

Chapter 4

Advection algorithms II. Flux conservation, subgrid models and flux limiters

In this chapter we will focus on the flux-conserving formalism of advection algorithms, and we shall discuss techniques to have higher order accuracy without the risk of getting spurious oscillations in the solutions. This chapter follows in many respects the book of LeVeque, but at times with slightly modified notation, to remain consistent with the rest of the lecture notes.

4.1 Flux conserving formulation: the principles

So far we have focused mostly on how to move a function over a grid. If we want to apply this to the equations of hydrodynamics, then we need to keep in mind that these equations are in fact *conservation equations*. This property is of fundamental importance. If we make systematic errors in the conservation of conserved quantities, then we acquire behavior of our solutions that is clearly unphysical. Moreover, the fact that we have conserved quantities in our system is also a big help: if we can formulate our algorithm to identically conserve these quantities numerically (down to machine precision of 16 digits in doubleprecision), we have eliminated at least a few of possible things that can go wrong.

To numerically conserve conserved quantities is not just a matter of precision. Even with a reasonably precise algorithm, one could make small but *systematic* errors each time step. Suppose the relative error in the total energy is 10^{-4} per time step, but we need to perform 10^5 time steps to reach the end-time of the simulation, then if the error is always in one direction, the error has accumulated to a factor of 10 by the end of the simulation. Such results are clearly useless as we have created energy from nothing. Therefore it is a highly desirable property of a hydrodynamical code (or any solver of hyperbolic equations) if it is formulated in a *numerically flux conserving form*.

Of course, even the best flux-conserving formulation is not perfectly flux conserving, as numerical errors are always made when performing floating-point operations. However, the IEEE math routines used by any programming language (or built into any chip) are such that such errors are always only at the last digit (8th digit for floating points, 16th digit for doubleprecision floats), and generally such errors are not systematic, so that they do not propagate to lower digits quickly.

4.1.1 Cells and fluxes

The idea of flux conserving schemes is to create *cells* out of the grid, instead of just sampling grid points. The grid points now become *cell centers* located at x_i , and we now have *cell interfaces* or *cell walls* located at:

$$x_{i+1/2} = \frac{1}{2}(x_i + x_{i+1}) \quad (4.1)$$

where the $+1/2$ is just a notational trick to denote “in between i and $i + 1$ ”. Note that if we have a grid of N cell centers we have $N - 1$ cell interfaces. But it is also important to note that the cells $i = 1$ (left boundary) and $i = N$ (right boundary) also have to have a cell wall at resp. $i = 1/2$ and $i = N + 1/2$. Eq. (4.1) does not clearly define the location of these boundary cell walls. We will take them to be at:

$$x_{1/2} = x_1 - \frac{1}{2}(x_2 - x_1) \quad (4.2)$$

$$x_{N+1/2} = x_N + \frac{1}{2}(x_N - x_{N-1}) \quad (4.3)$$

This then makes a total of $N + 1$ cell interfaces. Let us define, as before, the cell spacing to be $\Delta x_{i+1/2} = x_{i+1} - x_i$ and $\Delta x_i = x_{i+1/2} - x_{i-1/2}$. For constant spacing we can write Δx for both quantities.

If we now assume this 1-D problem to be in fact a 3-D flow through a pipe with cross-sectional surface S , then the volume of each cell is: $V = \Delta x S$. In real 3-D hydrodynamic problems the cell volumes will be something like $V = \Delta x \Delta y \Delta z$, and the surface is $S = \Delta y \Delta z$.

We can now regard the cells as volumes containing our conserved quantity q_i^n . The total content of each cell is $Q_i^n = q_i^n V$. The fact that the equation to solve is a conservation equation means that Q_i^n can only be reduced by moving some of it to neighboring cells. Conversely, Q_i^n can only be increased by moving something from neighboring cells into cell i . Therefore the change in Q_i can be seen as in- and out-flow through the cell interfaces. We can formulate at each cell interface a *flux* $f_{i+1/2}$ and say that

$$\frac{dQ_i}{dt} = (f_{i-1/2} - f_{i+1/2}) \cdot S \quad (4.4)$$

In discrete form:

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} = (f_{i-1/2} - f_{i+1/2}) \cdot S \quad (4.5)$$

Stricly speak we must do this at half-time:

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} = (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \cdot S \quad (4.6)$$

Overall the sum of Q_i^n is now naturally constant, except for in- or outflow at $x = x_{1/2}$ or $x = x_{N+1/2}$:

$$\frac{d}{dt} \left(\sum_{i=1}^N Q_i \right) = (f_{1/2} - f_{N+1/2}) S \quad (4.7)$$

If we make use of the expressions $Q_i^n = q_i^n V$ and $V = S \Delta x$ (assume constant grid spacing for now) then Eq. (4.6) becomes:

$$\frac{q_i^{n+1} - q_i^n}{\Delta t} = \frac{f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}}{\Delta x} \quad (4.8)$$

or in explicit form:

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{\Delta x} (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \quad (4.9)$$

We have now formulated our advection problem in flux-conserving form.

