Boundary conditions

At the grid boundaries, finite-difference and finite-vlume algoirthms need values for mesh points that are not on the computational grid. For example, in the first zone of the grid, we need to define the left state of the Riemann problem when we're using Godunov's method.
In a numerical code, one can either switch to a different finite-difference operator at the boundaries (that is chosen to respect the *mathematical* boundary conditions) or use so called *ghost zones* that are populated with values by copying/reflecting the values of zones inside the computational domain. Common cases include:

- Periodic boundary conditions
- Reflecting (symmetric/antisymmetric) boundary conditions
- Fixed boundary conditions

Periodic boundary conditions are often particularly easy to implement if a programming language offers a command for a cyclic permutation of an array.

Sometimes, more sophisticated boundary conditions are needed (hydrostatic, characteristic boundary conditions).

Conservative and Primitive Variables

When we're using a conservative finite-volume scheme, we evolve the *conserved variables*

$$(U_1, U_2, U_3, U_4, U_5) = (\rho, \rho v_x, \rho v_y, \rho v_z, \rho(\epsilon + v^2/2)),$$

in time using (in 1D) equations of the form

$$\frac{\partial}{\partial t} \int U \,\mathrm{d}x + \Delta t[F_L - F_R] = 0.$$

However, to calculate the fluxes (and other quantities like c_s that we need to solve the Riemann problem), we need the *primitive variables* ρ, $\mathbf{v}$ and ϵ from which we can compute P, c_s, wave speeds, etc. using the equation of state. Therefore we need to convert back from U to the primitives after updating the conserved variables:

$$\rho = U_1, v_x = \frac{U_2}{U_1}, v_y = \frac{U_3}{U_1}, v_z = \frac{U_4}{U_1}, \epsilon = \left(\frac{U_5}{U_1}\right) - \frac{v^2}{2}.$$

Setting up the initial conditions typically involves setting the primitive variables, so we also need to convert from primitives to conserved variables at the beginning.

Higher-Order Reconstruction

- Godunov's method turns out to be very dissipative, i.e. it considerably smears out the solution.
- This is due to the fact that it is only accurate to 1st order.
- **Can you verify this? Hint: Consider the advection equation and show that Godunov's scheme reduces to the 1st order upwind scheme.**
- The assumption of step functions in the reconstruction step is to blame.

Reconstruct-Solve-Average Algorithm – Piecewise Linear Reconstruction

- Higher-order accuracy can be achieved by using *non-constant* functions for the state vector **U** within a cell.
- For example, we can approximate the function u in terms of its cell average $\bar{u}$ and a centred approximation $\partial u/\partial x \approx (u_{i+1} - u_{i-1})/(2\Delta x)$ for the slope:

$$
\begin{aligned}
u(x) &= \bar{u} + \frac{\partial u}{\partial x} + (x - x_i) \\
u(x) &= \bar{u} + \frac{u_{i+1} - u_{i-1}}{2\Delta x} \times (x - x_i)
\end{aligned}
$$

This leads to *Fromm's method*.

Generalised Riemann Problem

- If we naively implement this for the advection equation, by choosing the reconstructed function on the upwind side of the cell interface to compute fluxes,

$$u_i^{n+1} = u_i^n + \left[F(u^n_{L,i-1/2}) - F(u^n_{R,i-1/2})\right] ,$$

  we run into trouble already for the advection equation. The solution is again unstable.
- The problem is that we now face a *generalised Riemann problem* – u is not constant on both side of the interface.

Generalised Riemann Problem

For the advection equation, the solution is simple: Knowing the characteristics, we can compute how the interface flux changes during a time step and use a time-averaged flux:

$$\begin{aligned} F_{i+1/2} &= \frac{1}{\Delta t}\int_0^{\Delta t} F(u(x_{i+1/2}-\lambda t))\,\mathrm{d}t \\ &= \frac{1}{\Delta t}\int_0^{\Delta t} u^n_{L,i+1/2} - v\left(\frac{\partial u}{\partial x}\right)_i t\,\mathrm{d}t \\ &= u^n_{L,i+1/2} - \frac{\Delta t}{2} v\left(\frac{\partial u}{\partial x}\right)_i . \end{aligned}$$

This suggests one possibility to deal with the generalised Riemann problem: Advance the left and right state by $\Delta t/2$ using a characteristic solution, and then simply using the unmodified Riemann solver gives an estimate for the flux that is second-order accurate in time. In general, this gives a stable scheme. Another possibility is to use the so-called method of lines and higher-order time integration (we'll come to that).

