computational Astrophysics. We have personally used the following books for inspiration for Ay190:

- *Numerical Methods in Astrophysics, An Introduction* by P. Bodenheimer, G. P. Laughlin, M. Rozyczka, and H. Yorke. Taylor & Francis, New York, NY, USA, 2007.
- *Numerical Mathematics and Computing*, 6th edition, by W. Cheney and D. Kincaid, Thomson Brooks/Cole, Belmont, CA, USA, 2008.
- *Finite Difference Methods for Ordinary and Partial Differential Equations*, by R. Leveque, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007.
- *An Introduction to Computational Physics*, 2nd edition, by Tao Pang, Cambridge University Press, Cambridge, UK, 2006
- *Numerical Recipes*, 3rd. edition, by W. H. Press, S. Teukolsky, W. Vetterling, and B. Flannery, Cambridge University Press, Cambridge, UK, 2007.

I.2 Acknowledgements

CDO wishes to thank Ian Hawke for sharing his class notes with him for inspiration.

The authors are indebted to Mislav Balokovic, Kristen Boydstun, Yi Cao, Trevor David, Ben Montet, and Arya Farahi for spotting typos and factual errors.

I.3 Future Additions

The idea of these lecture notes is that they will be continuously improved and expanded with more background material, more details, figures, and example problems. In this section we summarize concrete plans of what will be added in the near future.

- ODE: More details on stiff equations and stability.
- ODE integrators: Crank-Nicholson implicit integrator.
- PDEs/ODEs: More on the Method of Lines. Studies of the wave equation, parabolic, and elliptic PDEs.
- Application: Stellar structure with Henyey Solver.

Chapter II

Fundamentals: The Basic of Numerical Computation

II.1 Computers & Numbers

II.1.1 Floating Point Numbers

Computers have finite memory and compute power. Hence, real numbers must be represented as floating point (FP) numbers with finite precision. Is this a problem? Yes, because small approximation errors can in principle (and in unlucky circumstances) be amplified and grow out of bound.

$$\pm\, 0.a_1a_2a_3a_4 \ldots a_m \times 10^c \ , \tag{II.1.1}$$

where $\pm 0.a_1a_2a_3a_4 \ldots a_m$ is the mantissa, the integer c is the exponent, and the interger 10 is the base of the FP number. The digits a_i are the *significant digits* and the number of significant digits is the FP precision m.

In general, a real number x can be expressed as a FP number $\overline{x}$ plus an absolute FP error (*round off* error) $e(x)$:

$$\underbrace{x}_{\in \mathbb{R}} \;=\; \underbrace{\overline{x}}_{\text{FP number}} \;+\; \underbrace{e(x)}_{\text{absolute FP error}} . \tag{II.1.2}$$

Converting a real nunber x to a floating point number $\overline{x}$ is accomplished either by truncation or (better) rounding.

Example: $m = 4$

$$\begin{aligned} x &= \pi = 3.141592653589... \ . \\ \overline{x} &= 3.141 + e(x) \ ; \ \ e(x) = \pi - \overline{x} \ . \end{aligned}$$

In computer terminology, one frequently speaks of *single precision* and *double precision* numbers. Single precision generally corresponds to 32-bit FP numbers with $m \simeq 7$ and double precision generally stands for 64-bit FP numbers with $m \simeq 14 - 16$. For reasons that we will see later, FP data types in computers actually have a mantissa of size $\geq 2m$, but their accuracy is still limited to m digits.

In addition, there are limits to the maximum and minimum number that can be stored in a FP variable. This is controlled by the maximum size of the exponent. For standard base-10 numbers,

the maximum (minimum) number is of $\mathcal{O}(10^{38})$ ($\mathcal{O}(10^{-37})$) and $\mathcal{O}(10^{308})$ ($\mathcal{O}(10^{-307})$) for single and double precision, respectively.

II.1.2 Errors

We define two kinds of errors

$$\begin{array}{rrll} \text{absolute error:} & E_a(x) & = e(x) & = x - \overline{x}\ , \\ \text{relative error:} & E_r(x) & = \frac{x-\overline{x}}{x}\ . & \end{array} \tag{II.1.3}$$

II.1.2.1 Errors & Floating Point Arithmetic

The basic arithmetic operations are $x + y$, $x - y$, $x \cdot y$, and $\frac{x}{y}$. We must investigate how errors propagate in these operations.

$x + y$:

$$x + y = \overline{x} + \overline{y} + e(x) + e(y)\,, \tag{II.1.4}$$

with $x = \overline{x} + e(x)$ and $y = \overline{y} + e(y)$. $e(x)$ and $e(y)$ are the absolute round-off error for x and y, respectively.

Note that:

(a) $\overline{x+y} \neq \overline{x} + \overline{y}$
Example: $m = 3$

$$\begin{array}{rcl} x = 1.313 & \rightarrow & \overline{x} = 1.31 \\ y = 2.147 & \rightarrow & \overline{y} = 2.14 \end{array} \rightarrow \overline{x} + \overline{y} = 3.45\ ,$$

but

$$x + y = 3.460 \rightarrow \overline{x+y} = 3.46\ .$$

(b) $x + y$ could be larger (or smaller) than the largest (smallest) number that can be represented. This leads to an *overflow* or *underflow*.

(c) Addition of FP numbers is not associative. So order matters:

$$\overline{\overline{(\overline{x} + \overline{y})} + \overline{z}} \neq \overline{\overline{x} + \overline{(\overline{y} + \overline{z})}}\,. \tag{II.1.5}$$

Exercise II.1.1 (FP addition is not associative)
Show by contradiction that the assumption that FP is associative is false.

$x - y$:

$$x - y = \overline{x} - \overline{y} + e(x) - e(y)\,, \tag{II.1.6}$$

which can be problematic when x and y are nearly equal. Example for $m = 3$:

$$\begin{aligned} &x = 42.102\,,\ y = 42.043\,,\\ &x - y = 0.059\,,\\ &\overline{x} - \overline{y} = 42.1 - 42.0 = 0.1\ . \end{aligned}$$

$x \cdot y$:

$$x \cdot y = \overline{x} \cdot \overline{y} + \overline{y} \cdot e(x) + \overline{x} \cdot e(y) + \underbrace{e(x) \cdot e(y)}_{\text{usually small}} \tag{II.1.7}$$

$\overline{x} \cdot \overline{y}$ is at most $2m$ digits long. This is why computers use $2m$ digits for computations on m-digit numbers. The result is then rounded to m digits.

$\frac{x}{y}$:

$$\frac{x}{y} = \frac{\overline{x} + e(x)}{\overline{y} + e(y)} \overset{\text{expansion}}{=} \frac{\overline{x}}{\overline{y}} + \frac{e(x)}{\overline{y}} - \frac{\overline{x} e(y)}{\overline{y}^2} + \mathcal{O}(e(x) \cdot e(y))\,. \tag{II.1.8}$$

The expansion shows that the error terms will grow immensely as $y \to 0$. Dividing by small numbers is a thing to avoid.

II.1.3 Stability of a Calculation

A numerical calculation is unstable if small errors made at one stage of the process are magnified in subsequent steps and seriously degrade the accuracy of the overall solution.

A very simple example for $m = 3$:

$$\begin{aligned} &(1)\quad \overline{y} = \overline{\sqrt{x}}\ ,\\ &(2)\quad \overline{y} = \overline{\sqrt{\overline{\overline{x} + 1} - 1}}\,. \end{aligned}$$

If we were not dealing with FP numbers, the results of (1) and (2) would be identical.

Let us now choose $\overline{x} = x = 0.02412 = 2.41 \times 10^{-2}$. Then

$$(1)\quad \overline{\sqrt{\overline{x}}} = 0.155\,,$$

but

$$\begin{aligned} (2)\quad &\overline{\overline{x} + 1} = 1.02\,,\\ &\overline{\overline{\overline{x} + 1} - 1} = 0.02\,,\\ &\overline{\sqrt{0.02}} = 0.141\ . \end{aligned}$$

The relative error between (1) and (2) is 9% and is due purely to additional unnecessary operations that amplified the FP error!

Exercise II.1.2 (An unstable Calculation)

Consider the following sequence and recurrence relation:

$$x_0 = 1\,,\ x_1 = \frac{1}{3}\,,\ x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1}\,,$$

which is equivalent to

$$x_n = \left(\frac{1}{3}\right)^n\,.$$

Implement the recurrence relation for $n = 0, ..., 15$ using *single precision* FP numbers and compute absolute and relative error with respect to the closed-form expression. How big are relative and absolute error at $n = 15$?

Now do the same for $x_1 = 4$, and compare it to $x_n = 4^n$. Go to $n = 20$. Why is this calculation stable? Should absolute or relative error be used to measure accuracy and stability?

II.2 Finite Differences

II.2.1 Forward, Backward, and Central Finite Difference Approximations

We will often need to numerically find the derivative of a function $f(x)$,

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}(x) \,, \quad \text{where} \quad \Delta f = f\left(x + \frac{\Delta x}{2}\right) - f\left(x - \frac{\Delta x}{2}\right) \,. \tag{II.2.1}$$

Now let us assume $\delta x/2 = h$. We can use Taylor's theorem to find an approximation for f' at x_0, namely:

$$\begin{aligned} f(x_0 + h) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} h^n \\ &= f(x_0) + h f'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(x_0) + \mathcal{O}(h^4) \,. \end{aligned} \tag{II.2.2}$$

By truncating at the h^2 term, we get

$$f(x_0 + h) = f(x_0) + h f'(x_0) + \mathcal{O}(h^2) \,, \tag{II.2.3}$$

and can solve for $f'(x_0)$:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + \mathcal{O}(h) \,. \tag{II.2.4}$$

This is called the *forward difference* estimate for f'. In other words, the slope of $f(x)$ at x_0 is approximated by the straight line through $(x_0, f(x_0))$ and $(x_0 + h, f(x_0 + h))$.

The expression $\mathcal{O}(h)$ on the far right of Eq. (II.2.4) indicates that the error of our estimate decreases linearly ($\propto h$) with step size h. Hence, we call this approximation *first order* (in h).

Now let us consider $f(x_0 - h)$,

$$f(x_0 - h) = f(x_0) - h f'(x_0) + \frac{h^2}{2} f''(x_0) - \frac{h^3}{6} f'''(x_0) + \mathcal{O}(h_0^4) \,, \tag{II.2.5}$$

which gets us the first-order *backward difference* estimate for $f'(x)$ at x_0:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \mathcal{O}(h) \,. \tag{II.2.6}$$

Now, subtracting Eq. (II.2.6) from Eq. (II.2.4), we get

$$f(x_0 + h) - f(x_0 - h) = 2h f'(x_0) + \frac{h^3}{3} f'''(x_0) + \mathcal{O}(h^4) \,, \tag{II.2.7}$$

and, because the h^2 term has vanished,

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \mathcal{O}(h^2) \,. \tag{II.2.8}$$

This is the *central difference* estimate for $f'(x)$ at x_0.

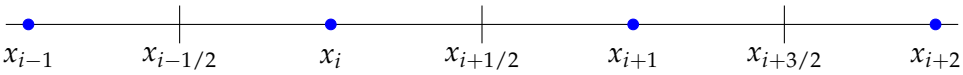

Figure II.2.1: Basic logical grid setup in 1D. Grid cell centers are marked with a filled circle. Cell interfaces are marked by vertical lines.

Exercise II.2.1 (Finite-Difference Estiate of the Second Derivative)

Use the methods demonstrated in this section to derive a second-order (central) expression for the second derivative of a function $f(x)$ at x_0.

II.2.2 Finite Differences on evenly and unevenly spaced Grids

So far, we have simply assumed that the step size h is constant between discrete nodes x_i at which we know the values of function $f(x)$.

Figure II.2.1 shows a typical basic grid setup in a one-dimensional problem. A computational cell has cell center x_i and is demarkated by $x_{i-1/2}$ on the left and $x_{i+1/2}$ on the right. If the grid is evenly spaced (on speaks of an *equidistant grid*) then the distance between cell centers $\Delta x = x_{i+1} - x_i$ is constant throughout the grid. In this case, we may, for example, express the centered derivative of a function $f(x)$ at x_i by

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + \mathcal{O}(\Delta x^2)\,. \tag{II.2.9}$$

Equidistant grids are the conceptually simplest way of discretizing a problem. They are, however, in many cases by far not the most efficient way, since many problems need more resolution in some parts of their domain than in others. A great example is the modeling of stellar collapse of an iron core to a neutron star. To resolve the steep density gradients near the neutron star, a resolution of order 100 m is required within ~ 30 km of the origin, while a cell size of order 10 km is sufficient at radii above $\sim$1000 km. One efficient way of dealing with these different resolution requirements is to increase the radial extent of computational cells with increasing radius.

Taking numerical derivatives on such unevenly spaced (or *non-equidistant*) grids is more complicated, but still doable. Obviously, first-order estimates are trivial, since they only involve two points. In the following, we derive a second-order estimate of $f'(x)$ for non-equidistant grids that reduces to the standard central difference estimate in the limit of equidistant grids.

We want to evaluate $f'(x)$ at $x = x_i$. We define $h_1 = x_i - x_{i-1}$ and $h_2 = x_{i+1} - x_i$. Then,

$$\begin{aligned} f(x_i + h_2) &= f(x_i) + h_2 f'(x_i) + \frac{h_2^2}{2} f''(x_i) + \mathcal{O}(h_2^3)\,, \\ f(x_i - h_1) &= f(x_i) - h_1 f'(x_i) + \frac{h_1^2}{2} f''(x_i) + \mathcal{O}(h_1^3)\,. \end{aligned} \tag{II.2.10}$$

This allows us to eliminate the $f''(x_i)$ term and to solve for $f'(x_i)$:

$$f'(x_i) = \frac{h_1}{h_2(h_1+h_2)} f(x_{i+1}) - \frac{h_1 - h_2}{h_2 h_1} f(x_i) - \frac{h_2}{h_1(h_1+h_2)} f(x_{i-1}) \,. \tag{II.2.11}$$

It is trivial to see that this reduces to the standard central difference estimate (Eq. II.2.8) if $h_1 = h_2$.

Exercise II.2.2 (Second Derivative Estimate on non-equidistant Grids)

Derive a second-order estimate for the second derivative of a function $f(x)$ at x_i on a non-equidistant grid.

II.2.3 Convergence

A numerical method to solve a problem with a true solution $y(x)$ is said to be convergent if the discrete solution $y(x;h)$ approaches the true solution for vanishing discretization step size h:

$$\lim_{h\to 0} y(x;h) = y(x) \,. \tag{II.2.12}$$

In other words, if the resolution is increased, the numerical result converges to the true result.

Three important things:

1. For a numerical scheme that is n-th order accurate, a decrease in step size by a factor m leads to a decrease of the deviation from the true solution by a factor m^n. So, for $n = 2$, the error is reduced by a factor of 4 if the resolution is doubled, while for $= 1$, the error goes down only by a factor of 2.

   We can express this more mathematically by defining the convergence factor

   $$Q = \frac{|y(x;h_2) - y(x)|}{|y(x;h_1) - y(x)|} = \left(\frac{h_2}{h_1}\right)^n \,. \tag{II.2.13}$$

   Here $h_1 > h_2$ are two different discretization step sizes. Since h_2 is smaller than h_1, the calculation with h_2 has higher resolution.

2. **Checking convergence of a numerical algorithm to known solutions is crucial for validation.** If an algorithm does not converge or converges at the wrong rate, a systematic bug in the implementation is likely!

3. **Self Convergence**. In many cases, the true solution to a problem is not known. In this case, we can use results obtained at 3 different discretization step sizes $h_1 > h_2 > h_3$.

   We define the self-convergence factor Q_S:

   $$Q_S = \frac{|y(x;h_3) - y(x;h_2)|}{|y(x;h_2) - y(x;h_1)|} \,, \tag{II.2.14}$$

   which, at convergence order n, must equal

$$Q_S = \frac{h_3^n - h_2^n}{h_2^n - h_1^n}\,. \tag{II.2.15}$$

Exercise II.2.3 (Convergence of a Finite-Difference Estimate)

Discretize

$$f(x) = x^3 - 5x^2 + x$$

on the interval $[-2, 6]$ and compute its first derivative with (i) forward differencing and (ii) central differencing. Demonstrate that (i) is first-order convergent and (ii) is second-order convergent by plotting the absolute error $f'(x; h_i) - f'(x)$ at resolutions h_1 and $h_2 = h_1/2$. At h_2 the absolute error should be reduced by the expected convergence factor.

II.3 Interpolation

We are frequently confronted with the situation that we know the values of a function f only at discrete locations x_i, but want to know its values at general points x.

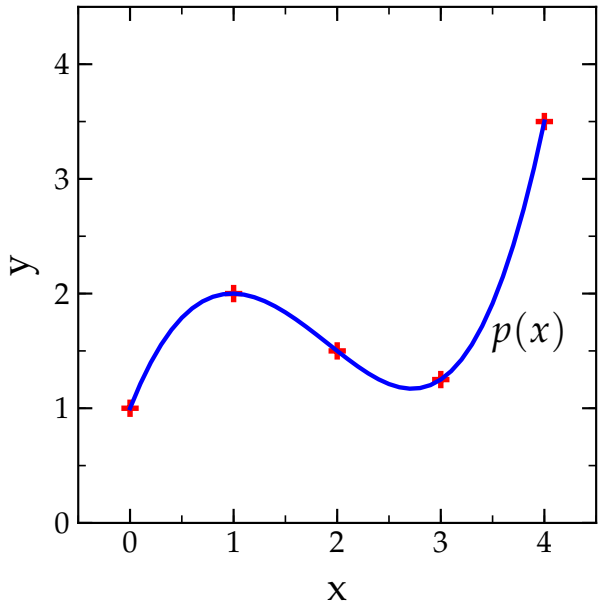

Figure II.3.1: The interpolation problem.

In order to solve this problem, we must look for an approximation $p(x)$ that uses the discrete information about $f(x)$ at x_i to interpolate $f(x)$ between the x_i with $p(x_i) = f(x_i)$. If x is outside $[x_{\rm min}, x_{\rm max}]$, where $x_{\rm min} = \min\{x_i; \forall i\}$ and $x_{\rm max} = \max\{x_i; \forall i\}$, $p(x)$ extrapolates $f(x)$.

II.3.1 Direct Polynomial Interpolation

Polynomial interpolation of $f(x)$ involves finding a polynomial $p(x)$ of *degree* n that passes through $n+1$ points. Note that the literature will frequently talk about the *order* of a polynomial. We will stick to *degree* in order not to confuse polynomial order with order of approximation, i.e., with the exponent of the leading-order error term. For example a polynomial $p(x)$ of degree 1 is linear in x and has a quadratic error term ("second order").

In general, our intergration polynomial will have the following form:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \,, \tag{II.3.1}$$

where the a_i are $n+1$ real constants that can be determined by solving a set of $n+1$ linear equations:

$$\underbrace{\begin{pmatrix} 1 & x_0^1 & x_0^2 & \cdots & x_0^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^1 & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{\text{Vandermonde Matrix}} \begin{pmatrix} a_0 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ \vdots \\ \vdots \\ f(x_n) \end{pmatrix} \tag{II.3.2}$$

For large n this obviously gets very complicated (we will talk later about how to solve systems of linear equations efficiently), but it is useful to consider the two simplest cases, linear ($n = 1$) and quadradtic ($n = 2$) interpolation.

II.3.1.1 Linear Interpolation

We obtain the linear approximation $p(x)$ for $f(x)$ in the interval $[x_i, x_{i+1}]$ by

$$p(x) = f(x_i) + \underbrace{\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}}_{\text{1st-order forward difference}} (x - x_i) + \mathcal{O}(h^2)\,, \tag{II.3.3}$$

where $h = x_{i+1} - x_i$. Linear interpolation is the most robust method for interpolation; the result can be differentiated once, but the derivative will be discontinuous at the locations x_{i+1} and x_i.

II.3.1.2 Quadratic Interpolation

The quadratic approximation $p(x)$ for $f(x)$ in the interval $[x_i, x_{i+1}]$ is given by

$$\begin{aligned} p(x) = &\frac{(x - x_{i+1})(x - x_{i+2})}{(x_i - x_{i+1})(x_i - x_{i+2})} f(x_i) \\ &+ \frac{(x - x_i)(x - x_{i+2})}{(x_{i+1} - x_i)(x_{i+1} - x_{i+2})} f(x_{i+1}) \\ &+ \frac{(x - x_i)(x - x_{i+1})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} f(x_{i+2}) \\ &+ \mathcal{O}(h^3)\,, \end{aligned} \tag{II.3.4}$$

where $h = \max\{x_{i+2} - x_{i+1}, x_{i+1} - x_i\}$.

Note that the results will be sensitive to which three points are chosen, since there are two choices: $\{x_i, x_{i+1}, x_{i+2}\}$ or $\{x_{i-1}, x_i, x_{i+1}\}$ for interpolating $f(x)$ in $[x_i, x_{i+1}]$.

$p(x)$ is twice differentiable. Its first derivative will be continuous, but $p''(x)$ will have finite-size steps.

II.3.2 Lagrange Interpolation

So far, we have looked at linear and quadratic polynomial interpolation. Sometimes it will be necessary to use a higher-order method. *Lagrange Interpolation* provides a means of constructing general interpolating polynomials of degree n using data at $n + 1$ points.

There are alternative methods to Lagrange Interpolation. These are, for example, *Newton's Divided Differences*, *Chebychev Polynomials*, *Aitken's method*, and *Taylor Polynomials*. We will not discuss these methods, but rather refer the reader to the literature listed in §I.1.

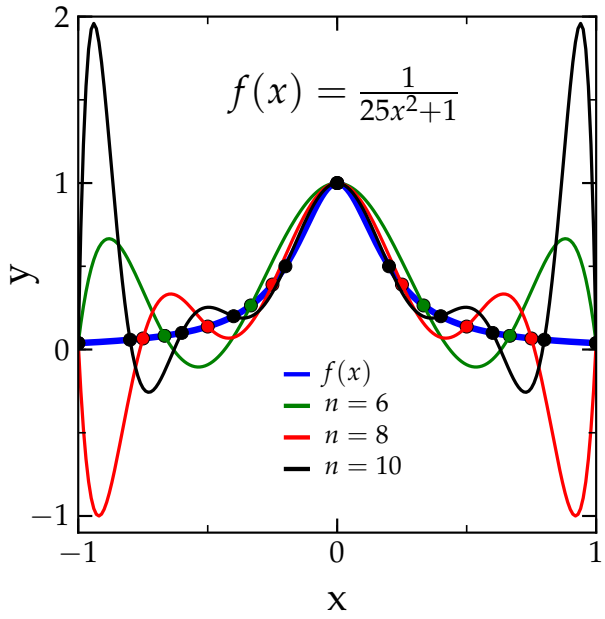

Figure II.3.2: Runge's phenomenon of non-convergence as exhibited by polynomials of degree 6, 8, and 10 for the continuous function $f(x) = (25x^2 + 1)^{-1}$.

For a moment, let us reconsider linear interpolation (Eq. [II.3.3]) and rewrite it slightly to

$$\begin{aligned} f(x) \approx p(x) &= \frac{x - x_{i+1}}{x_i - x_{i+1}} f(x_i) + \frac{x - x_i}{x_{i+1} - x_i} f(x_{i+1}) + \mathcal{O}(h^2)\,, \\ &= \sum_{j=i}^{i+1} f(x_j) L_{1j}(x) + \mathcal{O}(h^2)\,, \\ \text{where } L_{1j}(x) &= \left.\frac{x - x_k}{x_j - x_k}\right|_{k \neq j}\,. \end{aligned} \tag{II.3.5}$$

We can now easily generalize this to an n-th degree polynomial that passes through all $n + 1$ data points:

$$p(x) = \sum_{j=0}^{n} f(x_j) L_{nj}(x) + \mathcal{O}(h^{n+1})\,, \tag{II.3.6}$$

with

$$L_{nj}(x) = \prod_{k \neq j}^{n} \frac{x - x_k}{x_j - x_k}\,. \tag{II.3.7}$$

Note that this clearly satisfies the interpolating condition, $p(x_i) = f(x_i)$.

II.3.3 Convergence of Interpolating Polynomials

If our data describe a continuous function $f(x)$ on an interval $[a, b]$, we would expect that, if we constuct interpolating polynomials $p_n(x)$ with increasing degree n, the interpolation error con-

verges to zero. Mathematically this is expressed by

$$\lim_{n\to\infty}\left[\max_{x\in[a,b]}|f(x)-p_n(x)|\right]=0\,. \tag{II.3.8}$$

Unfortunately, it turns out that for most continuous functions, this is not the case and the error does not converge to 0. This is called *Runge's Phenomenon*, an example of which is shown by Fig. II.3.2 for the well behaved continuous function $f(x)=\frac{1}{25x^2+1}$. While the interpolating polynomials $p_n(s)$ converge to $f(x)$ near the peak, they exhibit strong oscillations and diverge dramatically at the edges of the interval.

Runge's phenomenon teaches us that we need to be extremely careful with polynomial interpolation. A higher polynomial degree could leat to a greater error and unwanted oscillations!

High degree polynomial interpolation ($n\gtrsim 2$) should only be used if one is certain that the function to be interpolated can be approximated well globally by a single polynomial. If this is not the case, one must resort to other methods. One solution is to use *piecewise-polynomial* interpolation, which, in essence, uses many low-degree polynomials rather than a single global polynomial to approximate a function or interpolate a set of data. The simplest (and safest!) form of piecewise polynomial interpolation is piecewise linear interpolation – essentially connecting data points with straight lines.

Exercise II.3.1 (Runge's Phenomenon)

1. Implement a routine that generates Lagrangean interpolating polynomials of arbitrary degree n based on $n+1$ data points. Then reproduce Fig. II.3.2 for $f(x)=\frac{1}{25x^2+1}$ in $[-1,1]$ with $n=6$, $n=8$, $n=10$, $n=12$.
2. Discretize the domain with $m=100$ equally spaced points and compute the error norm-2,
$$\mathrm{EN2}=\frac{1}{m}\sqrt{\sum_{i=1}^{m}\left(\frac{p(x)-f(x)}{f(x)}\right)^2}\,,$$
for the 4 cases $n=\{6,8,10,12\}$.
3. Now introduce another discretization with m_2 = 50 and implement a routine that will interpolate $f(x)$ piecewise linearly between these m_2 data points. Now evaluate EN2 at the m points used in part 2 and compare your result to the results of part 2.

II.3.4 Hermite Interpolation (in Lagrange form)

Hermite interpolation is a special form of polynomial interpolation. It uses data points as well as derivatives of the data to construct an interpolating polynomial. This can significantly reduce unwanted oscillations, in particular if it is applied in piecewise fashion.

In the following, we will consider only the simplest case of Hermite interpolation, which interpolates a function and its first derivative. If we have $n+1$ data points, then we need to find a polynomial that satisfies

$$p(x_i) = c_{i0} \ , \ \ p'(x_i) = c_{i1} \quad \text{for } i \in [0,n] \ , \tag{II.3.9}$$

where $c_{i0} = f(x_i)$ and $c_{i1} = f'(x_i)$.

In analogy with the Lagrange interpolation formula (Eq. II.3.6), we write

$$p(x) = \sum_{i=0}^{n} c_{i0} A_i(x) + \sum_{i=0}^{n} c_{i1} B_i(x) \ , \tag{II.3.10}$$

in which $A_i(x)$ and $B_i(x)$ are polynomials with certain properties:

$$\begin{aligned} A_i(x_j) &= \delta_{ij} \ , \quad B_i(x_j) = 0 \ , \\ A_i'(x_j) &= 0 \ , \quad B_i'(x_j) = \delta_{ij} \ . \end{aligned} \tag{II.3.11}$$

Using

$$L_{nj}(x) = \prod_{\substack{j=0 \\ k\neq j}}^{n} \frac{x - x_k}{x_j - x_k} \ , \tag{II.3.12}$$

Without proof: A_i and B_i can be defined as follows for points $0 \leq i \leq n$:

$$\begin{aligned} A_i(x) &= [1 - 2(x - x_i) L_{ni}'(x_i)] L_{ni}^2(x) \ , \\ B_i(x) &= (x - x_i) L_{ni}^2(x) \ . \end{aligned} \tag{II.3.13}$$

Note that each L_{ni} is of degree n and therefore A_i and B_i are of degree $2n+1$. This is also the maximal degree of $p(x)$. A derivation of Eq. (II.3.13) can be found in Doron Levy's lecture notes on Numerical Analysis, http://www.math.umd.edu/~dlevy/books/na.pdf.

II.3.4.1 Piecewise Cubic Hermite Interpolation

When performing piecewise cubic Hermite interpolation on a large dataset, we set $n = 1$ and for each x at which we want to approximate a function $f(x)$, we find two neighboring (bracketing) points x_i and x_{i+1}, where $f(x_i)$, $f(x_{i+1})$, $f'(x_i)$, and $f'(x_{i+1})$ are known or can be evaluated numerically.

The cubic Hermite polynomial that interpolates $f(x)$ in $[x_i, x_{i+1}]$ is then given by

$$\begin{aligned} H_3(x) = & f(x_i)\psi_0(z) + f(x_{i+1})\psi_0(1-z) \\ & + f'(x_i)(x_{i+1} - x_i)\psi_1(z) \\ & - f'(x_{i+1})(x_{i+1} - x_i)\psi_1(1-z) \ , \end{aligned}$$

where

$$z = \frac{x - x_i}{x_{i+1} - x_i} \ , \tag{II.3.14}$$

and

$$\psi_0(z) = 2z^3 - 3z^2 + 1 \ , \tag{II.3.15}$$

$$\psi_1(z) = z^3 - 2z^2 + z \ . \tag{II.3.16}$$

Of course, the ψ_i are straightforwardly related to the A_i and B_i of the previous section. They have been written in this form for convenience.

II.3.5 Spline Interpolation

Splines are the ultimate method for piecewise polynomial interpolation of strongly varying data if continuity and differentiability (= smoothness) of the interpolation result is important.

Spline interpolation achieves smoothness by requiring continuity at data points not only for the function values $f(x_i)$, but also for a number of its derivatives $f^{(l)}(x_i)$. Assuming that we know the values $f(x_i)$ at points x_i ($i \in [0,n]$), we can construct piecewise polynomials on each segment $[x_i, x_{i+1}]$ (there are n such segments) of degree m,

$$p_i(x) = \sum_{k=0}^{m} c_{ik} x^k \ , \tag{II.3.17}$$

to approximate $f(x)$ for $x \in [x_i, x_{i+1}]$. Note that m is the degree of the spline and there are $m+1$ c_{ik} for each i and there are n intervals. Hence we have $n(m+1)$ coefficients c_{ik} that we must determined by

(a) requiring $(n-1)(m-1)$ smoothness conditions at non-boundary points: $p_i^l(x_{i+1}) = p_{i+1}^l(x_{i+1})$ for $l = 1, \cdots, m-1$,

(b) requiring $2n$ interpolation conditions: $p_i(x_i) = f(x_i) = p_{i+1}(x_i)$,

(c) choosing the remaining $m-1$ values of some of the $p_o^l(x_o)$ and $p_{n-1}^l(x_n)$ for $l = 1, \cdots, m-1$.

Note that the simplest spline, the linear spline $(m = 1)$, is equivalent to piecewise linear interpolation.

II.3.5.1 Cubic Natural Spline Interpolation

Note: The following discussion is based on Tao Pang's book *An Introduction to Computational Physics*.

$m = 3$, a cubic, is the most widely used spline basis function. In this case, a total of $4n$ conditions are necesssary to find the coefficients c_{ik}. $m-1 = 2$ must be picked by choosing values for some of the derivatives at both boundary points. One possible choice is to set the highest derivative to zero at boths ends of the interval. This is called the *natural spline*. For the cubic spline this means

$$p_0''(x_0) = 0 \ , \ \ p_{n-1}''(x_n) = 0 \ . \tag{II.3.18}$$

To construct the cubic spline, we start with the linear interpolation of its second derivative in $[x_i, x_{i+1}]$,

$$p_i''(x) = \frac{1}{x_{i+1} - x_i} \left[(x - x_i) p_{i+1}'' - (x - x_{i+1}) p_i'' \right] \ , \tag{II.3.19}$$

where we have set $p_i'' = p_i''(x_i) = p_{i-1}''(x_i)$ and $p_{i+1}'' = p_{i+1}''(x_{i+1}) = p_i''(x_{i+1})$.

