Clawpack

Christophe Ancey

A tutorial for hydraulic applications

C. Ancey,

EPFL, ENAC/IIC/LHE,

Ecublens, CH-1015 Lausanne, Suisse

christophe.ancey@epfl.ch, lhe.epfl.ch

Clawpack tutorial / C. Ancey

version 1.1 from 17 June 2020, Lausanne

Picture credit. **Cover page** : Val Ferret (VS). **Table of contents**: William Turner, (). **Chapter 1**: William Turner, *The Falls of the Rhine at Schaffhausen* (Courtauld collection, London). **Chapter 2**: William Turner, *dawn after the wreck*(Courtauld collection, London). **Chapter 3**: William Turner, *Lausanne (cathedral and bridge)* (Tate Britain, London). **Chapter 4**: William Turner, *The fall of anarchy* (Tate Britain, London). **Chapter 5**: William Turner, *The Dogano, San Giorgio, Citella, from the Steps of the Europa hotel* (Tate Britain, London). **Chapter 6**: William Turner, *Sea disaster* (Tate Britain, London). **Chapter 7**: William Turner, *The Parting of Hero and Leander* (National Gallery, London). **Bibliography**: William Turner, *Lausanne (War: The Exile and the Rock Limpet* (Tate Britain, London). **Index**: William Turner, *Lausanne (Two Figures on a Beach with a Boat* (Tate Britain, London).

Table of Contents

Foreword

This tutorial is primarily based on the material written by Randall LeVeque and his collaborators

References

Hyperbolic equation theory is described in a few books, including:

- Numerical Methods for Conservation Laws (LeVeque, 1992)
- Finite Volume Methods for Hyperbolic Problems (LeVeque, 2002)
- Riemann Problems and Jupyter Solutions (Ketcheson *et al.*, 2020)

Online material

Clawpack can be downloaded from its official website www.clawpack.org. It can also be downloaded from github: github.com/clawpack.

The new book based on jupyter notebooks by David Ketcheson et al. offers a convenient way of learning clawpack and the theoretical fundamentals through a series of examples. Many of these examples will be used here. The html version of this book is available online.

Jupyter notebook viewers: cocalc.com and nbviewer.jupyter.org.

Additional ressources

- Shaltop
- Basilisk
- Iber
- Basement
- OpenFoam

Notation

The notation used in this tutorial differs from that used by Randall LeVeque. I use the classic tensorial notation: vectors and tensors are denoted by boldface symbols. I also use the operator · to refer to the

contracted product ("produit une fois contracté" in French) and : for the double-contracted product: for tensors $\boldsymbol{A}$ and $\boldsymbol{B}$ whose matrix representation is A_{ij} (i row, j column index) and B_{ij}, respectively, $\boldsymbol{v}$ and $\boldsymbol{w}$ two vectors of coordinates v_i and w_i in a given basis, then

$$\begin{aligned}
\boldsymbol{A} \cdot \boldsymbol{B} &= \sum_j A_{ij} B_{jk} \\
\boldsymbol{A} : \boldsymbol{B} &= \sum_{i,j} A_{ij} B_{ji} \\
\boldsymbol{A} \cdot \boldsymbol{v} &= \sum_j A_{ij} v_j \\
\boldsymbol{w} \cdot \boldsymbol{v} &= \sum_i w_i v_i
\end{aligned}$$

CHAPTER 1

Hyperbolic equations

Let us start with linear hyperbolic systems. Nonlinear equations are more complex, but the solutions to the Riemann problem have a similar structure to that exhibited by linear systems. Furthermore, finite-volume numerical solvers involve approximate (linearised) solutions to this Riemann problem.

1.1 Riemann problems for linear hyperbolic equations

1.1.1 Linear system

For one-dimensional problems, a linear hyperbolic equation is defined by an equation of the form

$$\frac{\partial}{\partial t}\boldsymbol{q} + \boldsymbol{A}\cdot\frac{\partial}{\partial x}\boldsymbol{q} = \boldsymbol{S}, \tag{1.1}$$

where $\boldsymbol{q}$ is a vector with m components representing the unknowns, $\boldsymbol{A}$ is a $m \times m$ matrix whose eigenvalues are assumed to be real and distinct, and $\boldsymbol{S}$ is a vector (of dimension m) called the *source term*, x is the spatial dimension, and t is time. For the moment, we assume that $\boldsymbol{S} = 0$ (the equation is said to be *homogenous*). The matrix $\boldsymbol{A}$ has m real eigenvalues λ_i, which are associated with m left $\boldsymbol{v}_i$ and m right eigenvectors $\boldsymbol{w}_i$:

$$\boldsymbol{A}\cdot\boldsymbol{w}_i = \lambda_i\boldsymbol{w}_i \text{ and } \boldsymbol{v}_i\cdot\boldsymbol{A} = \lambda_i\boldsymbol{v}_i. \tag{1.2}$$

In the following, the eigenvalues are ranked in ascending order: $\lambda_1 < \lambda_2 \cdots < \lambda_m$.

1.1.2 Diagonalization

If we multiply Eq. (1.1) by $\boldsymbol{v}_i$, we obtain:

$$\boldsymbol{v}_i\cdot\frac{\partial}{\partial t}\boldsymbol{q} + \boldsymbol{v}_i\cdot\boldsymbol{A}\cdot\frac{\partial}{\partial x}\boldsymbol{q} = \boldsymbol{v}_i\cdot\boldsymbol{S}. \tag{1.3}$$

We introduce *characteristic variable* or *Riemann variable* (also called *Riemann invariant* when $\boldsymbol{S} = 0$):

$$r_i = \boldsymbol{v}_i\cdot\boldsymbol{q} \text{ and the vector } \boldsymbol{r} = (r_1, \cdots, r_n), \tag{1.4}$$

and the diagonal matrix $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \cdots \lambda_m)$. With this notation, we transform Eq. (1.3) into a system of m uncoupled equations:

$$\frac{\partial}{\partial t}\boldsymbol{r} + \boldsymbol{\Lambda}\cdot\frac{\partial}{\partial x}\boldsymbol{r} = \boldsymbol{L}\cdot\boldsymbol{S}, \tag{1.5}$$

where $\boldsymbol{L}$ is a matrix whose rows are made of the left eigenvectors: $\boldsymbol{L} = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m]^T$.

