

Chapter 7

Riemann solvers II

In this chapter we will see how the concepts of a Riemann solver are in practice implemented. We will discuss several often-used schemes.

7.1 Roe's linearized Riemann solver

The Roe solver uses the technique of linearization of the equations, and then applying the Riemann method to the linear perturbations. The elegance of the method lies in the fact that the linearization is done in such a way that it also correctly recognises *non-linear* jumps such as shocks and contact discontinuities. The disadvantage of the method is that, in practice, it sometimes creates a unphysical results such as “expansion shocks” (which should be smooth rarefaction waves) or the infamous carbuncle phenomenon.

7.1.1 The equations of hydrodynamics revisited

Before we construct our linearized Riemann solver, let us make a slightly modified definition of the state vector q , which allows us an easy generalization of our algorithms to 3-D. We define q to be

$$q = \begin{pmatrix} \rho e_{\text{tot}} \\ \rho u \\ \rho v \\ \rho w \\ \rho \end{pmatrix} \quad (7.1)$$

For convenience we shall index it from 0 to 4:

$$q_0 = \rho e_{\text{tot}} \quad q_1 = \rho u \quad q_2 = \rho v \quad q_3 = \rho w \quad q_4 = \rho \quad (7.2)$$

The full set of equations for 3-D hydrodynamics is then:

$$\partial_t q + \partial_x f_x(q) + \partial_y f_y(q) + \partial_z f_z(q) = 0 \quad (7.3)$$

where

$$f_x = \begin{pmatrix} \rho h_{\text{tot}} u \\ \rho u^2 + P \\ \rho v u \\ \rho w u \\ \rho u \end{pmatrix} \quad f_y = \begin{pmatrix} \rho h_{\text{tot}} v \\ \rho v^2 + P \\ \rho w v \\ \rho v \end{pmatrix} \quad f_z = \begin{pmatrix} \rho h_{\text{tot}} w \\ \rho w^2 + P \\ \rho v w \\ \rho w \end{pmatrix} \quad (7.4)$$

7.1.2 Linearized Riemann solvers

We have seen that for linear problems the Riemann solver reduces to a characteristic solver. For the full non-linear set of equations of hydrodynamics this is no longer the case. A Riemann solver, such as Godunov's method, is then a rather complex solver because it involves complex and non-linear solutions to the Riemann problem at each cell interface. It is also rather costly to solve numerically. Cheaper and elegant simplifications are *linearized Riemann solvers*. The way this is done is by expressing our interface fluxes as much as possible only using the differences in the state variables:

$$\Delta q_{k,i-1/2} \equiv q_{k,i-1/2,R} - q_{k,i-1/2,L} \quad (7.5)$$

If these differences are small, then much of the algebra can be linearized to first order in $\Delta q_{k,i-1/2}$.

If we now set the state at the beginning of each time step constant within each cell, then the cell interfaces have jumps of the state, i.e. they define a Riemann problem. The way to linearize this is to define an average state at the interface $\hat{q}_{k,i-1/2}$ (note: here we retain the index k of the index notation) in some way:

$$\hat{q}_{k,i-1/2} = \text{Average}[q_{k,i}, q_{k,i-1}] \quad (7.6)$$

where the precise definition of the average will be defined later. For now we can simply set $\hat{q}_{k,i-1/2} = (q_{k,i} + q_{k,i-1})/2$ for instance. Now we can express the Riemann problem in the deviation from this average:

$$\delta q_{k,i-1/2,L} \equiv q_{k,i-1} - \hat{q}_{k,i-1/2} \quad (7.7)$$

$$\delta q_{k,i-1/2,R} \equiv q_{k,i} - \hat{q}_{k,i-1/2} \quad (7.8)$$

If $|\delta q_{k,i-1/2,L/R}| \ll |\hat{q}_{k,i-1/2}|$ then, locally, the Riemann problem can be regarded as a linear Riemann problem, which we have extensively discussed in Section 6.5. The advection matrix in x direction is now simply the Jacobian $\partial f_x(q)/\partial q$, so the equation, locally between $x_{i-1} < x < x_i$ becomes:

$$\partial_t \delta q + \left(\frac{\partial f_x}{\partial q} \right) \partial_x \delta q = 0 \quad (7.9)$$