We now integrate Eq. (II.3.19) twice and identify $f_i = p_i(x_i) = f(x_i)$. We obtain

$$p_i(x) = \alpha_i (x - x_i)^3 + \beta_i (x - x_{i+1})^3 + \gamma_i (x - x_i) + \eta_i (x - x_{i+1}) \, , \tag{II.3.20}$$

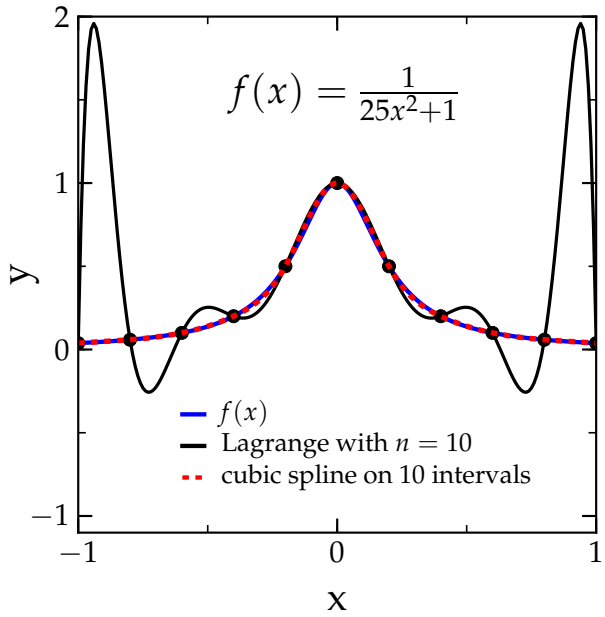

Figure II.3.3: Same as Fig. II.3.2, but this time showing only $f(x)$, the result of Lagrangean interpolation with $n = 10$, and of cubic natural spline interpolation on the same 10 intervals.

where

$$\alpha_i = \frac{p''_{i+1}}{6h_i} \qquad \beta_i = \frac{-p''_i}{6h_i}\,, \tag{II.3.21}$$

$$\gamma_i = \frac{f_{i+1}}{h_i} - \frac{h_i p''_{i+1}}{6} \qquad \eta_i = \frac{h_i p''_i}{6} - \frac{f_i}{h_i}\,, \tag{II.3.22}$$

with $h_i = x_{i+1} - x_i$. Hence, all that is needed to find the spline is to find all of its second derivatives $p''_i = p''_i(x_i)$.

We may now apply the condition $p'_{i-1}(x_i) = p'_i(x_i)$ to Eq. (II.3.20) to obtain

$$h_{i-1}p''_{i-1} + 2(h_{i-1}+h_i)p''_i + h_i p''_{i+1} = 6\left(\frac{g_i}{h_i} - \frac{g_{i-1}}{h_{i-1}}\right)\,, \tag{II.3.23}$$

where $g_i = f_{i+1} - f_i$. This is a linear system with $n-1$ unknowns p''_i for $i = 1, \cdots, n-1$ and $p''_0 = p''_n = 0$ (as set by the natural spline condition).

If we set $d_i = 2(h_{i-1}+h_i)$ and $b_i = 6\left(\frac{g_i}{h_i} - \frac{g_{i-1}}{h_{i-1}}\right)$, then we can write

$$Ap'' = b \quad \text{with } A_{ij} = \begin{cases} d_i & \text{if } i = j\,, \\ h_i & \text{if } i = j-1\,, \\ h_{i-1} & \text{if } i = j+1\,, \\ 0 & \text{otherwise.} \end{cases} \tag{II.3.24}$$

Note that the coefficient matrix A_{ij} is real, symmetric, and tri-diagonal. We will discuss later how to solve linear systems of equations of this type.

Figure II.3.3 shows how much better cubic spline interpolation deals with the example we used in §II.3.2 to show Runge's phenomenon.

II.3.6 Multi-Variate Interpolation

In many situations, on ehas to deal with data that depend on more than one variable. A typical example are stellar opacities that may depend on frequency, gas density, temperature, and composition. Since they are too complicated to compute on the fly, one pre-tabulates them and then interpolates between table entries.

For deciding which approach to take for multi-variate interpolation, it is important to differentiate between two general kinds of multi-variate data:

(a) Data in a grid arrangement that are given on a regularly distributed (equidistantly or non-equidistantly) set of points (nodes).

(b) Scattered data, which are not arranged in a regular grid.

As we shall see, for case (a), we can use the so-called *Tensor Product Interpolation* to construct a multi-variate interpolation function.

For case (b), the situation is more complicated. One can try to first introduce a regular gridding of the data by linear interpolation between nearest neighbors or by piecewise constant nearest neighbor interpolation in which the value at the nearest irregular node is assigned to a node of an introduced regular grid. Alternatively, one may use a special method for multi-variate interpolation of scattered data, e.g., Shepard's method.

A good summary of methods for interpolation of scattered data can be found in Alfeld, *Scattered Data Interpolation in 3 or more Variables*, in Mathematical Methods in Computational Geometric Design, 1989.

II.3.6.1 Tensor Product Interpolation

For data arranged in a grid, a multi-variate (here: n-variate) interpolation function can be obtained by forming the tensor product

$$g(x_1, \cdots, x_n) = g_1(x_1) \circ g_2(x_2) \circ \cdots \circ g_n(x_n) \ , \tag{II.3.25}$$

using n univariate interpolation functions g_i.

This sounds much more complicated than it really is. The above simply means that true multivariate (multi-dimensional) interpolation is mathematically equivalent to executing multiple univerative (1D) interpolations sequentially. So, if, for example, we want to interpolate a function $U(x, y, z)$, we would first interpolate it in x, then interpolate the result in y, and then that result in z. This, of course, requires many interpolation steps. Hence, it is generally more efficient to analytically work out and simplify the multi-variate interpolation function and then interpolate in only one step.

Exercise II.3.2 (Bilinear Interpolation)

Construct a bilinear interpolation polynomial that interpolates a function $f(x, y)$ for points (x, y) in $[x1, x2]$ and $[y1, y2]$, respectively.

II.3.6.2 Shepard's Method for Scattered Data

Shepard's Method uses inverse distance weighting for finding an interpolated value u at a given point $\vec{x}$ based on irregularly arranged data (samples) $u_i(\vec{x}_i)$ ($i = 0, \cdots, n$):

$$u(\vec{x}) = \sum_{i=0}^{n} \frac{w_i(\vec{x})\, u_i}{\sum_{j=0}^{n} w_j(\vec{x})} \ , \quad \text{with} \quad w_i(\vec{x}) = \frac{1}{[d(\vec{x}, \vec{x}_i)]^p} \ . \tag{II.3.26}$$

Here, the $w_i(\vec{x})$ are distance-dependent weights, $d(\vec{x}, \vec{x}_i)$ is the distance function, and p is the power parameter ($p > 0$). The weight decreases distance increases from the known points. The power parameter p controls the smoothness of the interpolation. For $0 < p \leq 1$, $u(\vec{x})$ is rather smooth and for $p > 1$ it quickly becomes sharp.

The *Modified Shepard's Method* uses a modified weight functions

$$w_i(\vec{x}) = \left(\frac{R - d(\vec{x}, \vec{x}_i)}{R d(\vec{x}, \vec{x}_i)} \right)^2 \ , \tag{II.3.27}$$

which emphasises points within a sphere of radius R around $\vec{x}$.

II.4 Integration

There are many situations in which we need to evalute the integral

$$Q = \int_a^b f(x)dx \ . \tag{II.4.1}$$

However, integrals can be evaluated anlytically only for very few well behaved functions $f(x)$, plus, frequently, we have to deal with discrete data $f(x_i)$ that, of course, cannot be integrated analytically. Hence, we have to resort to numerical integration (quadrature).

II.4.1 Integration based on Piecewise Polynomial Interpolation

Assume that we know (or can evaluate) $f(x)$ at a finite set of points (nodes) $\{x_j\}$ with $j = 0, \cdots, n$ in the interval $[a, b]$. Then we can replace $f(x)$ with a simpler function $p(x)$ whose analytic integral we know and which interpolates $f(x)$: $p(x_i) = f(x_i)$. Such integration formulae based on interpolation polynomials are generally called Newton-Cotes quadrature formulae.

In the following, we will assume that the full interval over which we intend to integrate can be broken down into N sub-intervals $[a_i, b_i]$ that encompass $N+1$ nodes x_i ($i = 0, \cdots, N$) at which we know the integrand $f(x)$. We can then express the full integral Q as the sum of the sub-integrals Q_i:

$$Q = \sum_{i=0}^{N-1} Q_i = \sum_{i=0}^{N-1} \int_{a_i}^{b_i} f(x)dx \ . \tag{II.4.2}$$

II.4.1.1 Midpoint Rule

The simplest approximation is to assume that the function $f(x)$ is constant on the interval $[a_i, b_i]$ and to use its central value value:

$$Q_i = \int_{a_i}^{b_i} f(x)dx = (b_i - a_i)\, f\left(\frac{a_i + b_i}{2}\right) + \mathcal{O}([b_i - a_i]^2) \ . \tag{II.4.3}$$

This is also called the "rectangle rule" and the error is quadratic in the interval size $b_i - a_i$. Note that we need to be able to evaluate $f(x)$ at the midpoint. We may not be able to do this if we know $f(x)$ only at discrete locations. This integration scheme is locally second order, but globally first order, since we must apply it $\mathcal{O}(N)$ times.

II.4.1.2 Trapezoidal Rule

Here we approximate $f(x)$ with a linear polynomial on $[a, b]$,

$$Q_i = \int_{a_i}^{b_i} f(x)dx = \frac{1}{2}(b_i - a_i)\,[f(b_i) + f(a_i)] + \mathcal{O}([b_i - a_i]^3) \ . \tag{II.4.4}$$

The error scales with the cube of the size of the interval $[a_i, b_i]$, but global convergence is only $\mathcal{O}([b - a]^2)$.

II.4.1.3 Simpson's Rule

Simpson's Rule approximates $f(x)$ on $[a_i, b_i]$ by a quadratic (a parabola; a polynomial of degree 2). Writing out the interpolating polynomial on $[a_i, b_i]$ and integrating it, one obtains

$$Q_i = \frac{b_i - a_i}{6}\left[f(a_i) + 4f\left(\frac{a_i + b_i}{2}\right) + f(b_i)\right] + \mathcal{O}([b_i - a_i]^5) \ . \tag{II.4.5}$$

Note that in the above we have assumed that we know or can evaluate $f((a_i + b_i)/2)$. If we know $f(x)$ only at discrete locations, all hope is not lost. We can always combine two intervals $[a_i, b_i]$ and $[b_i, b_{i+1}]$ to $[a_i, b_{i+1}]$ and identify $a_i = x_i$, $b_i = a_{i+1} = x_{i+1}$, and $b_{i+1} = x_{i+2}$. Of course, the error will then scale with the fifth power of the combined interval locally, but globally with the fourth power of the interval size (because there are many sub-intervals).

Note that special care must be taken at the boundaries of the interpolation interval. If it is not possible to analytically extend the integrand beyond the boundary to obtain the needed integrand values, one must (a) use a lower-order integration method at the end points (but this can spoil the global convergence rate) or (b) resort to a one-side interpolation polynomial that relies exclusively on data that is in the integration interval. Details on (b) can be found in Tao Pang's book (see §I.1).

So far, we have considered only equidistant intervals of size $h = [a_i, b_i]$. If $f(x)$ is non-equidistantly sampled, we need to use a modification of Simpson's rule:

Note that at least two people have pointed out that there is something wrong in the next two paragraphs. I need to re-derive everything to find out what the problem is. This will take a while and may not happen this term :-/ - Christian

First we rewrite the interpolating quadratic as

$$p(x) = ax^2 + bx + c \qquad \text{for } x \in [x_{i-1}, x_{i+1}] \ , \tag{II.4.6}$$

where

$$\begin{aligned} a &= \frac{h_{i-1}f(x_{i+1}) - (h_{i-1} + h_i)f(x_i) + h_i f(x_{i-1})}{h_{i-1}h_i(h_{i-1} + h_i)} \ , \\ b &= \frac{h_{i-1}^2 f(x_{i+1}) + (h_i^2 - h_{i-1}^2)f(x_i) - h_i^2 f(x_{i-1})}{h_{i-1}h_i(h_{i-1} + h_i)} \ , \\ c &= f(x_i) \ , \end{aligned} \tag{II.4.7}$$

with $h_i = x_{i+1} - x_i$ and $h_{i-1} = x_i - x_{i-1}$. Because

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx$$

is independent of the origin of the coordinates, we may set $x_i = 0$.

Now, with $x_i = 0$, $-h_{i-1} = x_{i-1}$, $h_i = x_{i+1}$ we obtain

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = \int_{-h_{i-1}}^{h_i} f(x)dx = \alpha f(x_{i+1}) + \beta f(x_i) + \gamma f(x_{i-1}) \ , \tag{II.4.8}$$

with

$$\begin{aligned} \alpha &= \frac{2h_i^2 + h_i h_{i-1} - h_{i-1}^2}{6h_i} \ , \qquad \beta = \frac{(h_i + h_{i-1})^3}{6h_i h_{i-1}} \ , \\ \gamma &= \frac{-h_i^2 + h_i h_{i-1} - 2h_{i-1}^2}{6h_i} \ , \end{aligned} \tag{II.4.9}$$

Example: $f(x) = x^2$, to be integrated on $[-0.5, 1]$ with $x_{i-1} = 0.5$, $x_i = 0$, $x_{i+1} = 1$.

$$\alpha = \frac{2 + 0.5 - 0.25}{6} = 0.375 \ , \ \beta = \frac{3.375}{3} = 1.125 \ , \ \gamma = 0 \ ,$$

and using Eq. II.4.8, we obtain

$$\int_{x_{i-1}}^{x_{x+1}} f(x)dx = 0.375 \ .$$

The analytic result is, of course,

$$\int_{-0.5}^{1} x^2 dx = \left[\frac{x^3}{3}\right]_{-0.5}^{1} = \frac{1}{3} + \frac{1}{3}\frac{1}{8} = 0.375.$$

Hence, the integral of our quadratic interpolating polynomial integrates the quadratic $f(x) = x^2$ exactly.

Note that as it is the case with the previously discussed methods, Simpson's rule is globally convergent to one order less than its local convergence rate, so $\mathcal{O}([b-a]^4)$

II.4.2 Gaussian Quadrature

> *As I have received your paper about the approximate integration, I no longer can withstand to thank you for the great pleasure you have given to me.*
>
> From a letter written by Bessel to Gauss.

Gaussian quadrature is an integration method that gives an exact result for polynomials of degree $2n-1$ for a set of n nodes x_i at which f is known:

$$\int_a^b f(x)dx = \int_a^b \underbrace{W(x)}_{\substack{\text{weighting}\\ \text{function}}} \underbrace{g(x)}_{\substack{\text{approximating}\\ \text{polynomial}}} dx \approx \sum_{i=1}^{n} \underbrace{w_i}_{\text{weights}} g(x_i) \ . \tag{II.4.10}$$

The nodes x_i are the roots of the approximating polynomial $g(x)$.

II.4.2.1 2-point Gaussian Quadrature

We set $W(x) = 1$, so $\int_a^b f(x)dx = w_1 f(x_1) + w_2 f(x_2)$. The unknowns are w_1, w_2, x_1, and x_2. We can find them by demanding that the formula give exact results for integrating a general polynomial of degree 3:

$$\begin{aligned} \int_a^b f(x)dx &= \int_a^b (c_0 + c_1 x + c_2 x^2 + c_3 x^3)dx \ , \\ &= c_0(b-a) + c_1\left(\frac{b^2-a^2}{2}\right) + c_2\left(\frac{b^3-a^3}{3}\right) + c_3\left(\frac{b^4-a^4}{4}\right) \ . \end{aligned} \tag{II.4.11}$$

We also have

$$\int_a^b f(x)dx = w_1 f(x_1) + w_2 f(x_2) = w_1(c_0 + c_1 x_1 + c_2 x_1^2 + c_3 x_1^3) + w_2(c_0 + c_1 x_2 + c_2 x_2^2 + c_3 x_2^3) \ . \tag{II.4.12}$$

Table II.1: Types of Gaussian Quadrature

$[a,b]$	$W(x)$	Type of Gaussian Quadrature
$[-1,1]$	1	Gauss-Legendre
$[-1,1]$	$(1-x^2)^{-1/2}$	Gauss-Chebychev
$[0,\infty)$	$x^c e^{-x}$	Gauss-Laguerre
$(-\infty,\infty)$	e^{-x^2}	Gauss-Hermite

Setting Eqs. (II.4.11) and (II.4.12) equal gives

$$\begin{aligned} & c_0(b-a) + c_1\left(\frac{b^2-a^2}{2}\right) + c_2\left(\frac{b^3-a^3}{3}\right) + c_3\left(\frac{b^4-a^4}{4}\right) \ , \\ & \quad = w_1(c_0 + c_1x_1 + c_2x_1^2 + c_3x_1^3) + w_2(c_0 + c_1x_2 + c_2x_2^2 + c_3x_2^3) \ , \\ & \quad = c_0(w_1+w_2) + c_1(w_1x_1 + w_2x_2) + c_2(w_1x_1^2 + w_2x_2^2) + c_3(w_1x_1^3 + w_2x_2^3) \ . \end{aligned} \tag{II.4.13}$$

Since the c_i are arbitrary, we can now demand:

$$(1)\ b-a = w_1 + w_2 \qquad (3)\ \frac{b^3-a^3}{3} = w_1x_1^2 + w_wx_2^2 \ , \tag{II.4.14}$$

$$(2)\ \frac{b^2-a^2}{2} = w_1x_1 + w_2x_2 \qquad (4)\ \frac{b^4-a^4}{4} = w_1x_1^3 + w_2x_2^3 \ . \tag{II.4.15}$$

From this, we obtain

$$w_1 = \frac{b-a}{2} \ , \qquad w_2 = \frac{b-a}{2} \ , \tag{II.4.16}$$

$$x_1 = \left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \left(\frac{b+a}{2}\right) \ , \tag{II.4.17}$$

$$x_2 = \left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \left(\frac{b+a}{2}\right) \ . \tag{II.4.18}$$

$$\tag{II.4.19}$$

So, in the special case of the interval $[-1,1]$:

$$w_1 = w_2 = 1 \ , \qquad x_1 = -\frac{1}{\sqrt{3}} \ , \qquad x_2 = \frac{1}{\sqrt{3}} \ , \tag{II.4.20}$$

and

$$\int_{-1}^{1} f(x)dx = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) + \mathcal{O}(h^3) \ . \tag{II.4.21}$$

Of course, for this to work $f(x_1)$ and $f(x_2)$ must be known.

II.4.2.2 General Higher-Point Gaussian Quadrature Formulae

General Gaussian quadrature formulae are usually given for special, fixed intervals. To make practical use of them, the integral in question must be transformed. For the special case $[-1,1]$,

this works in the following way:

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)dx \ . \tag{II.4.22}$$

In formal terms, this is an affine transformation that maps $[a,b] \rightarrow [-1,1]$ via

$$t = \frac{b-a}{2}x + \frac{a+b}{2} \tag{II.4.23}$$

and a change of variables

$$dt = \frac{b-a}{2}dx \ . \tag{II.4.24}$$

There is a number of different Gaussian quadrature formulae that are named after the polynomials $g(x)$ that they use. Table II.1 summarizes various types of Gaussian quadrature that work over different intervals and have differing weight functions. The corresponding x_i and w_i are tabulated and can be found, for example, in Abramowitz & Stegun.

II.5 Ordinary Differential Equations (Part I)

A system of first-order ordinary differential equations (ODEs) is a relationship between an unknown (vectorial) function $\vec{y}(\vec{x})$ and its derivative $\vec{y}\,'(\vec{x})$. The general system of first-order ODEs has the form

$$\vec{y}\,'(x) = \vec{f}(\vec{x}, \vec{y}(\vec{x}))\,. \tag{II.5.1}$$

To make life easier, we will drop the vector $\vec{}$ notation in the following and just keep in mind that the symbols are vectors if more than one equation is to be solved.

A solution to the differential equation (II.5.1) is, obviously, any function $y(x)$ that satisfies it.

There are two general classes of first-order ODE problems:

(a) Initial value problems: $y(x_i)$ is given at some starting point x_i.

(b) Two-point boundary value problems: y is known at two ends ("boundaries") of the domain and these "boundary conditions" must be satisfied simultaneously.

II.5.1 Reduction to First-Order ODE

Any ODE can be reduced to first-order form by introducing additional variables. Here is an example:

$$y''(x) + q(x)y'(x) = r(x)\ . \tag{II.5.2}$$

We reduce this ODE to first-order form by introducing a new function $z(x)$ and solve for

$$\begin{aligned} &(1)\quad y'(x) = z(x)\,,\\ &(2)\quad z'(x) = r(x) - q(x)z(x)\ . \end{aligned}$$

II.5.2 ODEs and Errors

For studying the kinds of errors we have to deal with when working with ODEs, let us restrict ourselves to initial value problems. All procedures to solve numerically such an ODE consist of transforming a continuous differential equation into a discrete iteration procedure that starts from the initial conditions and returns the values of the dependent variable $y(x)$ at points $x_m = x_0 + m * h$, where h is the discretization step size (which we have assumed to be constant here).

Two kinds of errors can arise in this procedure:

(a) Round-off error: Due to limited FP accuracy. The global round-off is the sum of the local FP errors.

(b) **Truncation error**.

Local: The error made in one step when we replace a continuous process (e.g. a derivative) with a discrete one (e.g., a forward difference).

Global: If the local truncation error is $\mathcal{O}(h^{n+1})$, then the global truncation error must be $\mathcal{O}(h^n)$, since the number of steps used in evaluating the derivatives to reach an arbitrary point x_f, having started at x_0, is $\frac{x_f - x_0}{h}$.

II.5.3 Euler's Method

We want to solve $y' = f(x, y)$ with $y(x_0) = y_0$. We introduce a fixed stepsize h and we first obtain an estimate of $y(x)$ at $x_1 = x_0 + h$ using Taylor's theorem:

$$\begin{aligned} y(x_1) = y(x_0 + h) &= y(x_0) + y'(x_0)h + \mathcal{O}(h^2)\,, \\ &= y(x_0) + hf(x_0, y(x_0)) + \mathcal{O}(h^2)\,. \end{aligned} \tag{II.5.3}$$

By analogy, we obtain that the value y_{n+1} of the function at the point $x_{n+1} = x_0 + (n+1)h$ is given by

$$y_{n+1} = y(x_{n+1}) = y_n + hf(x_n, y(x_n)) + \mathcal{O}(h^2)\,. \tag{II.5.4}$$

This is called the *forward Euler Method*. It is obviously extremely simple, but rather inaccurate and potentially unstable.

II.5.3.1 Stability of Forward Euler

Forward Euler is an *explicit* method. This means that y_{n+1} is given explicitly in terms of known quantities y_n and $f(x_n, y_n)$.

Explicit methods are simple and efficient, but the drawback is that the step size must be small for stability. Example:

$$y' = -ay\,, \qquad \text{with} \quad y(0) = 1\,,\ a > 0\,,\ y' = \frac{dy}{dt}\,. \tag{II.5.5}$$

As we know, the exact solution to this problem is $y^{\rm ex} = exp(-at)$, which is stable & smooth with $y^{\rm ex}(0) = 1$ and $y^{\rm ex}(\infty) = 0$.

Now applying forward Euler:

$$y_{n+1} = y_n - a\,h\,y_n = (1 - ah)^2 y_{n-1} = \cdots = (1 - ah)^{n+1} y_0\,. \tag{II.5.6}$$

This implies that in order to prevent any potential amplification of errors, we must require that $|1 - ah| < 1$. In fact, we can decide between 3 cases:

(i) $0 < 1 - ah < 1$: $(1 - ah)^{n+1}$ decays (good!).
(ii) $-1 < 1 - ah < 0$: $(1 - ah)^{n+1}$ oscillates (not so good!).
(iii) $1 - ah < -1$: $(1 - ah)^{n+1}$ oscillates and diverges (bad!).

With this, we arrive at an overall stability criterion of $h < 2/a$. This means that forward Euler is unstable overall, but conditionally stable if $h < 2/a$.

II.5.4 Backward Euler

If we use

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})\,, \tag{II.5.7}$$

we get what is called the *backward Euler Method*, and *implicit* method. Implicit, because y_{n+1} depends on unknown quantities, i.e. on itself!

For our toy problem of the previous section II.5.3.1, we get with backward Euler

$$\begin{aligned} y_{n+1} &= y_n + h(-a y_{n+1}) \ , \\ y_{n+1} &= \frac{1}{1+ha}\, y_n \ . \end{aligned} \tag{II.5.8}$$

Since $ha > 0$, $(1-ha)^{-1} < 1$ and the solution decays. This means that backward Euler is *unconditinally stable* for all h! But do not forget that the error could still be large; it will just not wreck the scheme by leading to blow up.

II.5.5 Predictor-Corrector Method

We can improve upon the forward Euler method in many ways. One way is to try

$$y_{n+1} = y_n + h\,\frac{f(x_n, y_n) + f(x_{n+1}, y_{n+1})}{2} \ , \tag{II.5.9}$$

which will be a better estimate as it is using the "average slope" of y. However, we don't know y_{n+1} yet.

We can get around this problem by using forward Euler to estimate y_{n+1} and then use Eq. (II.5.9) for a better estimate:

$$\begin{aligned} y^{(\mathrm{P})}_{n+1} &= y_n + h f(x_n, y_n) \ , && \text{(predictor)} \\ y_{n+1} &= y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y^{(\mathrm{P})}_{n+1}) \right] \ . && \text{(corrector)} \end{aligned} \tag{II.5.10}$$

One can show that the error of the predictor-corrector method decreases locally with h^3, but globally with h^2. One says it is *second-order accurate* as opposed to the Euler method, which is first-order accurate. The predictor-corrector method is also considerably more stable than the Euler method, but we spare the reader the proof.

II.5.6 Runge-Kutta Methods

The idea behind Runge-Kutta (RK) methos is to match the Taylor expansion of $y(x)$ at $x = x_n$ up to the highest possible and convenient order.

As an example we shall consider derivation of a second order RK method (RK2). For

$$\frac{dy}{dx} = f(x, y) \ , \tag{II.5.11}$$

we have

$$y_{n+1} = y_n + a k_1 + b k_2 \ , \tag{II.5.12}$$

with

$$\begin{aligned} k_1 &= h\, f(x_n, y_n) \ , \\ k_2 &= h\, f(x_n + \alpha h, y_n + \beta k_1) \ . \end{aligned} \tag{II.5.13}$$

We now fix the four parameters a, b, α, β so that Eq. (II.5.12) agrees as well as possible with the Taylor series expansion of $y' = f(x, y)$:

$$\begin{aligned} y_{n+1} &= y_n + hy'_n + \frac{h^2}{2} y''_n + \mathcal{O}(h^3) \ , \\ &= y_n + hf(x_n, y_n) + \frac{h^2}{2} \frac{d}{dx} f(x_n, y_n) + \mathcal{O}(h^3) \ , \\ &= y_n + hf_n + h^2 \left(\frac{1}{2} \frac{df_n}{dx} + \frac{df_n}{dy} f_n \right) + \mathcal{O}(h^3) \ , \end{aligned} \tag{II.5.14}$$

where we have used $f_n = f(x_n, y_n)$.

On the other hand, we have, using Eq. (II.5.12),

$$y_{n+1} = y_n + ahf_n + bhf(x_n + \alpha h, y_n + \beta h f_n) \ . \tag{II.5.15}$$

Now we expand the last term of Eq. (II.5.15) in a Taylor series to first order in terms of (x_n, y_n),

$$y_{n+1} = y_n + ahf_n + bh \left[f_n + \frac{df}{dx}(x_n, y_n)\alpha h + \frac{df}{dy}(x_n, y_n)\beta h f_n \right] \ , \tag{II.5.16}$$

and can now compare this with Eq. (II.5.12) to read off:

$$a + b = 1 \ , \qquad \alpha b = \frac{1}{2} \qquad \beta b = \frac{1}{2} \ . \tag{II.5.17}$$

So there are only 3 equations for 4 unknowns and we can assign an arbitrary value to on eof the unknowns. Typical choices are:

$$\alpha = \beta = \frac{1}{2} \ , \qquad a = 0 \ , \qquad b = 1 \ . \tag{II.5.18}$$

With this, we have for RK2:

$$k_1 = hf(x_n, y_n) \ , \tag{II.5.19}$$

$$k_2 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \ , \tag{II.5.20}$$

$$y_{n+1} = y_n + k_2 + \mathcal{O}(h^3) \ . \tag{II.5.21}$$

Note that this method is locally $\mathcal{O}(h^3)$, but globally only $\mathcal{O}(h^2)$ (see §II.5.2). Also note that for $a = b = 1/2$ and $\alpha = \beta = 1$ we recover the predictor-corrector method!

II.5.7 Other RK Integrators

II.5.7.1 RK3

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \ , \\ k_3 &= hf(x_n + h, y_n - k_1 + 2k_2) \ , \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) + \mathcal{O}(h^4) \ . \end{aligned} \tag{II.5.22}$$

II.5.7.2 RK4

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \ , \qquad \text{(II.5.23)}\\
k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \ ,\\
k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2) \ ,\\
k_4 &= hf(x_n + h, y_n + k_3) \ ,\\
y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5) \ . \qquad \text{(II.5.24)}
\end{aligned}
$$

II.5.7.3 Implementation Hint

When implementing RK integration, write a subroutine that evaluates

$$\frac{dy}{dx} = f(x, y) \ , \qquad \text{(II.5.25)}$$

for a given x and y, since you will call it many times (with different arguments). $f(x, y)$ in the above equation is usually referred to as the *right-hand side* (RHS) of the ODE problem.

II.5.8 Runge-Kutta Methods with Adaptive Step Size

This section of the lecture notes has been contributed by Mark Scheel, `scheel@tapir.caltech.edu`.

The RK methods we have discussed thus far require choosing a fixed step size h. How should one choose h? How does one know if one is making the right choice?

Better would be to choose an *error tolerance* and have h be chosen automatically to satisfy this error tolerance. To do this, we need:

(a) A method for estimating error.

(b) A way to adjust the stepsize h, if the error is too large/small.

II.5.8.1 Embedded Runge-Kutta Formulae

Embedded RK formulae provide an error estimator for (almost) free.

The simplest example (not used in practice) is the following:

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \ ,\\
k_2 &= hf(x_n + h, y_n + k_1) \ ,\\
y_{n+1} &= y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 \qquad &&+\mathcal{O}(h^3) \ , \quad \text{“2nd order global error”; predictor-corrector}\\
y^*_{n+1} &= y_n + k_1 \qquad &&+\mathcal{O}(h^2) \ , \quad \text{“1st order global error”; forward Euler} \qquad \text{(II.5.26)}
\end{aligned}
$$

So one formula gives two approximations to y_{n+1}: 2nd order (y_{n+1}) and 1st order (y^*_{n+1}). For updating y, we would of course use y_{n+1}, but the error can be estimated by $\delta y_{n+1} = y_{n+1} - y^*_{n+1} = \mathcal{O}(h^2)$.

II.5.8.2 Bogaki-Shampine Embedded Runge-Kutta

Bogaki and Shampine developed the following useful 2nd/3rd order embedded RK scheme.

$$\begin{aligned} k_1 &= hf(x_n, y_n) \ , \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \ , \\ k_3 &= hf(x_n + \frac{3}{4}h, y_n + \frac{3}{4}k_2) \ , \\ y_{n+1} &= y_n + \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3 + \mathcal{O}(h^4) \\ k_4 &= hf(x_n + h, y_{n+1}) \\ y^*_{n+1} &= y_n + \frac{7}{24}k_1 + \frac{1}{4}k_2 + \frac{1}{3}k_3 + \frac{1}{8}k_4 + \mathcal{O}(h^3) \ . \end{aligned} \tag{II.5.27}$$

The error is then again

$$\delta y_{n+1} = y_{n+1} - y^*_{n+1} \ . \tag{II.5.28}$$

Note that k_4 of step n is the same as k_1 of step $n+1$. So k_1 does not need to be recomputed on step $n+1$; simply save k_4 and re-use it on the next step. This trick is called FSAL, "first same as last."

There exist embedded Runge-Kutta formulae for other orders, e.g. 4th/5th, 7th/8th, etc. Formulae for given orders are, however, not unique. They are derived, for a given order, say 2nd/3rd, by (*1*) trying to match stability regions of the 2nd and 3rd order formulae and (*2* choosing coefficients so that leading-order error terms are dominant in the error estimate. This is a complicated business and we strongly recommend that you use well-studied embedded formulae rather than trying to derive new ones yourself.

II.5.8.3 Adjusting the Step Size h

Now we have an error estimate $\delta y_{n+1} = y_{n+1} - y^*_{n+1}$. Our goal must now be to keep the error small: $|\delta y_{n+1}| \leq \epsilon$ by adjusting h.

Usually, one sets

$$\epsilon = \underbrace{\epsilon_a}_{\substack{\text{absolute error} \\ \text{tolerance}}} + |y_{n+1}| \underbrace{\epsilon_r}_{\substack{\text{relative error} \\ \text{tolerance}}} \ . \tag{II.5.29}$$

Now define

$$\Delta = \frac{|\delta y_{n+1}|}{\epsilon} \ , \tag{II.5.30}$$

and we want $\Delta \approx 1$.

Note that for a p-th-order formula, $\Delta \sim \mathcal{O}(h^p)$. So if you took a step h and got a value Δ, then the step $h_{\rm desired}$ you need to get $\Delta_{\rm desired}$ is

$$h_{\rm desired} = h \left| \frac{\Delta_{\rm desired}}{\Delta} \right|^{\frac{1}{p}} \ , \tag{II.5.31}$$

and $\Delta_{\rm desired} = 1$.

The algorithm to adjust h can be written as follows:

(a) Take step h, measure Δ.

(b) If $\Delta > 1$ (error too large), then
- set $h_{\rm new} = h \left|\frac{1}{\Delta}\right|^{\frac{1}{p}} S$, where S is a fudge factor ($\sim$0.9 or so).
- *reject* the old step, redo with $h_{\rm new}$.