Similarly to $\boldsymbol{L}$, we define $\boldsymbol{R}$ is a matrix whose columns are made of the right eigenvectors: $\boldsymbol{R} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m]$. The following relationships hold true

$$\boldsymbol{A} \cdot \boldsymbol{R} = \boldsymbol{R} \cdot \boldsymbol{\Lambda}, \tag{1.6}$$
$$\boldsymbol{L} \cdot \boldsymbol{A} = \boldsymbol{\Lambda} \cdot \boldsymbol{L}. \tag{1.7}$$

We also have:

$$\boldsymbol{A} = \boldsymbol{R} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{R}^{-1}, \tag{1.8}$$
$$\boldsymbol{A} = \boldsymbol{L}^{-1} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{L}. \tag{1.9}$$

Because when taking the transpose of $\boldsymbol{v}_i \cdot \boldsymbol{A} = \lambda_i \boldsymbol{v}_i$ we have

$$(\boldsymbol{v}_i \cdot \boldsymbol{A})^T = \boldsymbol{A}^T \cdot \boldsymbol{v}_i^T = \lambda_i \boldsymbol{v}_i^T, \tag{1.10}$$

the left eigenvectors $\boldsymbol{v}_i$ of $\boldsymbol{A}$ is also the right eigenvector of $\boldsymbol{A}^T$.

Multiplying Eq. (1.6) by $\boldsymbol{L}$ and Eq. (1.7) by $\boldsymbol{R}$, we get

$$\boldsymbol{L} \cdot \boldsymbol{A} \cdot \boldsymbol{R} = \boldsymbol{L} \cdot \boldsymbol{R} \cdot \boldsymbol{\Lambda} = \boldsymbol{\Lambda} \cdot \boldsymbol{L} \cdot \boldsymbol{R}. \tag{1.11}$$

When two matrice $\boldsymbol{M}$ and $\boldsymbol{D}$ (where $\boldsymbol{D}$ is diagonal) satisfy $\boldsymbol{D} \cdot \boldsymbol{M} = \boldsymbol{M} \cdot \boldsymbol{D}$, then $\boldsymbol{M}$ is diagonal. This means here that $\boldsymbol{M} = \boldsymbol{L} \cdot \boldsymbol{R}$ is diagonal. There is no unique choice as any multiple of an eigenvector is also an eigenvector. We can define the right eigenvectors such that:

$$\boldsymbol{R} = \boldsymbol{L}^{-1}. \tag{1.12}$$

A geometrical interpretation of $\boldsymbol{R}$ and $\boldsymbol{L}$ is the following: as we $\boldsymbol{R} \cdot \boldsymbol{L} = \boldsymbol{L}^{-1} \cdot \boldsymbol{L} = \mathbf{1}$ (where $\mathbf{1}$ denotes the identity matrix), then the left and right eigenvectors are orthogonal two by two: $\boldsymbol{v}_i \cdot \boldsymbol{w}_i \neq 0$ and $\boldsymbol{v}_i \cdot \boldsymbol{w}_k = 0$ for $k \neq i$.

In practice, we determine the right eigenvectors $\boldsymbol{w}_i$. The left eigenvectors are the right eigenvectors of the transpose of $\boldsymbol{A}$. The resulting matrices $\boldsymbol{R}$ and $\boldsymbol{L}$ satisfy: $\boldsymbol{R} \cdot \boldsymbol{L}^T = \text{diag}(\boldsymbol{w}_k \cdot \boldsymbol{v}_k)_{1 \leq k \leq 1}$. Furthermore, by normalizing the right eigenvectors ($\tilde{\boldsymbol{w}}_i = \boldsymbol{w}_i / |\boldsymbol{w}_i|$), we can enforce $\boldsymbol{L} = \boldsymbol{R}^{-1}$, a relationship that turns out to be helpful thereafter.

Example Let us consider the 3×3 matrix

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 2 & 8 & 2 \end{pmatrix}.$$

The eigenvalues are $\lambda_1 = -3$, $\lambda_2 = -1$, $\lambda_3 = 12$ associated with the right eigenvectors

$$\boldsymbol{w}_1 = \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix}, \ \boldsymbol{w}_2 = \begin{pmatrix} -3 \\ 0 \\ 2 \end{pmatrix}, \ \boldsymbol{w}_3 = \begin{pmatrix} 11 \\ 26 \\ 23 \end{pmatrix},$$

and the left eigenvectors

$$\boldsymbol{v}_1 = \begin{pmatrix} 4 \\ -7 \\ 6 \end{pmatrix}, \ \boldsymbol{v}_2 = \begin{pmatrix} 5 \\ -3 \\ 1 \end{pmatrix}, \ \boldsymbol{v}_3 = \begin{pmatrix} 2 \\ 4 \\ 3 \end{pmatrix}.$$

1.1.3 Characteristic form and solution to the Cauchy problem for homogenous equations

Each uncoupled equation of the system (1.3) can be put into a characteristic form

$$\frac{\partial r_i}{\partial t} + \lambda_i \frac{\partial r_i}{\partial x} = \boldsymbol{v}_i \cdot \boldsymbol{S} \Leftrightarrow \frac{\mathrm{d}r_i}{\mathrm{d}t} = \boldsymbol{v}_i \cdot \boldsymbol{S} \text{ along the straight line } \frac{\mathrm{d}x}{\mathrm{d}t} = \lambda_i. \tag{1.13}$$