4.1.2 Non-constant grid spacing

In flux conserving schemes it is very simple to handle non-regular grids. Suppose that we have a grid spacing in x as follows: $x_i = \{0., 0.1, 0.3, 0.6, 1.0, 1.5, \dots\}$, then the Δx is not anymore a globally constant value. Moreover, it now becomes an interesting question where we define the cell interfaces $x_{i+1/2}$ to be. In general there is no unique recipe to define the $x_{i+1/2}$ corresponding to the x_i for some non-regular grid spacing. As long as $x_{i+1/2}$ somewhere in between x_i and x_{i+1} in a sensible way, then it should be fine. The algorithm then becomes:

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{x_{i+1/2} - x_{i-1/2}} (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \quad (4.10)$$

with the fluxes $f_{i\pm 1/2}^{n+1/2}$ given by the particular algorithm used, for instance the donor-cell algorithm or one of the algorithms described in the sections to come.

All flux conserving algorithms that are first-order in time are based on Eq.(4.10). All the cleverness of the advection algorithm is hidden in how $f_{i\pm 1/2}^{n+1/2}$ is formulated in terms of the quantities $q_{i\pm k}^n$. Method which use this type of flux-conserving schemes are called Finite Volume Methods.

4.1.3 Non-constant advection velocity

In most practical purposes (in particular in the case of hydrodynamics) the advection velocity is not a global constant. If it were, then we would not have had to go through all the trouble in the last chapter and in this chapter to invent good advection algorithms, because the solution would have been analytically known beforehand. So the equation we wish to solve is now:

$$\partial_t q(x, t) + \partial_x (u(x)q(x, t)) = 0 \quad (4.11)$$

Note that now the distinction between a conservation equation and a non-conserving advection equation becomes apparent, and therefore it now becomes even more important to use flux conserving schemes if we wish to solve the conservation equation.

So how do we handle non-constant $u(x)$? First of all, we would still use the fundamental form of the update described in the previous subsection: Eq. (4.10). Exactly how we construct the fluxes $f_{i\pm 1/2}^{n+1/2}$ from q_i^n and the velocity field $u(x)$ depends very much on the algorithm that we use for the advection. We will discuss this in each of the examples below. In general, though, it requires the definition of the velocity u at the interfaces: $u_{i+1/2}$ because the fluxes $f_{i+1/2}$ are defined at the interfaces. Depending on the definition of the problem these velocities are already defined on these interfaces (i.e. the problem definition specifies $u_{i+1/2}$), or they have to be computed by linear interpolation from cell-centered values u_i (i.e. if the problem definition specifies u_i). Now the discrete algorithm is then Eq. (4.10) with

$$f_{i+1/2}^{n+1/2} = \tilde{q}_{i+1/2}^{n+1/2} u_{i+1/2} \quad (4.12)$$

where $\tilde{q}_{i+1/2}^{n+1/2}$ is some estimate of what the average state is of q at the interface:

$$\tilde{q}_{i+1/2}^{n+1/2} = \text{estimate of } \frac{1}{t_{n+1} - t_n} \int_{t_n}^{t_{n+1}} q(x_{i+1/2}, t) dt \quad (4.13)$$

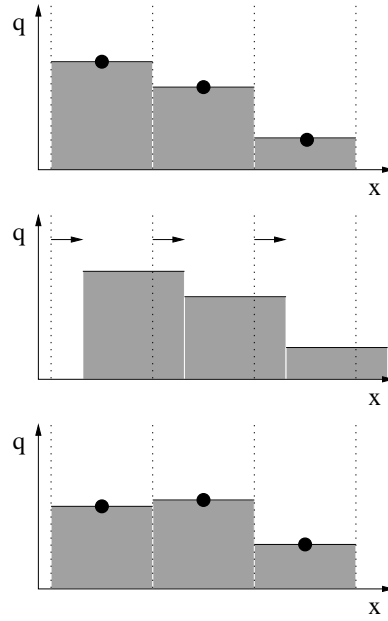


Figure 4.1. Illustration of the piecewise constant (donor-cell) advection algorithm.

Since we (by definition of the fact that we solve a numerical problem) do not know exactly what this average state is, it is the task of the algorithm to provide a recipe that estimates this as well as possible. In the two examples below we shall describe two such algorithms.

4.2 Example 1: Donor-cell advection

The simplest flux conserving scheme is the donor-cell scheme. In this scheme the “average interface state” is simply:

$$\tilde{q}_{i+1/2}^{n+1/2} = \begin{cases} q_i^n & \text{for } u_{i+1/2} > 0 \\ q_{i+1}^n & \text{for } u_{i+1/2} < 0 \end{cases} \quad (4.14)$$

This means that the donor-cell interface flux is:

$$f_{i+1/2}^{n+1/2} = \begin{cases} u_{i+1/2} q_i^n & \text{for } u_{i+1/2} > 0 \\ u_{i+1/2} q_{i+1}^n & \text{for } u_{i+1/2} < 0 \end{cases} \quad (4.15)$$

The physical interpretation of this method is the following. One assumes that the density is constant within each cell. We then let the material flow through the cell interfaces, from left to right for $u_{i+1/2} > 0$. Since the density to the left of the cell interface is constant, and since the CFL condition makes sure that the flow is no further than 1 grid cell spacing at maximum, we know that for the whole time between time t_n and t_{n+1} the flux through the cell interface (which is $\tilde{q}_{i+1/2}^{n+1/2} u_{i+1/2}$) is constant, and is equal to Eq. (4.15). Once the time step is finished, the state in each cell has the form of a step function (Fig. 4.1). To get back to the original sub-grid model we need to average the quantity $q(x)$ out over each cell, to obtain the new q_i^{n+1} . This is what happens in the donor-cell algorithm.