The eigenvalues of the Jacobian $\partial f_x/\partial q$ at the interface $i - 1/2$ are (for convenience we leave out the $i - 1/2$ index):

$$\lambda_1 = \hat{u} - \hat{C}_s \quad (7.10)$$

$$\lambda_2 = \hat{u} + \hat{C}_s \quad (7.11)$$

$$\lambda_3 = \hat{u} \quad (7.12)$$

$$\lambda_4 = \hat{u} \quad (7.13)$$

$$\lambda_5 = \hat{u} \quad (7.14)$$

with eigenvectors:

$$e_1 = \begin{pmatrix} \hat{h}_{\text{tot}} - \hat{C}_s \hat{u} \\ \hat{u} - \hat{C}_s \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad e_2 = \begin{pmatrix} \hat{h}_{\text{tot}} + \hat{C}_s \hat{u} \\ \hat{u} + \hat{C}_s \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad (7.15)$$

$$e_3 = \begin{pmatrix} \frac{1}{2} \hat{u}^2 \\ \hat{u} \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad e_4 = \begin{pmatrix} \hat{v}^2 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad e_5 = \begin{pmatrix} \hat{w}^2 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad (7.16)$$

where $\hat{h}_{\text{tot}} = \hat{e}_{\text{tot}} + \hat{P}/\hat{\rho}$ is the total specific enthalpy and $\hat{C}_s = \sqrt{\gamma \hat{P}/\hat{\rho}}$ is the adiabatic sound speed. In all symbols the caret $\hat{}$ indicates that these are the primitive variables as derived from the ‘‘average state’’ $\hat{q}_{i-1/2}$ at the location of interface $i - 1/2$. Note that we can derive similar expressions for the advection in y and z direction.

We can now directly insert those formulae to Eq. (6.56) and apply this to the values of $\delta q_{k,i-1/2,L/R}$. Now the special form of Eq.(6.56) comes to our advantage, because since this expression (and the expressions for the flux limiters) only depend on the difference $\delta q_{k,i-1/2,R} - \delta q_{k,i-1/2,L} \equiv q_{k,i-1/2,R} - q_{k,i-1/2,L} \equiv \Delta q_{k,i-1/2}$, we can now forget about $\delta q_{k,i-1/2,L/R}$ and focus entirely on $\Delta q_{k,i-1/2}$, which is the jump of the state over the interface. We can now decompose $\Delta q_{k,i-1/2}$ into the eigenvectors Eq. (7.15...7.16):

$$\Delta q_{k,i-1/2} = \sum_{m=1 \dots 5} \tilde{\Delta} q_{m,i-1/2} e_{m,k,i-1/2} \quad (7.17)$$

where (again for clarity we omit the index $i - 1/2$):

$$\tilde{\Delta} q_1 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ \hat{e}_{\text{kin}} \Delta q_4 - \xi \} - \frac{\Delta q_1 - \hat{u} \Delta q_4}{2\hat{C}_s} \quad (7.18)$$

$$\tilde{\Delta} q_2 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ \hat{e}_{\text{kin}} \Delta q_4 - \xi \} + \frac{\Delta q_1 - \hat{u} \Delta q_4}{2\hat{C}_s} \quad (7.19)$$

$$\tilde{\Delta} q_3 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ (\hat{h}_{\text{tot}} - 2\hat{e}_{\text{kin}}) \Delta q_4 + \xi \} \quad (7.20)$$

$$\tilde{\Delta} q_4 = \Delta q_2 - v \Delta q_4 \quad (7.21)$$

$$\tilde{\Delta} q_5 = \Delta q_3 - w \Delta q_4 \quad (7.22)$$

where $\hat{e}_{\text{kin}} = (\hat{u}^2 + \hat{v}^2 + \hat{w}^2)/2$ and $\xi \equiv u \Delta q_1 + v \Delta q_2 + w \Delta q_3 - \Delta q_0$. With these expressions for $\tilde{\Delta} q_{i-1/2}$ the flux at the interface becomes (cf. Eq. 6.56) becomes¹

$$f_{k,i-1/2}^{n+1/2} = \frac{1}{2} (f_{k,i-1/2,R}^n + f_{k,i-1/2,L}^n) - \frac{1}{2} \sum_{m=1 \dots 5} \lambda_{m,i-1/2} \tilde{\Delta} q_{m,i-1/2} e_{m,k,i-1/2} [\theta_{m,i-1/2} + \tilde{\phi}_{m,i-1/2} (\epsilon_{m,i-1/2} - \theta_{m,i-1/2})] \quad (7.23)$$

¹Warning: There was a typo in this formula until Philipp Girichidis found it, 10 January 2011.

where we retained the index k in the expression for the flux: $f_{k,i-1/2}^{n+1/2}$ according to index notation. We now see that the interface flux for this linearized Riemann solver consists of the average of the non-linear fluxes plus a correction term in which the difference of the flux over the interface is decomposed into eigenvectors and each component advected in its own upwind fashion.