For a homogenous problem, this means that r_i is constant along the line $x = \lambda_i t + x_0$. If we know the initial value $\boldsymbol{q}_0 = \boldsymbol{q}(x, t = 0)$, then we know the initial condition for $\boldsymbol{r}$: $\boldsymbol{r}_0 = \boldsymbol{r}(x, t = 0) = \boldsymbol{L} \cdot \boldsymbol{q}_0$. For a homogenous equation, the solution to Eq. (1.13) is

$$r_i(x, t) = r_{i,0}(x - \lambda_i t), \tag{1.14}$$

and thus the solution to the initial-value (Cauchy) problem is

$$\begin{aligned} \boldsymbol{q}(x,t) &= \boldsymbol{R} \cdot \boldsymbol{r} = \sum_{i=1}^{m} r_i(x,t)\boldsymbol{w}_i, & (1.15)\\ &= \sum_{i=1}^{m} r_{0,i}(x - \lambda_i t)\boldsymbol{w}_i, & (1.16)\\ &= \sum_{i=1}^{m} (\boldsymbol{v}_i \cdot \boldsymbol{q_0}\ (x - \lambda_i t))\boldsymbol{w}_i. & (1.17) \end{aligned}$$

The solution $\boldsymbol{q}$ is a combination of the right eigenvectors. In other words, the initial conditions propagate along the directions $\boldsymbol{w}_i$. This propagation is a consequence of the travelling-wave structure. Indeed, the linear hyperbolic system (1.1) is invariant to the travelling wave group. If we seek a solution in the form $\boldsymbol{s}(x, t) = \boldsymbol{s}(\xi)$ where $\xi = x - at$ and a is the wave velocity, then Eq. (1.1) leads to:

$$-a \frac{\mathrm{d}}{\mathrm{d}\xi} \boldsymbol{s} + \boldsymbol{A} \cdot \frac{\mathrm{d}}{\mathrm{d}\xi} \boldsymbol{s} = 0. \tag{1.18}$$

This shows that $\boldsymbol{s}'$ is an eigenvector and a must be one of the eigenvalues, say λ_i. Substituting the Cauchy solution Eq. (1.15) into Eq. (1.18) shows that this condition is met. For strictly hyperbolic systems (i.e., when all eigenvalues are real and different), the right eigenvectors form a basis, and the decomposition (1.15) is unique.

The solution to the Cauchy problem is the superposition of m waves, each is advected independently at the velocity λ_i along the direction $\boldsymbol{w}_i$, with no change in shape when the system is homogenous.

1.1.4 Simple wave

When the initial conditions are constant for all but one value k

$$r_{i,0}(x) = r_i \text{ for } i \neq k \text{ and } r_{k,0}(x) = r_{k,0}(x - \lambda_k t), \tag{1.19}$$

then the solution

$$\boldsymbol{q}(x,t) = \boldsymbol{q}_0(x - \lambda_k t) = r_{k,0}(x - \lambda_k t)\boldsymbol{w}_k + \sum_{i \neq k} r_i \boldsymbol{w}_i \tag{1.20}$$

is called a *simple wave*. Propagation concerns the direction k alone.

1.1.5 Riemann problem: definition

A Riemann problem is an initial-value problem for which the initial value is piecewise constant with a single jump discontinuity at some point, by default at $x = 0$:

$$\boldsymbol{q} = \begin{cases} \boldsymbol{q}_l \text{ if } x < 0, \\ \boldsymbol{q}_r \text{ if } x > 0. \end{cases} \tag{1.21}$$

Because the right eigenvectors form a basis, we can decompose $\boldsymbol{q}_l$ and $\boldsymbol{q}_r$ in this basis

$$\boldsymbol{q}_l = \sum_{i=1}^{m} r_{l,i}\boldsymbol{w}_i \text{ and } \boldsymbol{q}_r = \sum_{i=1}^{m} r_{r,i}\boldsymbol{w}_i \tag{1.22}$$

and each Riemann variable satisfies the initial condition

$$r_{i,0} = \begin{cases} r_{i,l} \text{ if } x < 0, \\ r_{i,r} \text{ if } x > 0. \end{cases} \tag{1.23}$$

Each initial discontinuity propagates with speed λ_i:

$$r_i = \begin{cases} r_{i,l} \text{ if } x < \lambda_i t, \\ r_{i,r} \text{ if } x > \lambda_i t. \end{cases} \tag{1.24}$$

Let us consider a point P at (x, t). We refer to I as the maximum index i for which $x > \lambda_i t$. As illustrated by the example of Fig. 1.1, we can decompose the solution into two parts, either reflecting the left or right initial conditions

$$\boldsymbol{q} = \sum_{i=1}^{I} r_{i,r}\boldsymbol{w}_i + \sum_{i=I+1}^{m} r_{i,l}\boldsymbol{w}_i. \tag{1.25}$$

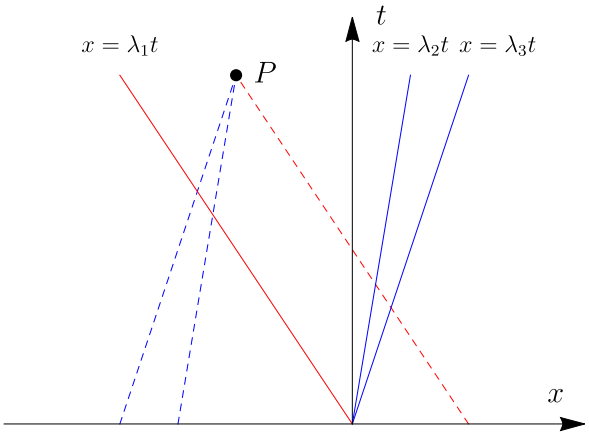


Figure 1.1 Characteristic lines emanating from the origin point (solid lines) and joining P (dashed lines). In this figure, we have $I = 1$: P is on the right of the first characteristic curve $x = \lambda_1 t$, and on the left of the two others. Here we have $\boldsymbol{q} = r_{1,r}\boldsymbol{w}_1 + r_{2,l}\boldsymbol{w}_2 + r_{3,l}\boldsymbol{w}_3$.