(c) If $\Delta < 1$ (error too small), then
- set $h_{\rm new} = h \left|\frac{1}{\Delta}\right|^{\frac{1}{p}} S$.
- accept old step, take next step with $h_{\rm new}$.

II.5.8.4 Other Considerations for improving RK Methods

PI Control

In control systems language, $h_{\rm new} = h \left|\frac{1}{\Delta}\right|^{1/p} S$ is an "integral controller" for $\log h$. One can improve the stability of the controller by adding a "proportional" term. The resulting algorithm is then $h_{n+1} = S h_n \Delta_n^{-\alpha} \Delta_{n-1}^{\beta}$, with $\alpha \approx \frac{1}{p} - \frac{3}{4}\beta$ and $\beta \approx \frac{0.4}{p}$.

Dense Output

Some RK formulae allow free accurate interpolation to get values of y at arbitrary x within a single step by using the k_i values that have been computed, i.e.,

$$y(x_n + \theta h) = y_n + b_1(\theta) k_1 + b_2(\theta) k_2 + \cdots \,, \tag{II.5.32}$$

where $0 \leq \theta \leq 1$.

With this feature, one can (for example) output at every $\Delta x = 0.2$ while still allowing fully adaptive h, including $h > 0.2$, if the error tolerance allows.

II.6 Root Finding

In a broad range of applications one encounters situations in which it is necessary to find the roots of an equation, i.e., the values of x (x could be a scalar or a vector) for which $f(x) = 0$ (where f could be a single equation or a system of equations). f can either directly depend on x (explicitly) or have and implicit dependence on x.

There are a variety of ways to find the roots of an equation.

II.6.1 Newton's Method

Newton's Method is also referred to as the "Newton-Raphson Method". If we expand a function $f(x)$ about its root x_r, we get:

$$f(x_r) = f(x) + (x_r - x)f'(x) + \mathcal{O}(x^2) = 0 \ . \tag{II.6.1}$$

In this, x_r can be seen a trial value for the root x_r at the n-th step of an iterative procedure. The $n+1$-th step is then

$$f(x_{n+1}) = f(x_n) + \underbrace{(x_{n+1} - x_n)}_{\delta x} f'(x_n) \approx 0 \ , \tag{II.6.2}$$

and, thus,

$$x_{n+1} = x_n + \delta x = x_n - \frac{f(x_n)}{f'(x_n)} \ . \tag{II.6.3}$$

The iteration is stopped when $f(x_{n+1}) = 0$ (which rarely happens, because we are dealing with floating point numbers) or when the fractional change between iteration n and $n+1$ is smaller than some small number: $|[f(x_{n+1}) - f(x_n)]/f_{x_n}| < \epsilon$. One should not expect ϵ to be smaller than floating point accuracy.

Newton's Method as quadratic convergence, provided $f(x)$ is well behaved and that one has a good initial guess for the root. It also requires the ability to evaluate the derivative $f'(x_n)$ directly. If this is not possible, one resorts to the Secant Method.

II.6.2 Secant Method

The Secant Method is just like Newton's method, but this time, we have to evaluate the first derivative $f'(x_n)$ numerically. This is usually done with a backward difference:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f_n - f_{n-1}} \ . \tag{II.6.4}$$

Of course, this method will converge less rapidly than Newton's method, because we have introduced a first-order derivative. Also, we need now two points to start the iteration (or some other guess at $f'(x_n)$ at $n = 0$). Like Newton's method, the secant method may fail for misbehaved functions and a good initial guess at the root helps a lot in finding it.

II.6.3 Bisection

The intermediate-value theorem states that a continuous function $f(x)$ has at least one root in the interval $[a, b]$ if $f(a)$ and $f(b)$ are of opposite sign (this seems self-evident, but mathematicians like to have theorems and proofs for such things).

The bisection method – which is really more of an algorithm than anything – exploits the intermediate-value theorem. It goes as follows:

(i) Pick initial values of a and b so that $f(a)$ and $f(b)$ have opposite sign.

(ii) Compute the midpoint $c = \frac{a+b}{2}$. If $f(c) = 0$ or $|[f(c) - f(a)]/f(a)| < \epsilon$ or $|[f(c) - f(b)]/f(b)| < \epsilon$, then one is done.
If not:
  (1) If $f(a)$ and $f(c)$ have opposite sign, then they bracket a root. Go to (i) with $a = a$, $b = c$.
  (2) If $f(c)$ and $f(b)$ have opposite sign, then they bracket a root. Got to (i) with $a = c$, $b = b$.

The bisection method is very effective, more robust, but generally not as fast as Newton's Method, requiring more iterations until a root is found.

II.6.4 Multi-Variate Root Finding

It is often the case that one needs to simultaneously find the roots of multiple equations with multiple independent variables.

$\mathbf{f}(\mathbf{x})$ is a multi-variate vector function and we are looking for $\mathbf{f}(\mathbf{x}) = 0$. Analogously to the scalar case, we write the multi-variate Newton's/Secant Method,

$$\mathbf{f}(\mathbf{x}_{n+1}) \approx \mathbf{f}(\mathbf{x}_n) + \nabla \otimes \mathbf{f}(\mathbf{x}_n)(\mathbf{x}_{n+1} - \mathbf{x}_n) = 0 \ , \tag{II.6.5}$$

and

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [\nabla \otimes \mathbf{f}(\mathbf{x}_n)]^{-1} \mathbf{f}(\mathbf{x}) \ . \tag{II.6.6}$$

Here,

$$\mathbf{J} \equiv \nabla \otimes \mathbf{f}(\mathbf{x}_n) \ , \tag{II.6.7}$$

is the Jacobian matrix. In index notation, it is given by

$$J_{ij} = \frac{\partial f_i}{\partial x_j} \ . \tag{II.6.8}$$

There are also multi-variate variants of the bisection method, but they are rather complex and we will not discuss them here.

II.7 Linear Systems of Equations

Linear systems of equations (LSEs) are everywhere:

- Interpolation (e.g., for the computation of the spline coefficients).
- ODEs (implicit time integration).
- Solution methods for elliptic PDEs.
- Solution methods for non-linear equations by linearization and Newton iterations.

Example applications in astrophysics are plenty: Stellar structure and evolution, Poisson solvers, radiation transport and radiation-matter coupling, nuclear reaction networks. One encounters LSEs basically anywhere where implicit solutions are necessary, because balance/equilibrium must be found.

II.7.1 Basics

A system of linear equations can be written in matrix form:

$$A\mathbf{x} = \mathbf{b} \ . \tag{II.7.1}$$

Here, A is a real $n \times n$ matrix with coefficients a_{ij}. $\mathbf{b}$ is a given real vector. $\mathbf{x}$ is the vector of n unknowns.

In the following, we will use $\mathbf{I}$ to indicate the identity matrix and $\mathbf{O}$ to indicate the zero matrix.

A quick flash-back from Linear Algrebra:

A LSE of the form of Eq. II.7.1 has a unique solution if and only if $\det A = |A| \neq 0$ and $\mathbf{b} \neq 0$. The solution then is

$$\mathbf{x} = A^{-1}\mathbf{b} \ , \tag{II.7.2}$$

where A^{-1} is the inverse of A with $AA^{-1} = A^{-1}A = \mathbf{I}$. For the case of $\det A = 0$, the equations either have no solution (i.e., they form an inconsistent set of equations) or an infinite number of solutions (i.e, they are an undetermined set of equations).

II.7.2 Matrix Inversion – The Really Hard Way of Solving an LSE

The inverse of a matrix A is given by

$$A^{-1} = \frac{1}{|A|} \underbrace{\mathrm{adj}A}_{\text{adjugate}} \ . \tag{II.7.3}$$

the adjugate of A is the transpose of A's cofactor matrix C:

$$\mathrm{adj}A = C^T \ . \tag{II.7.4}$$

So the problem boils down to finding C and $\det A$. How to do this, you learned in your Linear Algebra class, but here is a quick reminder:

II.7.2.1 Finding $\det A$ for an $n \times n$ Matrix

If the $n \times n$ matrix A is triangular, that is

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ & a_{22} & \vdots \\ 0 & & a_{nn} \end{pmatrix} \quad \text{upper triangular matrix,} \tag{II.7.5}$$

or

$$A = \begin{pmatrix} a_{11} & & 0 \\ \vdots & a_{22} & \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \quad \text{lower triangular matrix,} \tag{II.7.6}$$

then computing the determinant is trivial:

$$\det A = \prod_{i=1}^{n} a_{ii} \ . \tag{II.7.7}$$

If A is not triangular, one resorts either to Laplace's Formula or to the Leibniz Formula.

Laplace's Formula:

$$\det A = \sum_{j=1}^{n} a_{ij} c_{ij} \tag{II.7.8}$$

for any row i. c_{ij} are the coefficients of the cofactor matrix of A (see §II.7.2.2).

Leibniz Formula:

$$\det A = \sum_{\sigma \in S_n} \operatorname{sign} \sigma \prod_{i=1}^{n} a_{i\sigma} \ , \tag{II.7.9}$$

where:

σ : is a permuatation of the set $\{1, 2, \cdots, n\}$,
S_n : is the complete set of possible permutations, namely a symmetric group on n elements,

and

$$\operatorname{sign} \sigma = \begin{cases} +1 & \text{if } \sigma \text{ is an even permutation,} \\ -1 & \text{if } \sigma \text{ is an odd permutation.} \end{cases} \tag{II.7.10}$$

Here, *even* means that σ can be obtained with by an even number of replacements and *odd* means that σ can be obtained by an odd number of replacements.

Here is an example for $n = 3$:

σ	Result	# of replacements	$\operatorname{sign} \sigma$
1	123	0	+1
2	231	2	+1
3	312	2	+1
4	213	1	-1
5	132	1	-1
6	321	1	-1

Note that there are $n!$ possible permutations!

II.7.2.2 The Cofactor Matrix of an $n \times n$ Matrix

$$c_{ij} = (-1)^{i+j} m_{ij} \ , \tag{II.7.11}$$

where m_{ij} is the *minor* for the coefficient a_{ij} of our $n \times n$ matrix A.

Finding the minors m_{ij} of A is a multi-step process that is best implemented in a recursive algorithm:

To find the minors m_{ij} of A,

(1) choose an entry a_{ij} of A,

(2) create a submatrix B that contains all coefficients of A that do not belong to row i and column j,

(3) obtain the determinant of B.
This is trivial, if B is 2×2, easy if it is 3×3. If B is larger than 3×3, then use

$$\det B = \sum_{j=1}^{n} b_{ij} c_{ij}^{B}$$

for any convenient i of B. $c_{ij}^{B} = (-1)^{i+j} m_{ij}^{B}$ is determined by recursion through (1).

II.7.3 Cramer's Rule

Cramer's Rule is a smarter way to solve LSEs of the kind

$$A\mathbf{x} = \mathbf{b} \ . \tag{II.7.12}$$

Provided A is invertible (i.e., has non-zero $\det A$), the solution to Eq. II.7.12 is then

$$x_i = \frac{\det A_i}{\det A} \ , \tag{II.7.13}$$

where A_i is the matrix formed from A by replacing its i-th column by the column vector $\mathbf{b}^T$.

Cramer's rule is more efficient that matrix inversion. The latter scales in complexity with $n!$ (where n is the number of rows/columns of A), while Cramer's rule has been shown to scale with n^3, so is more efficient for large matrixes and has comparable efficiency to direct methods such as Gauss Elimination, which we will discuss in §II.7.4.1.

II.7.4 Direct LSE Solvers

There are two main classes of sensible algorithms to solve LSEs. *Direct methods* consist of a finite set of transformations of the original coefficient matrix that reduce the LSE to one that is easily solved. *Indirect methods* (also called *iterative methods*) consist of algorithms that specify a series of steps that lead closer and closer to the solution without, however, ever exactly reaching it.

II.7.4.1 Gauss Elimination

The idea behind Gauss elimination – and all other direct methods – is to bring a LSE into a form that has an easy solution. Let us consider the following:

$$A\mathbf{x} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \ . \tag{II.7.14}$$

This LSE is solved trivially by simple back-substitution:

$$x_3 = \frac{b_3}{a_{33}} \ , \qquad x_2 = \frac{1}{a_{22}}(b_2 - a_{23}x_3) \ , \quad \text{and} \quad x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3) \ . \tag{II.7.15}$$

The Gauss algorithm consists of a series of steps to bring any $n \times n$ matrix into the above upper triangular form. It goes as follows:

(1) Sort the rows of A so that the diagonal coefficient a_{ii} (called *the pivot*) of row i (for all i) is non-zero. If this is not possible, the LSE cannot be solved.

(2) Replace the j-th equation with

$$-\frac{a_{j1}}{a_{11}} \times (\text{1-st equation}) + (j\text{-th equation}) \ , \tag{II.7.16}$$

where j runs from 2 to n. This will zero-out column 1 for $i > 1$.

(3) Repeat the previous step, but starting with the next row down and with $j >$ (current row number). The current row be row k. Then we must replace rows j, $j > k$, with

$$-\frac{a_{jk}}{a_{kk}} \times (k\text{-th equation}) + (j\text{-th equation}) \ , \tag{II.7.17}$$

where $k < j \leq n$.

(4) Repeat (3) until all rows have been reduced and the matrix is in upper triangular form.

(5) Back-substitute to find $\mathbf{x}$.

While the Gauss Elimination algorithm is straightforward, it is not very computationally efficient, since it requires $\mathcal{O}(n^3)$ operations for n equations. This can also lead to build-up of floating point errors.

II.7.4.2 Pivoting

Gauss Elimination suffers from poor accuracy if the matrix coefficients are of very different sizes. Here is an example:

$$\begin{aligned} 10^{-5}x_1 + \quad x_2 &= 1 \\ x_1 + \quad x_2 &= 2 \ . \end{aligned} \tag{II.7.18}$$

Using Gauss Elimination, the exact solution is:

$$x_1 = \frac{10^5}{99999} \approx 1 \ , \quad x_2 = \frac{-99998}{-99999} \approx 1 \ . \tag{II.7.19}$$

If, however, we solve the problem using floating point numbers, the result may be quite different. Let's say $m = 4$,

$$\left(\begin{array}{cc|c} 0.1000 \times 10^{-4} & 0.1000 \times 10^{1} & 0.1000 \times 10^{1} \\ 0.1000 \times 10^{1} & 0.1000 \times 10^{1} & 0.2000 \times 10^{1} \end{array} \right) \ , \tag{II.7.20}$$

becomes

$$\left(\begin{array}{cc|c} 0.1000 \times 10^{-4} & 0.1000 \times 10^{1} & 0.1000 \times 10^{1} \\ 0.0000 \times 10^{0} & -0.1000 \times 10^{6} & -0.1000 \times 10^{6} \end{array} \right) \ , \tag{II.7.21}$$

and, thus,

$$x_1 = 1 \quad x_2 = 0 \ , \tag{II.7.22}$$

which is an incorrect result.

The above is an example of what is called *poor scaling*. Whenever the coefficients of an LSE are of greatly varying sizes, we must expect that rounding errors may build up due to loss of significant figures.

Poor scaling can be fixed by the following simple procedure:

At each of the intermediate Gauss steps, compare the size of the pivot a_{kk} with all a_{jk}, $k < j \leq n$, and exchange row k with the row of the largest a_{jk}.

This simple procedure minimized the floating point error in Gaussian elimination. It is called *partial pivoting*. Here is the example from above, but this time with pivoting:

$$\left(\begin{array}{cc|c} 0.1000 \times 10^{1} & 0.1000 \times 10^{1} & 0.2000 \times 10^{1} \\ 0.1000 \times 10^{-4} & 0.1000 \times 10^{1} & 0.1000 \times 10^{1} \end{array} \right) \ , \tag{II.7.23}$$

which becomes

$$\left(\begin{array}{cc|c} 0.1000 \times 10^{1} & 0.1000 \times 10^{1} & 0.2000 \times 10^{1} \\ 0.0000 \times 10^{0} & 0.1000 \times 10^{1} & 0.1000 \times 10^{1} \end{array} \right) \ , \tag{II.7.24}$$

and

$$x_1 = 1 \quad x_2 = 1 \ , \tag{II.7.25}$$

which is an almost fully correct result.

Essentially, pivoting modifies the replacement of row j with row $j - \epsilon$ row k so that $0 < \epsilon < 1$ is as small as possible.

Total pivoting exists as well and exchanges also columns so that the largest element in the matrix becomes the pivot. This is very time consuming and we do not discuss it here.

II.7.4.3 Decomposition Methods (LU Decomposition)

The idea behind decomposition methods is to split a given LSE into smaller, considerably easier to solve parts. The simplest such method is the Lower-Upper (LU) decomposition.

Given $A\mathbf{x} = \mathbf{b}$, suppose we can write the matrix A as

$$A = LU \ , \tag{II.7.26}$$

where L is lower triangular and U is upper triangular. The solution of the LSE than becomes straightforward:

$$A\mathbf{x} = \mathbf{b} \longrightarrow (LU)\mathbf{x} = \mathbf{b} \longrightarrow L(U\mathbf{x}) = \mathbf{b} \ . \tag{II.7.27}$$

If we now set $\mathbf{y} = U\mathbf{x}$, then we have transformed the original system into two systems

$$\begin{aligned} &(1) \quad L\mathbf{y} = \mathbf{b} \ , \\ &(2) \quad U\mathbf{x} = \mathbf{y} \ . \end{aligned} \tag{II.7.28}$$

Both these LSEs are triangular. (1) can be trivially solved by forward-substitution and (2) can be trivially solved by back-substitution. So we have now to solve two LSEs instead of one, but both are very easy to solve! The difficult part is now to find the L and U parts of A! This is done via matrix factorization.

II.7.4.4 Factorization of a Matrix

The process of decomposing A into L and U parts is called factorization. Any matrix A that can be brought into U form without swapping any rows has a LU decomposition. This decomposition is not generally unique and there are multiple ways of factorizing A.

A is an $n \times n$ matrix with n^2 coefficients. L and U are triangular and have $n(n+1)/2$ entries each for a total of $n^2 + n$ entries. Hence, L and U together have n coefficients more than A and these can be chosen to our own liking.

To derive the LU factorization, we begin by writing out $A = LU$ in coefficients:

$$a_{ij} = \sum_{s=1}^{n} l_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} l_{is} u_{sj} \ , \tag{II.7.29}$$

where we have used that $l_{is} = 0$ for $s > i$ and $u_{sj} = 0$ for $s > j$.

Let's start with entry $a_{ij} = a_{11}$:

$$a_{11} = l_{11} u_{11} \ . \tag{II.7.30}$$

We can now make use of the freedom of choosing n coefficients of L and U.

In *Doolittle's factorization*, one sets $l_{ii} = 1$, which makes L *unit triangular*. In *Crout's factorization*, one sets $u_{ii} = 1$, which means that U is unit triangular.

Following Doolittle, we set $l_{11} = 1$ and, with this, $u_{11} = a_{11}$. We can now compute all the elements of the first row of U and of the first column of L by setting $i = 1$ or $j = 1$,

$$\begin{aligned} u_{1j} &= a_{1j} \qquad && i = 1, j > 1 \ , \\ l_{i1} &= \frac{a_{i1}}{u_{11}} \qquad && j = 1, i > 1 \ . \end{aligned} \tag{II.7.31}$$

Consider now a_{22}:

$$a_{22} = l_{21} u_{12} + l_{22} u_2 2 \ . \tag{II.7.32}$$

With Doolittle, $l_{22} = 1$, thus $u_{22} = a_{22} - l_{21}u_{12}$, where u_{12} and l_{21} are known from the previous steps. The second row of U and the second column of L are now calculated by setting either $i = 2$ or $j = 2$:

$$\begin{aligned} u_{2j} &= a_{2j} - l_{21}u_{1j} \qquad & i = 2, j > 2 \ , \\ l_{i2} &= \frac{a_{i2} - l_{i1}u_{12}}{u_{22}} \qquad & j = 2, i > 2 \ . \end{aligned} \tag{II.7.33}$$

This procedure can be repeated for all the rows and columns of U and L. In the following, we provide a compact form of the algorithm in pseudocode.

For $k = 1, 2, \cdots, n$ do

- Choose either l_{kk} (Doolittle) or u_{kk} (Crout) [the choice must be non-zero] and compute the other from
$$l_{kk}u_{kk} = a_{kk} - \sum_{s=1}^{k-1} l_{ks}u_{sk} \ . \tag{II.7.34}$$
- Build the k-th row of U:
For $j = k+1, \cdots, n$ do:
$$u_{kj} = \frac{1}{l_{kk}} \left(a_{kj} - \sum_{s=1}^{k-1} l_{ks}u_{sj} \right) \ . \tag{II.7.35}$$
- Build the k-th column of L:
For $i = k+1, \cdots, n$ do:
$$l_{ik} = \frac{1}{u_{kk}} \left(a_{ik} - \sum_{s=1}^{k-1} l_{is}u_{sk} \right) \ . \tag{II.7.36}$$

The LU decomposition algorithm has a complexity of $\mathcal{O}(n^3)$, but the constant in front is smaller than with pure Gaussian Elimination.

II.7.4.5 Tri-Diagonal Systems

Consider the following 4×4 LSE:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 \\ a_1 & b_2 & c_2 & 0 \\ 0 & a_2 & b_3 & c_3 \\ 0 & 0 & a_3 & b_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} \ . \tag{II.7.37}$$

Tri-diagonal LSEs like the above give rise to particularly simple results when using Gaussian elimination. Forward elimination at each step yeilds:

$$\begin{pmatrix} 1 & c_1/d_1 & 0 & 0 \\ 0 & 1 & c_2/d_2 & 0 \\ 0 & 0 & 1 & c_3/d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \ , \tag{II.7.38}$$

with

$$\begin{aligned} d_1 &= b_1\,, & y_1 &= f_1/d_1\,,\\ d_2 &= b_2 - a_1c_1/d_1\,, & y_2 &= (f_2 - y_1a_1)/d_2\,,\\ d_3 &= b_3 - a_2c_2/d_2\,, & y_3 &= (f_3 - y_2a_2)/d_3\,,\\ d_4 &= b_4 - a_3c_3/d_3\,, & y_4 &= (f_4 - y_3a_3)/d_4\,. \end{aligned} \tag{II.7.39}$$

Back-substitution than gives

$$\begin{aligned} x_1 &= y_1 - x_2c_1/d_1\,,\\ x_2 &= y_2 - x_3c_2/d_2\,,\\ x_3 &= y_3 - x_4c_3/d_3\,,\\ x_4 &= y_4\,. \end{aligned} \tag{II.7.40}$$

In the general $n \times n$ case, the procedure is as follows:

- Forward Elimination:
  (1) At the first step: $d_1 = b_1$ and $y_1 = f_1/d_1$.
  (2) At the k-th step:
  $$d_k = b_k - a_{k-1}c_{k-1}/d_{k-1}\,,$$
  $$y_k = (f_k - y_{k-1}a_{k-1})/d_k\,.$$
- Backward Substitution: Determine all x_k by

$$x_n = y_n\,,$$
$$x_{k-1} = y_{k-1} - x_kc_{k-1}/d_{k-1}\,.$$

The complexity of this algorithm is still $\mathcal{O}(n^3)$, but the constant in front is much smaller than that of LU decomposition for this kind of matrix.

II.7.5 Iterative Solvers

Direct methods for the solution of LSEs generally have a computational complexity of $\mathcal{O}(n^3)$ and also suffer from round-off errors due to the many operations necessary for a direct solutio. If one is satisfied with an approximate solution, one can resort to iterative methods that start from some initial "guess" $\mathbf{x}^{(0)}$ and define a sequence of approximations $\mathbf{x}^{(1)}$, ..., $\mathbf{x}^{(m)}$, which, in principle, converge to the exact solution.

A large class of iterative methods can be defined as follows:

Given

$$A\mathbf{x} = \mathbf{b}\,, \qquad \text{with } \det A \neq 0\,, \tag{II.7.41}$$

then the coefficient matrix can be split – in an infinite number of ways – into the form

$$A = N - P\,, \tag{II.7.42}$$

where N and P are matrices of the same order as A. The LSE is then written as

$$N\mathbf{x} = P\mathbf{x} + \mathbf{b}\,. \tag{II.7.43}$$

Starting from some vector $\mathbf{x}^{(0)}$, we define a sequence of vectors $\{\mathbf{x}^{(i)}\}$ by the recursion

$$N\mathbf{x}^{(i)} = P\mathbf{x}^{(i-1)} + \mathbf{b} \ , \qquad i = 1, 2, \cdots \ . \tag{II.7.44}$$

The various iteration methods available in the literature are characterized by their choices of N and P. Right away, one notes from Eq. II.7.44 that N must be chosen such that $\det N \neq 0$. It should also be chosen such that the LSE of the form $N\mathbf{y} = \mathbf{z}$ that must be solved at each iteration step is easily solved by a direct method.

It is possible to show the following (we will not show proofs):

(1) An iterative method converges if and only if the *convergence matrix* $M = N^{-1}P$ exists and has all eigenvalues λ_i with modulus less than 1:

$$\rho(M) = \max_i |\lambda_i| < 1 \ . \tag{II.7.45}$$

(2) A weaker, but more straightforwardly useful statement is that the method converges if at least one norm of the convergence matrix is strictly less than one:

$$||M|| < 1 \ . \tag{II.7.46}$$

Note, however, that the reverse is not true. That is a method may converge, but some of the norms may be equal or larger than 1. In the above, we have used the matrix norm, which is defined analogously to the vector norm,

$$||\mathbf{x}||_p = \left[\sum_j x_j^p\right]^{1/p} \ , \qquad p > 0 \ . \tag{II.7.47}$$

The matrix norm is then

$$||A||_p = \max_{||\mathbf{x}||_p = 1} ||A\mathbf{x}||_p \ . \tag{II.7.48}$$

In the following, we will assume that all diagonal elements of A are 1. This is easily achieved by dividing each row by its diagonal element. If a row happens to have a zero diagonal elment, we just have to re-order the rows.

II.7.5.1 Jacobi Iteration

A (with $a_{ii} = 1$) is split into diagonal and off-diagonal parts

$$A = \mathbf{I} - (A_L + A_U) \ , \tag{II.7.49}$$

so $N = I$ and $P = (A_L + A_U)$ is the sum of upper and lower triangular matrices with diagonal 0. The iteration scheme is then

$$\mathbf{x}^{(i+1)} = \mathbf{b} + (A_L + A_u)\mathbf{x}^{(i)} \ , \tag{II.7.50}$$

and the convergence matrix is simply $M = N^{-1}P = P$.

II.7.5.2 Gauss-Seidel Iteration

In Jacobi's method, the old guess is used to estimate all the elements of the new guess. In Gauss-Seidel iteration, each new iterate is used as soon as it becomes available:

$$\begin{aligned} A &= \mathbf{I} - (A_L + A_U) \ , \\ \mathbf{x}^{(i+1)} &= \mathbf{b} + A_L \mathbf{x}^{(i+1)} + A_U \mathbf{x}^{(i)} \ . \end{aligned} \tag{II.7.51}$$

Hence, $N = \mathbf{I} - A_L$ and $P = A_U$. As before, A_L and A_U are the lower and upper diagonal parts of A with diagonal elements set to zero. If we start computing the elements of $\mathbf{x}^{(i+1)}$ from the first, i.e., from x_1^{i+1}, then all the terms on the RHS are known by the time the are used. Specifically, we have

$$x_k^{(i+1)} = b_k + \sum_{l>j} a_{jl}^U \, x_l{}^{(i)} + \sum_{l<j} a_{jl}^L \, x_l^{(i+1)} \ , \tag{II.7.52}$$

where we denote the coefficients of A_U and A_L as a_{jl}^U and a_{jl}^L, respectively.

II.7.5.3 Successive Over-Relaxation (SOR) Method

One can interpret Gauss-Seidel iteration as a method that applies at each guess $\mathbf{x}^{(i)}$ a correction term $\mathbf{c}$. We can rewrite this scheme as

$$\begin{aligned} \mathbf{x}^{(i+1)} &= \mathbf{b} + A_L \mathbf{x}^{(i+1)} + A_U \mathbf{x}^{(i)} \ , \\ &= \mathbf{x}^{(i)} + \underbrace{\left[\mathbf{b} + A_L \mathbf{x}^{(i+1)} + (A_U - \mathbf{I}) \mathbf{x}^{(i)} \right]}_{\mathbf{c}} \ , \\ &= \mathbf{x}^{(i)} + \mathbf{c} \ . \end{aligned} \tag{II.7.53}$$

The idea of SOR is now that the convergence of the method can be pushed by using a slightly larger correction factor $\mathbf{c}' = \omega \mathbf{c}$ with $\omega \gtrsim 1$. Since this can also accelerate divergence, ω is chosen to be close to 1.

II.7.6 Aside on Determinants and Inverses

Now that we have studied how to solve LSEs, it is much easier to find determinants and inverses of matrices!

Determinant

The determinant of a well-behaved matrix is easily found via LU decomposition.

$$\begin{aligned} A &= LU \ , \\ \det A &= \det L \det U \ . \end{aligned} \tag{II.7.54}$$

Since both L and U triangular, we have

$$\det L = \prod_{i=1}^{n} l_{ii} \quad \text{and} \quad \det U = \prod_{i=1}^{n} u_{ii} \ . \tag{II.7.55}$$

Furthermore, in Doolite factorization $\det L = 1$ and in Crout factorization $\det U = 1$. Note that if pivoting was used, then the sign of $\det A$ depends on the number of even and odd permutations that were carried out.

Inverse

We can solve for A^{-1},

$$AA^{-1} = \mathbf{I} \ , \tag{II.7.56}$$

by solving n systems of n linear equations. Let e_i be the i-th unit vector (only its i-th component is non-zero), then we have to solve

$$A\mathbf{c}_i = \mathbf{e}_i \tag{II.7.57}$$

for $i = 1, \cdots, n$. c_i is the i-th column of the inverse A^{-1}.

This method is faster than normal inversion the requries the evaluation of $n + 1$ determinants in Cramer's rule.

II.8 Ordinary Differential Equations: Boundary Value Problems

As mentioned already at the beginning of §II.5, a boundary value problem (BVP) consists of finding a solution of an ODE in an interval $[a, b]$ that satisfies constraints at both ends (boundary conditions).

A typical example of a BVP is

$$y'' = f(x, y, y') \ , \ \ y(a) = A \ , \ \ y(b) = B \ , \ \text{and } x \in [a, b] \ . \tag{II.8.1}$$

II.8.1 Shooting Method

The shooting method is a very frequently used approach to BVPs. The idea behind it is to reduce a BVP to an initial value problem by making an educated guess on unknown inner boundary conditions and then iterating until a modified guessed inner boundary condition leads to the correct known outer boundary value.

In the example given in Eq. II.8.1, the value of $y'(a)$ is not known, but is needed for the problem's solution. We can make a guess $y'(a) = z$ and then we know $f(a, y(a), y'(a))$, can reduce the second-order problem to two first-order problems and just integrate the two ODEs out to b. Since we have chosen z, we have now solved $y = y(x, z)$, but our goal is to find y such that $y(b, z) = B$.

In other words, we can define a new function

$$\Phi(z) = y(b, z) - B \tag{II.8.2}$$

and search for a z so that $\Phi(z) = 0$. Hence, we are looking for the root of $\Phi(z)$! For this we can use what was discussed in §II.6.

The full shooting algorithm for $y'' = f(x, y, y')$ goes than as follows:

(1) Guess a starting value $z_0 = y'(a)$, set the iteration counter $i = 0$.

(2) Compute $y = y(x, z_i)$ by integrating the IVP.

(3) Compute $\Phi(z_i) = y(b, z_i) - B$. If z_i does not give a sufficiently accurate solution of the full problem, increment i to $i + 1$ and find a value for z_{i+1} using a root finder on $\Phi(z_i) = 0$. Then go back to (2).

Note that one typically ends up with the secant method, since the derivative of $\Phi(z)$ is not known in the general case and one is stuck with having to numerically compute it. For this, at least two guesses for z are needed.

II.8.2 Finite-Difference Method

BVPs of the kind given by Eq. (II.8.1) can be solved by Taylor expanding the ODE itself to linear order (assuming there are no non-linearities in y and y'):

$$y'' = g(x) - p(x)y' - q(x)y \ , \tag{II.8.3}$$

where $g(x)$, $p(x)$, and $q(x)$ are functions of x only and the sign convention is arbitrary.

We can now discretize y' and y'' on an evenly spaced grid with step size h,

$$\begin{aligned} y'(x_i) &= \frac{y(x_{i+1}) - y(x_{i-1})}{2h} \ , \\ y''(x_i) &= \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} \ , \end{aligned} \tag{II.8.4}$$

where $x_i = a + ih$, $i = 0, \cdots, n+1$, and $h = (b-a)/(n+1)$.