A linearized Riemann solver is evidently not an exact Riemann solver, since the Riemann problem is solved in an approximate way only. This is why linearized Riemann solvers are part of the (larger) family of *approximate Riemann solvers*.

7.1.3 Roe's average interface state

The final missing piece of the algorithm is a suitable expression for the “average interface state”, or better, the “average interface primitive variables” $\hat{u}_{i-1/2}, \hat{v}_{i-1/2}, \hat{w}_{i-1/2}, \hat{\rho}_{i-1/2}, \hat{h}_{\text{tot},i-1/2}$. As long as the numerical solution is very smooth, i.e. that $|\Delta q_{k,i-1/2}| \ll |\hat{q}_{k,i-1/2}|$, then any reasonable average would do and would probably give the right results. However, when contact discontinuities and/or shock waves are present in the solution, then it becomes extremely important to define the proper average such that the “linearization” (which is then strictly speaking no longer valid) still produces the right propagation of these discontinuities.

A *Roe solver* is a linearized Riemann solver with a special kind of averaged state at the interface. These state variables are defined as:

$$\hat{u} = \frac{\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.24)$$

$$\hat{v} = \frac{\sqrt{\rho_L}v_L + \sqrt{\rho_R}v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.25)$$

$$\hat{w} = \frac{\sqrt{\rho_L}w_L + \sqrt{\rho_R}w_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.26)$$

$$\hat{h}_{\text{tot}} = \frac{\sqrt{\rho_L}h_{\text{tot},L} + \sqrt{\rho_R}h_{\text{tot},R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.27)$$

With these expressions for the average interface state and the above defined eigenvector decomposition and interface flux expressions we now have a Riemann solver that acts as a *linearized* Riemann solver for smooth regions of the flow and acts as a *non-linear shock-capturing* Riemann solver whenever it encounters strong discontinuities. Together with the superbee fluxlimiter it is one of the least diffusive Riemann solvers around. This is good, in the sense that it does not smear out details very much, but it can also be bad, in the sense that it can produce numerical artifacts much more easily than more diffusive solvers. It is therefore a method that requires a lot of “supervision” by the user.

7.2 The HLL family of solvers

We have seen that the Roe solver splits the Δq (where q is the state vector) into their projections onto the basis of eigenvectors of the Jacobian. Let us write each of these components of Δq as $\Delta_k q$ where k is the index of which eigenvector is meant. Each of the $\Delta_k q$ jumps is advected with its own speed: the characteristic velocity λ_k , which is the eigenvalue of the Jacobian matrix. This is precisely the scenario depicted in Fig. 6.1. Each of these moving jumps is in fact a *wave*, and λ_k is the wave speed. We know from the true solutions of the Riemann problem for the Euler equations (Fig. 6.3) that in reality not every wave is a jump: in particular the expansion wave

is not a jump. In the Roe solver we just approximate each of them to be a jump, based on the semi-linearized set of equations.

Another Riemann solver, which also approximates all waves as jumps, is the HLL Riemann solver and its various derivatives. HLL stands for Harten, Lax and van Leer, who first proposed a method of this kind. The difference between the HLL family of solvers and the Roe solver is the wave propagation speeds λ_k and the wave decompositions of Δq into $\Delta_k q$ are not rigorously derived from the eigenvectors and eigenvalues of some approximation of the Jacobian matrix. Instead some simpler physical arguments are used to “guess” these values. Strictly speaking the HLL family of solvers allow an infinite number of variants because it is partly left to the developer of the HLL method which recipe to take to construct the wave propagation speeds λ_k and the decomposition of Δq into waves $\Delta_k q$.

Let us first look at HLL type solvers in a general sense, and later worry about how exactly to compute the λ_k and the waves $\Delta_k q$.