Examples for 2nd-Order Schemes

- Using centered, upwind, and downwind differences for the slope result in stable 2nd order schemes: Fromm's method, Beam-Warming, and Lax-Wendroff.
- The Lax-Wendroff method can also be interpreted as a symmetric method where we get the fluxes by advancing the interpolated state at cell interfaces $U_i^n + U_{i+1/2}$ by half a time-step and then compute the interface fluxes from this:

$$\begin{aligned} U_{i+1/2}^{n+1/2} &= \frac{U_i^n + U_{i+1/2}}{2} + \frac{\Delta t}{2} \frac{F(U_i^n) - F(U_{i+1}^n)}{\Delta x} \\ U_i^{n+1} &= U_i^n + \Delta t \frac{F(U_{i-1/2}^n) - F(U_{i+1/2}^n)}{\Delta x} \end{aligned}$$

- These schemes produce oscillations (wiggles) at discontinuities.
- The reason for this is that the reconstructed function overshoots too much at the discontinuities.
- *Monotone* schemes avoid this by using *slope limiters*.

Slope Limiters – Minmod

The minmod slope is computed from the left and right approximation for the slope $s_L = (u_i - u_{i-1})/(x_i - x_{i-1})$ and $s_R = (u_{i+1} - u_i)/(x_{i+1} - x_i)$ as follows:

$$\left(\frac{\partial u}{\partial x}\right)_{\mathrm{minmod}} = \begin{cases} \min(|s_L|, |s_R|), & s_L > 0 \wedge s_R > 0 \\ -\min(|s_L|, |s_R|), & s_L < 0 \wedge s_R < 0 \\ 0, & s_L s_R \leq 0 \end{cases}$$

The reconstructed function in two adjacent cells is then always monotonic (**Prove this!**). Of course, it may not be monotonic over the entire domain because there may extrema in u, but the reconstruction itself does not introduce any new non-monotonicities.

Slope Limiters – van Leer

The minmod limited still smears out the solution quite a bit. Actually, the restrictions on the limited slope can be relaxed quite a bit while still preserving the monotonicity of the solution. An example is the *van Leer limiter*:

$$\left(\frac{\partial u}{\partial x}\right)_{\mathrm{van\ Leer}} = \begin{cases} \frac{2 s_L s_R}{s_L + s_R}, & s_L s_R > 0 \\ 0, & s_L s_R \le 0 \end{cases}$$

Slope Limiters – MC

Van Leer's monotonised central (MC) limiter is also quite robust and accurate. In fact, it gives a solution that is almost 3rd order accurate!

$$\left(\frac{\partial u}{\partial x}\right)_{\mathrm{MC}} = \begin{cases} \mathrm{sign}(s_L)\min\left(\left|\frac{s_L+s_R}{2}\right|, 2|s_L|, 2|s_R|\right), & s_L s_R > 0 \\ 0, & s_L s_R \leq 0 \end{cases}$$

Slope Limiters – Superbee

- The *Superbee* limiter is constructed to capture discontinuities even more sharply.
- $$\left(\frac{\partial u}{\partial x}\right)_{\text{Superbee}} = \begin{cases} \operatorname{sign}(s_L)\max\left(\min\left(|s_R|, 2|s_L|\right), \min\left(2|s_R|, |s_L|\right)\right), & s_L s_R > 0 \\ 0, & s_L s_R \leq 0 \end{cases}$$
- It is actually the “sharpest” 2nd order limiter in a well-defined mathematical sense (which we don’t discuss here).
- However, this limiter tends to make the numerical solution “polygonal”.
- Making the slopes as steep as possible *everywhere* is obviously not desirable: *Superbee* still flattens the solution near extrema, not smearing it out elsewhere systematical distorts the shape of the solution.