When crossing the ith characteristic, there is a jump from $r_{i,l}$ to $r_{i,r}$ while the other coefficients remain constant. As illustrated in Fig. 1.1, the plane is split into different wedges separated by characteristic lines oriented by $\boldsymbol{w}_i$. Across the ith characteristic, the solution $\boldsymbol{q}$ experiences a jump:

$$\Delta\boldsymbol{q} = (r_{i,r} - r_{i,l})\boldsymbol{w}_i, \tag{1.26}$$

which can be written as

$$\Delta \boldsymbol{q} = \alpha_i \boldsymbol{w}_i \text{ with } \alpha_i = r_{i,r} - r_{i,l}. \tag{1.27}$$

For linear hyperbolic systems, a strategy of solving the Riemann problem is to decompose the initial jump $\Delta \boldsymbol{q}_r - \boldsymbol{q}_l$ in the right eigenvector basis

$$\Delta \boldsymbol{q} = \sum_{i=1}^{m} \alpha_i \boldsymbol{w}_i, \tag{1.28}$$

which requires determining the coefficient α_i

$$\boldsymbol{R} \cdot \boldsymbol{\alpha} = \Delta \boldsymbol{q} \Rightarrow \boldsymbol{\alpha} = \boldsymbol{R}^{-1} \cdot \Delta \boldsymbol{q} = \boldsymbol{L} \cdot \Delta \boldsymbol{q}. \tag{1.29}$$

As this decomposition is central to Clawpack, we introduce the wave

$$\boldsymbol{W}_i = \alpha_i \boldsymbol{w}_i. \tag{1.30}$$

The solution to the Riemann problem can thus be written

$$\Delta \boldsymbol{q} = \sum_{i=1}^{m} \alpha_i \boldsymbol{w}_i, \tag{1.31}$$

$$\boldsymbol{q} = \boldsymbol{q}_l + \sum_{i=1}^{I} \boldsymbol{W}_i, \tag{1.32}$$

$$\boldsymbol{q} = \boldsymbol{q}_r - \sum_{i=I+1}^{m} \boldsymbol{W}_i, \tag{1.33}$$

$$\boldsymbol{q} = \boldsymbol{q}_l + \sum_{i=1}^{m} H(x - \lambda_i t) \boldsymbol{W}_i, \tag{1.34}$$

where H is the Heaviside function. Equation (1.32) can also be written

$$\boldsymbol{q} = \boldsymbol{q}_l + \sum_{\lambda_i < x/t} \boldsymbol{W}_i, \tag{1.35}$$

which can be interpreted as follows (see an example in Fig. 1.1): at time t and position x, the state $\boldsymbol{q}$ is the left initial state to which contributions from the right initial state are added if this point is on the right of the characteristic $x = \lambda_i t$ (that is, when $x > \lambda_i t$).

1.1.6 Phase plane representation for $m = 2$ equations

For a linear system of two hyperbolic equations, the solution consists of two discontinuities $x = \lambda_1 t$ and $x = \lambda_2 t$, and within the wedge formed by these two discontinuities there is an intermediate (constant) state

$$\boldsymbol{q}_* = r_{1,r} \boldsymbol{w}_1 + r_{r,l} \boldsymbol{w}_2. \tag{1.36}$$

The jump from $\boldsymbol{q}_l$ to $\boldsymbol{q}_*$ is $(r_{1,r} - r_{1,l})\boldsymbol{w}_1$, while the jump from $\boldsymbol{q}_r$ to $\boldsymbol{q}_*$ is $(r_{2,r} - r_{2,l})\boldsymbol{w}_2$. In other words, starting from the left state $\boldsymbol{q}_l$, we follow the direction $\boldsymbol{w}_l$ to reach the intermediate state $\boldsymbol{q}_*$, and finally the direction $\boldsymbol{w}_2$ to reach the right state $\boldsymbol{q}_r$, shown by Fig. 1.2.

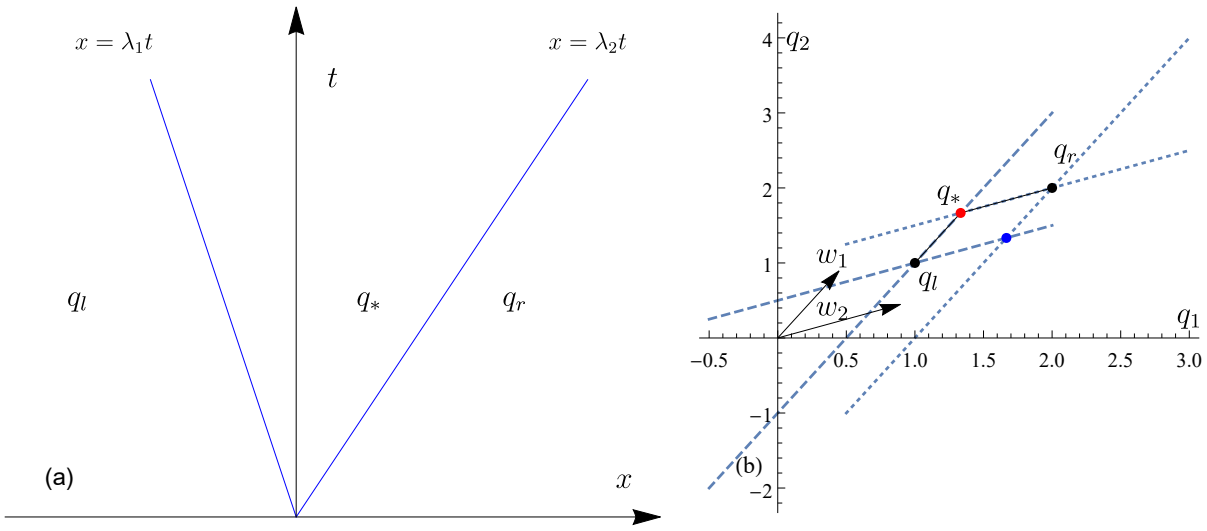


Figure 1.2 (a) Solution to the Riemann problem in the $x - t$ plane. (b) Phase plane representation.