The discrete version of Eq. (II.8.3) is then a system of $n+2$ linear algebraic equations,

$$\begin{aligned} y_0 &= A \ , \\ y_{i-1}(1 - \frac{h}{2}p_i) - y_i(2 - h^2 q_i) + y_{i+1}(1 + \frac{h}{2}p_i) &= h^2 g_i \ , \\ y_{n+1} &= B \ , \end{aligned} \tag{II.8.5}$$

where $p_i = p(x_i)$, $g_i = g(x_i)$, and $q_i = q(x_i)$. One ends up with a tri-diagonal matrix of dimension $n \times n$:

$$\begin{pmatrix} -2 + h^2 q_1 & 1 + \frac{h}{2}p_1 & 0 & & \cdots & & 0 \\ 1 - \frac{h}{2}p_2 & \ddots & \ddots & 0 & & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \ddots & 1 + \frac{h}{2}p_{n-1} \\ \vdots & 0 & 0 & \cdots & 0 & 1 - \frac{h}{2}p_n & -2 + h^2 q_n \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} h^2 g_1 - A(1 - \frac{h}{2}p_1) \\ h^2 g_2 \\ \vdots \\ \vdots \\ \vdots \\ h^2 g_{n-1} \\ h^2 g_n - B(1 + \frac{h}{2}p_n) \end{pmatrix} \tag{II.8.6}$$

This system can be solved in a straightforward fashion using the methods discussed in §II.7.

II.9 Partial Differential Equations

One can fill an entire term just talking about formal aspects of partial differential equations (PDEs). Here we just provide a quick and dirty introduction to PDEs in order to familiarize you with the basics and with key terminology. There are multiple specialized PDE classes offered in Caltech's applied math department (this is probably true for any applied math department in the world) and we encourage you to attend them should you be interested to learn about the details of PDEs.

Preliminaries and Definitions

- A PDE is a relation between the partial derivatives of an unknown function and the independent variables.
- The order of the highest derivative sets the *order* of the PDE. If the highest derivative is a second derivative, we are dealing with a second-order PDE.
- A PDE is *linear* if it is of the first degree in the dependent variable (i.e. the unknown function) and in its partial derivatives.
- If each term of a PDE contains either the dependent variable or one of its partial derivatives, the PDE is called *homogeneous*. Otherwithe it is *non-homogeneous*.

II.9.1 Types of PDEs

There are three general types of PDEs: hyperbolic PDEs, parabolic PDEs, and elliptic PDEs. Not all PDEs fall into one of these three types, but many PDEs used in practice do. These classes of PDEs model different sorts of phenomena, display different behavior, and require different numerical techniques for their solution.

It is not always straighforward to see (to show and proof) what type of PDE a given PDE one is dealing with is.

For the special case of linear second-order differential equations for an unknown function u in two independent variables x and y of the form

$$a\partial^2_{xx}u + b\partial^2_{xy}u + c\partial^2_{yy}u + d\partial_x u + e\partial_y u + fu = g \ , \tag{II.9.1}$$

can be straightforwardly categorized based on the discriminant,

$$b^2 - 4ac \begin{cases} < 0 & \rightarrow \quad \text{elliptic}, \\ = 0 & \rightarrow \quad \text{parabolic}, \\ > 0 & \rightarrow \quad \text{hyperbolic}, \end{cases} \tag{II.9.2}$$

and the names come from analogy with conic sections in the theory of ellipses.

II.9.1.1 Hyperbolic PDEs

Hyperbolic equations in physics and astrophysics describe *dynamical* processes and systems that generally start at some initial time $t_0 = 0$ with some initial conditions. The equations are then integrated in time.

The prototypical linear second-order hyperbolic equation is the homogeneous wave equation,

$$c^2\partial^2_{xx}u - \partial^2_{tt}u = 0 \ , \tag{II.9.3}$$

where c is the wave speed.

Another important class of hyperbolic equations are the *first-order hyperbolic systems*. In one space dimension and assuming a linear problem, such a PDE system has the form

$$\partial_t u + A\partial_x u = 0 \ , \tag{II.9.4}$$

where $u(x,t)$ is the state vector with s components and A is a $s \times s$ matrix. The problem is called *hyperbolic* if A has only real eigenvalues and is diagonizable, i.e., has a complet set of linearly independent eigenvectors so that one can construct a matrix

$$\Lambda = Q^{-1}AQ \ , \tag{II.9.5}$$

where Λ is diagonal and has real numbers on its diagonal. A key example of such an equation is the linear *advection equation*:

$$\partial_t u + v\partial_x u = 0 \ , \tag{II.9.6}$$

which is first order, linear, and homogeneous.

More complicated and non-linear first-order systems can still be written in the form of Eq. II.9.4). Consider

$$\partial_t u + \partial_x F(u) = 0 \ , \tag{II.9.7}$$

where $F(u)$ is the *flux* and may or may not be non-linear in u. We can re-write this PDE in *quasi-linear* form, by introducing the Jacobian

$$\bar{A} = \frac{\partial F}{\partial u} \ , \tag{II.9.8}$$

and writing

$$\partial_t u + \bar{A}\partial_x u = 0 \ . \tag{II.9.9}$$

This PDE is hyperbolic if $\bar{A}$ has real eigenvalues and is diagonizable. The *equations of hydrodynamics* are a key example of a non-linear, first-order hyperbolic PDE system.

Initial Conditions for Hyperbolic Problems

One must specify $u(x, t = 0)$ (at all x) and (order-1) time derivatives.

Boundary Conditions for Hyperbolic Problems

One must specify either von Neumann, Dirichlet, or a mixture of von Neumann and Dirichlet boundary conditions:

(1) **Dirichlet Boundary Conditions**.
Let 0 be the inner and L be the outer boundary of the domain.

$$\begin{aligned} u(x = 0, t) &= \Phi_1(t) \ , \\ u(x = L, t) &= \Phi_2(t) \ . \end{aligned} \tag{II.9.10}$$

(2) **von Neumann Boundary Conditions**.
Let 0 be the inner and L be the outer boundary of the domain.

$$\begin{aligned} \partial_x u(x=0,t) &= \Psi_1(t) \ , \\ \partial_x u(x=L,t) &= \Psi_2(t) \ . \end{aligned} \tag{II.9.11}$$

Note that in a multi-D problem ∂_x turns into the derivative in the direction of the normal to the boundary.

II.9.1.2 Parabolic PDEs

Parabolic PDEs describe processes that are slowly changing, such as the slow diffusion of heat in a medium, sediments in ground water, and radiation in an opaque medium. Parabolic PDEs are second order and have generally the shape

$$\partial_t u - k\partial^2_{xx} u = f \ . \tag{II.9.12}$$

Initial Conditions for Parabolic Problems

One must specify $u(x, t = 0)$ at all x.

Boundary Conditions for Parabolic Problems

Dirichlet or von Neumann or mixed boundary conditions. If the boundary conditions are independent of time, the system will evolve towards a steady state ($\partial_t u = 0$). In this case, one may set $\partial_t u = 0$ for all times and treat Eq. (II.9.12) as an elliptic equation.

II.9.1.3 Elliptic PDEs

Elliptic equations describe systems that are static, in steady state and/or in equilibrium. There is no time dependence. A typical elliptic equation is the Poisson equation:

$$\nabla^2 u = f \ , \tag{II.9.13}$$

which one encounters in Newtonian gravity and in electrodynamics. ∇^2 is the Laplace operator, and f is a given scalar function of position. Elliptic problems may be linear (f does not depend on u or its derivatives) or non-linear (f depends on u or its derivatives).

Initial Conditions for Elliptic Problems

Do not apply, since there is no time dependence.

Boundary Conditions for Elliptic Problems

Dirichlet or von Neumann boundary conditions.

II.9.2 Numerical Methods for PDEs: A Very Rough Overview

There is no such thing as a general robust method for the solution of generic PDEs. Each type (and each sub-type) of PDE requires a different approach. Real-life PDEs may be of mixed type or may

have special properties that require knowledge about the underlying physics for their successful solution.

There are three general classes of approaches to solving PDEs:

(1) Finite Difference Methods.
The differential operators are approximated using their finite-difference representation on a given grid. A sub-class of finite-difference methods, so-called finite-volume methods, can be used for PDEs arising from conservation laws (e.g., the hydrodynamics equations).

Finite difference/volume methods have polynomial convergence for smooth functions.

(2) Finite Element Methods.
The domain is divided into cells ("*elements*"). The solution is represented as a single function (e.g., a polynomial) on each cell and the PDE is transformed to an algebraic problem for the matching conditions of the simple functions at cell interfaces.

Finite element methods can have polynomial or exponential convergence for smooth functions.

(3) Spectral Methods.
The solution is represented by a linear combination of known functions (e.g. trigonometric functions or special polynomials). The PDE is transformed to a set of algebraic equations (or ODEs) for the amplitudes of the component functions. A sub-class of these methods are the collocation methods. In them, the solution is represented on a grid and the spectral decomposition of the solution in known functions is used to estimate to a high degree of accuracy the partial derivatives of the solution on the grid points.

Spectral methods have exponential convergence for smooth functions.

II.9.3 The Linear Advection Equation, Solution Methods, and Stability

We have already seen when talking about ODEs in §II.5 that accuracy is not the only consideration when devising a numerical scheme. Stability should not be neglected and it is one of the key considerations when it comes to hyperbolic and parabolic PDEs.

Analyzing a general discretization scheme for a general PDE for stability can be a tricky and involved task. For linear PDEs, however, we can use the rather straightforward stability analysis proposed by von Neumann. *von Neumann stability analysis* can also be applied to non-linear problems if the non-linearity is weak.

The idea is that the complete solution of a PDE can be decomposed into a linear combination of eigenmodes e^{ikx} (where k is a spatial wave number). Each of these eigenmodes must individually be a solution of the PDE, so

$$u(x,t) = \gamma e^{ikx} \ , \tag{II.9.14}$$

where $\gamma = \gamma(k)$ is the complex amplitude of mode k.

Assuming that we can write the evolution of the discretized solution from time n to time $n+1$ as a (quasi)-linear operation,

$$u_j^{(n+1)} = \mathcal{D}(\Delta t, \Delta x) u_j^{(n)} \ , \tag{II.9.15}$$

then we can apply this to the case $u^{(n)} = e^{ikx}$ (we have set $\gamma = 1$ for convenience):

$$\begin{aligned} u_j^{(n+1)} &= \mathcal{D} e^{ikx_j} \ , \\ &= \xi e^{ikx_j} \end{aligned} \tag{II.9.16}$$

where ξ is a complex number and e^{ikx_j} has been factored out (which is always possible). ξ is called the *amplification factor*, since the evolution of a single eigenmode from time 0 to time $n+1$ leads to a product $\prod_{i=0}^{n+1} \xi = \xi^n$. Obviously, this will only be stable if $|\xi| \leq 1$.

Let us now study the linear advection equation,

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0 \ . \tag{II.9.17}$$

This is the simplest example of a hyperbolic equation and its exact solution is simply given by

$$u(x,t) = u(t=0, x - vt) \ , \tag{II.9.18}$$

hence, the advection equation does (as one might expect) just translate the given data along the x-axis with constant advection velocity v.

II.9.3.1 First-order in Time, Centered in Space (FTCS) Discretization

There are many ways in which we could discretize Eq. (II.9.18). We could try a first-order in time, centered (second-order) in space (FTCS) method, for example:

$$u_j^{(n+1)} = u_j^{(n)} - \frac{v\Delta t}{2\Delta x} \left(u_{j+1}^{(n)} - u_{j-1}^{(n)} \right) \ . \tag{II.9.19}$$

We may now introduce $u(x, t^n) = e^{ikx}$ into the above equation,

$$\begin{aligned} u_j^{(n+1)} &= e^{ik\Delta x j} - \frac{v\Delta t}{2\Delta x} \left(e^{ik\Delta x (j+1)} - e^{ik\Delta x (j-1)} \right) \ , \\ &= \left(1 - \frac{v\Delta t}{2\Delta x} \left(e^{ik\Delta x} - e^{-ik\Delta x} \right) \right) e^{ik\Delta x j} \ , \\ &= \underbrace{\left(1 - \frac{v\Delta t}{\Delta x} i \sin(k\Delta x) \right)}_{=\xi} e^{ik\Delta x j} \ , \end{aligned} \tag{II.9.20}$$

and

$$|\xi| = \sqrt{\xi \xi^*} = \sqrt{1 + \left(\frac{v\Delta t}{\Delta x} \sin(k\Delta x) \right)^2} > 1 \ . \tag{II.9.21}$$

Hence, the FTCS method is *unconditionally unstable* for the advection equation!

II.9.3.2 Upwind Method

The spatially second-order FTCS method is unconditionally unstable. We could go a step back and see if a first-order in space, first-order in time method could be stable. If we discretize by

expansion of the spatial derivative to first order, we get two possiblities:

$$\begin{aligned} \frac{\partial u}{\partial x} &\approx \frac{1}{\Delta x}(u_j - u_{j-1}) \qquad \text{upwind finite difference,} \\ \frac{\partial u}{\partial x} &\approx \frac{1}{\Delta x}(u_{j+1} - u_j) \qquad \text{downwind finite difference.} \end{aligned} \tag{II.9.22}$$

These are both so-called one-sided approximations, because they use data only from one side or the other of point x_j. Coupling one of these equations with forward Euler time integration yields

$$\begin{aligned} u_j^{(n+1)} &= u_j^{(n)} - \frac{v\Delta t}{\Delta x}\left(u_j^{(n)} - u_{j-1}^{(n)}\right) \qquad \text{upwind,} \\ u_j^{(n+1)} &= u_j^{(n)} - \frac{v\Delta t}{\Delta x}\left(u_{j+1}^{(n)} - u_j^{(n)}\right) \qquad \text{downwind .} \end{aligned} \tag{II.9.23}$$

While these one-sided approximations are (in general terms) less accurate than a centered difference, they exploit the asymmetry of the advection equation: it describes translation with speed v. This translation is either to the right (when $v > 0$) or to the left (when $v < 0$). So when $v > 0$, the upwind difference will rely only on information that is behind and at the location of point x_j. It thus "follows" the flow direction and thus is named *upwind* (or sometimes *upstream*). If $v < 0$, the *downwind* (or sometimes *downstream*) method follows the flow, as the flow is now going in the opposite direction.

Stability analysis shows that the upwind method is stable for

$$0 \leq \frac{v\Delta t}{\Delta x} \leq 1 \ , \tag{II.9.24}$$

while the downwind method is stable for

$$-1 \leq \frac{v\Delta t}{\Delta x} \leq 0 \ , \tag{II.9.25}$$

which just confirms that for $v > 0$ one should use the upwind method and for $v < 0$ the downwind method.

In the above, the demand

$$\alpha = \left|\frac{v\Delta t}{\Delta x}\right| \leq 1 \tag{II.9.26}$$

plays a major role. It is a mathematical realization of the physical causality principle – the propagation of information (here via advection) in one times step Δt must not jump ahead more than one grid interval of size Δx.

The demand that $\alpha \leq 1$ is usually referred to as the *Courant-Friedrics-Lewy (CFL) condition* . In practise, the CFL condition is used to determine the allowable time step for a spatial grid size Δx for a certain accuracy and stability,

$$\Delta t = c_{\rm CFL}\frac{\Delta x}{|v|} \ , \tag{II.9.27}$$

where $c_{\rm CFL} \leq 1$ is the CFL factor.

II.9.3.3 Lax-Friedrich Method

The Lax-Friedrichs method, like FTCS, is first order in time, but second order in space. It is given by

$$u_j^{(n+1)} = \frac{1}{2}\left(u_{j+1}^{(n)} + u_{j-1}^{(n)}\right) - \frac{v\Delta t}{2\Delta x}\left(u_{j+1}^{(n)} - u_{j-1}^{(n)}\right) \ , \tag{II.9.28}$$

and, compared to FTCS, has been made stable for $\alpha \leq 1$ (Eq. II.9.26) by using the average of the old result at points $j+1$ and $j-1$ to compute the update at point j. This is equivalent to adding a dissipative term to the equations (which damps the instability of the FTCS method) and leads to rather poor accuracy.

II.9.3.4 Leapfrog Method

A fully second-order method is the *Leapfrog* Method (also called the midpoint method) given by

$$u_j^{(n+1)} = u^{(n-1)} - \frac{v\Delta t}{\Delta x}\left(u_{j+1}^{(n)} - u_{j-1}^{(n)}\right) \ . \tag{II.9.29}$$

One can show that it is stable for $\alpha < 1$ (Eq. II.9.26). A special feature of this method is that it is *non-dissipative*. This means that any initial condition simply translates unchanged. All modes (in a decomposition picture) travel unchanged, but they do not generally travel at the correct speed, which can lead to high-frequency oscillations that cannot damp (since there is no numerical viscosity).

II.9.3.5 Lax-Wendroff Method

The *Lax-Wendroff* method is an extension of the Lax-Friedrich method to second order in space and time. It is given by

$$u_j^{(n+1)} = u_j^n - \frac{v\Delta t}{2\Delta x}\left(u_{j+1}^{(n)} - u_{j-1}^{(n)}\right) + \frac{v^2(\Delta t)^2}{2(\Delta x)^2}\left(u_{j-1}^{(n)} - 2U_j^{(n)} + u_{j+1}^{(n)}\right) \ . \tag{II.9.30}$$

It has considerably better accuracy than the Lax-Friedrich method and is stable for $\alpha \leq 1$ (Eq. II.9.26).

II.9.3.6 Methods of Lines (MoL) Discretization

So far we have discussed methods that have discretized the advection equation (and may discretize other PDEs) in both space and time.

The *Method of Lines* is a semi-discrete approach. The PDE is discretized in space using a discretization $\mathcal{D}(u)$. Since the spatial part is now discretized, the remaining equation,

$$\frac{du}{dt} = \mathcal{D}(u) \ , \tag{II.9.31}$$

is now an ODE that can be integrated forward in time with a well-known stable ODE integrator such as Runge-Kutta. This is the method of choice if high-order accuracy in time is required. It also simplifies the implementation of complex equations significantly, since one must worry only about the spatial part of the discretization.

II.9.4 A Linear Elliptic Equation Example: The 1D Poisson Equation

One of the simplest elliptic equations is the linear Poisson equation for the Newtonian gravitational potential Φ,

$$\nabla^2\Phi = \Delta\Phi = 4\pi G\rho \ . \tag{II.9.32}$$

For simplicity, let us work with a spherically symmetric mass distribution and a 1D spherical equidistant grid. The Laplacian Δ reduces in spherical symmetry to

$$\begin{aligned}\Delta[\cdot] &= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}[\cdot]\right) \ , \\ &= \frac{2}{r}\frac{\partial}{\partial r}[\cdot] + \frac{\partial^2}{\partial r^2}[\cdot] \ .\end{aligned} \tag{II.9.33}$$

There are two ways of solving the 1D poisson equation. One relies on the fact that in spherical symmetry Φ will depend only on the single variable r, hence the character of Eq. (II.9.32) changes from PDE to ODE and the equation can be straighforwardly integrated. The other method is more general, does not exploit the ODE character in 1D, and can be applied also to multi-D and non-linear problems.

II.9.4.1 Direct ODE Method

We will work with the second-order ODE

$$\frac{d^2}{dr^2}\Phi + \frac{2}{r}\frac{d}{dr}\Phi = 4\pi G\rho \ . \tag{II.9.34}$$

We reduce this second-order equation to two first-order equations by setting

$$\begin{aligned}\frac{d}{dr}\Phi &= z \ , \\ \frac{d}{dr}z + \frac{2}{r}z &= 4\pi G\rho \ .\end{aligned} \tag{II.9.35}$$

This can be straightforwardly integrated using the methods discussed in §II.5 using inner and outer boundary conditions,

$$\begin{aligned}\frac{d}{dr}\Phi(r=0) &= 0 \ , \\ \Phi(r=R_{\rm surface}) &= -\frac{GM(r=R_{\rm surface})}{R_{\rm surface}} \ .\end{aligned} \tag{II.9.36}$$

In practice, one sets first $\Phi(r=0)=0$, integrates out, and then corrects by adding a correction factor to obtain the correct outer boundary condition set by the well-known analytic result for the gravitational potential outside a spherical mass distribution. Not that this is, in some sense, equivalent to the analytic calculation of the gravitational potential – it is determined only up to an additive constant whose choice is convention, since only derivatives of the potential have physical meaning.

II.9.4.2 Matrix Method

In the matrix method, we directly discretize

$$\frac{d^2}{dr^2}\Phi + \frac{2}{r}\frac{d}{dr}\Phi = 4\pi G\rho \ . \tag{II.9.37}$$

With centered differences, we obtain for an interior (non-boundary) grid point i

$$\begin{aligned} \left.\frac{\partial}{\partial r}\Phi\right|_i &\approx \frac{1}{2\Delta r}\left(\Phi_{i+1} - \Phi_{i-1}\right) \ , \\ \left.\frac{\partial^2}{\partial r^2}\phi\right|_i &\approx \frac{1}{(\Delta r)^2}\left(\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}\right) \ . \end{aligned} \tag{II.9.38}$$

With this, we get

$$\begin{aligned} \frac{1}{r_i}\frac{1}{\Delta r}\left(\Phi_{i+1} - \Phi_{i-1}\right) + \frac{1}{(\Delta r)^2}\left(\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}\right) &= 4\pi G\rho_i \ , \\ f_i &= 4\pi G\rho_i \ . \end{aligned} \tag{II.9.39}$$

From the $1/r$ term we note that we either must regularize at $r = 0$ by some kind of expansion or simply stagger the grid so that there is no point exactly at $r = 0$. The latter is easily achieved by moving the entire grid over by $0.5dr$. At the inner boundary, we have $\partial/\partial r\Phi = 0$, so

$$\Phi_{-1} = \Phi_0 \tag{II.9.40}$$

and the finite difference terms in the innermost zone are adjusted to

$$\begin{aligned} \left.\frac{\partial}{\partial r}\Phi\right|_{i=0} &= \frac{1}{\Delta r}(\Phi_1 - \Phi_0) \ , \\ \left.\frac{\partial^2}{\partial r^2}\Phi\right|_{i=0} &= \frac{1}{(\Delta r)^2}(\Phi_1 - \Phi_0) \ . \end{aligned} \tag{II.9.41}$$

Let us press ahead by computing the Jacobian

$$J_{ij} = \frac{\partial f_i}{\partial \Phi_j} \ , \tag{II.9.42}$$

and solving

$$J\mathbf{\Phi} = \mathbf{f} \ , \tag{II.9.43}$$

for $\Phi = (\Phi_0, \cdots, \Phi_{n-1})^T$ (for a grid with n points labeled 0 to $n-1$) and with $\mathbf{b} = 4\pi G(\rho_0, \cdots, \rho_{n-1})^T$. After finding the solution to (II.9.43), one corrects for the analytic outer boundary value $\Phi(r = R_{\rm surface}) = -GM/R_{\rm surface}$.

The Jacobian J has tri-diagonal form and can be explicitely given as follows:

(a) $i = j = 0$:

$$J_{00} = -\frac{1}{(\Delta r)^2} - \frac{1}{r_0 \Delta r} \ , \tag{II.9.44}$$

(b) $i = j$:

$$J_{ij} = \frac{-2}{(\Delta r)^2} \ , \tag{II.9.45}$$

(c) $i + 1 = j$:

$$J_{ij} = \frac{1}{(\Delta r)^2} + \frac{1}{r_i \Delta r} \ , \tag{II.9.46}$$

(d) $i - 1 = j$:

$$J_{ij} = \frac{1}{(\Delta r)^2} - \frac{1}{r_i \Delta r} \ . \tag{II.9.47}$$

Chapter III

Applications in Astrophysics

III.1 Nuclear Reaction Networks

Nuclear reaction networks (NRNs) are a direct application of ODE theory and methods to solve (non-)linear equations. In this section, we will introduce the basics of nuclear reaction networks. A much more detailed discussion can be found, for example, in Arnett's book [1].

Thermonuclear reactions are very sensitive to temperature, since the reactants must have sufficient kinetic energy to have significant probability for quantum tunneling through the Coloumb barrier blocking the reaction. For example, the triple-α reaction

$$3\,^4\mathrm{He} \longrightarrow {}^{12}C + \gamma \ , \tag{III.1.1}$$

which is the reaction in which helium burns into carbon, has an energy generation rate (which is proportional to the reaction rate) that scales with

$$\epsilon_{3\alpha} \propto \left(\frac{T_8}{2}\right)^{18.5} \tag{III.1.2}$$

near $T_8 = 2$, where T_8 is the temperature measured in 10^8 K ($T_8 = T/10^8$ K).

Since generally many different reactions are going on at the same time at generally very different rates, the equations governing the complete set of reactions are stiff systems of ODEs.

III.1.1 Some Preliminaries and Definitions

- Characterizing Isotopes
  An isotope is characterized by dimensionless integers:

  Z # of protons (the atomic number)

  N # of neutrons

  A $Z + N$ = # of nucleons.

- Avogadro's Number
  $N_A = 6.02214179 \times 10^{23}\,\mathrm{mole}^{-1}$ is the number of entities ("units", nuclei etc.) in one mole of material.

- Atomic Weight
  The *atomic weight* (also called *molar mass*) of one entity with mass m in grams is $W = mN_A\,\mathrm{g\,mole}^{-1}$.

- Atomic Mass Unit
  The *atomic mass unit* is given by

$$\begin{aligned} 1\,m_u = 1\,\mathrm{amu} = \frac{1}{N_A} &= 1.660538782 \times 10^{-24}\,\mathrm{g} \\ &= 931.494061\,\mathrm{MeV/c^2} \ . \end{aligned} \tag{III.1.3}$$

  ($1\,\mathrm{MeV} = 1.6 \times 10^{-10}\,\mathrm{erg}$). For ${}^{12}C$, $W = 12\,\mathrm{g\,mole}^{-1}$. So 1 amu has then $W = 1\,\mathrm{g\,mole}^{-1}$.

- Isotope Rest Mass
  The isotope rest mass of a single isotope k is

$$\begin{aligned} m_k &= Nm_n + Zm_p + Z(1-f)m_e - \Delta m \ , \\ &= Nm_n + Zm_p + Z(1-f)m_e - \frac{|B_k|}{c^2} \ \mathrm{g} \ . \end{aligned} \tag{III.1.4}$$

  Here f is the ionization fraction and we have neglected the contribution from the electron binding energy, which is usually negligible. We have also neglected the small negative electron binding energy. Δm is the *mass deficit*, which describes the amount by which the rest mass of the isotopes is reduced below the sum of the its constituent nucleon rest masses by the nuclear binding energy B_k. The molar mass of the isotope is $W_k = m_k N_A$.

  The binding energy has a negative sign, i.e. it reduces the rest mass of the isotope in the same way gravitational binding energy reduces the observed mass of neutron stars below their baryonic mass.

- Baryon Mass Density
  For a mixture of isotopes we define the baryon mass density

$$\rho = \frac{\sum_i n_i A_i}{N_A} = \sum_i m_i n_i A_i \tag{III.1.5}$$

  where n_i is the number density of isotope i, the number of entities of this isotope per unit volume.

- Mass Fraction and Molar Fraction

  The *mass fraction* of isotope i is its relative abundance per mass defined as

$$X_i = \frac{A_i n_i}{\rho N_A} \ . \tag{III.1.6}$$

  And all mass fractions must add up to one: $\sum_i X_i = 1$.

  The *molar fraction* of an isotope is its relative abundance in one mole of material (so it is a number fraction).

$$Y_i = \frac{X_i}{A_i} = \frac{n_i}{\rho N_A} \ . \tag{III.1.7}$$

  The use of the molar fraction is somewhat historic and comes from the origins of nuclear reaction networks in chemical reaction networks. It makes sense to use it, since nuclear reactions are reactions between entities and depend directly on the number of entities present. Using the molar fraction, while less intuitive than mass fraction or number density, has also the advantage that the energy generation by a nuclear reaction is very easily evaluated:

$$\frac{d\epsilon}{dt} = N_A \sum_i |B_i| \frac{dY_i}{dt} \ , \tag{III.1.8}$$

  where ϵ is the specific internal energy per gram in erg g^{-1}, $|B_i|$ is the binding energy in units of erg g^{-1}, and the Y_i are formally dimensionless.

- Average Nucleus and the Electron Fraction Y_e
  We define the average nucleus described by $\overline{A}$ and $\overline{Z}$ and the electron fraction (number of electrons per baryon) by

$$\begin{aligned} \overline{A} &= \frac{\sum_i n_i A_i}{\sum n_i} = \frac{1}{\sum_i Y_i} \ , \\ \overline{Z} &= \frac{\sum_i n_i Z_i}{\sum n_i} = \overline{A} \sum_i Y_i Z_i \ , \\ Y_e &= \frac{\overline{A}}{\overline{Z}} \ . \end{aligned} \tag{III.1.9}$$

- Mass Excess and Energy Generation
  The *mass excess* of an isotope is defined as

$$\Delta M = \left(\frac{m}{m_u} - A \right) m_u c^2 \ , \tag{III.1.10}$$

  which is just the difference of its actual mass to its mass in atomic mass units. This difference is due to differences in binding energy between the isotope in question and ^{12}C.

  For a reaction $1 + 2 \longrightarrow 3$, the liberated energy Q can be expressed in terms of the mass excess as

$$Q = \Delta M_1 + \Delta M_2 - \Delta M_3 \ , \tag{III.1.11}$$

  which is equivalent to expressing it in terms of the binding energies of the reactants:

$$Q = |B_3| - (|B_1| + |B_2|) \ . \tag{III.1.12}$$

III.1.2 A 3-Isotope Example

Let's consider a nuclear reaction network with three reactants:

$$\begin{aligned} 1 + 2 &\longleftrightarrow 3 + \gamma \\ 2 &\longrightarrow 1 \end{aligned} \tag{III.1.13}$$

Isotopes 1 and 2 combine to make isotopes 3 and the released energy goes into a photon. Isotope 3 can be photodissociated, making isotopes 1 and 2. Also, isotope 2 is unstable and could decay to isotope 1 under the emission of an electron or positron (which has been emitted from this schematic equation). We also leave out changes in the thermodynamics due to the excess heat generated by the reaction and keep density and temperature fixed.

We can write out the change in the number fraction for isotope 1:

$$\begin{aligned} \frac{dn_1}{dt} = & -n_1 n_2 \langle \sigma v \rangle_{12} && \text{reaction } 1 + 2 \rightarrow 3 \\ & +\lambda_{\gamma 3} n_3 && \text{reaction } 3 + \gamma \rightarrow 1 + 2 \\ & +\lambda_2 n_2 && \text{decay } 2 \rightarrow 1 \ . \end{aligned} \tag{III.1.14}$$

The first reaction is a so-called binary reaction and its rate is proportional to the product of the number densities of the reactants. $\langle \sigma v \rangle_{12}$ is the velocity-integrated cross section (units cm^3 s^{-1}) for the reaction $1 + 2 \rightarrow 3$. This is where nuclear physics and the local thermodynamics, controlling

the velocity distribution of the reactants, come in. In the following, we will use simplified notation and define

$$\lambda_{ij} = \langle \rho v \rangle_{ij} \ . \tag{III.1.15}$$

The second reaction involves $\lambda_{\gamma 3}$, which is the rate for photodissociation of reactant 3. This depends on the temperature and on reactant 3's binding energy. λ_2 in the decay reaction is the decay rate.

The rate equations in terms of the number density for the other isotopes are

$$\begin{aligned} \frac{n_2}{dt} &= -n_1 n_2 \lambda_{12} + \lambda_{\gamma 3} n_3 - \lambda_2 n_2 \ , \\ \frac{n_3}{dt} &= +n_1 n_2 \lambda_{12} - \lambda_{\gamma 3} n_3 \ . \end{aligned} \tag{III.1.16}$$

Next we convert to molar fractions using Eq. (III.1.7):

$$\frac{dn_i}{dt} = \rho N_A \frac{dY_i}{dt} + N_A Y_i \frac{d\rho}{dt} \tag{III.1.17}$$

and we set $d\rho/dt = 0$ for the purpose of this example.

We can now write out the reaction network for the change in the molar fractions Y_i ($i = 1, 2, 3$):

$$\begin{aligned} f_1 = \frac{d}{dt} Y_1 &= -N_A \rho \lambda_{12} Y_1 Y_2 + \lambda_{\gamma 3} Y_3 + \lambda_2 Y_2 \ , \\ f_2 = \frac{d}{dt} Y_2 &= -N_A \rho \lambda_{12} Y_1 Y_2 + \lambda_{\gamma 3} Y_3 - \lambda_2 Y_2 \ , \\ f_3 = \frac{d}{dt} Y_3 &= +N_A \rho \lambda_{12} Y_1 Y_2 - \lambda_{\gamma 3} Y_3 \ . \end{aligned} \tag{III.1.18}$$

This system of ODEs is non-linear in the Y_i and if the reaction rates λ_{12}, $\lambda_{\gamma 3}$, and λ_2 may be very different, it will be stiff and difficult to solve with standard explicit methods that tend to lead to instability.