Modern Methods

- Piecewise Parabolic Method (Colella & Woodward 1984, 4th order for uniform spacing; Colella & Sekora 2008, 6th order)
- Essentially Non-Oscillatory (ENO) Schemes (Shu & Osher 1999), arbitrary order in principle
- Weighted Essentially Non-Oscillatory Schemes

Outline of Piecewise Parabolic Method

- Determine a polynomial parabolic interpolant such that

$$\int_{x_{i-1/2}}^{x_{i+1/2}} = X_i \Delta x_i$$

  for any state variable X. This can be done by interpolating the antiderivative $\sum_{j=1}^{i} X_j \Delta x_j$.

- For uniform spacing, fourth-order accurate values at zone interfaces are given by

$$a_{i+1/2} = \frac{7}{12}(a_i + a_{i+1}) - \frac{1}{12}(a_{i-1} + a_{i+2})$$

- Monotonize to ensure that $a_{i+1/2}$ lies between a_i and a_{i+1}. Also make sure that the parabolic interpolant has no extremum *within* the cell.
- Check for contact discontinuities by looking for cells where the third derivative of ρ is large and the second derivative switches sign. Steepen the interpolant in these zones.
- Detect shock by looking for the characteristic jumps in pressure and density and flatten profiles near shocks.

Multi-Dimensional Fluid Flow

$$
\begin{aligned}
\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} + \frac{\partial \rho v_z}{\partial z} &= 0 \\
\frac{\partial \rho v_x}{\partial t} + \frac{\partial \rho v_x^2}{\partial x} + \frac{\partial \rho v_x v_y}{\partial y} + \frac{\partial \rho v_x v_z}{\partial z} + \frac{\partial P}{\partial x} &= -\rho g_x \\
\frac{\partial \rho v_y}{\partial t} + \frac{\partial \rho v_x v_y}{\partial x} + \frac{\partial \rho v_y^2}{\partial y} + \frac{\partial \rho v_y v_z}{\partial z} + \frac{\partial P}{\partial z} &= -\rho g_y \\
\frac{\partial \rho v_z}{\partial t} + \frac{\partial \rho v_x v_x}{\partial x} + \frac{\partial \rho v_y v_z}{\partial y} + \frac{\partial \rho v_z^2}{\partial z} + \frac{\partial P}{\partial z} &= -\rho g_z \\
\frac{\partial \rho e}{\partial t} + \frac{\partial (\rho e + P) v_x}{\partial x} + \frac{\partial (\rho e + P) v_y}{\partial y} + \frac{\partial (\rho e + P) v_z}{\partial z} &= -\rho (g_x v_x + g_y v_y + g_z v_z)
\end{aligned}
$$

Unsplit Schemes with Higher-Order Time Integration

One way to handle multi-D flow with higher-order time-integration accuracy is based on re-interpreting the *discretised* finite-volume equations as ordinary differential equations (ODE) for the cell averages, e.g.:

$$\begin{aligned}
\frac{\partial \rho_{i,j,k}\Delta V_{i,j,k}}{\partial t} &= \left[(\rho v_x)_{i-1/2,j,k}\Delta A_{i-1/2,j,k} - (\rho v_x)_{i+1/2,j,k}\Delta A_{i+1/2,j,k}\right] \\
&+ \left[(\rho v_x)_{i,j-1/2,k}\Delta A_{i,j-1/2,k} - (\rho v_x)_{i,j+1/2,k}\Delta A_{i,j+1/2,k}\right] \\
&+ \left[(\rho v_x)_{ij,k-1/2}\Delta A_{i,j,k-1/2} - (\rho v_x)_{i,j,k+1/2}\Delta A_{i,j,k+1/2}\right] \\
&= f[\rho(t)],
\end{aligned}$$

where f is a complicated function that depends on $\rho(t)$ in different cells, constructed from the solution of the Riemann problem at all cell interfaces. Then we use a standard method for ODE integration to solve the semi-discrete finite-volume equations. This is called the *method of lines* (MoL).