1.2 Nonlinear scalar problem

Let us consider the nonlinear hyperbolic equation (nonlinear advection equation):

$$\frac{\partial}{\partial t}q + \frac{\partial}{\partial x}f(q) = s(q,\, x,\, t) \Leftrightarrow \frac{\partial}{\partial t}q + c(q)\frac{\partial}{\partial x}q = S(q,\, x,\, t), \tag{1.37}$$

where f is the flux function (a function of q, and possibly of x and t), q is the unknown, S is the source term, and $c = f'(q)$ is the celerity. We assume that the celerity is an increasing function of q, which implies that the flux function is convex ($f'' > 0$). Nonconvex functions are possible, but they lead to difficulties that we will not address here.

1.2.1 Characteristic form

Equation (1.37) can be put in the characteristic form

$$\frac{\mathrm{d}}{\mathrm{d}t}q = s(q,\, x,\, t) \text{ along } + \frac{\mathrm{d}x}{\mathrm{d}t} = c(q). \tag{1.38}$$

When the source term is zero ($S = 0$), then q is constant along the characteristic curve, which is therefore a straight line of slope c.

1.2.2 Rankine-Hugoniot equation

As the celerity $c(q)$ is function of q, the characteristic curves are not parallel like in the linear case, and may intersect. As multivalued functions are not possible (this would otherwise break the assumptions of smoothness and uniqueness of the solution), then a shock takes place and connects two continuous branches of the solution. By taking a control volume around the shock position $x = s(t)$, we can deduce that its velocity $\dot{s}$ is given by the Rankine-Hugoniot equation:

$$\dot{s} = \frac{[\![f(q)]\!]}{[\![q]\!]}, \tag{1.39}$$

where the double brackets denote the flux jump across the shock wave

$$[\![f(u)]\!] = \lim_{x \to s,\, x>s} f(q) - \lim_{x \to s,\, x<s} f(q).$$

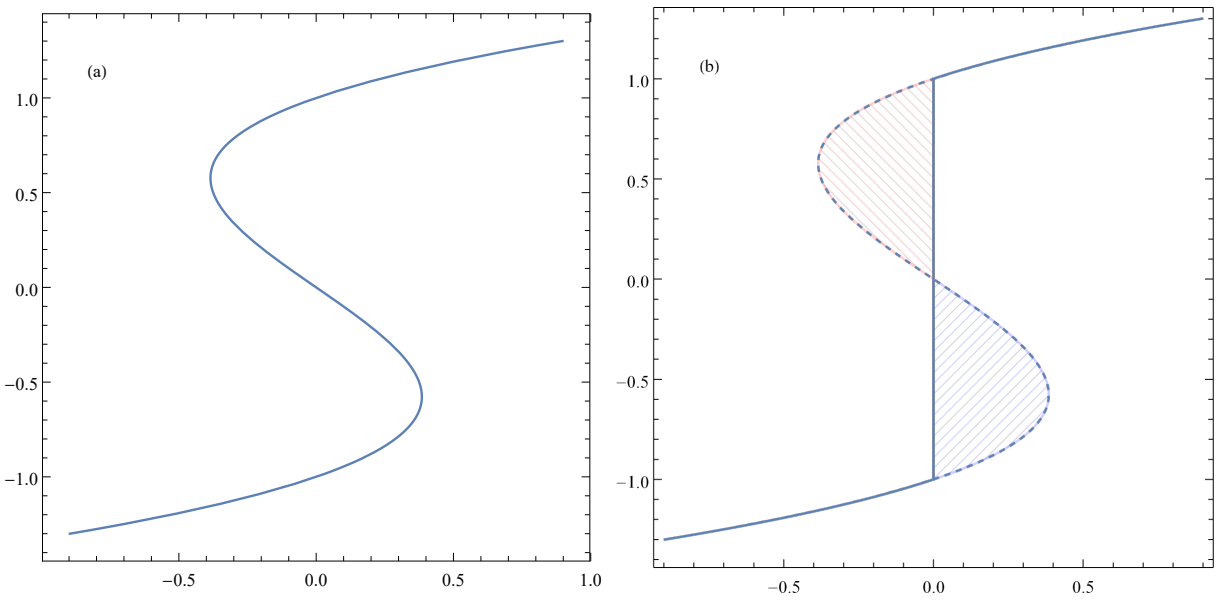


Figure 1.3 (a) multivalued function. (b) The multivalued part is replaced by a discontinuities. The areas of the two lobes are identical.

1.2.3 Riemann problem

Let us consider the Riemann problem for a homogeneous hyperbolic equation:

$$\frac{\partial}{\partial t} q + \frac{\partial}{\partial x} f(q) = 0, \tag{1.40}$$

subject to the initial condition

$$q(x,\ 0) = q_0(x) = \begin{cases} q_L & \text{if } x < 0, \\ q_R & \text{if } x > 0, \end{cases}$$

where q_L and q_R are constant. When the flux function is convex, two solutions are possible depending on these constants:

- rarefaction waves,
- show waves.