Our goal is thus to write an implicit scheme. For simplicity, we will keep it first order in time. So, for evaluating $Y_i(t + \Delta t)$ with the first-order backward Euler scheme, we write

$$Y_i(t + \Delta t) = Y_i(t) + \Delta t f_i(t + \Delta t) \ . \tag{III.1.19}$$

Furthermore, we denote

$$\Delta_i = \frac{dY_i}{dt} \Delta t \approx Y_i(t + \Delta t) - Y_i(t) \ . \tag{III.1.20}$$

Since the f_i are nonlinear in $Y_i Y_j$, we need to *linearize* the system to find an approximation for $f_i(t + \Delta t)$ that we can use in our integration scheme. We expand

$$f_i(t + \Delta t) \approx f_i(t) + \frac{df_i(t)}{dt} \Delta t = f_i(t) + \sum_j \frac{df_i}{dY_j} \frac{dY_j}{dt} \Delta t = f_i(t) + \sum_j \frac{df_i}{dY_j} \Delta_j \ . \tag{III.1.21}$$

Using (III.1.21) in (III.1.19) with (III.1.20), we have

$$\Delta_i - \sum_j \frac{df_i}{dY_j}\Delta_j \Delta t = f_i(t)\Delta t \ . \tag{III.1.22}$$

We can write out the left-hand-side using Eq. (III.1.18),

$$\begin{aligned}
(1 + N_A\rho\lambda_{12}Y_2(t)\Delta t)\Delta_1 + (N_A\rho\lambda_{12}Y_1(t) - \lambda_2)\Delta t\Delta_2 - \lambda_{\gamma 3}\Delta t\Delta_3 &= f_1(t)\Delta t \\
N_A\rho\lambda_{12}Y_2(t)\Delta t\Delta_1 + (1.0 + N_A\rho\lambda_{12}Y_1(t)\Delta t + \lambda_2\Delta t)\Delta_2 - \lambda_{\gamma 3}\Delta t\Delta_3 &= f_2(t)\Delta t \\
-N_A\rho\lambda_{12}Y_2(t)\Delta t\Delta_1 + -N_A\rho\lambda_{12}Y_1(t)\Delta t\Delta_2 + (1 + \lambda_{\gamma 3}\Delta_t)\Delta_3 &= f_3(t)\Delta t \ .
\end{aligned} \tag{III.1.23}$$

And by writing this in matrix form, we obtain

$$\begin{pmatrix} 1 + N_A\rho\lambda_{12}Y_2(t)\Delta t & N_A\rho\lambda_{12}Y_1(t)\Delta t - \lambda_2\Delta t & -\lambda_{\gamma 3}\Delta t \\ N_A\rho\lambda_{12}Y_1(t)\Delta t & 1 + N_A\rho\lambda_{12}Y_1(t)\Delta t + \lambda_2\Delta t & -\lambda_{\gamma 3}\Delta t \\ -N_A\rho\lambda_{12}Y_2(t)\Delta t & -N_A\rho\lambda_{12}Y_1(t)\Delta t & 1 + \lambda_{\gamma 3}\Delta_t \end{pmatrix} \begin{pmatrix} \Delta_1 \\ \Delta_2 \\ \Delta_3 \end{pmatrix} = \begin{pmatrix} f_1(t)\Delta t \\ f_2(t)\Delta t \\ f_3(t)\Delta t \end{pmatrix} \ , \tag{III.1.24}$$

which can be solved for the Δ_i needed for our update,

$$Y_i(t + \Delta t) = Y_i(t) + \Delta_i \ . \tag{III.1.25}$$

III.1.3 Reactant Multiplicity

A more complex situation arises when multiple entities of the same reactant are involved in a reaction. For example,

$$^{12}\mathrm{C} + {}^{12}\mathrm{C} \longrightarrow {}^{24}\mathrm{Mg} + \gamma \ . \tag{III.1.26}$$

So for each ^{24}Mg nucleus we make, two ^{12}C nuclei are needed. For simplicity, we will work with number densities in the following. We can always convert to molar fractions once the reaction equations have been derived. Naively, we would now assume that the change in the number density was given by

$$\frac{dn_{^{12}\mathrm{C}}}{dt} = -2n_{^{12}\mathrm{C}}^2\lambda_{^{12}\mathrm{C}^{12}\mathrm{C}} \ , \tag{III.1.27}$$

which, however is not quite right, since reactions between each pair of nuclei are counted $2! = 2$ times ($N!$ is the number of possible permutations that would lead to an identical result when N nuclei are involved). So we really have only

$$\frac{dn_{^{12}\mathrm{C}}}{dt} = -\frac{2}{2!}n_{^{12}\mathrm{C}}^2\lambda_{^{12}\mathrm{C}^{12}\mathrm{C}} \ . \tag{III.1.28}$$

So for an a bit more general binary reaction $N_i\, i + N_j\, j \rightarrow 1\, k$, we have

$$\begin{aligned}
\frac{dn_i}{dt} &= -\frac{|N_i|}{|N_i|!\,|N_j|!}n_i n_j \lambda_{ij} \ , \\
\frac{dn_j}{dt} &= -\frac{|N_j|}{|N_i|!\,|N_j|!}n_i n_j \lambda_{ij} \ , \\
\frac{dn_k}{dt} &= +\frac{1}{|N_i|!\,|N_j|!}n_i n_j \lambda_{ij} \ .
\end{aligned} \tag{III.1.29}$$

This is straightforwardly extended to triple reactions. For example, for the triple-α reaction given in Eq. (III.1.1) we have:

$$\begin{aligned} \frac{dn_{^4\mathrm{He}}}{dt} &= -\frac{3}{3!} n_{^4\mathrm{He}}^3 \lambda_{3\alpha} \ , \\ \frac{dn_{^{12}\mathrm{C}}}{dt} &= +\frac{1}{3!} n_{^4\mathrm{He}}^3 \lambda_{3\alpha} \ . \end{aligned} \tag{III.1.30}$$

III.1.4 Open Source Resources

Fortunately, a large number of nuclear reaction networks and nuclear data and reaction rate sets are available as open source / open physics.

- Frank Timmes (a nuclear astrophysicist at Arizona State University) provides an assortment of state-of-the-art reaction networks at http://cococubed.asu.edu/code_pages/burn.shtml.
- The Modules for Explorations in Stellar Astrophysics (MESA) code comes with extensive nuclear reaction networks. http://mesa.sourceforge.net.
- The Clemson University nuclear astrophysics group provides reaction networks and online tools, e.g., a calculator for abundances in nuclear statistical equilibrium, at http://www.webnucleo.org.
- The Joint Institute for Nuclear Astrophysics (JINA), a National Science Foundation Physics Frontier Center, provides lots of resources on nuclear reactions as well as a data base of isotope masses and reaction rates. http://www.jinaweb.org.

References

[1] D. Arnett. *Supernovae and nucleosynthesis. an investigation of the history of matter, from the Big Bang to the present*. Princeton Series in Astrophysics, Princeton, NJ: Princeton University Press, 1996.

III.2 N-body Methods

We will begin by exploring the gravitational N-body problems and algorithms that have been developed to solve it efficiently.

III.2.1 Specification of the Problem

Frequently in astrophysics we have systems of "particles" which interact only gravitationally. Examples include dark matter and purely stellar systems (in which each star is a particle). Ignoring relativistic effects (which we can in a wide variety of situations) the evolution of such systems is entirely specified by the Newtonian $1/r^2$ law of gravity, such that the acceleration of particle i is given by:

$$\ddot{\mathbf{x}}_i = \sum_{j=1; j\neq i}^{N} -\frac{G m_j}{|\mathbf{x}_{ij}|^2}\hat{\mathbf{x}_{ij}}, \tag{III.2.1}$$

where $\mathbf{x}$ is position, $\mathbf{x}_{ij} \equiv \mathbf{x}_i - \mathbf{x}_j$ and a hat indicates a unit vector.

III.2.1.1 How Big Must N Be?

We typically want N, the number of particles, to be as large as possible given available computational resources. When simulating a system of fixed total mass, increasing N will allow each particle to have a smaller mass and, therefore, will allow the simulation to resolve structure on smaller scales (and more closely approach the idealized fluid that we're attempting to simulate). The number of particles used will also affect how long we can run our simulation for, before the discrete nature of our particle representation becomes a problem. Imagine an isolated, self-gravitating system of particles that is in virial equilibrium (such as a galaxy). In the limit of an infinite number of particles, the potential of this system will be unchanging in time and so each particle will conserve its total energy as it orbits through the system. However, if there are a finite number of particles, the potential will no longer remain constant in time, instead fluctuating as particles move around. This will allow for energy exchange between particles. If our simulation contains fewer particles than the real system (which is almost always will do) this energy exchange is unphysical. In a non-gravitating system this would eventually lead to equipartition and thermal equilibrium. Gravitating systems, because of their negative specific heat, have no thermal equilibrium and this process will eventually lead to the phenomenon of "core collapse" (observed in globular clusters) in which the core of the system collapses to arbitrariily high densities while a diffuse envelope of material is ejected to large radii.

To estimate how long we can run a simulation before this relaxation process becomes important, we want an order-of-magnitude estimate how long it takes for encounters with other stars to significantly change the energy of a star. Typically we are interested in collisionless systems in which the acceleration of a given particle never receives a dominant contribution from a single other particle, instead being determined by the combined effects of many particles. Consider an encounter between two stars. Assuming the collision is a small perturbation to the motion of the star we can approximate the force experienced by the star as:

$$\dot{\mathbf{v}}_\perp = \frac{Gm}{b^2+x^2}\cos\theta = \frac{Gmb}{(b^2+x^2)^{3/2}} \approx \frac{Gm}{b^2}\left[1+\left(\frac{vt}{b}\right)^2\right]^{-3/2}. \tag{III.2.2}$$

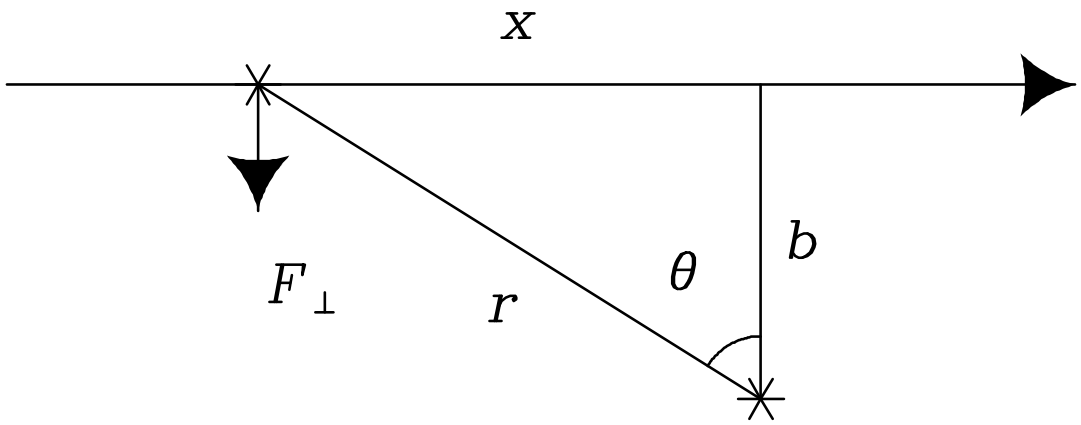

Figure III.2.1: Geometry of a gravitational collision between two stars.

Integrating over the entire collision gives us the change in velocity of the star:

$$\mathbf{v}_\perp \approx \frac{Gm}{bv}\int_{-\infty}^{\infty}(1+s^2)^{-3/2}\mathrm{d}s = \frac{2Gm}{bv}, \tag{III.2.3}$$

whcih is approximately the force at closest approach times the duration of the encounter, b/v. The surface density of stars in a galaxy is of order $N/\pi R^2$, so in crossing the galaxy once, the star experiences

$$\delta n = \frac{N}{\pi R^2}2\pi b\mathrm{d}b = \frac{2N}{R^2}b\mathrm{d}b, \tag{III.2.4}$$

encounters with impact parameter between b and $b+\mathrm{d}b$. The encounters cause randomly oriented changes in velocity, so $\overline{\delta\mathbf{v}_\perp}=0$, but there can be a net change in $v_\perp^2$:

$$\delta v_\perp^2 \approx \left(\frac{2Gm}{bv}\right)^2\frac{2N}{R^2}d\mathrm{d}b. \tag{III.2.5}$$

Our perturbation approach breaks down if $\delta v_\perp \sim v_\perp$ which occurs is $b \lesssim b_{\min} = Gm/v^2$, so integrating over all impact parameters[1] from $b_{\min}$ to R (the largest possible impact parameter):

$$\Delta v_\perp^2 = \int_{b_{\min}}^{R}\delta v_\perp^2 \approx 8N\left(\frac{Gm}{Rv}\right)^2\ln\Lambda, \tag{III.2.6}$$

where $\Lambda = R/b_{\min}$ ("Coulomb logarithm"). The typical speed of a star in a self-gravitating galaxy is

$$v^2 \approx \frac{GNm}{R}. \tag{III.2.7}$$

[1] For a more careful treatment of small b encounters, take a look at the treatment of dynamical friction, §7.1 of Binney & Tremaine.

Therefore, we find

$$\frac{\Delta v_\perp^2}{v^2} = \frac{8\ln\Lambda}{N}, \tag{III.2.8}$$

and the number of crossings required for order unity change in velocity is:

$$n_{\rm relax} = \frac{N}{8\ln\Lambda}. \tag{III.2.9}$$

Since $\Lambda = R/b_{\rm min} \approx Rv^2/Gm \approx N$ we find $t_{\rm relax} \approx [0.1N/\ln N]t_{\rm cross}$.

Relaxation is important for systems up to globular cluster scales, but is entirely negligible for galaxies.

System	N	$t_{\rm relax}/t_{\rm cross}$	$t_{\rm relax}$
Small stellar group	50	1.3	
Globular cluster	10^5	870	10^8yr
Galaxy	10^{11}	4×10^8	4×10^7Gyr

It's worth taking a moment to consider what we're actually doing in an N-body calculation. Specifically, representing a fluid in 6-D phase space by a set of discrete points. We associate a mass with each of those points (usually the same mass for each point) such that we can use the points to compute the density distribution of the fluid and therefore its gravitational potential. Additionally, we move the points in phase space according to that same gravitational potential. As such, there are fundamentally two roles for the points: to represent the mass distribution at any time and to flow as would the actual fluid such that they continue to represent the mass distribution at later times. For most systems of interest (e.g. galaxies, dark matter halos) the number of particles we can use (given current computational techniques and resources) is much less than the number of particles in the real system. For example, a galaxy may contain 10^{11} stars, but the best current simulations of galaxies contain only $\sim 10^6$. Similarly, the best current simulations of dark matter halos contain around 10^9 particles, but the Milky Way's halo contains maybe 10^{70} dark matter particles. Therefore, the relaxation times in simulations will be much shorter than in the real systems (particularly in dense regions) unless we do something to mitigate this problem.

We can break down the N-body problem into three sub-problems:

(1) Specifying initial conditions;

(2) Computing forces/accelerations on particles;

(3) Stepping particles forward in time.

III.2.2 Force Calculation

The N-body problem is fundamentally an $\mathcal{O}(N^2)$ problem—direct summation over all particles involves a number of calculations that increases as the square of the number of particles. Not surprisingly therefore, force computation is typically the slowest step in any N-body calculation and so numerous techniques have been devised to speed this up. We'll review these techniques, as well as the direct summation approach (since it's still useful in some cases).

III.2.2.1 Direct Summation (Particle-Particle)

The direct summation approach is inherently simple—as we wrote already, given some set of N particles we evaluate the force on each one using

$$\ddot{\mathbf{x}}_i = \sum_{j=1;j\neq i}^{N} -\frac{Gm_j}{|\mathbf{x}_{ij}|^2}\hat{\mathbf{x}}_{ij}. \tag{III.2.10}$$

You can write yourself a direct summation code in a few lines. If you have a newish desktop computer with a GPU (Graphics Processing Unit) you can write a direct summation code which can handle $\gtrsim 10^4$ particles in a reasonable amount of time. Specialized hardware (GRAvity PipE or GRAPE boards) can handle several million—enough to simulate a globular cluster at one particle per star. The direct summation approach is obsolete for galactic and cosmological simulations.

In cases where we don't have one particle per star/particle two-body encounters between particles will be much more common in our simulation than in the real system. This can lead to effects (for example, binary formation) which would not occur in the real system on relatively short timescales. To circumvent this problem we recognize that each particle is better thought of as representing an extended distribution of mass. It will therefore not have a point mass potential. Instead, its potential will be "softened". We might therefore modify the force law to be

$$\ddot{\mathbf{x}}_i = \sum_{j=1;j\neq i}^{N} -\frac{Gm_j}{(|\mathbf{x}_{ij}|+\epsilon)^2}\hat{\mathbf{x}}_{ij}, \tag{III.2.11}$$

where ϵ is a length scale (called the *softening length*) of order the "size" of the particle and which limits the maximum force between two particles. Note that this doesn't do much to change the relaxation time—the Coulomb logarithm term in our derivation of the relaxation time shows that the relaxation process gains equal contributions from particles in each logarithmic interval of radius, and so softening the force on small scales only slightly reduces the rate of relaxation.

Different functional forms have been used for the softening. For example, a common approach has been to represent particles by so-called Plummer spheres—density distributions of the form $\rho(x) \propto (1+x^2)^{-5/2}$ (where $x = r/\epsilon$) which have a potential $\Phi(x) \propto 1/\sqrt{\epsilon^2+x^2}$, such that

$$\ddot{\mathbf{x}}_i = \sum_{j=1;j\neq i}^{N} -\frac{Gm_j|\mathbf{x}_{ij}|}{(|\mathbf{x}_{ij}|^2+\epsilon^2)^{3/2}}\hat{\mathbf{x}}_{ij}. \tag{III.2.12}$$

Another common approach is to use a cubic spline density distribution of the form:

$$\rho(x) \propto \begin{cases} 4-6x^2+3x^3 & \text{for } x<1 \\ (2-x)^3 & \text{for } 1\leq x<2 \\ 0 & \text{for } x\geq 2. \end{cases} \tag{III.2.13}$$

This form has the advantage that the density distribution is truncated (i.e. goes to zero beyond $x=2$) and so the force law becomes precisely Newtonian at $2>2\epsilon$. In general, softening kernels of this type (known as compact kernels) are superior to non-compact kernels (e.g. Plummer).

Choosing a value for the softening length is something of an art form. It shouldn't be too large as any structure on scales smaller than the softening length will be smoothed away. However, it shouldn't be too small, or two-body collisional effects will become important again. A good discussion of choosing softening lengths is given by [3], who explores further refinements to this idea, such as compensating the reduced forces on small scales by slightly enhancing forces on larger scales and the possibility of adapative softening lengths (i.e. making ϵ a function of, for example, local density).

III.2.2.2 Particle-Mesh

The fundamental limitation of the direction summation technique is that it is $\mathcal{O}(N^2)$. So, let's explore some ways to reduce the computational load. One of the first ideas was to compute forces by solving Poisson's equation rather than by direct summation. Suppose we have a density field $\rho(\mathbf{x})$. Poisson's equation tells us that the gravitational potential is related to this density field by

$$\nabla^2\Phi(\mathbf{x}) = 4\pi G\rho(\mathbf{x}). \tag{III.2.14}$$

If we can solve this equation, then the acceleration on particle i can be found from the potential using

$$\ddot{\mathbf{x}}_i = -\boldsymbol{\nabla}\Phi(\mathbf{x}_i). \tag{III.2.15}$$

In particular, if we know the density field on a uniform grid then we can use Fourier transforms (in particular, Fast Fourier Transforms) to quickly solve these equations. If we represent the Fourier transform of some quantity q on our grid as

$$q_i = \sum_{\mathbf{k}_j} \widetilde{q}_j \exp(i\mathbf{k}_j \cdot \mathbf{x}_i), \tag{III.2.16}$$

where the sum is taken over all wavenumbers $\mathbf{k}_j$ then from Poisson's equation we have

$$\sum_{\mathbf{k}_j} k_j^2\, \widetilde{\Phi}_j \exp(i\mathbf{k}_j \cdot \mathbf{x}_i) = 4\pi G \sum_{\mathbf{k}_j} \widetilde{\rho}_j \exp(i\mathbf{k}_j \cdot \mathbf{x}_i), \tag{III.2.17}$$

which can only hold in general if

$$-k_j^2\, \widetilde{\Phi}_j = 4\pi G\, \widetilde{\rho}_j\ . \tag{III.2.18}$$

Thus, to compute the potential, we proceed as follows:

(1) Find the density field on a grid;

(2) Compute its Fourier transform;

(3) Multiply each Fourier component by $-4\pi G/k_j^2$;

(4) Take the inverse Fourier transform.

We can actually skip the final step since we're interested in the force on each particle which is given by

$$\ddot{\widetilde{\mathbf{x}}}_j = -i\mathbf{k}_j\, \widetilde{\Phi}\ . \tag{III.2.19}$$

So, we simply multiply the potential by $-i\mathbf{k}_j$, take the inverse Fourier transform and are left with the particle accelerations on a grid. We can interpolate accelerations to the precise location of each particle if necessary.

The density field is, of course, constructed from the particle distribution. There are numerous ways to do this. The most common are nearest grid point (NGP), cloud-in-cell (CIC) and triangular-shaped cloud (TSC) methods which are illustrated in Figure. III.2.2.

Particle-mesh algorithms of this sort are $\mathcal{O}(N + N_g \log N_g)$ where N_g is the number of grid points and can therefore be substantially faster than direct summation techniques. Their main

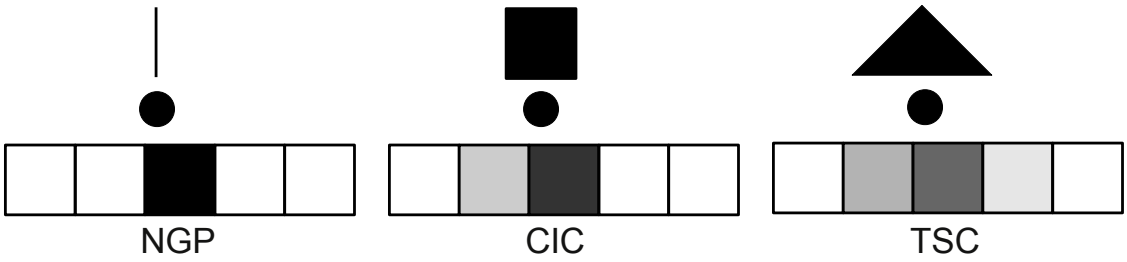

Figure III.2.2: Methods for assigning particles to grid cells. This illustrates the 1-D case—the generalization to two additional dimensions is straightforward.

limitation is that information about the density distribution (and, therefore, the potential and acceleration fields) on scales smaller than the size of a grid cell are lost. Consequently, structures on smaller scales will be lost or fail to form. This loses one of the nice features of the N-body approach, namely that it puts the resolution where it's needed. More elaborate techniques, using nested grids of different resolution can be used to help mitigate this limitation.

III.2.2.3 Particle-Particle/Particle-Mesh (P3M)

This approach aims to combine the advantages of direct summation (no loss of small scale resolution) and particle-mesh (fast) techniques. Briefly, it computes the forces by dividing contributions from particles nearby and far away. For particles that are "far away" (typically more than about 3 grid cell lengths away) the contribution to the force is computed using the particle-mesh technique. For particles that are closer by a direct summation is used. (Frequently this means doing the full particle-mesh calculation and then, for each nearby particle, subtracting the force it contributed in the particle-mesh approximation before adding its contribution using the direct summation approach). The biggest problem with this approach is that it can easily become dominated by the direct summation (particle-particle) part of the calculation and become about as slow as direct summation. This will happen in any simulation where particles become strongly clustered (and, unfortunately, gravity is attractive...).

III.2.2.4 Tree Algorithms

Tree algorithms take a rather different approach. In a "top down" approach, the entire simulation volume (typically a cubic region, or, in the 2-D example shown in Fig. III.2.3, a square) is placed into the top level cell. The next level in the tree is created by splitting this cell in half in each dimension such that the 2^{nd} level contains 8 cells (in 3-D; a so-called oct-tree) or 4 cells (in 2-D; a quad-tree). Further tree levels are made by repeatedly splitting cells in the next higher level in this way until each cell contains only a single particle. The resulting tree structure is illustrated in Fig. III.2.4. In the original Barnes-Hut [1] tree algorithm, the center of mass of the particles in each cell at each level is computed. Then, to compute the force on any given particle, one simply computes the contribution due to the total mass of particles located at the center of mass of a cell. For nearby particles, we will use the finest levels of the tree to accurately determine the forces. For more distant particles we can use a coarser level of the tree and therefore compute the force due to many particles in one go. In this way, the calculation can be made faster than direct summation

while retaining good spatial resolution on small scales. The tree approach has an advantage over particle-mesh techniques in that doesn't waste time on empty regions of the simulation volume (useful if one is simulating the collision of two galaxies for example). However, there is a memory overhead associated with constructing and storing the tree structure itself.

The original Barnes-Hut tree algorithm computed forces directly by treating particles in each cell as a monopole (i.e. a point mass). More recent algorithms work with the potential instead and account for higher order moments of the particle distribution in each cell. Essentially, the potential due to particles in a cell is described by a multipole expansion—more or less like expanding a function as a Taylor series. The more terms we include, the more accurate the representation (but the slower the calculation), and the functional forms used are easily differentiable making it simple to compute the force from the potential. This method is more accurate than the simple Barnes-Hut method, but is computationally more expensive if higher order multipoles are used.

How do we determine which level of the tree we should use to compute the force on any given particle? First, keep in mind that we're always going to have a trade off between speed and accuracy with the tree algorithm. If we decide to use the tips of each branch (call them "leaves") then we have precisely one particle per cell and we would have effectively the particle-particle algorithm, which would be very accurate, but very slow (slower than the original particle-particle algorithm because we wasted a whole lot of time building the tree!). Alternatively, if we used the base of the tree (call it the "trunk") then the calculation would be extremely fast, but very inaccurate, as we'd only consider the interaction of each particle with the center of mass of the distribution. Obviously, we want something in between.

The usual approach is to adopt an "opening angle criterion". Consider the circled particle in Fig. III.2.5. We want to compute the force on it due to all other particles. We can begin at the "trunk" of the tree and ask the following questions: Is the ratio of the size of the tree cell at this level divided by the distance from the particle to the center of mass of the cell less than some angle θ? More specifically, we ask if

$$\frac{d}{r} < \theta \tag{III.2.20}$$

where d is the size of the tree cell and r is the distance from the particle in question to the center of mass of this cell. We then proceed as follows:

(1) If the opening angle condition is satisfied, then compute the force from all particles in this tree cell by treating them as a single particle at the center of mass of the cell.

(2) Otherwise, walk along the tree branches to the next most refined level of the tree and recheck the opening angle condition.

In this way, as we get closer to a set of particles we will use more refined branches of the tree to compute the force from them. The parameter θ is a parameter that we can tune—making it smaller will make for a more accurate but slower calculation, while making it larger will reduce accuracy and increase speed. Typically, values of $\theta \lesssim 1$ are found to work quite well (although this depends on the details of the algorithm, such as whether or not higher order multipoles of the gravitational potential are included or not). A tree algorithm with this type of opening angle criterion is relatively easy to code as a recursive function, which simply walks along the branches of the tree accumulating forces from sets of particles as it goes and truncating the walk along each branch once the opening angle has been specified. From a coding point of view, this makes for a compact and elegant algorithm.

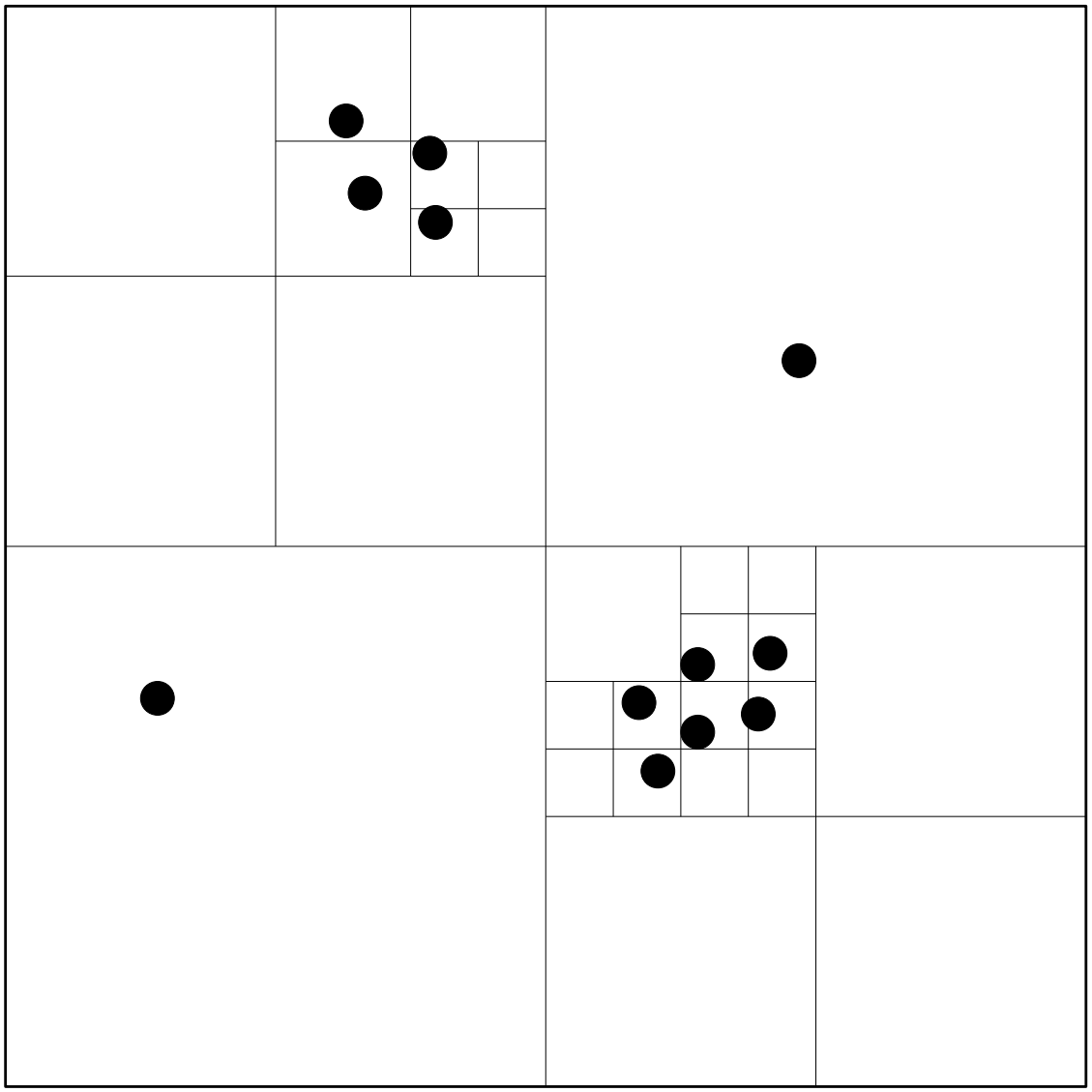

Figure III.2.3: Example of the construction of a quad-tree.

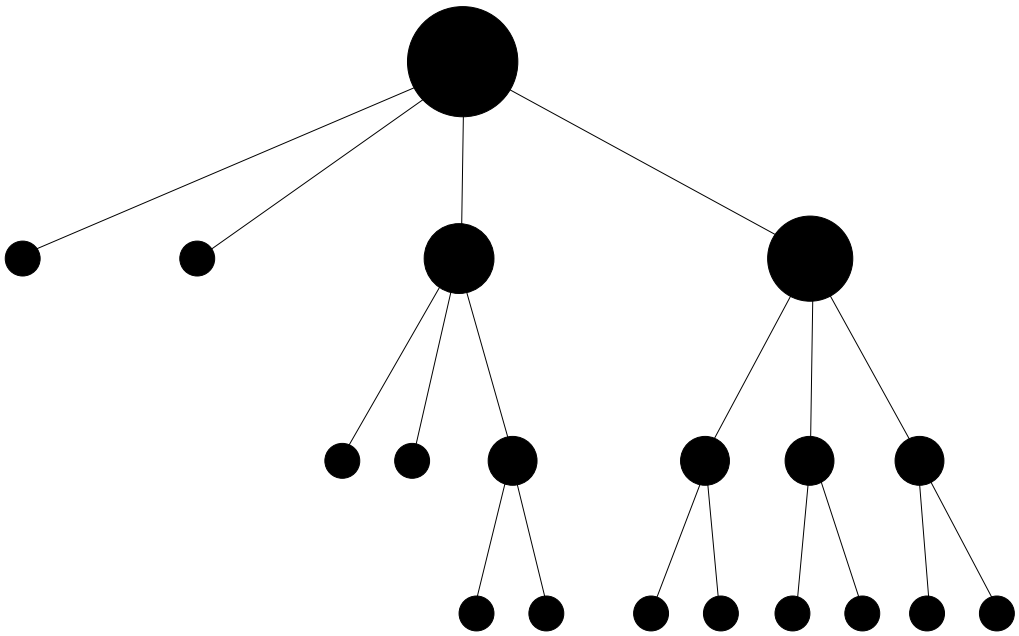

Figure III.2.4: The tree structure resulting from Fig. III.2.3.

Why this opening angle criterion and not some other criterion? We can think about the physics the underlies it. Consider the mass distribution in a tree cell. We can represent any density distribution as a sum over various multipoles. Therefore, we can write the gravitational potential for this mass distribution as a multipole expansion in spherical coordinates

$$\Phi(r,\theta,\phi) = \sum_{l=0}^{\infty}\sum_{m=0}^{l}\left(A_l r^l + B_l r^{-l-1}\right)\left[C_{lm}Y^c_{lm}(\theta,\phi) + S_{lm}Y^s_{lm}(\theta,\phi)\right], \tag{III.2.21}$$

where A_l, B_l, C_{lm} and S_{lm} are coefficients determined by the mass distribution and the Y_{lm}'s are the sine and cosine real spherical harmonics. (This is actually a Laplace series, valid outside the mass distribution—it is applicable to any potential which satisfies the Laplace equation $\nabla^2\Phi = 0$.) Since we expect the gravitational potential to decline with distance from the source then $A_l = 0$ for all l. Then we see that the potential due to the l^{th} multipole declines with distance as r^{-l-1} (e.g. for a monopole distribution, $l = 0$, the usual r^{-1} relation is obtained). Therefore, higher order multipole contributions to the potential will be reduced by a factor of approximately $(d/r)^l$ relative to the monopole contribution if d is the characteristic size of the mass distribution. By ensuring that $d/r < \theta$ we guarantee that, for suitably small θ, all terms beyond the monopole will be negligible.