Method of Lines

Common ODE integrators for $\partial_t \mathbf{U} = \mathbf{f}[\mathbf{U}]$ used within the method of lines include:

- Method of Heun (2nd order):

$$\begin{aligned} \mathbf{U}'_{n+1} &= \mathbf{U}_n + \Delta t\ \mathbf{f}[\mathbf{U}_n] \\ \mathbf{U}_{n+1} &= \frac{1}{2}\left(\mathbf{U}_n + \mathbf{U}'_{n+1} + \Delta t\ \mathbf{f}[\mathbf{U}'_{n+1}]\right) \end{aligned}$$

- Method of Runge (=modified Euler or midpoint method, 2nd order):

$$\begin{aligned} \mathbf{U}'_{n+1/2} &= \mathbf{U}_n + \frac{\Delta t}{2}\ \mathbf{f}[\mathbf{U}_n] \\ \mathbf{U}_{n+1} &= \mathbf{U}_n + \Delta t\ \mathbf{f}[\mathbf{U}'_{n+1/2}] \end{aligned}$$

- 3rd order Runge-Kutta:

$$\begin{aligned} \mathbf{U}'_{n+1} &= \mathbf{U}_n + \Delta t \mathbf{f}[\mathbf{U}_n] \\ \mathbf{U}''_{n+1} &= \frac{1}{4}\left(3\mathbf{U}_n + \mathbf{U}'_{n+1} + \Delta t\ \mathbf{f}[\mathbf{U}'_{n+1}]\right) \\ \mathbf{U}_{n+1} &= \frac{1}{3}\left(2\mathbf{U}_n + \mathbf{U}''_{n+1} + \Delta t\ \mathbf{f}[\mathbf{U}''_{n+1}]\right) \end{aligned}$$

Primes or doubles primes denote intermediate results.
Usually, spatial accuracy is the limiting factor, so one rarely goes beyond 3rd order time integration.

Dimensional Splitting, Operator Splitting

Splitting techniques are another way to handle multiple dimensions. Unsplit MoL has some advantages, but so does operator splitting. Moreover, operator splitting is still the only viable choice for some applications (e.g. when the hydro is coupled to “stiff” equations for nuclear reactions or radiation transport).

Dimensional Splitting, Operator Splitting

In abstract notation, we can write a time-evolution problem like the equations of hydrodynamics as an equation with a *differential operator* $\mathcal{L}$, which contains all terms aside from the time derivative:

$$\frac{\partial \mathbf{U}}{\partial t} = \mathcal{L}[\mathbf{U}].$$

Formally, the solution to this equation can be written in terms of the exponential of the operator $\mathcal{L}$:

$$\mathbf{U}(t+\Delta t) = (\exp \mathcal{L})[\mathbf{U}(0)] = (1 + \Delta t \mathcal{L} + \frac{\Delta t^2}{2}\mathcal{L}_1^2 + \ldots)[\mathbf{U}(t)].$$

For the advection equation, this would look as follows:

$$\mathcal{L} = -v \frac{\partial}{\partial x}$$

And the solution would look like:

$$U(t+\delta t) = (1 + v\,\Delta t \frac{\partial}{\partial x} + \frac{v^2 \Delta t^2}{2}\frac{\partial^2}{\partial x^2} + \ldots)[U(t)].$$

How is this related to the explicit solution of the advection equation that we discused earlier?

Dimensional Splitting

We can view our numerical solution as the result of an operator acting on the hydrodynamic state on the previous time step. If our solution is accurate to $\mathcal{O}(\Delta t^n)$ in time, then this means that our numerical solution operator agrees with

$$(\exp \mathcal{L})[\mathbf{U}(0)] = (1 + \Delta t \mathcal{L} + \frac{\Delta t^2}{2} \mathcal{L}_1^2 + \ldots)[\mathbf{U}(t)].$$

up to the term in Δt^n.

Dimensional Splitting

For the sake of simplicity, consider the hydro equations in 2D (with the $\partial/\partial z$ terms omitted) without gravitational source terms. We could then collect the terms into two differential operators $\mathcal{L}_1$ and $\mathcal{L}_2$ containing the $\partial/\partial x$ and $\partial/\partial x$ terms respectively:

$$\frac{\partial \mathbf{U}}{\partial t} = \mathcal{L}_1[\mathbf{U}] + \mathcal{L}_2[\mathbf{U}]$$

Now the solution will be:

$$e^{(\mathcal{L}_1+\mathcal{L}_2)\Delta t} = 1 + \mathcal{L}_1\Delta t + \mathcal{L}_2\Delta t + \frac{\Delta t^2}{t}\left(\mathcal{L}_1^2 + \mathcal{L}_2^2 + \mathcal{L}_1\mathcal{L}_2 + \mathcal{L}_2\mathcal{L}_1\right) + \ldots$$