Suppose we have K waves (i.e. $k = 1 \dots K$) ordered such that $\lambda_k \leq \lambda_{k+1}$. Once we have decomposed Δq into $\Delta_k q$ with wave speeds λ_k we can write the state $q(x, t)$ near the interface $i + 1/2$ as:

$$q(x, t + \Delta t) = q_i + \sum_{k:x > x_{i+1/2} + \lambda_k \Delta t} \Delta_k q_{i+1/2} \equiv q_{i+1} - \sum_{k:x \leq x_{i+1/2} + \lambda_k \Delta t} \Delta_k q_{i+1/2} \quad (7.28)$$

That is: the state $q(x, t)$ near the cell interface is split into $K + 1$ regions. The first and the last have $q = q_i$ and $q = q_{i+1}$ respectively. And in the wave fan region we have $K - 1$ sub-regions of constant state q given by the above equation. For the flux that use for the cell updates we are only interested in $\bar{q}_{i+1/2} \equiv q(x = x_{i+1/2}, t + \Delta t)$, meaning we get:

$$\bar{q}_{i+1/2} \equiv q_i + \sum_{k:\lambda_k \Delta t < 0} \Delta_k q_{i+1/2} \equiv q_{i+1} - \sum_{k:\lambda_k \Delta t \geq 0} \Delta_k q_{i+1/2} \quad (7.29)$$

Strictly speaking, we can now compute a flux $\bar{f}_{i+1/2} = f(\bar{q}_{i+1/2})$ and we are done: we now have a way to express the interface flux of this approximation of the Riemann solution. However, it turns out that a better (more stable and reliable) way to compute the interface flux is to note that one can construct also wave jumps in the flux $\Delta_k f$ and use the property that:

$$\Delta_k f = \lambda_k \Delta_k q \quad (7.30)$$

This property is the equivalent of the Rankine-Hugoniot condition for shocks, but now applied to any wave. It basically guarantees that wave k indeed propagates with speed λ_k . In this way we can now construct the flux at the interface by adding up the jumps, like in the case of the construction of $\bar{q}_{i+1/2}$:

$$\bar{f}_{i+1/2} \equiv f_i + \sum_{k:\lambda_k \Delta t < 0} \lambda_k \Delta_k q_{i+1/2} \equiv f_{i+1} - \sum_{k:\lambda_k \Delta t \geq 0} \lambda_k \Delta_k q_{i+1/2} \quad (7.31)$$

Constructing the interface flux in this way is the basis of the HLL family of approximate Riemann solvers.

The above expression is still first order. To make the scheme second order one can use Eq. (6.56), which we repeat here using the Δ_k notation:

$$\begin{aligned} f_{i+1/2}^{n+1/2} &= \frac{1}{2} (f_{i+1/2,R}^n + f_{i+1/2,L}^n) \\ &\quad - \frac{1}{2} \sum_{k=1 \dots K} [\theta_{k,i+1/2} + \tilde{\phi}_{k,i+1/2} (\epsilon_{k,i+1/2} - \theta_{k,i+1/2})] \Delta_k f_{i+1/2}^n \end{aligned} \quad (7.32)$$

where $\phi_{k,i+1/2}$ is the flux limiter of wave k . This is still precisely the same method of making the scheme higher order as we used for the Roe solver.

To complete our scheme we must now choose λ_k and Δq_k in a clever way. This is the topic of the rest of this section.

7.2.1 The HLL solver

The simplest solver of this kind is the original one. It ignores the middle wave (the mass-advection wave) and decomposes Δq into *two* waves $\Delta_{(-)}q$ and $\Delta_{(+)}q$ which are the forward and backward moving sound waves. This gives us a left state $q_{L,i+1/2} = q_i$, a right state $q_{R,i+1/2} = q_{i+1}$ and a middle state $q_{M,i+1/2}$ which is assumed to be:

$$q_M = \frac{\lambda_{(+)}q_R - \lambda_{(-)}q_L + f_L - f_R}{\lambda_{(+)} - \lambda_{(-)}} \quad (7.33)$$

(where we dropped the $i + 1/2$ for notation convenience). Using the above expressions one can derive that the flux at the interface is then:

$$\bar{f}_{i+1/2} = \begin{cases} f_i & \text{if } \lambda_{(-)} \geq 0 \\ \frac{\lambda_{(+)}f_i - \lambda_{(-)}f_{i+1} + \lambda_{(+)}\lambda_{(-)}(q_{i+1} - q_i)}{\lambda_{(+)} - \lambda_{(-)}} & \text{if } \lambda_{(-)} < 0 < \lambda_{(+)} \\ f_{i+1} & \text{if } \lambda_{(+)} \leq 0 \end{cases} \quad (7.34)$$

So what about the expressions for $\lambda_{(-)}$ and $\lambda_{(+)}$? There is a whole variety of proposed expressions for these values. The simplest is due to Davis (1988):

$$\lambda_{(-),i+1/2} = u_i - a_i \quad \lambda_{(+),i+1/2} = u_{i+1} + a_{i+1} \quad (7.35)$$

where a is the sound speed. But there are many other versions too. See the book by Toro for an in-depth discussion.