This method is very strongly similar to the *upstream differencing* scheme of Section 3.3.2. The difference comes to light mainly when either the velocity u is space-dependent or the grid x_i is non-constantly spaced (see Section 4.1.2). The Donor-Cell algorithm is easily implemented but, as the upstream differencing method, it is very diffusive.

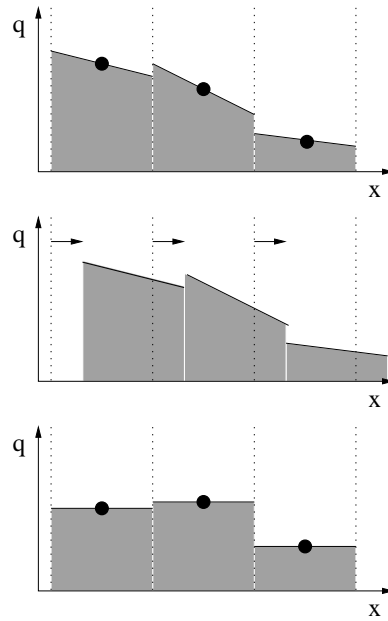


Figure 4.2. Illustration of the piecewise linear advection algorithm. The slope is chosen according to Lax-Wendroff's method.

→ **Exercise:** Quantify the difference between the donor-cell and upstream differencing schemes for non-constant grid spacing by writing the expressions for q^{n+1} in both cases and comparing them.

4.3 Example 2: Piecewise linear schemes

The donor-cell algorithm is a simple algorithm based on the idea that at the beginning of each time step the state within each cell is constant throughout the cell. The state at the grid center is therefore identical to that in the entire cell. The idea that the state is constant in the cell is, however, just an assumption. One must make *some* assumption, because evidently we have not more information about the state in the cell than just the cell-center state. But one could also make another assumption. One could, for instance, assume that the state within each cell is a linear function of position. One says that the state is assumed to be *piecewise linear* (see Fig. 4.2). The assumption that it is a linear function within the cell is called a *subgrid model*: it is a model of what the state looks like at spatial scales smaller than the grid spacing.

Within each cell the state at the beginning of the time step is now given by

$$q(x, t = t_n) = q_i^n + \sigma_i^n(x - x_i) \quad \text{for } x_{i-1/2} < x < x_{i+1/2} \quad (4.16)$$

where σ_i^n is some slope, which we shall discuss below. We should assure that

$$x_i = \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) \quad (4.17)$$

so that the state q_i^n is equal to the average of $q(x, t = t_n)$ over the cell irrespective of the slope σ_i^n . This is necessary for flux conservation.

Let us, from here on, assume that the grid is equally spaced, i.e. that there is a global Δx . Let us also assume that $u > 0$ is a constant.

At the interface the flux will now vary with time between t_n and t_{n+1} :

$$\begin{aligned} f_{i-1/2}(t) &= uq(x = x_{i-1/2}, t) \\ &= uq_{i-1}^n + u\sigma_{i-1}^n(x_{i-1/2} - x_{i-1} - u(t - t_n)) \\ &= uq_{i-1}^n + u\sigma_{i-1}^n \left(\frac{1}{2}\Delta x - u(t - t_n) \right) \end{aligned} \quad (4.18)$$

and similar for $f_{i+1/2}(t)$. The average flux over the time step $\Delta t = (t_{n+1} - t_n)$ is then:

$$\begin{aligned} \langle f_{i-1/2}(t) \rangle_{t_n}^{t_{n+1}} &= \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} uq(x = x_{i-1/2}, t) dt \\ &= uq_{i-1}^n + \frac{1}{2}u\sigma_{i-1}^n (\Delta x - u\Delta t) \end{aligned} \quad (4.19)$$

and similar for $f_{i+1/2}(t)$. The difference is:

$$\langle f_{i+1/2}(t) \rangle_{t_n}^{t_{n+1}} - \langle f_{i-1/2}(t) \rangle_{t_n}^{t_{n+1}} = u(q_i^n - q_{i-1}^n) + \frac{1}{2}u(\sigma_i^n - \sigma_{i-1}^n)(\Delta x - u\Delta t) \quad (4.20)$$

Using Eq. (4.10) we then obtain the update of the state after one time step:

$$q_i^{n+1} = q_i^n - \frac{u\Delta t}{\Delta x}(q_i^n - q_{i-1}^n) - \frac{u\Delta t}{\Delta x} \frac{1}{2}(\sigma_i^n - \sigma_{i-1}^n)(\Delta x - u\Delta t) \quad (4.21)$$

where we defined $f_{i+1/2}^{n+1/2} \equiv \langle f_{i+1/2}(t) \rangle_{t_n}^{t_{n+1}}$. Eq. (4.21) is the update of the state for a flux-conserving piecewise linear scheme (assuming that the grid spacing is constant). This is the higher-order version of the donor-cell algorithm. Note that it is identical to donor-cell if the slopes are chosen to be zero. Note also that since we chose the grid to be constantly spaced and the velocity to be globally constant, the algorithm is like an upwind scheme with a correction term.