Suppose we already have a solution for $\partial\mathbf{U}/\partial y = \mathcal{L}_1[\mathbf{U}]$ and $\partial\mathbf{U}/\partial x = \mathcal{L}_2[\mathbf{U}]$ (which just means that we can solve the hydro equations in 1D). Then we might try to approximate the full solution as follows:

- Do one time step Δt ignoring the $\partial/\partial y$ terms (i.e. ignoring the operator $\mathcal{L}_2$.
- Do one time step Δt ignoring the $\partial/\partial x$ terms (i.e. ignoring the operator $\mathcal{L}_1$.

This strategy is called *operator splitting* (or *dimensional splitting* in the special case of splitting operators containing derivatives with respect of the different coordinates). It can be applied to arbitrarily many operators, provided that these meet certain conditions.

Dimensional Splitting

x-sweep: Solve the following equation for one time step Δt:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_x}{\partial x} &= 0 \\
\frac{\partial \rho v_x}{\partial t} + \frac{\partial \rho v_x^2}{\partial x} + \frac{\partial P}{\partial x} &= 0 \\
\frac{\partial \rho v_y}{\partial t} + \frac{\partial \rho v_x v_y}{\partial x} &= 0 \\
\frac{\partial e}{\partial t} + \frac{\partial (e+P) v_x}{\partial x} &= 0
\end{aligned}$$

y-sweep: Solve the following equation for one time step Δt:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_y}{\partial y} &= 0 \\
\frac{\partial \rho v_x}{\partial t} + \frac{\partial \rho v_x v_y}{\partial y} &= 0 \\
\frac{\partial \rho v_y}{\partial t} + \frac{\partial \rho v_y^2}{\partial y} &= 0 \\
\frac{\partial e}{\partial t} + \frac{\partial (e+P) v_y}{\partial y} &= 0
\end{aligned}$$

Dimensional Splitting

This amounts to using the operator

$$\begin{aligned}
e^{\mathcal{L}_1 \Delta t} e^{\mathcal{L}_2 \ \Delta t} &= (1 + \mathcal{L}_1 \Delta t + \mathcal{L}_1^2 \frac{\Delta t^2}{t} + \ldots)(1 + \mathcal{L}_2 \Delta t + \mathcal{L}_2^2 \frac{\Delta t^2}{t} + \ldots) \\
&= 1 + \mathcal{L}_1 \Delta t + \mathcal{L}_2 \Delta t + \mathcal{L}_1^2 \frac{\Delta t^2}{t} + \mathcal{L}_2^2 \frac{\Delta t^2}{t} + +2\mathcal{L}_1 \mathcal{L}_2 \frac{\Delta t^2}{2} + \ldots\ldots)
\end{aligned}$$

instead of $e^{(\mathcal{L}_1 + \mathcal{L}_2)\Delta t}$. If $2\mathcal{L}_1\mathcal{L}_2 = \mathcal{L}_1\mathcal{L}_2 + \mathcal{L}_2\mathcal{L}_1$, then we're fine, but *in general operators do not commute:* $\mathcal{L}_1 \neq \mathcal{L}_2$ (remember quantum mechanics)! Hence the operator-split solution agrees with the true solution only up to order Δt – it's only first order accurate.

Strang Splitting

An idea by Strang helps to recover 2nd order accuracy in time (if the individual sweeps are 2nd order accurate):

- Do an x-sweep with time step $\Delta t/2$.
- Do a y-sweep with time step Δt (or *two* y-sweeps with time step $\Delta t/2$).
- Do an x-sweep with time step $\Delta t/2$.

In operator notation, the time evolution operator becomes $e^{\mathcal{L}_1 \Delta t/2} e^{\mathcal{L}_2 \Delta t} e^{\mathcal{L}_1 \Delta t/2}$.
Prove that the resulting scheme is 2nd order accurate in time.