One of the main disadvantages of this HLL solver is that it cannot keep contact discontinuities sharp. This is not surprising since we have no middle wave in this scheme.

7.2.2 The HLLC solver

A newer version of the HLL scheme is the HLLC scheme, where the C stands for central wave: this is a method which *does* include the middle wave that is missing from the standard HLL scheme. The general way to construct this scheme is the same as shown above. Instead of 2 waves and 3 regions of constant q we now have 3 waves and 4 regions of constant q . While the general method is the same, the details of the construction of the HLLC method (and its various estimates of the wave speeds) requires some more in-depth discussion. We refer to the book of Toro for these details.

7.3 Source extrapolation methods

One major disadvantage of Riemann solvers in general is that they are, by their structure, less capable of “recognizing” (semi-)static solutions in which pressure gradients are compensated by an external force (typically gravity). To demonstrate what is meant let us take the example of a hydrostatic atmosphere (e.g. Earth’s atmosphere) with small perturbations on it (e.g. the formation of clouds). We want that the unperturbed atmosphere is recognized by the method

in the sense that the method perfectly keeps the static atmosphere intact and does not produce wiggles, or worse: entropy. A hydrostatic atmosphere obeys:

$$\frac{dP}{dz} = -\rho g \quad (7.36)$$

where z is the vertical coordinate and g is the gravitational constant of the atmosphere ($g \simeq 1000\text{cm/s}^2$). Let us discretize this as:

$$\frac{P_{i+1} - P_i}{z_{i+1} - z_i} = -\frac{1}{2}(\rho_{i+1} + \rho_i)g \quad (7.37)$$

and let us construct the solution by choosing $P = K\rho^\gamma$ with K constant (an adiabatic atmosphere) and choosing $\rho(z=0)$ as the density at the base. We can then integrate (for a choice of K) from bottom to some height above the surface using Eq.(7.37). Since this equation is implicit in ρ_{i+1} one must solve for each new ρ_{i+1} using for instance an iteration at each new grid point until Eq.(7.37) is satisfied. We then get a hydrostatic atmosphere that is consistent.

If we insert this into a time-dependent hydro code we want that this hydrocode leaves this solution exactly intact (i.e. that it does not introduce perturbations). For classical numerical hydro schemes of Chapter 5 this is in fact not difficult, especially not if a staggered grid is used. This is because the $\partial P/\partial z$ term is treated as a source term in such methods. We then have two source terms: the $\partial P/\partial z$ and the $-\rho g$ term. If we discretize these two terms in exactly the same way as in Eq.7.37, then both terms cancel out exactly (for this hydrostatic solution) and the hydrostatic solution is kept exactly intact (to machine precision). We can then safely study tiny perturbations of this atmosphere without the worry that we may in fact be involuntarily studying the intrinsic noise of the method instead.

For Riemann solvers, however, this is not so easy. It is a fundamental property of these methods to include the $\partial P/\partial z$ term in the advection part, while the gravity force remains a source term (and can not be treated in the advection part). The pressure gradient force and the gravity force are therefore treated in fundamentally different ways and one cannot guarantee that they will *exactly* cancel for hydrostatic solutions.

To solve this problem Eulderink & Mellema (1995, A&ASup 110, 587) introduced the concept of *spatial flux extrapolation*. A very similar, but easier method was proposed by LeVeque (1998, J.Comp.Phys. 146, 346). We will use a combination of both formalisms here.

Consider the generalized hyperbolic equation with a source term:

$$\partial_t q(x, t) + \partial_x f(q(x, t)) = s(x) \quad (7.38)$$

The traditional way to do operator splitting is to first solve $\partial_t q(x, t) + \partial_x f(q(x, t)) = 0$ for one time step using for instance a Riemann solver and then solve $\partial_t q(x, t) = s(x)$ for the same time step (by simply adding the source). This gives us the problem of not recognizing steady states. A steady state is a solution of the equation

$$\partial_x f(q(x, t)) = s(x) \quad (7.39)$$

The *source extrapolation* method for time-dependent hydrodynamics is inspired on this stationary equation. At the start of each time step we construct a *subgrid model* $q(x, t = t_n)$ (for $x_{i-1/2} < x < x_{i+1/2}$) where $q(x = x_i, t = t_n) = q_i$ such that within the cell $q(x, t = t_n)$ is a solution of 7.39. To make things simpler we assume that $s(x)$ is constant within a cell, so we get:

$$\partial_x f(q(x, t)) = s_i \quad \text{for } x_{i-1/2} < x < x_{i+1/2} \quad (7.40)$$