The question is now: how shall we choose the slope σ_i^n of the linear function? The idea behind the piecewise linear scheme is that one uses the states at adjacent grid points in some reasonable way. There are three obvious methods:

$$\text{Centered slope: } \sigma_i^n = \frac{q_{i+1}^n - q_{i-1}^n}{2\Delta x} \quad (\text{Fromm's method}) \quad (4.22)$$

$$\text{Upwind slope: } \sigma_i^n = \frac{q_i^n - q_{i-1}^n}{\Delta x} \quad (\text{Beam-Warming method}) \quad (4.23)$$

$$\text{Downwind slope: } \sigma_i^n = \frac{q_{i+1}^n - q_i^n}{\Delta x} \quad (\text{Lax-Wendroff method}) \quad (4.24)$$

All these choices result in second-order accurate methods.

→ **Exercise:** Prove that the piecewise linear scheme with a downwind slope indeed produces the Lax-Wendroff scheme of Eq. (3.96) in Chapter 3.

If we now implement, for instance, Fromm's choice of slope into Eq. (4.21) then we obtain the following explicit update of the state (again, valid only for regular grid spacing and constant u):

$$\begin{aligned} q_i^{n+1} &= q_i^n - \frac{u\Delta t}{4\Delta x}(q_{i+1}^n + 3q_i^n - 5q_{i-1}^n + q_{i-2}^n) \\ &\quad - \frac{u^2\Delta t^2}{4\Delta x^2}(q_{i+1}^n - q_i^n - q_{i-1}^n + q_{i-2}^n) \end{aligned} \quad (4.25)$$

This is called Fromm's method of advection. One notices that the stencil of this scheme involves 4 points, and that these points are non-symmetric with respect to the updated cell. Only for the downwind slope we regain a 3-point stencil.

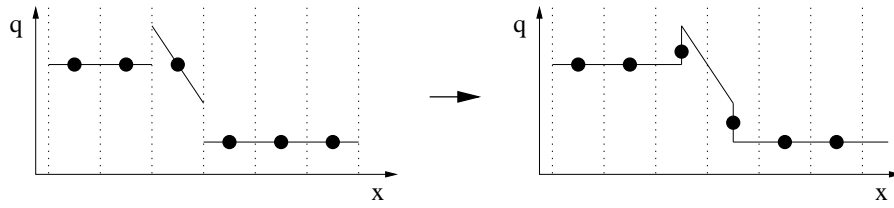


Figure 4.3. Illustration of why higher order schemes, such as the piecewise linear Lax-Wendroff scheme shown here, produce oscillations near discontinuities.

4.4 Slope limiters: non-linear tools to prevent overshoots

As we have already seen in chapter 3 higher order schemes tend to produce oscillations near jumps. We have seen that this can mathematically be understood in terms of phase errors and dispersion. With the current piecewise linear geometric picture of the higher order methods we can now also gain a geometrical understanding of why such an overshoot occurs. This is shown in Fig. 4.3

In this picture the overshoot happens because the piecewise linear elements can have overshoots. A succesful method to prevent such overshoots is the use of *slope limiters*. These are non-linear conditions that modify the slope σ_i^n if, and only if, this is necessary to prevent overshoots. This means that, if this is applied to for instance the Lax-Wendroff method, the slope limiter keeps the second order nature of the Lax-Wendroff method in regions of smooth variation of q while it will intervene whenever an overshoot threatens to take place.

4.4.1 The “Total Variation” (TV)

To measure oscillations in a solution we define the *Total Variation* (TV) of the discrete representation of q to be:

$$TV(q) = \sum_{i=1}^N |q_i - q_{i-1}| \quad (4.26)$$

where $i = 1$ and $i = N$ are the left and right boundaries of the numerical grid respectively. For a monotonically increasing function q the $TV(q) = |q_1 - q_N|$. If q_1 and q_N are taken to be constant, then, as long as the function remains monotonic, the $TV(q)$ is constant. However, if the values of q_i develop local minima or maxima, then the TV increases, by virtue of the absolute value in Eq. (4.26). The increase of TV is therefore a measure of the development of oscillations in a solution.

A numerical scheme is said to be *total variation diminishing* (TVD) if:

$$TV(q^{n+1}) \leq TV(q^n) \quad (4.27)$$

Clearly such a scheme will not develop oscillations near a jump, because a jump is a monotonically in/decreasing function and a total variation diminishing scheme will not increase the TV , and hence in this case keep the TV identically constant. This appears to be a desirable property of the numerical scheme.

What about if we already start with a solution that has some local minima and maxima? In principle a total variation diminishing scheme could conserve the total variation (as long as it does not increase it). In practice, however, the local minima and maxima will be gradually smoothed out by such a scheme. Good schemes will do this only very weakly, and will only really diminish oscillations near sharp jumps.