Let us start with rarefaction waves. This equation is invariant to the transformation $x \to \alpha x$ et $t \to \alpha t$. A solution can be sought in the form $q(\xi)$ with $\xi = x/t$. Substituting this form into Eq. (1.40):

$$\left(f'(q(\xi)) - \xi\right) q' = 0.$$

The solution is

$$q(x,\ t) = f'^{(-1)}(\xi)$$

where $f'^{(-1)}$ is the inverse of f'. The whole solution is

$$q(x,\ t) = \begin{cases} q_L & \text{if } \dfrac{x}{t} \leq f'(q_L), \\ f'^{(-1)}(\xi) & \text{if } f'(q_L) \leq \dfrac{x}{t} \leq f'(q_R) \\ q_R & \text{if } \dfrac{x}{t} \geq f'(q_R). \end{cases}$$

Let us now consider a shock wave. It position is $x = s(t) = \dot{s}t$. The Rankine-Hugoniot equation (1.39): $[\![f(q)]\!] = \dot{s}[\![q]\!]$. The whole solution is:

$$q(x,\ t) = \begin{cases} q_L & \text{if } x < \dot{s}t, \\ q_R & \text{if } x > \dot{s}t. \end{cases}$$

The shock velocity $\dot{s}$ is given by:

$$\dot{s} = \frac{f(q_L) - f(q_R)}{q_L - q_R}.$$

Let us summarise the two possible solutions: Recall that when $f'' > 0$, the celerity $c(q) = f'(q)$ is an increasing function of q, which is also the slope of the characteristic curves (straight lines):

- $q_R > q_L$, $\lambda(u_R) > \lambda(u_L)$. At time $t = 0$, the two families of characteristic curves fan out. Equation $\xi = f'(U(\xi))$ is an implicit solution over the interval $\lambda(q_R) > \xi > \lambda(q_L)$.
- $q_R < q_L$. The two families of characteristic curves cross each other as of $t = 0$. The shock wave moves at speed $\lambda(q_R) < \dot{s} < \lambda(q_L)$. This condition is called the *Lax condition*, which defines whether a shock is physically admissible.

1.3 Nonlinear systems

Let us now consider the nonlinear case for one-dimensional problems

$$\frac{\partial}{\partial t}\boldsymbol{q} + \frac{\partial}{\partial x}\boldsymbol{f}(q) = \boldsymbol{S} \Leftrightarrow \frac{\partial}{\partial t}\boldsymbol{q} + \boldsymbol{A}(\boldsymbol{q}) \cdot \frac{\partial}{\partial x}\boldsymbol{q} = \boldsymbol{S}, \tag{1.41}$$

where $\boldsymbol{q}$ is a vector with m components representing the unknowns, $\boldsymbol{f}$ is the flux function, $\boldsymbol{A} = \nabla \boldsymbol{f}$ is its Jacobian (the gradient involves the derivatives with respect to the $\boldsymbol{q}$ components). We assume that $\boldsymbol{A}$ is $m \times m$ matrix whose eigenvalues λ_i are assumed to be real and distinct—like for the linear case—over a certain domain.

1.3.1 Riemann invariants

The computational strategy closely follows the one taken for the linear case. It relies on the concept of differential invariants. Let us illustrate this concept for $m = 2$. The unknown vector $\boldsymbol{q}$ has components $(q_1,\ q_2)$. We seek a new variable $\boldsymbol{r} = \{r_1,\ r_2\}$ such that:

$$\boldsymbol{v}_1 \cdot \mathrm{d}\boldsymbol{q} = \mu_1 \mathrm{d}r_1,$$

$$\boldsymbol{v}_2 \cdot \mathrm{d}\boldsymbol{q} = \mu_2 \mathrm{d}r_2,$$

where μ_i are the integrating factors such that $\mathrm{d}r_i$ are exact differentials. By expanding the differential $\mathrm{d}r_1$, we get:

$$\mu_1 \mathrm{d}r_1 = \mu_1 \left(\frac{\partial r_1}{\partial q_1}\mathrm{d}q_1 + \frac{\partial r_1}{\partial q_2}\mathrm{d}q_2 \right) = v_{11}\mathrm{d}q_1 + v_{12}\mathrm{d}q_2.$$

Upon identification with the former equation, we deduce:

$$\frac{\partial r_1}{\partial q_1} = \frac{v_{11}}{\mu_1},$$

and

$$\frac{\partial r_1}{\partial q_2} = \frac{v_{12}}{\mu_1}.$$

We deduce the governing equations for r_1 and μ_1. By dividing the two equations above, we obtain:

$$\frac{\partial r_1}{\partial q_1} = \frac{v_{11}}{v_{12}} \frac{\partial r_1}{\partial q_2}, \tag{1.42}$$

while the integrating factor is obtained by applying the Schwarz theorem

$$\frac{\partial}{\partial q_1} \frac{v_{12}}{\mu_1} = \frac{\partial}{\partial q_2} \frac{v_{11}}{\mu_1}.$$

We have seen above that the left and right eigenvectors are orthogonal two by two, which means here that $\boldsymbol{v}_1 \cdot \boldsymbol{w}_2 = 0$ (that is, $v_{12} = w_{21}$ and $-v_{11} = w_{22}$). We can then transform Eq. (1.42) into

$$w_{21} \frac{\partial r_1}{\partial q_1} + w_{22} \frac{\partial r_1}{\partial q_2} = 0 \Rightarrow \boldsymbol{w}_2 \cdot \nabla r_1 = 0. \tag{1.43}$$

Note that :

- in the literature, the k-invariant r_k is defined simply as the solution to $\boldsymbol{w}_k \cdot \nabla r_k = 0$, and its label differs from the one used here: r_1 was associated with the eigenvalue λ_1, and is a 2-invariant of Eq. (1.41).
- when there are $m > 2$ equations, then there are usually $m - 1$ distinct functions r_k that are k-invariants.