From this equation we could in principle solve for $q(x, t)$ within cell i and thereby obtain values of q at the left and right sides of each interface: $q_{L,i+1/2}$ and $q_{R,i+1/2}$ (we come back in a minute to this). These are then the values we put into a Riemann solver to produce our interface flux $f_{i+1/2}$ which we use for the state update. It should be noted, however, that since now the states on both sides of the interface are no longer spatially constant (they follow the subgrid model). So in principle we have a *generalized Riemann problem* on the cell interface, in which the states are not constant. Such generalized Riemann problems are very difficult to solve. So instead, we simply ignore the fact that the states are strictly speaking not constant on both sides, and simply use $q_{L,i+1/2}$ and $q_{R,i+1/2}$ as the two states of a classical Riemann problem which we then solve with our favorite method.

Now let's come back to the solution of Eq. 7.40. If $f(q)$ is a linear function of q , say $f(q) = uq$ where u is a velocity, then the solution of Eq. 7.40 is simple:

$$q_{R,i-1/2} = q_i - \frac{1}{2}\Delta x_i s_i / u_i \quad (7.41)$$

$$q_{L,i+1/2} = q_i + \frac{1}{2}\Delta x_i s_i / u_i \quad (7.42)$$

This works fine as long as $u \neq 0$. Once u approaches zero we are in trouble. Moreover, if $f(q)$ is a *non-linear* function of q then we can write:

$$f_{R,i-1/2} = f(q_i) - \frac{1}{2}\Delta x_i s_i \quad (7.43)$$

$$f_{L,i+1/2} = f(q_i) + \frac{1}{2}\Delta x_i s_i \quad (7.44)$$

which is fine, but solving $q_{R,i+1/2}$ from a given flux $f_{R,i+1/2}$ is non-trivial, and in the case of the Euler equations typically has two different solutions, or one single solution or no solutions. It is therefore often unpractical to try to determine the $q_{L/R,i+1/2}$ from $f_{L/R,i+1/2}$. It turns out, however, that for Riemann solvers that are based entirely on propagating waves $\Delta_k f = \lambda_k \Delta_k q$ where Δ_k is the k -th wave and λ_k is its propagation speed, one can use $\Delta f_{i+1/2} = f_{R,i+1/2} - f_{L,i+1/2}$ as input to the wave-decomposition routine while keeping $q_{L,i+1/2} = q_i$ and $q_{R,i+1/2} = q_{i+1}$ as the primitive variables used to compute the eigenvectors and eigenvalues. In particular for the Roe solver this hybrid approach works well.

Now how does this solve the problem of recognizing a steady-state solution? The trick is that if the linear subgrid model described in Eq. (??) is sufficiently accurate, then one will obtain for the steady state solution:

$$f_{L,i+1/2} = f_{R,i+1/2} \quad (7.45)$$

meaning that

$$\Delta f_{i+1/2} = 0 \quad (7.46)$$

If we now use Eq.(7.32) then we get:

$$f_{i+1/2}^{n+1/2} = \frac{1}{2}(f_{i+1/2,R}^n + f_{i+1/2,L}^n) \quad (7.47)$$

because by virtue of $\Delta f_{i+1/2} = 0$ the entire Riemann solver part drops out of the equation now. What is left is:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x} [f(q_{i+1}^n) - f(q_{i-1}^n)] + \Delta t s_i^n \quad (7.48)$$

So if we construct our steady state initial condition such that

$$\frac{f(q_{i+1}) - f(q_{i-1})}{2\Delta x} + s_i = 0 \quad (7.49)$$

then the method finally recognizes the static solution down to machine precision.

7.4 Employing slope limiters *before* the Riemann solver step

So far we have studied solvers in which we could clearly distinguish two or three wave families. We were then able to apply flux limiters (or equivalently, slope limiters) to each wave (cf. Eq.7.32). This method ensures that the slope in each wave mode is taken into account and therefore gives second order accuracy in space.

Another method of making Riemann solvers higher-order is to apply slope limiters (not flux limiters) *before* the Riemann problem is solved. The trick is to extrapolate the cell-centered primitive variables ρ, P, \vec{u} to the cell-walls using a linear subgrid model. We can also extrapolate the conserved quantities $\rho, \rho\vec{u}, \rho e_{\text{tot}}$, whichever produces the best results. This then gives us the left and right states at the cell walls ($q_{L,i+1/2}$ and $q_{R,i+1/2}$), which defines a Riemann problem which we can then solve using any solver we want. Contrary to the flux limiter recipe of Eq.7.32 this slope limiter method can also be applied to Riemann solvers that do not approximate the problem into jump-like waves. This is therefore a way to make the original Godunov solver higher order.