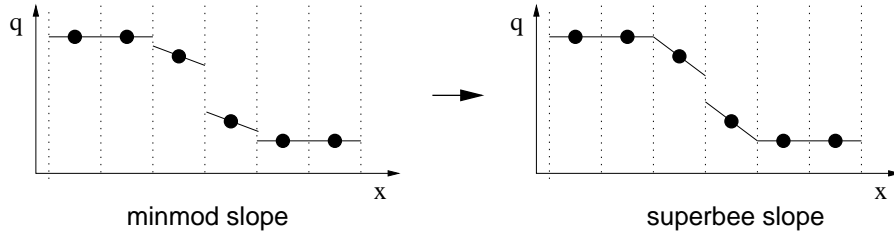


Figure 4.4. Illustration of the slope limiters minmod and superbee. Both clearly prevent overshoots in the subgrid models and hence are TVD.

4.4.2 Slope limiters

So now let us try to formulate a recipe to limit the slope σ_i^n of the piecewise linear scheme to make sure that the resulting scheme is TVD. One obvious choice is to take the slope always zero. We then recover the donor-cell algorithm. This algorithm is TVD because the piecewise constant function has the same TV as the continuous function. In fact, as we have discussed in Section 3.9, all first order schemes are TVD.

For the piecewise linear scheme there are several methods to guarantee TVD. One method is the *minmod slope*:

$$\sigma_i^n = \text{minmod} \left(\frac{q_i^n - q_{i-1}^n}{\delta x}, \frac{q_{i+1}^n - q_i^n}{\delta x} \right) \quad (4.28)$$

where

$$\text{minmod}(a, b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } ab > 0 \\ b & \text{if } |a| > |b| \text{ and } ab > 0 \\ 0 & \text{if } ab \leq 0 \end{cases} \quad (4.29)$$

This method chooses the smallest of the two slopes, provided they both have the same sign, and chooses 0 otherwise. The minmod slope is illustrated in Fig. 4.4-left.

Another efficient slope limiter is the *superbee* slope limiter introduced by Phil Roe:

$$\sigma_i^n = \text{maxmod}(\sigma_i^{(1)}, \sigma_i^{(2)}) \quad (4.30)$$

with

$$\sigma_i^{(1)} = \text{minmod} \left(\frac{q_{i+1}^n - q_i^n}{\delta x}, 2 \frac{q_i^n - q_{i-1}^n}{\delta x} \right) \quad (4.31)$$

$$\sigma_i^{(2)} = \text{minmod} \left(2 \frac{q_{i+1}^n - q_i^n}{\delta x}, \frac{q_i^n - q_{i-1}^n}{\delta x} \right) \quad (4.32)$$

The superbee slope is shown in Fig. 4.4-right.

In Fig. 4.5 the results are shown for all the piecewise linear advection algorithms discussed here.

4.5 Flux limiters

The concept of slope limiters is very closely related to the concept of *flux limiters*. For flux-conserving schemes the concept of flux-limiter is more appropriate and can be better implemented. Let us take another look at the time-averaged flux through a cell interface (Eq. 4.19).