Equation (1.42) can be cast in the form

$$\frac{\mathrm{d}q_1}{v_{12}} = \frac{\mathrm{d}q_2}{v_{11}} = \frac{\mathrm{d}r_1}{0},$$

whose integration provides r_1. Equation (1.41) leads to:

$$\boldsymbol{v}_1 \cdot \left. \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} \right|_{x = X_1(t)} + \boldsymbol{v}_1 \cdot \boldsymbol{S} = 0,$$

where the characteristic curve $x = X_1(t)$ satisfies $\mathrm{d}X_1/\mathrm{d}t = \lambda_1$. It is called the 1-characteristic. We have:

$$\mu_1 \left. \frac{\mathrm{d}r_1}{\mathrm{d}t} \right|_{x = X_1(t)} = \boldsymbol{v}_1 \cdot \boldsymbol{S}.$$

Similarly for r_2:

$$\mu_2 \left. \frac{\mathrm{d}r_2}{\mathrm{d}t} \right|_{x = X_2(t)} = \boldsymbol{v}_2 \cdot \boldsymbol{S}.$$

The compact form of Eq. (1.41) after the change of variable is:

$$\left. \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} \right|_{\boldsymbol{r} = \boldsymbol{X}(t)} = \boldsymbol{L} \cdot \boldsymbol{S}, \tag{1.44}$$

where $\boldsymbol{L} = [\boldsymbol{v}_1,\, \boldsymbol{v}_2]^T$ and $\boldsymbol{r} = \{r_1,\, r_2\}$.

1.3.2 Rarefaction wave

Definition

The homogeneous hyperbolic system

$$\frac{\partial}{\partial t}\boldsymbol{q} + \boldsymbol{A}(\boldsymbol{q}) \cdot \frac{\partial}{\partial x}\boldsymbol{q} = \boldsymbol{0} \tag{1.45}$$

is invariant to the stretching group $x \to \alpha x$ and $t \to \alpha t$. We define the similarity variable $\xi = x/t$. We are seeking a similarity solution $\boldsymbol{q} = \boldsymbol{q}(\xi)$. Substituting this form into Eq. (1.45) gives

$$-\xi \boldsymbol{q}' + \boldsymbol{A} \cdot \boldsymbol{q}' = 0,$$

which shows that $\boldsymbol{q}'$ is a right eigenvector of $\boldsymbol{A}$, imposing $\xi = \lambda_k$ and $\boldsymbol{q}'$ colinear with the right eigenvector $\boldsymbol{w}_k$. We can arbitrarily pose

$$\frac{\mathrm{d}}{\mathrm{d}\xi}\boldsymbol{q} = \boldsymbol{w}_k. \tag{1.46}$$

A geometric interpretation is that the curve $\boldsymbol{q}(\xi)$ is tangent to the vector field $\boldsymbol{w}_k$ ($\boldsymbol{q}(\xi)$ is called the *integral curve* of $\boldsymbol{w}_k$). Note that if we seek a function $R(\boldsymbol{q})$ that remains constant along this integral curve, then we recover the definition (1.42) of the Riemann invariant:

$$\frac{\mathrm{d}}{\mathrm{d}\xi}R(\boldsymbol{q}) = 0 \Rightarrow \nabla R \cdot \boldsymbol{q}' = 0,$$

and since $\boldsymbol{q}' = \boldsymbol{w}_k$, then the invariance condition is: $\nabla R \cdot \boldsymbol{w}_k = 0$.

Simple wave

Rarefaction waves are a special case of simple wave. As for the linear case (see § 1.1.4), a simple wave propagates in a single direction. If there is a smooth mapping $(x,\, t) \to \eta$, a simple wave is defined as the special solution

$$\boldsymbol{q}(x,\, t) = \boldsymbol{q}(\eta(x,t)).$$

Substituting this form into Eq. (1.45) gives

$$\frac{\partial \eta}{\partial t}\boldsymbol{q}' + \frac{\partial \eta}{\partial x}\boldsymbol{A} \cdot \boldsymbol{q}' = 0.$$

Reiterating the same reasoning as just above, we deduce that $\boldsymbol{q}$ is an integral curve of one right eigenvector $\boldsymbol{k}_k$, and thus η must satisfy the nonlinear advection equation

$$\frac{\partial \eta}{\partial t} + \lambda_k \frac{\partial \eta}{\partial x} = 0 \Leftrightarrow \frac{\mathrm{d}\eta}{\mathrm{d}t} = \text{ along } \frac{\mathrm{d}x}{\mathrm{d}t} = \lambda_k.$$

Characteristic curves in the $x - t$ plane are thus straight lines of slope $\lambda_k(\boldsymbol{q}(\eta))$.

Exemple: Saint-Venant equations

For water waves over horizontal frictionless beds, the governing equations (called Saint-Venant or shallow water equations) are given by Eq. (1.45) with:

$$\boldsymbol{q} = \begin{pmatrix} h \\ q \end{pmatrix} \text{ and } \boldsymbol{A} = \boldsymbol{f}' = \begin{pmatrix} 0 & 1 \\ -q^2/h^2 + gh & 2q/h \end{pmatrix}, \tag{1.47}$$

where h and $q = hu$ denote the flow depth and momentum, g is gravity acceleration, and u is velocity. The eigenvalues are

$$\lambda_1 = u - c \text{ and } \lambda_2 = u + c,$$

where $c = \sqrt{gh}$ and the right eigenvectors are

$$\boldsymbol{w}_1 = \begin{pmatrix} 1 \\ u - c \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} 1 \\ u + c \end{pmatrix}.$$