This method is in a way similar to Section 7.3, but there is an essential difference: here we use neighboring points to compute the slope of the linear subgrid model (instead of the source function). We therefore get non-zero slopes also for cases without source. We can employ all the machinery of Chapter 4 to produce the best possible linear subgrid model using e.g. MINMOD or SUPERBEE slope limiters.

One major problem that we encounter if we simply apply this method as-is, is that it quickly becomes unstable. It turns out (see Toro's book) that this instability problem can be elegantly solved by producing adapted left- and right interface states $\tilde{q}_{L,i+1/2}$ and $\tilde{q}_{R,i+1/2}$ from the interface states $q_{L,i+1/2}$ and $q_{R,i+1/2}$ that come out of the slope limiter method by advancing these half a time step in time in the following way:

$$\tilde{q}_{L,i+1/2} = q_{L,i+1/2} + \frac{1}{2} \frac{\Delta t}{\Delta x} [f(q_{R,i-1/2}) - f(q_{L,i+1/2})] \quad (7.50)$$

$$\tilde{q}_{R,i+1/2} = q_{R,i+1/2} + \frac{1}{2} \frac{\Delta t}{\Delta x} [f(q_{R,i+1/2}) - f(q_{L,i+3/2})] \quad (7.51)$$

(see book by Toro, but beware of different notation). Note that the left one is updated using $q_{R,i-1/2}$ and $q_{L,i+1/2}$, while the right one is updated using $q_{R,i+1/2}$ and $q_{L,i+3/2}$. These equations are saying that the subgrid model *within each cell individually* is advanced half a time step. Using this method we obtain a stable higher-order scheme.

The method is now completed by inserting $\tilde{q}_{L,i+1/2}$ and $\tilde{q}_{R,i+1/2}$ into our favorite Riemann solver and obtaining the interface flux with which we update the cell centered state variables.

NOTE: The use of the slope limiters before the Riemann step excludes the use of the source extrapolation method described in Section 7.3, and vice versa. If one wishes to use the source extrapolation method then only the flux limiter method after the Riemann step (Eq. 7.32) can be used.

7.5 The PPM Method

A very well-known Riemann solver method is the "Piecewise Parabolic Method" (Colella & Woodward 1984, J.Comp.Phys. 54, 174). It was designed before many of the above described methods were developed, and therefore it lies a bit off from the main track as described above, but in large part it follows (and indeed pioneered) the same ideas. It has proved to be a pretty

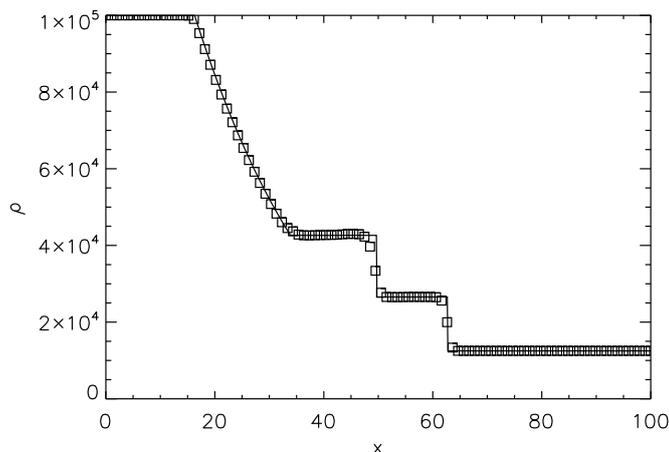


Figure 7.1. The result of the Roe solver with superbee flux limiter on a Sod shocktube test with $\rho_L = 10^5$, $P_L = 1$, $u_L = 0$, $\rho_R = 1.25 \times 10^4$, $P_R = 0.1$, $u_R = 0$ and $\gamma = 7/5$. Solid line: analytic solution; symbols: Roe solver.

powerful method and is still very often used. We will describe it very briefly here, but we will not go into details because the method is rather complex and has many fine-tuning aspects.