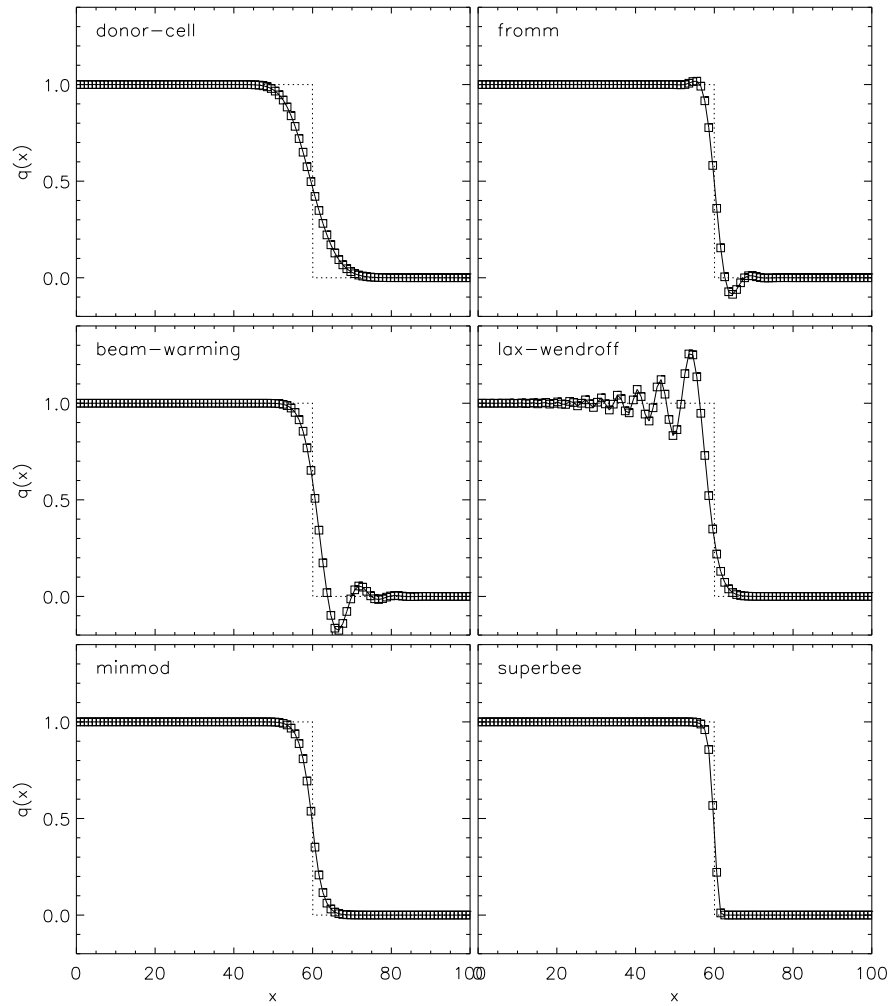


Figure 4.5. Advection with the piecewise linear advection algorithm with 6 different choices of the slope. Results are shown of the advection of a step function over a grid of 100 points with grid spacing $\Delta x = 1$, after 300 time steps with $\Delta t = 0.1$.

For arbitrary sign of u this becomes:

$$f_{i-1/2}^{n+1/2} = \begin{cases} uq_{i-1}^n + \frac{1}{2}u_{i-1/2}\sigma_{i-1}^n (\Delta x - u_{i-1/2}\Delta t) & \text{if } u_{i-1/2} \geq 0 \\ uq_i^n - \frac{1}{2}u_{i-1/2}\sigma_i^n (\Delta x + u_{i-1/2}\Delta t) & \text{if } u_{i-1/2} \leq 0 \end{cases} \quad (4.33)$$

where we dropped the $\langle \rangle_{t_n}^{t_{n+1}}$ for notational convenience and put instead a $^{n+1/2}$ to denote the fact that the flux average is similar to taking the flux at half-time¹. We have also now allowed for a varying velocity u . The velocity u is, in this formulation, defined on the cell interfaces, while the value of q is always defined at the cell center. If we, for convenience, introduce

$$\theta_{i-1/2} \equiv \theta(u_{i-1/2}) = \begin{cases} +1 & \text{for } u_{i-1/2} \geq 0 \\ -1 & \text{for } u_{i-1/2} \leq 0 \end{cases} \quad (4.34)$$

then we can write

$$f_{i-1/2}^{n+1/2} = \frac{1}{2}u_{i-1/2} \left[(1 + \theta_{i-1/2})q_{i-1}^n + (1 - \theta_{i-1/2})q_i^n \right] + \frac{1}{4}|u_{i-1/2}| \left(1 - \left| \frac{u_{i-1/2}\Delta t}{\Delta x} \right| \right) \Delta x \left[(1 + \theta_{i-1/2})\sigma_{i-1}^n + (1 - \theta_{i-1/2})\sigma_i^n \right] \quad (4.35)$$

This expression is identical to Eq. (4.33). It is just written in a single formula using the flip-flop function $\theta_{i-1/2}$ which is $+1$ for positive advection velocity and -1 for negative advection velocity. This expression is valid for any regularly or non-regularly spaced grid, and for any constant or varying advection velocity. It is therefore the kind of expression we need for numerical hydrodynamics in which non-regular grids and, more importantly, varying velocity fields are common.

In Eq. (4.35) it is now possible to replace

$$\frac{1}{2}\Delta x \left[(1 + \theta_{i-1/2})\sigma_{i-1}^n + (1 - \theta_{i-1/2})\sigma_i^n \right] \rightarrow \phi(r_{i-1/2}^n)(q_i^n - q_{i-1}^n) \quad (4.36)$$

where we introduced a *flux limiter* $\phi(r_{i-1/2}^n)$ where $r_{i-1/2}^n$ is defined as:

$$r_{i-1/2}^n = \begin{cases} \frac{q_{i-1}^n - q_{i-2}^n}{q_i^n - q_{i-1}^n} & \text{for } u_{i-1/2} \geq 0 \\ \frac{q_{i+1}^n - q_i^n}{q_i^n - q_{i-1}^n} & \text{for } u_{i-1/2} \leq 0 \end{cases} \quad (4.37)$$

We shall discuss expressions for $\phi(r_{i-1/2}^n)$ in a minute, but first let us see how Eq. (4.35) is modified by this replacement:

$$f_{i-1/2}^{n+1/2} = \frac{1}{2}u_{i-1/2} \left[(1 + \theta_{i-1/2})q_{i-1}^n + (1 - \theta_{i-1/2})q_i^n \right] + \frac{1}{2}|u_{i-1/2}| \left(1 - \left| \frac{u_{i-1/2}\Delta t}{\Delta x} \right| \right) \phi(r_{i-1/2}^n)(q_i^n - q_{i-1}^n) \quad (4.38)$$

We see that this expression of the flux is still the donor-cell flux (first line) plus some correction term (second line).

¹Note, however, that this does not mean that we have a Crank-Nicholson scheme here! This is not an implicit scheme!