Splitting – General strategy

- Do an x-sweep with time step $\Delta t/2$.
- Do a y-sweep with time step $\Delta t/2$.
- Do a z-sweep with time step $\Delta t/2$.
- Apply source terms (gravity, heating/cooling, nuclear burning), time step Δt.
- Do a z-sweep with time step $\Delta t/2$.
- Do a y-sweep with time step $\Delta t/2$.
- Do an x-sweep with time step $\Delta t/2$.

Sweeps: Practical Implementation

Dimensional splitting conveniently allows a solution of the hydro equations along individual "pencils" with constant (y,z), (x,z) and (x,y) in each sweep. This suggests the following implementation, e.g., for the x-sweep:

```
for all j,k do
    Get $\mathbf{U}_{1\dots N_x,j,k}$ and copy to a temporary array $\tilde{\mathbf{U}}_{1\dots N_x}$
    Reconstruct interface states using $\tilde{\mathbf{U}}_{1\dots N_x}$
    Solve Riemann problems and compute fluxes
    Update $\tilde{\mathbf{U}}_{1\dots N_x}$
    Copy $\tilde{\mathbf{U}}_{1\dots N_x}$ back to $\mathbf{U}_{1\dots N_x,j,k}$
end for
```

This has two advantages:

- We can recycle a 1D hydro solver and don't need to re-arrange its internal data layout (no additional array dimensions).
- The small amount of data for a 1D sweep can be stored in the low-level caches of a CPU, and can be accessed and worked on much faster by the CPU than data that needs to be accessed from the main memory.

Practical issues: Why do we need higher-order accuracy

- Suppose we have a cubical numerical domain with N^3 cells, and suppose we want to calculate for at least one sound-crossing time-scale t_{sound} (which is not much).
- Since $\Delta t < \Delta x / c_s \approx t_{\mathrm{sound}}/N$, we need N time steps.
- With $\sim$50 floating point operations per grid cell per time step, the total number of operations is $\sim 50N^4$.
- For a sustained performance of 5 GFLOPS per core we need:
    - $N = 10000$: 1157 days
    - $N = 1000$: 2.8 hours

  on one core.
- If a higher order method saves us a factor of ten in grid resolution to get the same accuracy, this clearly pays off even if it's more expensive per step by a factor of 10 or even 100.
- Remember: The total number of operations in 3D scales with N^4!
- More physics makes this more expensive, so we're talking about $\mathcal{O}(10^4)$ core-hours or more for 3D simulations.

Memory Requirements

- Suppose the data we store for each cell comprises density, pressure, energy density, and the three velocity components (6 numbers per cell) in double precision.
- For a 3D simulation with 1024^3 cells, we thus need at least 6 GB main memory.
- In a hydro solver with dimensional splitting and sweeps along pencils, we roughly get away with this – temporary arrays don't take up much space.
- For an unsplit 2nd order MoL solver, we need to store the fluxes (3×5 numbers per cell, actually (6×5) per cell with 2 shared between neighbouring cells), and the intermediate results for the state variables after the first Runge-Kutta step.
- This means: 26 numbers per cell, 26GB in total (although we could save some memory if we're smart).

Parallelisation

- The huge memory and computer time requirements of multi-D simulations imply that we need to distribute the data and the computational load across multiple CPUs.
- We can't cover this in this course, but we can outline the strategy.
- There are two basic paradigms: *shared-memory parallelism* and *distributed-memory parallelism*.
- In the shared-memory model, multiple processor cores work on data that resides in a shared main memory space; each of them has access to all the data all the time (in principle). Using special compiler directives (provided, e.g., by the OpenMP standard), one can tell the compiler do distribute works among different cores.
- Advantage: Relatively easy to implement (even in incremental steps, i.e. by parallelising one part of the code after the other).
- Disadvantage: Side effects if the programmer is not careful, technical limitations on the size of shared-memory units.

Domain Decomposition

- In the distributed-memory model, each process sees only its local data, if it is needed by another process it has to be sent explicitly, e.g., using the Message Passing Interface (MPI).
- In hydro simulations, the data is usually distributed by *domain decomposition*, i.e. each process gets a block of the grid to work on.
- At the beginning of each time step, the first few cells near an interface must be exchanged with the neighbouring process, where they are needed as boundary conditions in the reconstruction step.
- Global synchronisation is necessary, e.g., for determining the time step (which must be the same for each process).