The main idea of this Riemann solver is to use a reconstruction step *before* the Riemann step (see Section 7.4). But while we used linear reconstruction in Section 7.4, the PPM method uses quadratic reconstruction. In other words: starting from the cell-center values of the primitive variables ρ , p and \vec{u} each cell subgrid model is a parabola. In the PPM method care is taken that no overshoots happen (TVD scheme) and that in general the reconstruction is well-behaved. Using these parabolic subgrid models, the primitive variables on both sides of each of the interfaces can be calculated. We know from Section 7.4 that if we directly insert these values into our Riemann problem solver, we get an unstable scheme. The PPM method solves this using an approximation. It finds the fastest moving left- and right- characteristics and makes an average of the primitive variables over these domains. This is the PPM version of the half-step-interface-update described in Section 7.4. It is less mathematically rigorous, but in practice it works.

Now these left- and right- interface values can be inserted in a Riemann solver code. Again, as in Section 7.3, the solution to the true (generalized) Riemann problem in this case is very complex and no analytic solutions exist of generalized Riemann problems with parabolic spatial dependency of the variables on both sides. Therefore, as before, the approximation is made to solve the classical Riemann problem, with constant states on both sides, even though we know that the states on both sides are not constant.

We refer to the original paper by Colella & Woodward 1984 for details of the method and how the method is fine-tuned.

7.6 Code testing: the Sod shock tube tests

The Sod shock tube test of Section 6.3.2 can be used to test the performance of our computer program. We leave it to the reader to test the algorithms of the previous chapter on this kind of test. Here we merely shock the performance of the Roe solver on a test with $\rho_L = 10^5$, $P_L = 1$, $u_L = 0$, $\rho_R = 1.25 \times 10^4$, $P_R = 0.1$, $u_R = 0$ and $\gamma = 7/5$. The result is shown in Fig. 7.1.

Appendix A

The index notation of tensor calculus

The index notation is a very powerful way of dealing with vectors, matrices and tensors. The topic of tensor mathematics and the index notation is far too far-reaching to cover in depth in this short appendix. Here we only cover the absolute basics.

A vector (i.e. an object with spatial direction) is denoted in normal vector notation as \vec{u} , or sometimes also as \mathbf{u} (in this lecture we write \vec{u}). A vector is a tensor of rank 1. In index notation we refer to the individual components of the vector by writing it as u_i , where i is an index that can be chosen to be either 1, 2 or 3.

A matrix is often written as \mathbf{M} . It has two indices, i.e. 9 components (if related to true space, i.e. if it is a transformation in space like a rotation). A matrix is a tensor of rank 2. In index notation it is written as M_{ij} , where the indices i and j are 1, 2 or 3. The unit matrix is δ_{ij} , which is defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (\text{A.1})$$

which is called the *Kronecker delta function* in tensor mathematics.

A matrix is not the only kind of second rank tensor. The pressure tensor or stress tensor in hydrodynamics is also one, and in many respects that kind of object behaves very similar to a matrix. But formally it is not entirely the same. This can not be seen as long as one works in cartesian coordinates: then the pressure/stress tensor behaves exactly like a matrix. But in general coordinate transformations, in which the local basis is no longer cartesian, the two kinds of objects behave differently. For details see the tensor syllabus mentioned above, or other literature on tensor calculus. For completeness: a matrix is *contravariant* in the first index, and *covariant* in the second (or vice-versa) while the stress tensor is contravariant in both indices. Again, as long as one remains in cartesian coordinates (i.e. x , y and z or rotations thereof) this distinction is irrelevant.

A very important thing of the index notation is the *summation convention*. This convention says that *if an index appears twice in the same product, then it is automatically assumed that this is being summed over, from 1 to 3*. So the following expression is a matrix multiplication with a vector:

$$M_{ik}u_k \equiv \sum_{k=1}^3 M_{ik}u_k \equiv \sum_{m=1}^3 M_{im}u_m \equiv M_{im}u_m \quad (\text{A.2})$$

and all four expressions are (by the summation convention) the same.

The inner-product of a vector \vec{u} with \vec{w} is then written as:

$$\vec{u} \cdot \vec{w} = u_i w_i \quad (\text{A.3})$$

(Note: here u_i and w_k are not the (u, v, w) components of \vec{u} , as used often in this lecture, but they are the i -th component of \vec{u} and the k -th component of vector \vec{w} respectively here.)

The divergence of a vector field is written as:

$$\nabla \cdot \vec{u} = \nabla_i u_i \equiv \partial_i u_i \quad (\text{A.4})$$

The two ways of writing (with ∇_i or ∂_i) are both often used, and I will use both ways of writing in this lecture without notice. The divergence of a tensor field $\rho \vec{u} \vec{u}$ (in index notation $\rho u_i u_j$) is:

$$\nabla \cdot (\rho \vec{u} \vec{u})|_{\text{component } k} = \nabla_i (\rho u_i u_k) \quad (\text{A.5})$$