We can now verify that with particular choices of the flux limiter function $\phi(r)$ we regain the various recipes of Section 4.4:

$$\begin{aligned}
 \text{donor-cell :} & \quad \phi(r) = 0 \\
 \text{Lax-Wendroff :} & \quad \phi(r) = 1 \\
 \text{Beam-Warming :} & \quad \phi(r) = r \\
 \text{Fromm :} & \quad \phi(r) = \frac{1}{2}(1 + r)
 \end{aligned} \tag{4.39}$$

which are the linear schemes and

$$\begin{aligned}
 \text{minmod :} & \quad \phi(r) = \text{minmod}(1, r) \\
 \text{superbee :} & \quad \phi(r) = \max(0, \min(1, 2r), \min(2, r))
 \end{aligned} \tag{4.40}$$

which are the high-resolution non-linear schemes. Now that we are at it, there are also many other non-linear flux limiter schemes. Two examples are:

$$\begin{aligned}
 \text{MC :} & \quad \phi(r) = \max(0, \min((1 + r)/2, 2, 2r)) \\
 \text{van Leer :} & \quad \phi(r) = (r + |r|)/(1 + |r|)
 \end{aligned} \tag{4.41}$$

where MC stands for *monotonized central-difference limiter*.

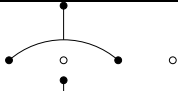

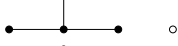
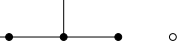





A flux limiter has the following effect:

- For smooth parts of a solution it will do second order accurate flux-conserved advection.
- For regions near a jump or a very sharp and sudden gradient it will switch to first order donor-cell (i.e. upwind) flux-conserved advection.

Flux limiters therefore make sure that one gets the best of both second order and first order methods. Flux limiters will turn out to play a major role for *shock capturing hydrodynamics codes*, in which shocks need to be kept tight and not smear out over too many grid cells. For the modeling of supersonic flows with shock waves (such as those often found in astrophysics) they are a highly useful tool. However, one should keep in mind that they may artificially steepen gradients that may not actually be meant to be steepened. They therefore have to be used with care.

4.6 Overview of algorithms

As a final remark we give here a table of the properties of the various algorithms:

Name	Order	Lin?	Stable?	TVD?	Stencil
Two-point symmetric	1	lin	-	-	
Upwind / Donor-cell	1	lin	+	+	
Lax-Wendroff	2	lin	+	-	
Beam-warming	2	lin	+	-	
Fromm	2	lin	+	-	
Minmod	2/1	non-lin	+	+	
Superbee	2/1	non-lin	+	+	
MC	2/1	non-lin	+	+	
van Leer	2/1	non-lin	+	+	

4.7 Computer implementation (adv-2): cell interfaces and ghost cells

The computer implementation is now a bit more complicated than in chapter 3 because we now have cells, and cell-interfaces. In this section we will see how to deal with this in a proper way. Also we will introduce the concept of *ghost cells* which is a simple trick to implement boundary conditions in a very easy way, even periodic boundary conditions. It will help us greatly when we will do the hydrodynamics in the later chapters.

Let us, as in the previous section, set up the problem in IDL/GDL. First the grid:

```

nx      = 100
x       = dblarr(nx+2)
xi      = dblarr(nx+3)

```

Here we have 100 regular grid points with a ghost cell on each side. We have therefore 102 cells with 103 cell walls. We now also define the array for q , for the temporary array q_{new} and for the flux f :

```

q       = dblarr(nx+2)
qnew    = dblarr(nx+2)
f       = dblarr(nx+3)

```

Now we set up the grid

```

dx      = 1.d0           ; Set grid spacing
for i=0,nx+1 do x[i]    = (i-1.d0) * dx
for i=1,nx+1 do xi[i]   = 0.5 * ( x[i] + x[i-1] )
xi[0]    = x[0] - 0.5 * ( x[1] - x[0] )
xi[nx+2] = x[nx+1] + 0.5 * ( x[nx+1] - x[nx] )

```

where we made the grid regular with grid spacing dx . Note that the cell walls are indexed such that wall i is left of cell i (see Fig. 4.6). Now we set up the initial condition and the advection velocity array

```
for i=0,nx-1 do if x[i] lt 30. then q[i]=1.d0 else q[i]=0.d0
u      = 1.d0 + dblarr(nx+3)
```

Note that the advection velocity array is also defined on the cell walls. Now the main part of the code is:

```
dt      = 2d-1
tend    = 70.d0
time    = 0.d0
while time lt tend do begin
  ;;
  ;; Check if end time will not be exceeded
  ;;
  if time + dt lt tend then begin
    dtused = dt
  endif else begin
    dtused = tend-time
  endelse
  ;;
  ;; Impose cyclic (=periodic) boundary conditions
  ;; using the ghost cells
  ;;
  q[0]    = q[nx]
  q[nx+1] = q[1]
  ;;
  ;; Make the flux
  ;;
  for i=1,nx+1 do begin
    if u[i] gt 0 then begin
      f[i] = q[i-1] * u[i]
    endif else begin
      f[i] = q[i] * u[i]
    endelse
  endfor
  ;;
  ;; Do the advection
  ;;
  for i=1,nx do begin
    qnew[i] = q[i] - dtused * ( f[i+1] - f[i] ) / $
                               ( xi[i+1]- xi[i] )
  endfor
  ;;
  ;; Copy back
  ;;
```