III.2.3 Timestepping Criteria

Having computed the force acting on each of our particles, we need to evolve the particle distribution forward in time. This amounts to solving the following differential equations

$$\dot{\mathbf{x}}_i = \mathbf{v}_i \tag{III.2.22}$$

$$\dot{\mathbf{v}}_i = \mathbf{F}_i/m_i \tag{III.2.23}$$

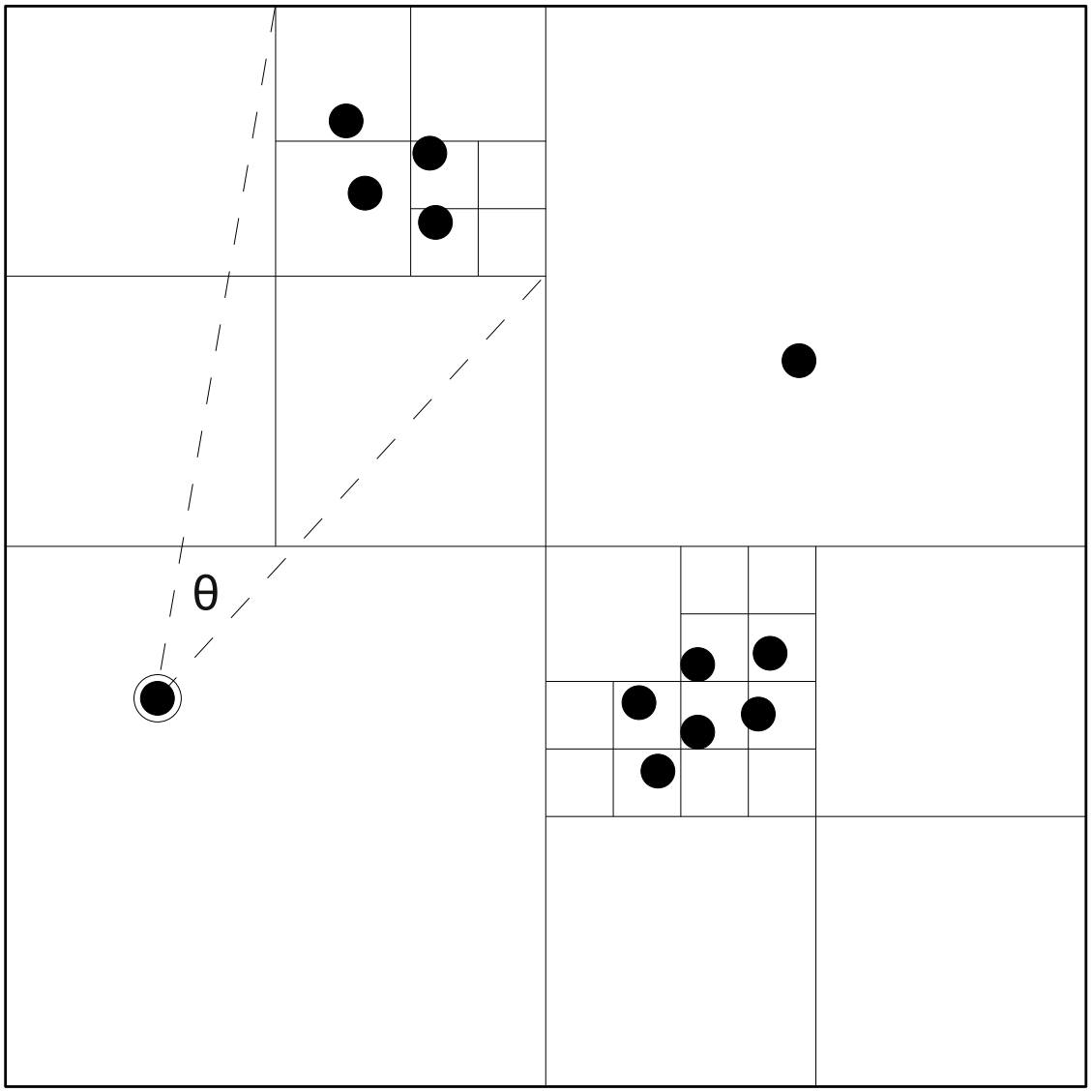

Figure III.2.5: Opening angle criterion.

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are the position and velocity of particle i, $\mathbf{F}_i$ is the force acting on that particle and m_i is its mass. Naively, we could choose some suitably small timestep, δt, and assume that the force is approximately constant over that period such that

$$\mathbf{x}_i \rightarrow \mathbf{x}_i + \mathbf{v}_i \delta t + \frac{1}{2}\mathbf{F}_i/m_i \delta t^2 \tag{III.2.24}$$

$$\mathbf{v}_i \rightarrow \mathbf{v}_i + \mathbf{F}_i/m_i \delta t. \tag{III.2.25}$$

Two problems immediately arise: First, how small should we make δt such that our approximation will be valid. Second, since characteristic timescales in gravitating systems scale as $\rho^{-1/2}$, the densest regions of our simulation will evolve on much shorter (i.e. orders of magnitude shorter) timescales so they'll force us to take very short timesteps. The second of these suggests that we perhaps want to have different timesteps for different particles (no problem in principle, but we have to correctly synchronize particles). The first suggests that we figure out some criterion to judge how small the timestep needs to be.

To make having individual timesteps for each the particles computationally manageable it's often useful to enforce the timesteps to by binary fractions of some overall timestep, T_0. That is, each particle will have a timestep $T_0/2^n$ for some integer n. Thus, a particle with $n = 1$ will take two steps for every one step taken by a particle with $n = 0$, while a particle with $n = 2$ will take four steps over the same time. The advantage of this approach is that we can easily maintain synchronization between particles (i.e. bring them all to the same time). To ensure sufficiently accurate evolution we then enforce

$$\delta t = \frac{T_0}{2^n} < \eta \frac{1}{\sqrt{G\rho}}, \tag{III.2.26}$$

where ρ is an estimate of the locally enclosed density. For a particle orbiting at radius r in a spherical potential for example, $\rho \sim M(r)/r^3$ if $M(r)$ is the mass inside radius r. Here, η is a numerical parameter that we can adjust to control the size of timesteps. We want timesteps small enough that any results we extract from the calculation are converged (i.e. unaffected by timestep size) but large enough that the calculation completes in a reasonable amount of time.

For specificity, we'll consider the time integration algorithm used in the original GADGET N-body code [8]. There are many other (and better) choices, but most share similar features (see, for example, Springel 9, Zemp et al. 10). GADGET-1 updates particle positions and velocities by first predicting the position of a particle at the middle of a timestep δt:

$$\mathbf{x}^{(n+\frac{1}{2})} = \mathbf{x}^{(n)} + \mathbf{v}^{(n)} \frac{\delta t}{2}, \tag{III.2.27}$$

where the superscripts refer to position/velocity at step n. This mid-step position is used to find the acceleration of the particle

$$\mathbf{a}^{(n+\frac{1}{2})} = -\nabla\Phi\big|_{\mathbf{x}^{(n+\frac{1}{2})}}. \tag{III.2.28}$$

The particle is then advanced to the next timestep using

$$\mathbf{v}^{(n+1)} = \mathbf{v}^{(n)} + \mathbf{a}^{(n+\frac{1}{2})} \delta t \tag{III.2.29}$$

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \frac{1}{2}\left[\mathbf{v}^{(n)} + \mathbf{v}^{(n+1)}\right] \delta t. \tag{III.2.30}$$

Note that we use the mean velocity through the timestep when updating the position. This is a variant of the so-called "leap-frog" method. Energy (kinetic plus potential) should always be

conserved along the particle's path, but in practice it won't be precisely since we take a finite timestep. It can be shown that the error in energy for the above time integrator is

$$\Delta E = \frac{1}{4}\frac{\partial^2\Phi}{\partial x_i \partial x_j} v_i^{(n)} a_j^{(n+\frac{1}{2})} \delta t^3 + \frac{1}{24}\frac{\partial^3\Phi}{\partial x_i \partial x_j \partial x_k} v_i^{(n)} v_j^{(n)} v_k^{(n)} \delta t^3 + \mathcal{O}(\delta t^4). \tag{III.2.31}$$

That is, this integrator is second order accurate (i.e. the first time in the above is third order in δt). In principle, the above could be used to choose δt to keep the error in energy below some predefined level. In practice this is not very useful as determining the gradients of the potential is expensive and we don't know for sure that the higher order terms are not important. Instead, GADGET-1 uses a criterion similar to that given above using a local estimate of the density. This local estimate is found by averaging over some number of nearby particles (similar to the approach used for smoothed particle hydrodynamics as we'll discuss later). This can be problematic in low density regions (where we have to go to large distances to find neighboring particles) and in dense regions resolved by only a small number of particles. For example, consider a cosmological simulation in which a low mass dark matter halo forms from 20 particles. Suppose we are using the 32 nearest particles to estimate the local density. In this case, the particles in the halo will have their local density estimate contaminated by particles that are not in the halo. As a result, the density estimate will be too low, the resulting timestep too long and so the halo will dissipate because of accumulating errors in energy conservation.

Therefore, GADGET-1 adds a second criterion

$$\delta t < \eta_\mathrm{a} \frac{\sigma}{|\mathbf{a}|}, \tag{III.2.32}$$

where η_a is an adjustable parameter and σ is an estimate of the local velocity dispersion (also found by averaging over nearby particles). These prevents the velocity changing by more than a fixed fraction of the local velocity dispersion in a given timestep. Consider again the case of the low mass dark matter halo described above. If we estimate the local velocity dispersion from the 32 nearest neighbors, then we'll underestimate it due to the contamination from particles outside the halo. In this case, however, this makes our timestep shorter, and so our integration more accurate. By combining these two criteria it's possible to maintain good energy conservation through a simulation while minimizing the number of timesteps that have to be taken.

III.2.4 Initial Conditions

So far we've considered how to compute the forces and evolve the particle positions/velocities for some set of particles for which we already know the position and velocity of each particle. But, we have to begin somewhere. Specifically, we need to know the positions and velocities of the particles at some initial time. These are the initial conditions of the calculation. Generally speaking, the goal of creating initial conditions is to set up the particles to represent some physical system while minimizing any numerical effects due to discreteness or assumptions made. We'll consider two scenarios which are common in the world of N-body simulation: an equilibrium system of dark matter, and the early Universe.

III.2.4.1 Equilibrium Dark Matter Halo

(Note that this could just as easily by an equilibrium system of stars, i.e. a galaxy.) A not uncommon N-body experiment is to see what happens when two systems, e.g. dark matter halos, merge.

To do this, we typically want to create isolated N-body representations of the halos when they are well separated and in internal equilibrium.

For cosmological halos in a cold dark matter universe, the Navarro-Frenk-White profile [6] is a good description of the run of density with radius. This profile has the form

$$\rho(r) = \frac{\rho_0}{(r/r_\mathrm{s})\left[1+(r/r_\mathrm{s})\right]^2}, \tag{III.2.33}$$

where ρ_0 is a normalization and r_s is a characteristic radius known as the scale radius. We can select values for these constants based on cosmological calculations, but the numerical values don't matter for our purposes. We want to make an N-body representation of an NFW halo. Let's assume that the halo is perfectly spherical to keep things simple. Also, we can note that the mass of the NFW profile is logarithmically divergent as we go to larger radii. This is kind of annoying for an N-body simulation because we can't model an infinite volume. So, let's introduce a truncation such that

$$\rho(r) = \frac{\rho_0}{(r/r_\mathrm{s})\left[1+(r/r_\mathrm{s})\right]^2} \times \begin{cases} 1 & \text{if } r < r_\mathrm{out} \\ \exp\left(-\frac{r}{r_\mathrm{out}}\right) & \text{if } r > r_\mathrm{out}, \end{cases} \tag{III.2.34}$$

where r_out is some outer radius. Like any good computational physicist we'll be sure to check that our adoption of a truncation radius doesn't affect out results (for example, by repeating our calculations with a larger value of r_out).

First, we want to select a position for each particle in our N-body representation. Since the halo is spherical it makes sense to use spherical coordinates. For ϕ things are easy, we just pick a number at random from a distribution that's uniform between 0 and 2π. For θ it's also easy—we know that the surface area on a sphere between θ and $\theta + \mathrm{d}\theta$ is proportional to $\sin\theta \mathrm{d}\theta \equiv \mathrm{d}\cos\theta$. So, if we select a number at random from a uniform distribution between -1 and 1 and call this $\cos\theta$ then take the inverse cosine to get θ we should have the correct distribution.

For the radial coordinate, r, we have to think a little more. If each of our particles has the same mass, then the chance of a particle drawn at random from the halo mass dsitribution lying between r and $r + \mathrm{d}r$ should be simply proportional to the fraction of mass in that region. That is

$$\mathrm{d}P(r) = \frac{4\pi r^2 \rho(r)\mathrm{d}r}{\int_0^\infty 4\pi r^2 \rho(r)\mathrm{d}r}. \tag{III.2.35}$$

if we integrate this, we get the cumulative probability that any randomly selected particle will be found within radius r:

$$P(r) = \frac{\int_0^r r'^2 \rho(r')\mathrm{d}r'}{\int_0^\infty 4\pi r'^2 \rho(r')\mathrm{d}r'}. \tag{III.2.36}$$

By definition, $P(r)$ runs from 0 to 1. Therefore, for each particle we can simply select a random value from a uniform distribution between 0 and 1 and call this $P(r)$. We then use the above equation to infer the r that corresponds to this $P(r)$. In this way we're guaranteed to build up a halo with the correct density profile. Note that the above integrals are not analytically tractable when the exponential term is included, so they usally have to be solved numerically.

After applying this procedure, we have 3-D positions for all of the particles in the N-body representation of the halo (which we can easily convert to Cartesian coordinates for input into our N-body solver). We also need velocities. For these, we make use of the fact that our halo is assumed to be in equilibrium. In collisionless gravitational dynamics, we know that any equilibrium

system must obey Jeans equation:

$$\frac{\mathrm{d}(\rho\sigma_r^2)}{\mathrm{d}r} + 2\frac{\beta(r)}{r}\rho(r)\sigma_r^2(r) = -\frac{GM(r)}{r^2}\rho(r), \tag{III.2.37}$$

where $\sigma_r(r)$ is the radial velocity dispersion at radius r, $M(r)$ is the mass enclosed with radius r and $\beta(r)$ is the anisotropy parameter defined as

$$\beta(r) = 1 - \frac{\sigma_\theta^2(r) + \sigma_\phi^2(r)}{2\sigma_r^2(r)}, \tag{III.2.38}$$

with $\sigma_\phi(r)$ and $\sigma_\theta(r)$ being the velocity dispersions in the other two directions. Since we know $\rho(r)$ (and, therefore, $M(r)$) we can solve this equation for $\sigma_r(r)$ if we choose some form for $\beta(r)$. There are many choices here, but for simplicity let's assume an isotropic velocity distribution which implies $\beta(r) = 0$. The Jeans equation is then easily solved for $\sigma_r(r) = \sigma_\phi(r) = \sigma_\theta(r)$. If we further assume that the velocity distribution in each coordinate is always a Gaussian then we have a fully specified velocity distribution function at all radii in the halo. Given the radius of a particle, we can then compute the corresponding σ_r and draw a random number from a normal distribution with standard deviation σ_r and repeat for the other two coordinates. This will give us a set of particle velocities consistent with Jeans equation and so consistent with an equilibrium halo.

Before moving on, we should think a little more about the final assumption made above, namely that the velocity distribution is Gaussian. Jeans equation does not tell us that the distribution is Gaussian. It merely tells us what the dispersion of the velocity distribution should be. We could construct an infinite number of non-Gaussian distributions which all have the same σ_r. So, we don't know for sure that we have the correct velocity distribution. Jeans equation can be extended to higher order moments of the velocity distribution, so we could (in principle) compute all of these higher moments and use them to construct a more accurate velocity distribution. In practice this is sometimes done, but often it is not. The reason is that the Gaussian approximation is often not too bad, and computing higher order moments rapidly becomes numerically challenging. When solving numerical problems we should always keep in mind which of the many approximations we have made is the most severe. For example, is it worth solving the Jeans equation for higher order moments of the velocity distribution when we've already approximated the halo as being spherically symmetric? Real halos are not spherically symmetric, so maybe this assumption is the biggest factor limiting the accuracy of our results. Maybe the fact that we're modelling purely dark matter with no gas component is a bigger limitation... There's no point wasting time doing one part of the calculation to arbitrarily high precision if other aspects are solved in a much cruder way.

III.2.4.2 Cosmological Initial Conditions

For cosmological calculations we're interested in setting up initial conditions that represent the distribution of matter in the early stages of the Universe. At early times, the Universe is almost perfectly uniform with just small perturbations in the density as a function of position (these are the source of the 10^{-5} level ripples seen in the cosmic microwave background). In our standard cosmological model there are two features that make the situation simpler still. First, if inflation is correct then the density perturbations should be Gaussian—that is the phases of individual Fourier modes of the density field are uncorrelated and so the density field is completely described (statistically) by a power spectrum (which measures the mean amplitude of Fourier modes of a

given wavelength). Second, in cold dark matter models the particles begin with zero random velocities—i.e. there is no velocity dispersion at a given point and so velocity is an entirely deterministic function of position, controlled entirely by the density field. A further useful feature is that, because the initial density perturbations are small they can be treated with a linear perturbation theory analysis. Providing we begin our simulation at an early enough time in the universe we can therefore use a linear analysis to set our initial conditions. We won't discuss the details of cosmological perturbation theory in this class (take Ay 127 for that). A good discussion of the details (including the unpleasant relativistic aspects that we're going to completely ignore!) can be found in [5], while a code for generating such initial conditions is given by [2].

Briefly, the standard method to construct initial conditions for such scenarios is as follows:

(1) Create a set of "pre-initial conditions" which is a random, but homogeneous set of particle positions in a periodic cube. We can construct this by simply selecting particle positions (x, y, z) at random within a cubic region[2].

(2) Compute the power spectrum, $P(k)$, that gives the mean amplitude of each Fourier mode of the desired density field. For a Gaussian random field the distribution of amplitude, $\hat{\delta}$, for a given mode can be shown to be a Rayliegh distribution

$$P(\hat{\delta})\mathrm{d}\hat{\delta} = \frac{\hat{\delta}}{\sigma^2}\exp\left(-\frac{\hat{\delta}^2}{2\sigma^2}\right)\mathrm{d}\hat{\delta}, \tag{III.2.39}$$

where $\sigma^2 = VP(k)/2)$ and V is the volume of the simulation cube[3].

(3) Take the Fourier transform to convert the $\hat{\delta}$ Fourier components into an actual density field. We then want to move the particles to recreate this density field in our N-body representation. For this we can use the Zel'dovich approximation which relates the Lagrangian position (position in the intial conditions) of a particle to its Eulerian position. We won't discuss the details, but note that this is a linear perturbation theory solution to the cosmological equations governing the growth of density perturbations. The Zel'dovich approximation states that $\mathbf{x} = \mathbf{q} + \Psi(\mathbf{q})$ where $\mathbf{q}$ is the initial (Lagrangian) position of a particle and $\mathbf{x}$ the Eulerian position. We can compute the gravitational potential $\Psi(\mathbf{q})$ from our density field and so can find $\mathbf{x}$. We can also take the derivative of this equation to get the velocity $\dot{\mathbf{x}}$.

There are problems with this method that make constructing accurate cosmological methods very difficuly. For example, what about the contribution from modes with wavelengths longer than the size of our simulation box? If we compute the density field on a grid, then we will only get Fourier modes whose wavelength is an integer number of grid cell lengths—do this missing modes matter? Fortunately, these issues have been considered extensively (see, for example, Sirko 7) and accurate initial condition generating codes exist.

[2]Frequently, this is not actually what is done, because it introduces fluctuations in the density field due to Poisson statistics, i.e. fluctuation in particle number in any region. If we began evolving this "homogeneous" distribution regions that randomly happen to be denser would quickly collapse under their own gravity. An alternative is to use a distribution in which the force on each particle is zero. For example, a regular lattice satisfies this condition (although it's an unstable equilibrium). Another common approach is to use "glass" initial conditions which can be made by starting with a Poisson-random distribution, then evolving it forward in time using a repulsive gravitational force. The particles will then all try to move apart until they reach positions of zero net force.

[3]Note that since the density field must be real, the Fourier components must satisfy the Hermitian constraints: $\hat{\delta}(\mathbf{k}) = \hat{\delta}^*(-\mathbf{k})$.

III.2.5 Parallelization

If you can do something with one computer, you can do it bigger and better with many computers. At least, you can in principle...

N-body codes are computationally demanding and so it's no surprise that a lot of work has been put into designing them to run efficiently on parallel computers. The idea is to divide the calculations that must be performed between a large number of processors and have the each solve part of the problem. To do this well there are a few considerations:

- How do we best divide the work between the processors? This may be limited by the amount of memory available to each processor (which will limit, for example, how many particles can be stored on each processor). Most importantly though, we want each processor to have about the same amount of work to do—if we have a situation where one processor is still crunching through its tasks while all others have already finished then we're not using the processors most efficiently. This is referred to as *load balancing*.
- How can we minimize the amount of communication between processors? Processors will need to communicate their results to each other (since, for example, particles on one processor will need to know the forces they feel from particles on some other processor). Communication is a relatively slow task so we want to minimize it as much as possible.

[4] gives a description of an approach to parallelizing a tree code. We'll follow his discussion. The basic approach is one of *domain decomposition* in which the simulation volume is broken up into sub-volumes and each sub-volume is assigned to a separate processor. The task is to find a sufficiently optimal domain decomposition to meet the criteria described above.

Let us assign to each particle a cost, C_i, which is a measure of the computational work required for this particle. In practice, this could be the number of gravitational force evaluations to evolve it over some fixed time perdiod for example. The cost is something we can compute relatively easily for each particle as the simulation proceeds. At the start of the calculation we don't know C_i, so we'll set $C_i = 1$ for all particles initially. We want to choose domains such that $\sum C_i$ (where the sum is taken over all particles in the domain) is about the same for all domains. To do this, we can use a technique to the tree construction algorithm we've used before. Suppose we have 2^n processors, with n some integer. We begin by splitting the simulation volume in two, with the division made such that $\sum C_i$ is the same (or almost the same) on each side of the division. We then proceeed to split each of these two domains into two sub-domains again balancing $\sum C_i$ in each subdomain. Repeating this process n times we get 2^n domains, corresponding to rectangular regions of the simulation cube, each with the same total computational cost.

As the simulation proceeds, the cost for each particle will change. For example, if a particle moves into a denser region it will require more computations to evolve its position and so its cost will increase. Therefore, our initially load balanced domain decomposition will not remain load balanced. Therefore, after each timestep (or, perhaps, once the load balancing becomes sufficiently bad) we can attempt to repartition into new domains to rebalance the workload. This means moving some particles from one processor to another, and therefore requires communication. We want to minimize this communication by moving as few particles as possible. Deciding which particles to move and when depends on the details of the simulation (e.g. how clustered the particles become), the algorithm used (which affects the distribution of costs) and the hardware used (which determines the relative costs of communication vs. work load imbalance). So, this

usually involves some tuning of parameters to find the optimal way in which to move particles from one processor to another.

An additional problem arises with the need to communicate gravitational forces between domains. Once we've performed the domain decomposition it's easy for each processor to build an oct-tree for its local set of particles. We could then have each processor share its local oct-tree with every other processor. Then, each processor would have the full tree and so could proceed to compute accelerations and update positions for all of its particles. In practice this is a bad idea for two reasons. First, the memory requirements for the full tree may be more than a single processor can handle. Second, this involves a lot of communication which slows down the calculation. However, we can make use of the opening angle criterion to limit the communication and memory requirements. Consider two widely spaced domains, which we'll label 1 and 2. Since all of the particles in domain 1 are far from all of the particles in domain 2 we'll only need to use the coarsest levels of the domain 2 tree to evaluate forces for particles in domain 1 (and vice versa). Therefore, we can apply the opening angle criterion to domain 2 in a conservative way to find the finest level of the domain 2 tree that will ever be needed by any particle in domain 1. We then communicate only those levels of domain 2 to the processor working on domain 1, and none of the finer levels of the tree. This allows the *essential tree* for domain 1 to be built—it contains just enough information to compute the forces on domain 1 particles at the required level of accuracy. This minimizes the communication and memory requirements.

Modern N-body codes use more complicated algorithms, but the basic ideas are the same.

References

[1] Josh Barnes and Piet Hut. A hierarchical O(N log n) force-calculation algorithm. *Nature*, 324: 446, December 1986. URL http://adsabs.harvard.edu/abs/1986Natur.324..446B.

[2] E. Bertschinger. ASCL: COSMICS: cosmological initial conditions and microwave anisotropy codes. http://ascl.net/cosmics.html. URL http://ascl.net/cosmics.html.

[3] Walter Dehnen. Towards optimal softening in three-dimensional n-body codes - i. minimizing the force error. *Monthly Notices of the Royal Astronomical Society*, 324:273, June 2001. URL http://adsabs.harvard.edu/abs/2001MNRAS.324..273D.

[4] John Dubinski. A parallel tree code. *New Astronomy*, 1:133–147, October 1996. URL http://adsabs.harvard.edu/abs/1996NewA....1..133D.

[5] Chung-Pei Ma and Edmund Bertschinger. Cosmological perturbation theory in the synchronous and conformal newtonian gauges. *The Astrophysical Journal*, 455:7, December 1995. URL http://adsabs.harvard.edu/abs/1995ApJ...455....7M.

[6] Julio F. Navarro, Carlos S. Frenk, and Simon D. M. White. A universal density profile from hierarchical clustering. *The Astrophysical Journal*, 490:493, December 1997. URL http://adsabs.harvard.edu/abs/1997ApJ...490..493N.

[7] Edwin Sirko. Initial conditions to cosmological N-Body simulations, or, how to run an ensemble of simulations. *The Astrophysical Journal*, 634:728–743, November 2005. URL http://adsabs.harvard.edu/abs/2005ApJ...634..728S.

[8] V. Springel, N. Yoshida, and S. D. M. White. GADGET: a code for collisionless and gas-dynamical cosmological simulations. *New Astronomy*, 6:79–117, April 2001. URL http://adsabs.harvard.edu/abs/2001NewA....6...79S.

[9] Volker Springel. The cosmological simulation code GADGET-2. *Monthly Notices of the Royal Astronomical Society*, 364:1105–1134, December 2005. URL http://adsabs.harvard.edu/abs/2005MNRAS.364.1105S.

[10] Marcel Zemp, Joachim Stadel, Ben Moore, and C. Marcella Carollo. An optimum time-stepping scheme for n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 376:273–286, March 2007. URL http://adsabs.harvard.edu/abs/2007MNRAS.376..273Z.

III.3 Hydrodynamics I – The Basics

Strictly speaking, hydrodynamics is a gross simplification: The dynamics of a distribution of particles (e.g., atoms, atomic nuclei, elementary particles, photons) $f(\mathbf{x}, \mathbf{p}, t)d\mathbf{x}d\mathbf{p}$ in 6D phase space and time is described by the Boltzmann equation

$$\frac{\partial f}{\partial t} + \frac{\mathbf{p}}{m}\nabla_{\mathbf{x}} f + \mathbf{F}\nabla_{\mathbf{p}} f = \left(\frac{\partial f}{\partial t}\right)_{\rm coll} \ . \tag{III.3.1}$$

Here, **F** is an external force, m is the mass of a particle, and the **coll**ision term on the right-hand side describes the interaction of particles with each other and with the environment: emission, absorption, scattering, and viscous effects. Details on the Boltzmann equation for classical and quantum gases (fluids, particles etc.) can be found in statistical mechanics books, e.g. in Huang's book [1].

III.3.1 The Approximation of Ideal Hydrodynamics

Two main assumptions are made in ideal hydrodynamcis:

- **Hydrodynamic Approximation** (local equilibrium). The mean free path λ between particle collisions is small compared to the size of a computational region, $\lambda \ll dx$, and the time τ between collision is shorter than the dynamical timescale of the system, $\tau \ll tau_{\rm dyn}$. This allows us to neglect individual particle collisions and treat all particles as a continuous fluid.
- **Inviscid Approximation**. The particles are interacting by hard-body collisions, but in no other way locally, i.e. there is, e.g., no viscosity. When this approximation is not appropriate, one needs to solve the equations of non-ideal hydrodynamics, the Navier-Stokes equations, which we shall not address here.

The ideal hydrodynamcis approximation works well for most astrophysical problems involving gas/fluid flow. Situations involving viscosity (e.g. in accretion flows) can usually be treated by adding approximate "viscous" terms to the inviscid equations and it is rarely necessary to solve the full Navier-Stokes equations.

III.3.2 The Equations of Ideal Hydrodynamics

The equations of ideal hydrodynamics are derived by taking moments of the Boltzmann euqation and represent *conservation laws* for mass, energy, momentum. They are called *Euler Equations*.

(1) **Continuity Equation** (mass conservation)

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \mathbf{v}) = 0 \ . \tag{III.3.2}$$

(2) **Momentum Equation** (momentum conservation)

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla(\rho \mathbf{v}\mathbf{v} + P) = \rho \mathbf{F} \ , \tag{III.3.3}$$

where **F** is an external force, e.g. gravity.

(3) **Total Energy Equation** (energy conservation)

$$\frac{\partial}{\partial t}\mathcal{E} + \nabla((\mathcal{E}+P)\mathbf{v}) = \rho\mathbf{v}\mathbf{F} \ . \tag{III.3.4}$$

Here $\mathcal{E}$ is the total specific energy per unit volume, defined by

$$\mathcal{E} = \rho\epsilon + \frac{1}{2}\rho v^2 \ , \tag{III.3.5}$$

where ϵ is the specific internal energy per unit mass and the $\frac{1}{2}\rho v^2$ term is the specific kinetic energy per volume.

The three equations specified in the above (2 scalar equations, 1 vector equation) are incomplete: An equation of state (EOS), connecting pressure to density, internal energy, and composition, is needed to close the system. In general, we may assume $P = P(\rho, \epsilon, \{X_i\})$, where the X_i are the mass fractions of the particle species present in the fluid.

III.3.3 Euler and Lagrange Frames

The Euler equations given in the previous section III.3.2 describe the evolution of a fluid at any fixed location $\mathbf{x}$ in space. This approach is called *Eulerian* and the Eulerian time derivative $\partial/\partial t$ refers to the changes occurring as a result of the flow of the fluid past the a fixed location $\mathbf{x}$.

An alternative view is called *Lagrangian* and assumes spatial coordinates that ar comoving with the fluid (they are also called *comoving coordinates*). The Lagrangian position vector $\mathbf{r}$ is defined by the instantaneous position of the fluid element. The Lagrangian time derivative D/Dt refers to the changes within a fluid element as it changes its state and location.

At the location occupied by the fluid element at an instant t, the Lagrangian velocity equals the Eulerian velocity with with the element sweeps past $\mathbf{x}$:

$$\frac{D}{Dt}\mathbf{r} = \mathbf{v}(\mathbf{x}) \ . \tag{III.3.6}$$

Using the definition of the Lagrangian derivative we may write for a quantity Q,

$$\frac{DQ}{Dt} = \lim_{\Delta t\to 0} \frac{Q(\mathbf{x}+\mathbf{v}\Delta t, t+\Delta t) - Q(\mathbf{x},t)}{} .\Delta t \tag{III.3.7}$$

Expanding this to first order yields

$$Q(\mathbf{x}+\mathbf{v}\Delta t, t+\Delta t) = Q(\mathbf{x},t) + \frac{\partial Q}{\partial_t}\Delta t + \sum_j v_j \frac{\partial Q}{\partial x_j}\Delta t \ . \tag{III.3.8}$$

So, to first order,

$$\frac{DQ}{Dt} = \frac{\partial Q}{\partial t} + \sum_j v_j \frac{\partial Q}{\partial x_j} \tag{III.3.9}$$

is the transformation equation between Eulerian and Lagrangian derivatives.

Consider the Eulerian continuity equation:

$$\frac{\partial \rho}{\partial t} + \sum_j \frac{\partial}{\partial x_j}(\rho v_j) = 0 \ . \tag{III.3.10}$$

Now,

$$\frac{\partial}{\partial x_j}(\rho v_j) = v_j \frac{\partial \rho}{\partial x_j} + \rho \frac{\partial v_j}{\partial x_j} \ , \tag{III.3.11}$$

and the Lagrangian variant is simply given by

$$\begin{aligned} \frac{D\rho}{Dt} &= \frac{\partial \rho}{\partial t} + \sum_j v_j \frac{\partial \rho}{\partial x_j} = -\rho \sum_j \frac{\partial v_j}{\partial x_j} \ , \\ \frac{D\rho}{Dt} &+ \rho \sum_j \frac{\partial v_j}{\partial x_j} = 0 \ . \end{aligned} \tag{III.3.12}$$

The Lagrangian expressions for the momentum and energy equations can be derived in similar fashion. They are

$$\frac{Dv_i}{Dt} + \frac{1}{\rho}\frac{\partial P}{\partial x_i} = F_i \ , \tag{III.3.13}$$

for momentum, and

$$\frac{D\mathcal{E}}{Dt} + \sum_j v_j \frac{\partial P}{\partial x_j} + (\mathcal{E} + P) \sum_j \frac{\partial v_j}{\partial x_j} = \rho \sum_j v_j F_j \ , \tag{III.3.14}$$

for energy.