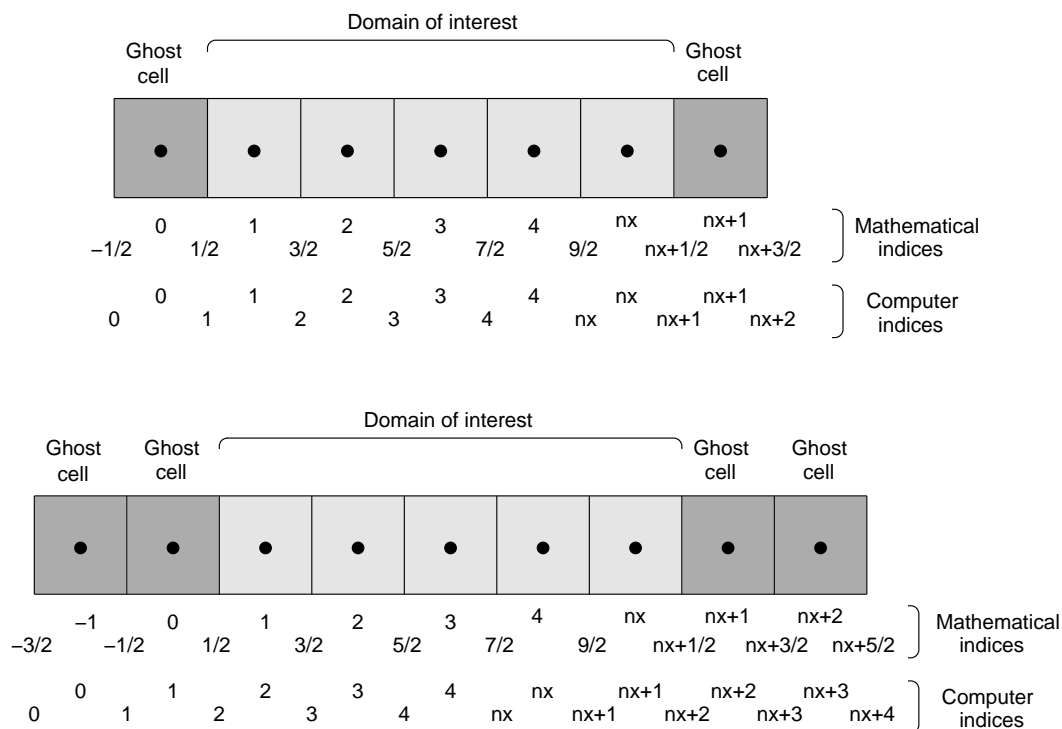


Figure 4.6. The grid of a flux-conserving advection program including ghost cells, for algorithms with a three-point stencil (upper panel) and for algorithms with a four- or five-point stencil (lower panel).

```

for i=1,nx do begin
    q[i] = qnew[i]
endfor
;;
;; Update time
;;
time = time + dtused
endwhile

```

Here we have imposed cyclic (periodic) boundary conditions, just as an example how useful ghost cells can be. We have also made sure that the algorithm works for any $u(x)$, also if it is a function of x and if it changes sign. If we now end the program with

```

plot,x[1:nx],q[1:nx],psym=-6
end

```

then we will see a figure on the screen like Fig. 4.7. Note that the periodic boundary conditions have the effect that the diffusive smearing now affects the front- *and* the back-side of the block function, in contrast to the case of Fig. 3.6. Also note that in the present program we went all the way to $t = 70$ instead of $t = 30$ of Fig. 3.6. The smearing has now almost entirely destroyed the block function.

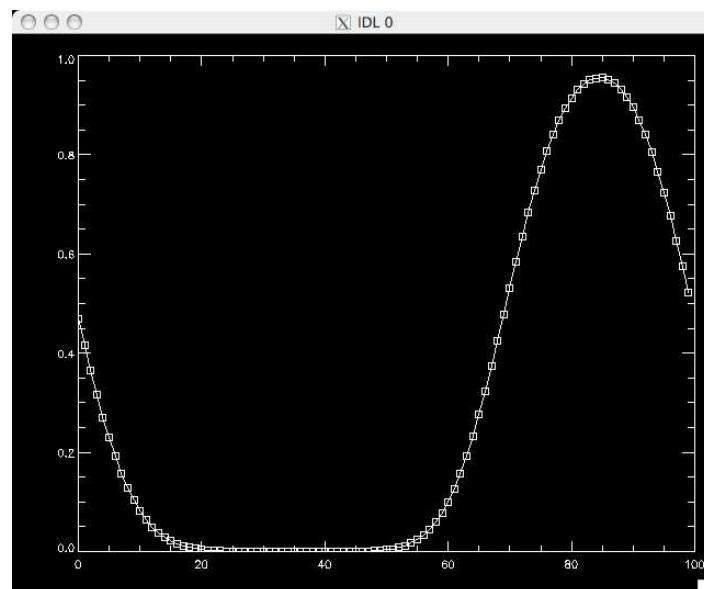


Figure 4.7. The plot resulting from the `donorcell.pro` program of Section 4.7. Note that in contrast to Fig. 3.6 we now have continued the simulation until time $t = 70$ and we have imposed periodic boundary conditions. Hence the difference to Fig. 3.6.