III.3.4 Special Properties of the Euler Equations

In the following, we shall consider the Eulerian picture and only one spatial dimension to keep the notation simple. All statements translate straightforwardly to the multi-dimensional case.

III.3.4.1 System of Conservation Laws

We can write the Euler equations as a *flux-conservative system* of PDEs

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F} = \mathbf{S} \ , \tag{III.3.15}$$

where the state vector $\mathbf{U}$ is

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho v \\ \rho\epsilon + \frac{1}{2}\rho v^2 \end{pmatrix} \ , \tag{III.3.16}$$

and the flux vector $\mathbf{F}$ is given by

$$\mathbf{F} = \begin{pmatrix} \rho v \\ \rho v v + P \\ (\rho\epsilon + \frac{1}{2}\rho v^2 + P) v \end{pmatrix} \ . \tag{III.3.17}$$

The source vector $\mathbf{S}$ represents an external force. If it is zero, the system of equations (III.3.15) is said to be conservative. Otherwise it is flux conservative.

III.3.4.2 Integral Form of the Equations

Assume a domain D and d dimensions. In the absence of sources, we can write

$$\underbrace{\frac{d}{dt}\int_D \mathbf{U} dx}_{\substack{\text{temporal change of the state vector in the domain}}} + \underbrace{\sum_{j=1}^{d}\int_{\partial D}\mathbf{F}_j \mathbf{n}_j dS}_{\substack{\text{gains \& losses throught the boundary of the domain}}} = 0 \tag{III.3.18}$$

III.3.4.3 Hyperbolicity

The Euler equations are hyperbolic. Consider

$$\frac{\partial}{\partial x}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F} = 0 \ . \tag{III.3.19}$$

and compute the spatial derivative. With that, we can rewrite the conservation law in *quasi-linear form*,

$$\frac{\partial}{\partial t}\mathbf{U} + \bar{A}\frac{\partial}{\partial x}\mathbf{U} = 0 \ , \tag{III.3.20}$$

where

$$\bar{A} = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \tag{III.3.21}$$

is the *Jacobian* matrix of the system. The qualification quasi-linear derives from the fact that $\bar{A}$ will generally depend on $\mathbf{U}$, $\mathbf{x}$, and t.

Following what we discussed in §II.9.1.1, our system of equations is hyperbolic if $\bar{A}$ is diagonizable and has only real eigenvalues. This is indeed the case for the Euler equations. There exists a matrix $\overline{Q}$ so that

$$\overline{\Lambda} = \overline{Q}^{-1}\bar{A}\overline{Q} \tag{III.3.22}$$

and $\overline{\Lambda}$ is diagonal with real numbers, the eigenvalues of $\bar{A}$, on its diagonal. The eigenvalues of $\bar{A}$ are

$$\lambda_i = \{v, v + c_s, v - c_s\} \ , \tag{III.3.23}$$

where c_s is the adiabatic sound speed,

$$c_s = \sqrt{\left.\frac{dP}{d\rho}\right|_s} \ , \tag{III.3.24}$$

where $|_s$ denotes "at constant (specific) entropy".

III.3.4.4 Characteristic Form

If we write $\mathbf{U}' = \overline{Q}^{-1}\mathbf{U}$, we can write

$$\frac{\partial}{\partial t}\mathbf{U}' + \overline{\Lambda}\frac{\partial}{\partial x}\mathbf{U}' = 0 \ . \tag{III.3.25}$$

Since $\overline{\Lambda}$ is diagonal, the equations for $\mathbf{U}'$ are decoupled and are called *characteristic equations*. For each i, we have

$$\frac{\partial}{\partial t}U_i' + \lambda_i \frac{\partial}{\partial x}U_i' = 0 \tag{III.3.26}$$

where the λ_i are the eigenvalues of $\bar{A}$ and are called the *characteristic speeds*.

III.3.4.5 Weak Solutions

Hyperbolic conservation laws like the Euler equations admit so-called *weak solutions* that satisfy the integral form of the conservation law, but may have a finite number of discontinuities with the variables of the system obeying certain *jump conditions* at these discontinuities.

In hydrodynamics, such discontinuities are called *shocks* or *contact discontinuities*.

III.3.5 Shocks

In any situation, we can decompose the velocity $\mathbf{v}$ into componets that are perpendicular and tangential to a surface $\mathbf{S}$, $\mathbf{v} = \mathbf{v}_\perp + \mathbf{v}_\parallel$.

We define a *shock* (a shock wave, a shock front) as a surface across which P, ρ, $\mathbf{v}_\perp$, T, and the entropy s change abruptly, whereas $\mathbf{v}_\parallel$ remains continuous. A typical example of a shock is the explosion of a supernova that expands into the interstellar space.

We define a *contact discontinuity* (a contact) as a surface across which (one or multiple of) ρ, T, s, or $\mathbf{v}_\parallel$ change, but P and $\mathbf{v}_\perp$ remain continuous. An example for a contact discontinuity is the sharp interface between a cold dense region in the interstellar medium and an adjacent warm, lower-density region at the same gas pressure. The key to stability is the constant pressure across the boundary (and the absence of significant external gravitational fields that could lead to bouancy).

III.3.5.1 How Shocks Develop

For simplicity, we will assume an isentropic fluid (a fluid with constant entropy). The will be clearly wrong once we have a shock, but it is an acceptable approximation for the phase before the shock appears. The following discussion is based on Mihalas & Mihalas [2].

So, our equation of state (EOS) will be a polytropic one:

$$P = K\rho^\gamma \ , \tag{III.3.27}$$

where K is the polytropic constant and γ is the adiabatic index. The adiabatic speed of sound is given by

$$c_s = \sqrt{\left.\frac{\partial P}{\partial \rho}\right|_s} = \sqrt{\gamma \frac{P}{\rho}} \ , \tag{III.3.28}$$

where $|_s$ denotes "at constant (specific) entropy".

Let us set up a one-dimensional pulse (a smalls spatial distribution of fluid density ρ) launched into an infinite homogeneous medium. Since our flow is isentropic, we have to consider only the

continuity and the momentum equations. The energy is always given by the first law of thermodynamics:

$$d\epsilon = -Pd\left(\frac{1}{\rho}\right) \ , \quad \text{and} \ \ P = K\rho^{\gamma} \ , \tag{III.3.29}$$

$$\epsilon = \frac{P}{(\gamma-1)\rho} \ , \tag{III.3.30}$$

where we have set the integration constant to zero.

Assume now that that ρ and the fluid velocity v are *single-valued* functions[4] of the time t. Then we can, using $\rho(t)$ and $v(t)$ (where we have suppressed the dependence on the spatial coordinate), find $\rho(v)$ and $v(\rho)$. We may also re-write the continuity equation to

$$\frac{\partial\rho}{\partial t} + \frac{\partial}{\partial x}(v\rho) = 0 \tag{III.3.31}$$

to

$$\begin{aligned} \left(\frac{d\rho}{dv}\right)\left(\frac{\partial v}{\partial t}\right) + \left[v\left(\frac{d\rho}{dv}\right) + \rho\right]\frac{\partial v}{\partial x} = 0 \ , \\ \frac{\partial v}{\partial t} + \left[v + \rho\left(\frac{dv}{d\rho}\right)\right]\frac{\partial v}{\partial x} = 0 \ . \end{aligned} \tag{III.3.32}$$

The momentum equation may similarly be rewritten to

$$\frac{\partial v}{\partial t} + \left[v + \frac{1}{\rho}\underbrace{\left(\frac{\partial P}{\partial\rho}\right)\Big|_s}_{=c_s^2}\frac{d\rho}{dv}\right]\frac{\partial v}{\partial x} = 0 \ . \tag{III.3.33}$$

Comparing (III.3.32) with (III.3.33), we find

$$\frac{dv}{d\rho} = \pm\frac{\sqrt{\left.\frac{\partial P}{\partial\rho}\right|_s}}{\rho} = \pm\frac{c_s}{\rho} \ . \tag{III.3.34}$$

Hence, the relation between v and ρ (or P) in the wave is

$$v = \pm\int_{\rho_0}^{\rho}\frac{c_s}{\rho}d\rho = \pm\int_{P_0}^{P}\frac{dP}{\rho c_s} \ , \tag{III.3.35}$$

where ρ_0 and P_0 are the ambient values in the unperturbed medium.

Now, using (III.3.34) in (III.3.32) or (III.3.33), we obtain

$$\frac{\partial v}{\partial t} + (v \pm c_s)\frac{\partial v}{\partial x} = 0 \ . \tag{III.3.36}$$

Similarly to before, we can again rewrite the continuity equation, this time saying $v = v(\rho)$:

$$\frac{\partial\rho}{\partial t} + \left(\rho\frac{dv}{d\rho} + v\right)\frac{\partial\rho}{\partial x} = 0 \ , \tag{III.3.37}$$

[4] A single-valued function is a unique map of one argument value to one function value or multiple argument values to one function value.

which with (III.3.34) implies

$$\frac{\partial \rho}{\partial t} + (v \pm c_s)\frac{\partial \rho}{\partial x} = 0 \ . \tag{III.3.38}$$

Equations (III.3.36) and (III.3.38) have general solutions of the kind

$$v = F_1[x - (v \pm c_s)t] \ , \tag{III.3.39}$$
$$\rho = F_2[x - (v \pm c_s)t] \ , \tag{III.3.40}$$

where F_1 and F_2 are functions representing simple traveling waves. Hence, a particular value of ρ or v will propagate through the medium with *phase space*

$$u_p(v) = v \pm c_s(v) \ , \tag{III.3.41}$$

where $c_s(v)$ is given by Eqs. (III.3.34) and (III.3.35). For the $\pm$ sign in the above equations, we now choose:

$+$ for waves traveling in the $+x$ direction,
$-$ for waves traveling in the $-x$ direction.

Since we can write $\rho = \rho(v)$ and $P = P(\rho(v))$ (and so on), all physical variables in the wave propagate in the same manner as v.

Now, for our perfect isentropic fluid,

$$\boxed{c_s^2 \propto \frac{P}{\rho} \propto \rho^{\gamma-1}} \ , \tag{III.3.42}$$

from which follows

$$(\gamma - 1)\frac{d\rho}{\rho} = 2\frac{dc_s}{c_s} \ . \tag{III.3.43}$$

Plugging this into (III.3.35) yields

$$\boxed{v = \pm\frac{2(c_s - c_{s_0})}{\gamma - 1}} \ , \tag{III.3.44}$$

or,

$$c_s = c_{s_0} \pm \frac{1}{2}(\gamma - 1)v \ . \tag{III.3.45}$$

This implies that the phase velocity is given by

$$u_p(v) = \frac{1}{2}(\gamma + 1)v \pm c_{s_0} \ . \tag{III.3.46}$$

Using the polytropic gas laws (e.g., [2]) for the polytropic EOS $P = K\rho^\gamma$,

$$\begin{aligned} P &= P_0\left(\frac{T}{T_0}\right)^{\gamma/(\gamma-1)} \ , \\ P &= P_0\left(\frac{\rho}{\rho_0}\right)^{\gamma} \ , \\ T &= T_0\left(\frac{\rho}{\rho_0}\right)^{\gamma-1} \ , \end{aligned} \tag{III.3.47}$$

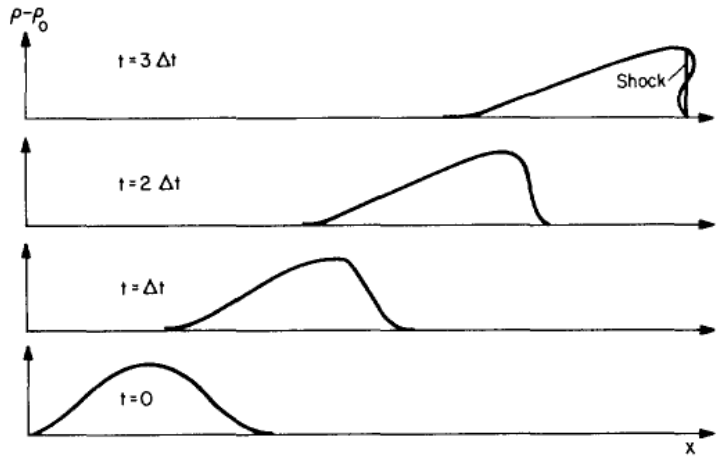

Figure III.3.1: Non-linear steepening of a sound wave into a shock. Taken from Mihalas & Mihalas [2].

and combining them with (III.3.44), we arrive at

$$\begin{aligned} \rho &= \rho_0 \left[1 \pm \frac{1}{2}(\gamma-1)\left(\frac{v}{c_{s_0}}\right)\right]^{2/(\gamma-1)} , \\ P &= P_0 \left[1 \pm \frac{1}{2}(\gamma-1)\left(\frac{v}{c_{s_0}}\right)\right]^{2\gamma/(\gamma-1)} , \\ T &= T_0 \left[1 \pm \frac{1}{2}(\gamma-1)\left(\frac{v}{c_{s_0}}\right)\right]^{2} . \end{aligned} \tag{III.3.48}$$

Interpretation

Consider a finite-size density wave pulse with an initial half-sinosoidal shape moving to the right (Fig. III.3.1). Eqs. (III.3.42), (III.3.44), and (III.3.48) show that more dense regions of the pulse have a higher sound speed (and pressure and temperature) and move faster than less dense regions. Hence, the wave crest (highest density) travels faster than the rest of the pulse and eventually overtakes the pulse front and the wave *breaks*. At thsi point, $\rho(x,t)$ becomes multi-valued adn the solution breaks down: We have a shock!

III.3.6 Rankine-Hugoniot Conditions

A shock is an irreversible (non-adiabatic, entropy generating) wave that cases a transiton from supersonic to subsonic flow (Fig. III.3.2).

The shock thickness, the actual discontinuity, is of the order of the mean free path of the gas/fluid particles. This is microscopic and much smaller than the scales of gradients in the gas/fluid. For all practical purposes we may assume it to be infinitely thin.

Using the Euler equations as conservation laws, one can derive useful expressions on how key fluid quantities change across a shock. We will work in a frame in which the shock is stationary (see Fig. III.3.2).

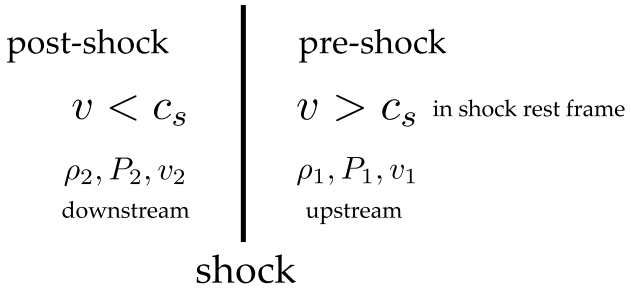

Figure III.3.2: Schematic view of a one-dimensional shock front moving to the right in the laboratory frame. In the shock rest frame, the velocity of the gas is subsonic behind and supersonic in front of the shock.

We must conserve mass across the shock:

$$\boxed{\text{Mass Conservation:} \quad \rho_1 v_1 = \rho_2 v_2 = \underbrace{\mathcal{F}_{\rm mass}}_{\text{Mass Flux}}} \; . \tag{III.3.49}$$

Momentum must also be conserved. We may assume a steady solution (which is OK if we are insterested in a snapshot) and set $\partial v/\partial t = 0$ and assume zero external force. Then

$$\begin{aligned} \rho v \frac{dv}{dx} + \frac{dP}{dx} &= 0 \; , \\ \frac{d}{dx}(P + \rho v^2) &= 0 \; , \end{aligned} \tag{III.3.50}$$

where we have used $(\rho v) = const.$ Integrating over the shock yields:

$$\boxed{\text{Momentum Conservation:} \quad P_1 + \rho_1 v_1^2 = P_2 + \rho_2 v_2^2 = \underbrace{\mathcal{F}_{\rm mom}}_{\text{Momentum Flux}}} \; . \tag{III.3.51}$$

Of course, energy is also conserved and neglecting radiative/conductive losses ($\partial \mathcal{E}/\partial t = 0$), we have

$$\frac{d}{dx}\left[v \left(\frac{1}{2}\rho v^2 + \rho\epsilon + P \right) \right] = 0 \; . \tag{III.3.52}$$

We now specialize to the case of an ideal gas/fluid with $\gamma = 5/3$ (we will go back to the general case at the end). With this

$$\epsilon = \frac{P}{(\gamma - 1)\rho} = \frac{3P}{2\rho} \; . \tag{III.3.53}$$

Using this together with $(\rho v) = const.$, we rewrite (III.3.52):

$$\rho v \frac{d}{dx}\left(\frac{1}{2}v^2 + \frac{5P}{2\rho} \right) = 0 \; . \tag{III.3.54}$$

This we integrate across the shock and obtain:

$$\boxed{\text{Energy Conservation:} \quad \frac{1}{2}v_1^2 + \frac{5}{2}\frac{P_1}{\rho_1} = \frac{1}{2}v_2^2 + \frac{5}{2}\frac{P_2}{\rho_2} = \mathcal{E}} \; , \tag{III.3.55}$$

where $\mathcal{E} = \frac{1}{2}v^2 + \frac{5}{2}\frac{P}{\rho}$ is the total specific energy.

Eqs. (III.3.49), (III.3.51), and III.3.55) are called the Rankine-Hugoniot shock jump conditions.

Let us now define the Mach number

$$\mathcal{M}^2 = \frac{v^2}{c_s^2} \overset{(\gamma=5/3)}{=} \frac{3}{5}\frac{\rho v^2}{P} \ . \tag{III.3.56}$$

Dividing (III.3.51) by (III.3.49) $\times v$:

$$\frac{\mathcal{F}_{\rm mom}}{\mathcal{F}_{\rm mass} v} = \frac{P}{\rho v^2} + 1 = \frac{3}{5\mathcal{M}^2} + 1 \ , \tag{III.3.57}$$

and

$$\mathcal{E} = \frac{1}{2}v^2 + \frac{5}{2}\frac{P}{\rho} = \frac{1}{2}v^2 + \frac{5}{2}\left(\frac{\mathcal{F}_{\rm mom} v}{\mathcal{F}_{\rm mass}} - v^2\right) \ , \tag{III.3.58}$$

or

$$v^2 - \frac{5}{4}\frac{\mathcal{F}_{\rm mom} v}{\mathcal{F}_{\rm mass}} + \frac{1}{2}\mathcal{E} = 0 \ . \tag{III.3.59}$$

The latter is a quadratic equation for v in terms of the conserved quantities $\mathcal{F}_{\rm mass}$, $\mathcal{F}_{\rm mom}$, and $\mathcal{E}$. It has two roots, which correspond to the velocities upstream (v_1) and downstream (v_2) of the shock. Solving Eq. (III.3.59), we find

$$v_1 + v_2 = \frac{5}{4}\frac{\mathcal{F}_{\rm mom}}{\mathcal{F}_{\rm mass}} \ , \tag{III.3.60}$$

and

$$\frac{v_2}{v_1} = \frac{5}{4}\frac{\mathcal{F}_{\rm mom}}{\mathcal{F}_{\rm mass} v_1} - 1 = \frac{5}{4}\left(\frac{3}{5\mathcal{M}_1^2} + 1\right) - 1 \ , \tag{III.3.61}$$

where we have used Eq. (III.3.57) and where $\mathcal{M}_1$ is the upstream Mach number.

In the *strong shock limit*, $\mathcal{M}_1 \gg 1$. In this case, Eq. (III.3.61) results in $v_2/v_1 = 1/4$ and from $\rho v = const.$, we have $\rho_2/\rho_1 = 4$.

These results can be generalized for any value of γ (and independent of Mach number):

$$\frac{\rho_1}{\rho_2} = \frac{(\gamma+1)P_1 + (\gamma-1)P_2}{(\gamma-1)P_1 + (\gamma+1)P_2} \ . \tag{III.3.62}$$

Using the ideal gas law, we can also write out the jump in temperature:

$$\frac{T_2}{T_1} = \frac{P_2\rho_1}{P_1\rho_2} = \frac{P_2}{P_1}\left[\frac{(\gamma+1)P_1 + (\gamma-1)P_2}{(\gamma-1)P_1 + (\gamma+1)P_2}\right] \ . \tag{III.3.63}$$

In terms of the upstream mach number $\mathcal{M}_1$ one can derive additional useful relations:

$$\frac{\rho_2}{\rho_1} = \frac{v_1}{v_2} = \frac{(\gamma+1)\mathcal{M}_1^2}{(\gamma-1)\mathcal{M}_1^2 + 2} \ , \tag{III.3.64}$$

and

$$\frac{P_2}{P_1} = \frac{2\gamma\mathcal{M}_1^2}{\gamma+1} - \frac{\gamma-1}{\gamma+1} \ . \tag{III.3.65}$$

The downstream Mach number $\mathcal{M}_2$ in terms of $\mathcal{M}_1$ is given by

$$\mathcal{M}_2^2 = \frac{2 + (\gamma - 1)\mathcal{M}_1^2}{2\gamma\mathcal{M}_1^2 - (\gamma - 1)} \ . \tag{III.3.66}$$

Because we have a shock, the upstream Mach number $\mathcal{M}_1$ is > 1. By writing out Eq. (III.3.64) as

$$\frac{\rho_2}{\rho_1} = \frac{\gamma + 1}{(\gamma - 1) + 2\mathcal{M}_1^{-2}} \ , \tag{III.3.67}$$

we see that $\rho_2/\rho_1 > 1$ for $\mathcal{M}_1 > 1$ and increases with an increase in $\mathcal{M}_1$. This means that the gas behind the shock is always compressed and that a shock involving stronger compression has to move faster.

References

[1] K. Huang. *Statistical Mechanics*. John Wiley & Sons, Hoboken, NJ, USA, 1987.

[2] D. Mihalas and B. Weibel-Mihalas. *Foundations of Radiation Hydrodynamics*. Dover Publications, Mineola, NY, USA, 1999.

III.4 Smoothed Particle Hydrodynamics

So far, I've only discussed gravitational forces in the N-body technique. This is OK if we're dealing with dark matter and/or stars since these both act as collisionless particles which experience only gravitational forces. But, in general, we want to include gas as well, and so we need to include hydrodynamic forces and, perhaps, things like radiative energy losses etc. You've already discussed hydrodynamical forces and their numerical solution using grid-based methods (i.e. Eulerian methods). We can certainly use those in conjunction with an N-body representation of dark matter for example. But, since we're dealing with particles anyway, it's worth considering if we can use a particle-based approach for hydrodynamical forces also. One advantage of this might be that a particle based approach naturally concentrates particles in dense regions, so we automatically get better resolution in dense regions of our simulation.

The first thing to note in considering this is that hydrodynamical forces involve things like density and pressure gradients. These are quantities which fundamentally can't be specified for a single particle. Instead, they depend on the relation between a particle and other nearby particles. For example, the density in a region clearly depends on how many particles are present in that region. So, we're going to need to consider collections of particles to estimate these quantities. The Smooth Particle Hydrodynamics (SPH) technique does this by smoothing over the local particle distribution in a specific way that we'll examine next. We'll follow the notation of [2] who gives a more detailed review of the SPH technique.

III.4.1 Smoothing Kernels

We begin by realizing that, for any field $F(\mathbf{r})$ (which could be density, pressure etc.), we can write a smoothed version of this field as a convolution

$$F_\mathrm{s}(\mathbf{r}) = \int F(\mathrm{r}) W(\mathbf{r}-\mathbf{r}', h) \mathrm{d}\mathbf{r}', \tag{III.4.1}$$

where $W(\mathbf{r}-\mathbf{r}', h)$ is a smoothing kernel and h is the characteristic length scale of the kernel. In the limit $h \to 0$ the kernel should approach a Dirac delta function in which case $F_\mathrm{s}(\mathbf{r}) = F(\mathrm{r})$. To be useful for our purposes we'll soon see that we want a kernel that is symmetric (i.e. no preferred direction) and is differentialable at least twice (so we can take derivatives of the smoothed field). In the early days of SPH people used Gaussian kernels, but most modern codes use a kernel with "finite support" (i.e. they're zero beyond some finite radius), typically a cubic spline. Writing $W(\mathbf{r}, h) = w(|\mathbf{r}|/2h)$ (so that our kernel is automatically symmetric and scales linearly with h) we can write

$$w(q) = \frac{8}{\pi} \begin{cases} 1 - 6q^2 + 6q^3, & 0 \le q \le \frac{1}{2} \\ 2(1-q)^3, & \frac{1}{2} < q \le 1, \\ 0, & q > 1. \end{cases} \tag{III.4.2}$$

This kernel (which belongs to a broader class of similar kernels) goes to zero beyond $r = 2h$.

In the case of a particle distribution our field $F(\mathbf{r})$ is actually a collection of delta functions. If each particle has a mass m_i and a density ρ_i (we'll worry about how we get this later) then they are associated with a volume $\Delta\mathbf{r})_i \sim m_i/\rho_i$. In this case, we can replace the convolution integral with a sum:

$$F_\mathrm{s}(\mathbf{r}) \approx \sum_j \frac{m_j}{\rho_j} F_j W(\mathbf{r} - \mathbf{r}_j, h). \tag{III.4.3}$$

This is then a smooth, differentialable representation of the field F described by our particles. We should be clear that this is an approximation, and so has some error compared to the true field (i.e. that we would obtain with infinite particles). If the particles are uniformly spaced by a distance d in 1-D, it can be shown that the interpolation is 2nd order accurate if $h = d$. For real distributions in 3-D it's not so simple to prove this, but it seems reasonable that we'd need $h \geq d$ which means $(4\pi/3)2^3 \approx 33$ neighbor particles should be within the non-zero part of the smoothing kernel.

So what about the density? We've already stated that we don't know the density for an individual particle, yet it appears in the above sum. Fortunately, we can use the SPH kernel to evaluate the density. Let $F_i = \rho_i$, then the smoothed density field becomes

$$\rho_{\rm s}(\mathbf{r}) \approx \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h), \tag{III.4.4}$$

which we can estimate knowing only the masses of the particles. The density for each particle is then found from the smoothed density field at the position of the particle.

We are free to choose whatever h we want and, in fact, to make it a function of position. This is advantageous because we can make h small in regions of high density and thereby avoid smoothing away too much information in these regions. There are essentially two choices for computing a suitably position-dependent h. In the "scatter" approach we use a kernel $W[\mathbf{r} - \mathbf{r}_j, h(\mathbf{r})]$ (i.e. h depends on the position $\mathbf{r}$) while in the "gather" approach we use $W[\mathbf{r} - \mathbf{r}_j, h(\mathbf{r}_i)]$ (i.e. h is determined at the position of the particle for which we're computing a smoothed field). The gather approach has the advantage that density of particle i can be determined using only h_i, the smoothing length for that same particle[5]. Using the gather method we have

$$\rho_i \approx \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h_i), \tag{III.4.5}$$

where we've dropped the subscript "s". From this, it becomes clear why kernels with compact support are useful—once we know h_i we need only find those particles within a distance $2h_i$ of particle i in order to find its density. This makes the calculation $\mathcal{O}(N_{\rm ngb}N)$ where N is total particle number and $N_{\rm ngb}$ is the number of neighbors within $2h_i$ (which will be about the same for all particles since we choose h_i based on the distance to these nearest neighbors). The $N_{\rm ngb}$ parameter is a key input to SPH simulations and it should always be checked to be sufficiently large to give converged answers, as ultimately the accuracy of SPH will be limited by the accuracy of the smoothing/interpolation that the kernel approach implies.

Note that we can apply this approach to any field. For example, the velocity field is

$$\mathbf{v}_i = \sum_j \frac{m_j}{\rho_j} \mathbf{v}_j W(\mathbf{r}_i - \mathbf{r}_j, h), \tag{III.4.6}$$

which, since this is now a smooth and continuous field, we can take the derivative of to get the local velocity divergence

$$(\nabla \cdot \mathbf{v})_i = \sum_j \frac{m_j}{\rho_j} \mathbf{v}_j \cdot \nabla_i W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{III.4.7}$$

[5]Technically if we integrate over the smoothed density field when using the gather method we don't get the correct total mass, so this method does not correctly conserve mass. This doesn't matter though since we explicitly conserve mass because mass is tied to particles which are conserved by construction.

Alternatively, we can make use of the identity $\rho\nabla\cdot\mathbf{v} = \nabla(\rho\mathbf{v}) - \mathbf{v}\cdot\nabla\rho$ and estimating the smoothed versions of the two fields on the right separately. This gives

$$(\nabla\cdot\mathbf{v})_i = \frac{1}{\rho_i}\sum_j m_j(\mathbf{v}_j - \mathbf{v}_i)\cdot\nabla_i W(\mathbf{r}_i - \mathbf{r}_j, h), \tag{III.4.8}$$

which turns out to be more accurate in practice and has the useful feature that it goes precisely to zero when all velocities are equal.

III.4.1.1 Variational Derivation

It's possible to derive the SPH equations from a Lagrangian. Why would we do this? It guarantees the various conservation laws hold (since they're explicitly encoded in the Lagrangian) and also ensures that the phase space structure imposed by Hamiltonian dynamics is retained. [1] showed that the Euler equations for inviscid ideal gas flow follow from the Lagrangian

$$L = \int \rho\left(\frac{\mathbf{v}^2}{2} - u\right)\mathrm{d}V. \tag{III.4.9}$$

This idea was first fully exploited by [3] who discretized this Lagrangian in terms of the SPH particles, yielding

$$L_{\mathrm{SPH}} = \sum_i\left(\frac{1}{2}m_i\mathbf{v}_i^2 - m_i u_i\right), \tag{III.4.10}$$

where u_i is the thermal energy per unit mass of the particle. For a gas with adiabatic index γ we can write

$$P_i = A_i\rho_i^\gamma = (\gamma-1)\rho_i u_i, \tag{III.4.11}$$

where we'll refer to A_i as the "entropy". More specifically, A_i is uniquely determined by the specific entropy of the particle and so is conserved in adiabatic flow. So, assuming A_i to be constant we can write

$$u_i = A_i\frac{\rho^{\gamma-1}}{\gamma-1}. \tag{III.4.12}$$

We then use the standard variational approach to get our equations of motion:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\mathbf{r}}_j} - \frac{\partial L}{\partial\mathbf{r}_j} = 0. \tag{III.4.13}$$

Some care is needed because the smoothing scale, h, varies from particle to particle (and so we can represent it too as a smooth field using our standard kernel). But, the bottom line is

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\sum_{j=1}^N m_j\left[f_i\frac{P_i}{\rho_i^2}\nabla_i W_{ij}(h_i) + f_j\frac{P_j}{\rho_j^2}\nabla_i W_{ij}(h_j)\right], \tag{III.4.14}$$

where

$$f_i = \left[1 + \frac{h_i}{3\rho_i}\frac{\partial\rho_i}{\partial h_i}\right]^{-1}, \tag{III.4.15}$$

and $W_{ij}(h) \equiv W(|\mathbf{r}_i - \mathbf{r}_j|, h)$. What's nice about this is that we've converted the set of partial differential equations that described inviscid flow of an ideal gas into a much simpler set of ordinary

differential equations. Mass and energy conservation are handled automatically (mass because we have a fixed number of particles and energy because our flow is adiabatic so the energy is determined by the (constant) entropy).

It's also possible to derive relativistic implementations of SPH using this variational principle approach.

III.4.2 Other Issues

III.4.2.1 Artificial Viscosity

In all of the above we've explicitly assumed that the specific entropy of a particle is conserved, i.e. that the flow is adiabatic. However, even if starting from smooth initial conditions the Euler equations can lead to shocks and contact doscontinuities at which the differential form of the Euler equations breaks down and their integral form (conservation laws) must be applied instead. Analysis shows that in shocks the entropy is always increased. How can we introduce this behavior into our SPH calculation? The usual approach is to introduce an artificial viscosity term. The role of this artificial viscosity is to broaden the shocks into a resolvable layer and dissipate kinetic energy into heat, thereby increasing the entropy. If introduced as a conservative force then the conservation laws implicity in the Euler equations guarantee that the correct amount of entropy is generated, independent of the details of the viscosity used. Of course, the problem with this is that shocks are now not perfect thin—instead they will be resolved by a few smoothing lengths.

The viscous forces is usually added in the form

$$\left.\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t}\right|_{\mathrm{visc}} = -\sum_{j=1}^{N} m_j \prod_{ij} \nabla_i \overline{W}_{ij}, \tag{III.4.16}$$

where

$$\overline{W}_{ij} = \frac{1}{2}[W_{ij}(h_i) + W_{ij}(h_j)] \tag{III.4.17}$$

is a symmetrized kernel. Provided that $\prod_{ij}$ is symmetric in i and j this viscous force is antisymmetric between pairs of particles so always conserves linear and angular momentum. It does not conserve energy (that's why we're including it) so a compensating change in the internal energy (or, equivalently, entropy) is introduced to balance this. We then have the desired conversion of kinetic to thermal energy and a corresponding entropy production.

A common form used for the viscosity factor is

$$\prod_{ij} = \begin{cases} [-\alpha c_{ij}\mu_{ij} + \beta\mu_{ij}^2]/\rho_{ij} & \text{if } \mathbf{v}_{ij}\cdot\mathbf{r}_{ij} < 0 \\ 0 & \text{otherwise,} \end{cases} \tag{III.4.18}$$

with

$$\mu_{ij} = \frac{h_{ij}\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^2 + \epsilon h_{ij}^2}. \tag{III.4.19}$$

Here, h_{ij} and ρ_{ij} are arithmetic means of the corresponding quantities for particles i and j, and c_{ij} being the mean sound speed. The strength of the viscosity is controlled by parameters α and β. The goal is to make this viscosity significant in shocks but weak elsewhere (so that we don't dissipate kinetic energy in regimes that are evolving adiabatically). Typical values of $\alpha \approx 0.5$–1.0 and $\beta = 2\alpha$ are used. The parameter $\epsilon \approx 0.01$ is introduced to avoid a singularity if two particles get very close.

III.4.2.2 Self-Gravity

Our SPH gas particles feel the force of gravity and, in particular, there is a self-gravity between the SPH particles. We can make use of standard N-body techniques to compute the gravitational forces, but there's also a neat way to do this using the variational approach discussed above. We simply add the gravitational self-energy of the SPH particles:

$$E_{\rm pot} = \frac{1}{2}\sum_i m_i \Phi(\mathbf{r}_i) = \frac{\rm G}{2}\sum_{ij} m_i m_j \phi(r_{ij}, \epsilon_j) \tag{III.4.20}$$

to the SPH Lagrangian giving

$$L_{\rm SPH} = \sum_i \left(\frac{1}{2} m_i \mathbf{v}_i^2 - m_i u_i\right) - \frac{\rm G}{2}\sum_{ij} m_i m_j \phi(r_{ij}, \epsilon_j). \tag{III.4.21}$$

When we apply the variational principle the equations of motion pick up extra terms due to gravity which look like:

$$\begin{aligned} m_i \mathbf{a}_i^{\rm grav} &= -\frac{\partial E_{\rm pot}}{\partial \mathbf{r}_i} \\ &= -\sum_j {\rm G} m_i m_j \frac{\mathbf{r}_{ij}}{r_{ij}} \frac{[\phi'(r_{ij}, \epsilon_i) + \phi'(r_{ij}, \epsilon_j)]}{2} \\ &\quad -\frac{1}{2}\sum_{jk} {\rm G} m_j m_k \frac{\partial \phi(r_{jk}, \epsilon_j)}{\partial \epsilon} \frac{\partial \epsilon_j}{\partial \mathbf{r}_i} \end{aligned} \tag{III.4.22}$$

The first time here is what you'd guess—it's a symmetrized gravitational interaction. The second term shows up if the softening length, ϵ, is allowed to be different for each particle (and so varies with position), and is required to make the interaction conservative in such cases.

III.4.2.3 Time Steps

One final issue: What time steps should we use when evolving the SPH particles? The usual approach is to impose a so called Courant time step criterion which requires:

$$\Delta t_i = C_{\rm CFL} \frac{h_i}{c_i} \tag{III.4.23}$$

where c_i is the sound speed and $C_{\rm CFL} \sim 0.1$–0.3. This limits each particle to moving significantly less than one smoothing length per timestep providing the particle is moving subsonically. This does not work so well in cases such as a blast wave propagating into cold gas (for which c_i will be small, allowing large timesteps which can result in cold gas particles moving through the blast wave). Improved algorithms attempt to catch such cases. This timestep is usually augmented by checking the gravitational timestep (and taking the minimum of the two) using the usual N-body techniques. In fact, in an N-body tree code the tree structure turns out to be extremely useful for one of the key SPH operations, namely finding the nearest neighbors. The cost of the operation can be reduced from $\mathcal{O}(N^2)$ for a simple search of all particles to $\mathcal{O}(N_{\rm ngb} N \ln N)$ when using the tree.

References

[1] Carl Eckart. Variation principles of hydrodynamics. *Physics of Fluids*, 3(3):421, 1960. ISSN 00319171. doi: 10.1063/1.1706053. URL http://link.aip.org/link/PFLDAS/v3/i3/p421/s1%26Agg=doi.

[2] Volker Springel. Smoothed particle hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48:391–430, September 2010. URL http://adsabs.harvard.edu/abs/2010ARA%26A..48..391S.

[3] Volker Springel and Lars Hernquist. Cosmological smoothed particle hydrodynamics simulations: the entropy equation. *Monthly Notices of the Royal Astronomical Society*, 333:649–664, July 2002. URL http://adsabs.harvard.edu/abs/2002MNRAS.333..649S.

III.5 Hydrodynamics III – Grid Based Hydrodynamics

In this section, we will get our feet wet in classical Newtonian hydrodynamics by tackling the Riemann problem, whose exact solution we will compute. We will then go on to grid-based hydrodynamics in its flux-conservative (= mass, momentum, energy are conserved up to source terms, e.g. external forces) formulation and work our way through the details of a building a 1D (planar) finite-volume hydrodynamics code.

III.5.1 Characteristics

In §III.3.4.3, we have already talked about the hyperbolicity of the Euler equations. Here we are coming back briefly to this, because we will need the *eigenstructure* (eigenvalues and eigenvectors) of the Euler equations.

III.5.1.1 Characteristics: The Linear Advection Equation and its Riemann Problem

Before going to the non-linear Euler equations, let's consider briefly the linear advection equation,

$$\frac{\partial}{\partial t}u + v\frac{\partial}{\partial x}u = 0 \ . \tag{III.5.1}$$

This is a linear conservation law and its general solution is given by $u(x,t) = u(x - vt, t = 0)$. The *characteristics* (also called *characteristic curves* of a PDE such as Eq. (III.5.1) are defined as curves $x = x(t)$ in the (t, x) plane along which the PDE becomes an ODE (see [5] for a full discussion; the following is an abridged version of what is in [5], §2.2).

Consider $x = (t)$ and regard u as a function of t, that is $u(x,t) = u(x(t), t)$. u changes along $x(t)$ according to

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{dx}{dt}\frac{\partial u}{\partial x} \ . \tag{III.5.2}$$

If the characteristic curve $x = x(t)$ satisfies

$$\frac{dx}{dt} = v \ , \tag{III.5.3}$$

then (III.5.1) together with (III.5.3) and (III.5.3) gives

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0 \ . \tag{III.5.4}$$

This means that u is constant along *any* characteristic curve $x = x(t)$ satisfying (III.5.2) with *characteristic speed* v, the slope of the characteristic curve $x = x(t)$ in the (t, x) plane. It is often more convenient to look at the (x, t) plane in which the slope of the characteristic $x = x(t)$ will, of course, be $1/v$ (Fig. III.5.1).

So the linear advection equation has a *family* of characteristic curves that is a *one-parameter family* that is completely determined by the initial position x_0 at $t = 0$. Then the characteristic curve passes through the point $(x_0, 0)$ and is completely determined by $x(t) = x_0 + vt$ and characteristic curves with different x_0 are parallel to each other (a feature of linear PDEs with constant coefficients).

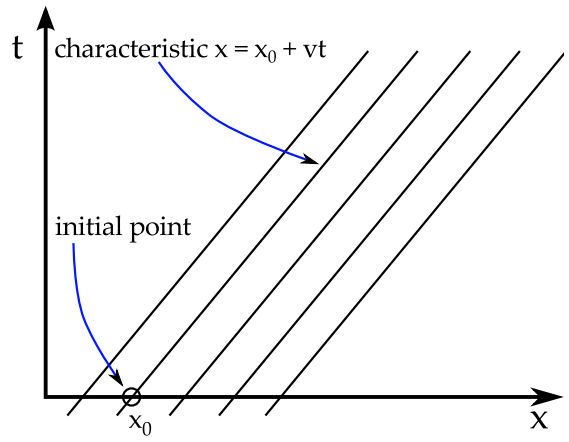

Figure III.5.1: Characteristic curves of the linear advection equation for positive characteristic speed v. The initial condition at $t = 0$ fixes the position x_0.

Eq. (III.5.3) shows that u remains constant along characteristic curves. So if the initial condition $u_0(x) = u(x, t = 0)$ is given, we know that it will stay constant along any characteristic curve $x = x_0 + vt$ and the solution for $u(x, t)$ along this curve at all times t is

$$u(x,t) = u_0(x_0) = u_0(x - vt) \ . \tag{III.5.5}$$

We can formulate and solve the so-called *Riemann problem* for the linear advection equation:

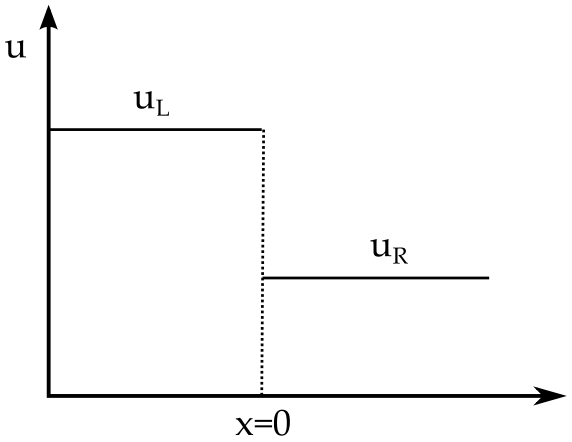

Figure III.5.2: Illustration of the initial data for the Riemann problem of the linear advection equation.

The initial conditions for this problem are such that at $t = 0$,

$$u(x, t = 0) = \begin{cases} u_L & \text{if } x < 0 \ , \\ u_R & \text{if } x > 0 \ , \end{cases} \tag{III.5.6}$$

Based on the just discussed solution, we expect any point on the initial profile to propagate a distance $d = vt$ in time t. In particular, we expect the initial discontinuity at $x = 0$ a distance $d = vt$ in time t. The particular characteristic curve $x = vt$ will then cleanly separate all characteristic curves to the left, on which the solution takes on the value u_L, from all characteristic curves to the right, on which the solution takes on the value u_R. This is visualized by Fig. III.5.3.

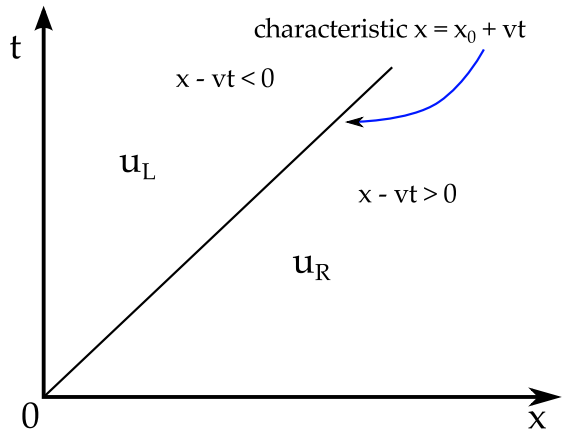

Figure III.5.3: Characteristic view of the Riemann problem for the linear advection equation. The characteristic curve passing through the initial discontinuity at $x = 0$ separates the evolution of the left and right states.

III.5.1.2 Eigenstructure of the Euler Equations

Things are a bit more complicated with the Euler Equations. This is due primarily to their non-linear nature. Here, we work out the eigenvalues and eigenvectors of the Euler equations and then apply the results to the Riemann problem in §III.5.2.

In the following, we will always assume that our problem is 1D planar and that there are no external forces. We will work in the Eulerian frame. The Euler Equations that we have first stated in §III.3 are then

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho v) = 0 \,, \tag{III.5.7}$$

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho v^2 + P) = 0 \,, \tag{III.5.8}$$

$$\frac{\partial}{\partial t}\mathcal{E} + \frac{\partial}{\partial x}((\mathcal{E} + P)v) = 0 \,, \tag{III.5.9}$$

where $\mathcal{E} = \rho\epsilon + \frac{1}{2}\rho v^2$. We can recast them into the simple form

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F} = \mathbf{S} \,, \tag{III.5.10}$$

where the state vector $\mathbf{U}$ is

$$\mathbf{U} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} \rho \\ \rho v \\ \rho\epsilon + \frac{1}{2}\rho v^2 \end{pmatrix} \,, \tag{III.5.11}$$

and the flux vector $\mathbf{F}$ is given by

$$\mathbf{F} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} \rho v \\ \rho v v + P \\ (\rho\epsilon + \frac{1}{2}\rho v^2 + P)v \end{pmatrix} \,. \tag{III.5.12}$$

We can re-write this system in quasi-linear form (as we did already in §III.3.4.3 by introducing the Jacobian

$$\bar{A} = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} = \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \frac{\partial f_1}{\partial u_3} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \frac{\partial f_2}{\partial u_3} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} & \frac{\partial f_3}{\partial u_3} \end{pmatrix} , \tag{III.5.13}$$

and writing

$$\frac{\partial}{\partial t}\mathbf{U} + \bar{A}\frac{\partial}{\partial x}\mathbf{U} = 0 . \tag{III.5.14}$$

In analogy with the linear advection equation (III.5.1) where v was the characteristic speed, $\bar{A}$ contains the (three) characteristic speeds of the Euler equations as its eigenvalues. The characteristic curves along which the Euler equations reduce to PDEs (they are not constant along these curves, because they are non-linear) are given by the eigenvectors of $\bar{A}$.

The eigenvalues and eigenvectors of $\bar{A}$ are computed using standard linear algebra:

Eigenvalues: λ_i

$$\det(\bar{A} - \lambda_i \mathbf{I}) = 0 , \tag{III.5.15}$$

and

Right Eigenvectors: $\mathbf{K}_i$

$$\bar{A}\mathbf{K}_i = \lambda_i \mathbf{K}_i . \tag{III.5.16}$$

The Euler equations are hyperbolic, so their eigenvalues are all real and the eigenvectors are a complete set of linearly independent vectors. Computing the eigenvalues and eigenvectors of the Euler equations is a bit tedious and we will just state the end results here:

$$\lambda_1 = v - c_s , \qquad \lambda_2 = v , \qquad \lambda_3 = v + c_s , \tag{III.5.17}$$

where c_s is the adiabatic sound speed, and

$$\mathbf{K}_1 = \begin{pmatrix} 1 \\ v - c_s \\ H - v c_s \end{pmatrix} , \qquad \mathbf{K}_2 = \begin{pmatrix} 1 \\ v \\ \frac{1}{2}v^2 \end{pmatrix} , \qquad \mathbf{K}_3 = \begin{pmatrix} 1 \\ v + c_s \\ H + v c_s \end{pmatrix} , \tag{III.5.18}$$

where $H = \frac{1}{2}v^2 + \epsilon + P/\rho$.

III.5.2 The Riemann Problem for the Euler Equations

The Riemann problem for the Euler equations is crucial to modern numerical methods for grid-based hydrodynamics. This is for two reasons: (1) The Riemann problem can be solved *exactly* by analytical means (involving Newton iterations to find a root) and thus can be used as a way of testing numerical implementations of the Euler equations. (2) The solution of the Riemann problem underlies the widely used *Godunov method* for numerical hydrodynamics, which we will discuss in §III.5.3.

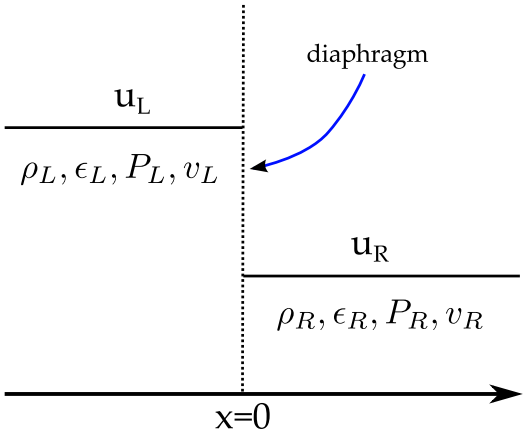

Figure III.5.4: Initial conditions for the Riemann problem for the Euler Equations. v_L and v_R are initially set to zero.

Detailed discussions of the Riemann problem including its complete and *exact* solution can be found in [2, 5]. Here we will only be able to outline its most salient aspects and list results of lengthy calculations whose details can be found in [5].

The initial conditions for the Riemann problem for the Euler Equations are very similar to the one for the linear advection equation. At $t = 0$ a diaphragm separates two regions with different values of the state variables. We shall call these left (L) and right (R) states. Initially, the velocity is zero. Let's consider the case in which the L state is a hot dense gas and the right state is a cooler, lower density gas. When the diaphragm is removed, the pressure difference between L and R states will push gas from left to right and a right-moving shock develops. This is why the Riemann problem is also referred to as the *shocktube* problem or as the *Sod shocktube problem* (because it was Sod who studied a particular Riemann problem in detail). The initial conditions for the Riemann problem are shown in Fig. III.5.4.

At $t = 0$, the diaphragm is removed and taking a snapshot at some time $t > 0$, a situation as shown in Fig. III.5.5 presents itself. We identify five distinct regions, whose development has been experimentally verified:

(1) Unchanged left (L) fluid.

(2) *Rarefaction*, expanding left fluid, rarefaction wave propagates to the left, but fluid moves to the right.

(3) Decompressed left fluid. Connected to (4) by a contact discontinuity ($P = const.$, $v = const.$)

(4) Compressed right fluid. Connected to (5) by a shock (everything discontinuous).

(5) Unchanged right (R) fluid.

The five regions are separated by characteristic waves, i.e., the characteristics of the Euler equations discussed in §III.5.1.2. Figure III.5.6 shows a schematic view of the characteristics of the Euler equations in the Riemann problem. The first eigenvalue, $\lambda_1 = v - c_s$, and its eigenvector $\mathbf{K}_1$ correspond to backward traveling waves that result in the observed rarefaction. The second eigenvalue, $\lambda_2 = v$, and its eigenvector $\mathbf{K}_2$ result in a contact discontinuity ($P = const.$, $v = const.$). The third eigenvalue, $\lambda_3 = v + c_s$, and its eigenvector $\mathbf{K}_3$ result in a shock wave.

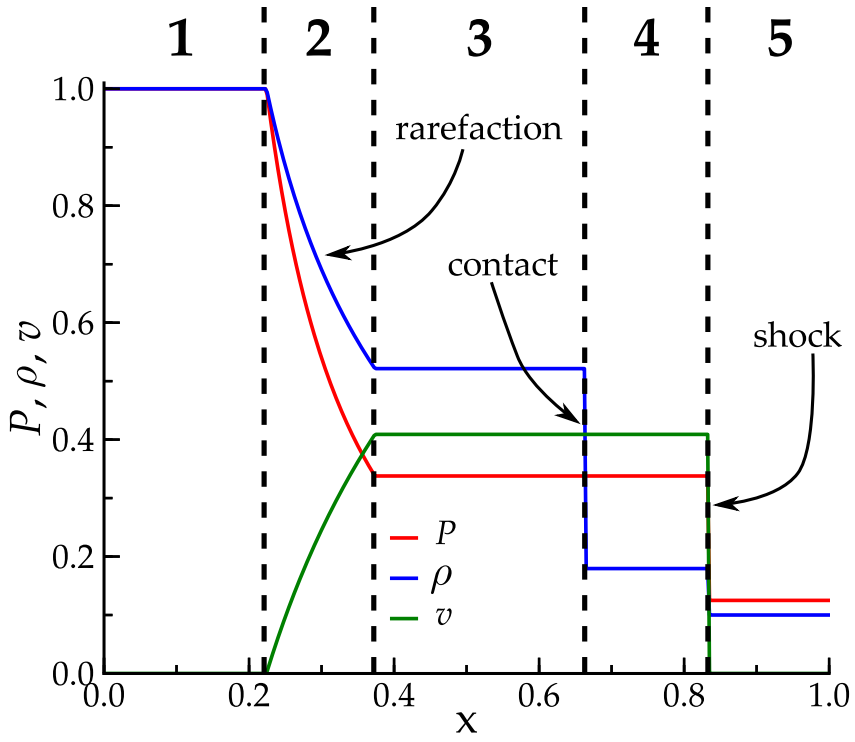

Figure III.5.5: Exact solution of a Sod shocktube Riemann problem for $\Gamma = 5/3$ at $t = 0.4$. The initial diaphragm was placed at $x_0 = 0.5$. Regions 1 and 5 contain the untouched initial left and right states, respectively. Region 4 and 5 are connected by the shock, region 3 and 4 are connected by a contact discontinuity, region 2 has a family of rarefaction waves. Note that we have used the special-relativistic solution of Martí & Müller [3], which uses units of the speed of light to measure velocities. Note: This solution will be replaced in a future version of these notes. It is qualitatively fine, but makes quantitative comparisons difficult.

In the following, we will a bit more closely look at rarefaction, contact, and shock. The discussion that we present is heavily inspired by [1, 2, 5] and readers interested in full derivations should check in particular [5], which has the most extended and clear discussion.

III.5.2.1 Rarefaction

The rarefaction wave connects the known state in region 1 with the unknown state 3 behind the contact and ρ, v, and P (and thus also ϵ) change across a rarefaction. Entropy is constant across a rarefaction, so we can work with the assumption of isentropic flow and use a polytropic EOS, $P = K\rho^\gamma$. In the present case, shock and contact are moving to the right, while the rarefaction wave is moving to the left. The rarefaction *head* (the part that is furthest left) is associated with the characteristic associated $\lambda^{HL} = v_1 - c_{s_1}$ and the rarefaction *tail* is associated with $\lambda^{TL} = v_3 - c_{s_3}$. These two characteristic curves enclose the *rarefaction fan*. So, since $v_1 = 0$, the rarefaction head moves to the left with velocity $-c_{s_1}$ (which is a constant) and the rarefaction tail moves to the left with velocity $v_3 - c_{s_3}$ (which is also a constant).

One can show [1, 5] that across a rarefaction, the so-called *Riemann Invariant of Flow* stays

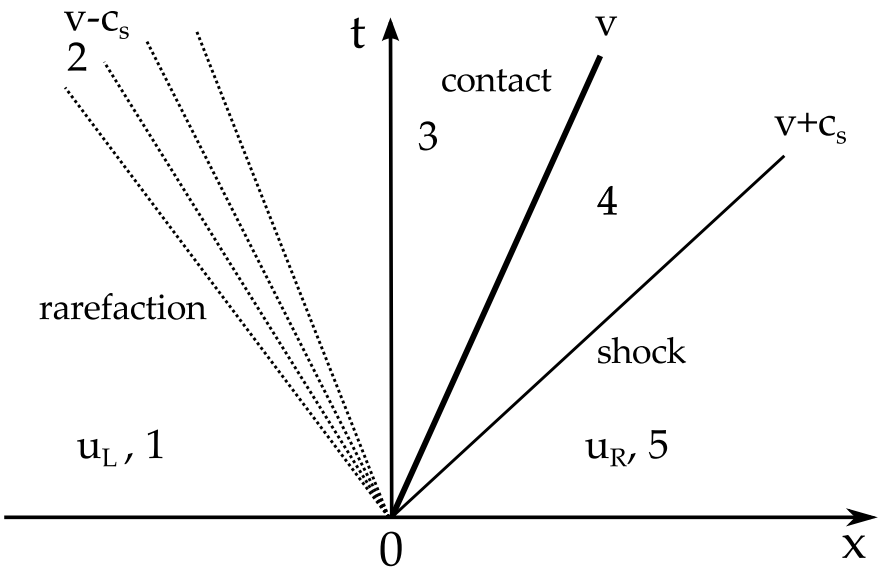

Figure III.5.6: Characteristic wave diagram for the Riemann problem for the Euler equations. For each characteristic wave, the corresponding eigenvalue is marked. We have also marked the various regions identified in Fig. III.5.5.

constant:

$$I_L(v, c_s) = v + \frac{2c_s}{\gamma - 1} \,. \tag{III.5.19}$$

With this and the assumption that all state variables are single valued and can be expressed in terms of each other, one derives the state inside the rarefaction,

$$\begin{aligned} v_2(x,t) &= \frac{2}{\gamma+1}(c_{s_1} + \frac{x-x_0}{t}) \,, \\ \rho_2(x,t) &= \rho_1 \left[\frac{2}{\gamma+1} - \frac{\gamma-1}{(\gamma+1)c_{s_1}}\frac{x-x_0}{t}\right]^{\frac{2}{\gamma-1}} \,, \\ P_2(x,t) &= P_1 \left[\frac{2}{\gamma+1} - \frac{\gamma-1}{(\gamma+1)c_{s_1}}\frac{x-x_0}{t}\right]^{\frac{2\gamma}{\gamma-1}} \,. \end{aligned} \tag{III.5.20}$$

III.5.2.2 Shock and Contact

We can use the Rankine-Hugoniot conditions laid out in §III.3.6 to infer information about the shock wave connecting region 4 with the unshocked state 5. The shock travels with a laboratory-frame velocity

$$v_s = c_{s,5}\left[\frac{\gamma+1}{2\gamma}\frac{P_4}{P_5} + \frac{\gamma-1}{2\gamma}\right]^{\frac{1}{2}} \,, \tag{III.5.21}$$

where we have used P_4, which we will determine in a second. First, let us write down (derivation via Rankine-Hugoniot) the velocity of the shocked fluid in region 4 behind the shock,

$$v_4 = v_s\frac{\rho_4 - \rho_5}{\rho_4} \overset{\text{(some algebra)}}{=} (P_4 - P_5)\left[\frac{1 - \frac{\gamma-1}{\gamma+1}}{\rho_5(P_4 + \frac{\gamma-1}{\gamma+1}P_5)}\right]^{\frac{1}{2}} \,. \tag{III.5.22}$$

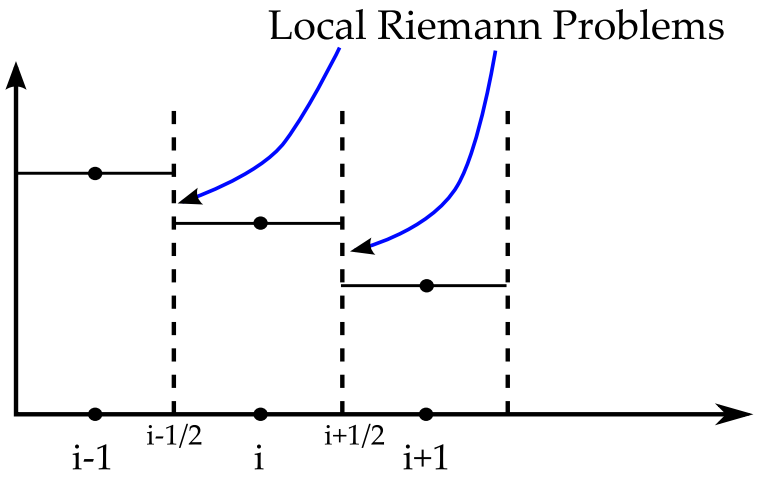

Figure III.5.7: Schematic view of discretized data. Godunov realized that one could solve local Riemann problems to determine the flow between grid cells and in this way to compute the time update of the overall solution.

Now, accross the contact discontinuity between regions 4 and 3, both pressure and velocity remain constant. So we have $P_4 = P_3$ and $v_3 = v_4$. Furthermore, at the tail of the rarefaction, we have $P_2(\text{RF tail}) = P_3$ and $v_2(\text{RF tail}) = v_3$ (see Fig. III.5.5). To proceed, we use a version of v_2 from Eq. (III.5.20) that has been rewritten to remove the explicit dependence on position [2, 5],

$$v_2 = (P_1^{\frac{\gamma-1}{2\gamma}} - P_2^{\frac{\gamma-1}{2\gamma}}) \left[\left(1 - \left(\frac{\gamma-1}{\gamma+1} \right)^2 \right) \frac{P_1^{\frac{1}{\gamma}}}{\left(\frac{\gamma-1}{\gamma+1} \right)^2 \rho_1} \right]^{\frac{1}{2}} . \tag{III.5.23}$$

Eqs. (III.5.22) and (III.5.23) represent two curves in the (P, v) plane. They intersect at the tail of the rarefaction, where regions 2 and 3 meet. Setting the two expressions equal, one can (via Newton iteration) find P_3 and thus P_4 and thus obtains the full solution.

Finally, the contact discontinuity propagates with $v_3 = v_4$, so its position at any time t is given by

$$x_{\text{CD}} = x_0 + v_3 t \ . \tag{III.5.24}$$

An implementation of the exact solution of the Riemann problem as described here can be found, for example, on Frank Timmes's webpage [4]. For relativistic flow, an exact solution of the Riemann problem is still possible and described in [3], which also contains an implementation of the solution.

III.5.3 The Godunov Method and Finite-Volume Schemes

The nature of the Euler equations leads to weak solutions, that is solutions that satisfy the integral form of the Euler equations (i.e., conserve mass, momentum, and energy), but have discontinuities in the differential representation. This makes it difficult to discretize the Euler equations, since weak solutions can arise from perfectly smooth flow and lead to oscillatory behavior that must be controlled.

Von Neumann and Richtmyer came up with the concept of *artificial viscosity* [6] to deal with shocks and oscillations in numerical simulations. We have already mentioned artificial viscosity in the context of SPH in §III.4.2.1. This method "smears out" the shock over multiple cells, in this way avoiding numerical instability, but sacrificing resolution.

In 1959, Godunov had a groundbreaking idea. Numerical data are by their very nature piecewise constant in the interior of a computational cell, but discontinuous at cell boundaries. A schematic view of this is given by Fig. III.5.7. Godunov realized that by the very discretization of the domain one is faced with a sequence of local Riemann problems that, in principle, can be solved exactly! Most modern hydrodynamics codes make use of schemes going back to Godunov's idea.

One class of methods for numerical hydrodynamics making use of Godunov's idea are the so-called *finite-volume* schemes. These are fully conservative schemes (in the absence of source terms, i.e. external forces acting on the fluid), which means they conserve mass, momentum, and energy by construction.

Let us consider a quantity q. In computational cell i with center coordinate x_i and interfaces $x_{i-1/2}$ and $x_{i+1/2}$, the *cell average* of q is given by

$$\bar{q}_i = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} q(x)dx \ , \tag{III.5.25}$$

where $\Delta x_i = x_{i+1/2} - x_{i+1/2}$. Now the change of q be governed by

$$\frac{\partial}{\partial t}q + \frac{\partial}{\partial x}f(q) = 0 \ . \tag{III.5.26}$$

Then the change of q from t_1 to t_2 is

$$q(x,t_2) = q(x,t_1) - \int_{t_1}^{t_2}\frac{\partial}{\partial x}f(q(x,t))dt \ . \tag{III.5.27}$$

Analoglously, the change of the cell average is

$$\bar{q}_i(t_2) = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}}\left\{q(x,t_1) - \int_{t_1}^{t_2}\frac{\partial}{\partial x}f(q(x,t))dt\right\}dx \ . \tag{III.5.28}$$

Provided the flux f is well behaved, the integration order may be reversed. Also, the flow is perpendicular to the unit area of the cell and, in 1D, $\partial f/\partial x \equiv \nabla f$, so we may use Gauss's law,

$$\int_V \nabla f d^3x = \int_{\partial V} f dS \ , \tag{III.5.29}$$

to substitute the "volume" integral of $\partial f/\partial x$ with the values of f at the "surface":

$$\bar{q}_i(t_2) = \bar{q}_i(t_1) - \frac{1}{\Delta x_i}\int_{t_1}^{t_2}\underbrace{\left[f(q(x_{i+1/2},t)) - f(q(x_{i-1/2},t))\right]}_{\text{"Flux Difference"}}dt \ . \tag{III.5.30}$$

This equation is *exact* for the cell average II_i (no approximation has been made). This is straighforwardly extended to the multi-D case by integrating over each direction separately and adding up the changes, which still yields an exact results for regular grids.

There are many ways of implementing finite-volume/Godunov schemes. One of the simplest is to follow the semi-discrete approach, discretize in space only and then use the Method of Lines (see §II.9.3.6) to treat the semi-discretized equation as an ODE and integrate it with standard integrator like Runge-Kutta.

In the semi-discrete case, the spatial order of accuracy of the scheme is set by the accuracy with which the states q at the cell interfaces $x_{i+1/2}$ and $x_{i-1/2}$ are "reconstructed". The temporal order of accuracy is set by the order of accuracy in which the time integral on the RHS of Eq. III.5.30 is handled. All the art is in computing the *flux differences* using either the exact or approximate solutions of local Riemann problems.

III.5.4 Anatomy of a 1D Finite-Volume Hydro Code

Available on Monday 02/20.

III.5.4.1 Spherical Symmetry

Available on Monday 02/20.

References

[1] R. Courant and K. O. Friedrichs. *Supersonic Flow and Shock Waves*. Springer Verlag, New York, NY, USA, 1976.

[2] J. F. Hawley, L. L. Smarr, and J. R. Wilson. A numerical study of nonspherical black hole accretion. I Equations and test problems. *Astrophys. J.*, 277:296, February 1984. doi: 10.1086/161696.

[3] J. M. Martí and E. Müller. Numerical Hydrodynamics in Special Relativity. *Liv. Rev. Rel.*, 6:7, December 2003.

[4] F. Timmes. An exact solution to the riemann problem (source code). URL `http://cococubed.asu.edu/code_pages/exact_riemann.shtml`.

[5] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, Berlin, Germany, 1999.

[6] J. von Neumann and R. D. Richtmyer. *J. Appl. Phys.*, 21:232, 1950.