If we define the 1-Riemann invariant r_1 as $\nabla r_1 \cdot \boldsymbol{w}_k = 0$, then r_1 is the solution to

$$\frac{\partial r_1}{\partial h} + (u + c)\frac{\partial r_1}{\partial q} = 0 \Leftrightarrow \frac{\mathrm{d}h}{1} = \frac{\mathrm{d}q}{u + c} = \frac{\mathrm{d}r_1}{0}.$$

Integrating the first pair of equations gives

$$q = -2h^{3/2}\sqrt{g} + ha \Leftrightarrow r_1 = u + 2\sqrt{gh},$$

where a is a constant of integration. As r_1 is an arbitrary function of a, we select the simplest form. Similarly for the 2-invariant r_2, we find

$$r_2 = u - 2\sqrt{gh}$$

The Saint-Venant equations are thus equivalent to

$$\frac{\mathrm{d}r_1}{\mathrm{d}t} = 0 \text{ along } \frac{\mathrm{d}x}{\mathrm{d}t} = \lambda_1 = u - c \text{ and } \frac{\mathrm{d}r_2}{\mathrm{d}t} = 0 \text{ along } \frac{\mathrm{d}x}{\mathrm{d}t} = \lambda_2 = u + c. \tag{1.48}$$

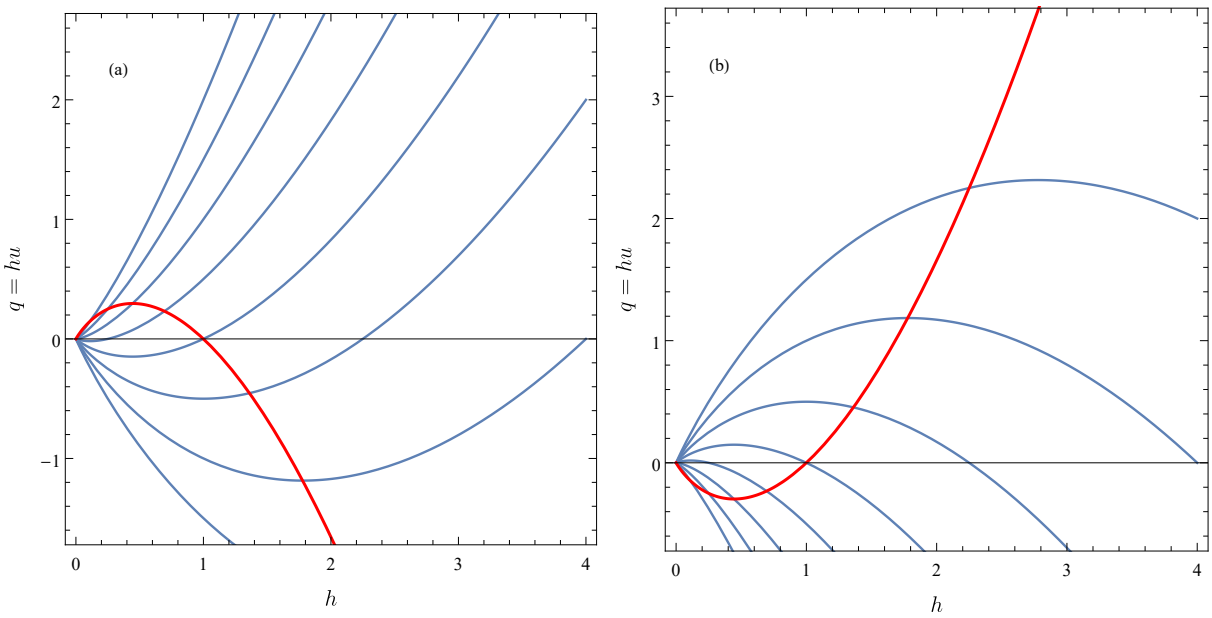


Figure 1.4 (a) lambda, -2.5, 1, 0.5. (b) lambda, -1, 2.5, 0.5.

CHAPTER 2

Finite volume methods

2.1 General formulation

Let us consider a hyperbolic equation in one space dimension and in a conservative form

$$\frac{\partial}{\partial t}\boldsymbol{q} + \frac{\partial}{\partial x}\boldsymbol{f}(\boldsymbol{q}) = 0, \tag{2.1}$$

where $\boldsymbol{q}$ is a vector with m components representing the unknowns and $\boldsymbol{f}$ is the flux function. We consider a uniform grid, whose mesh size is constant: Δx. We define the cell $C_i = [x_{i-1/2}, x_{i+1/2})$, centred around the cell middle $x_i = x_0 + i\Delta x$ and whose interfaces are $x_{i\pm1/2}$. The time step is denoted by $\Delta t = t_{n+1} - t_n$.

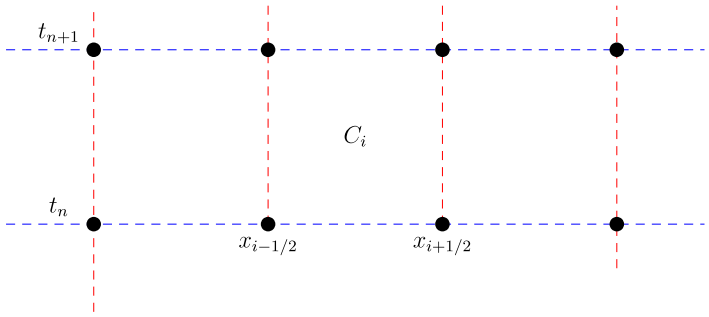


Figure 2.1 Computation grid in the $x - t$ plane.

We integrate Eq. (2.1) over the cell C_i:

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial \boldsymbol{q}}{\partial t}\mathrm{d}x + [\boldsymbol{f}(\boldsymbol{q})]_{x_{i-1/2}}^{x_{i+1/2}} = 0, \tag{2.2}$$

Integrating this equation over $(t_n, t_{n+1}]$ gives:

$$\int_{x_{i-1/2}}^{x_{i+1/2}} (\boldsymbol{q}(x, t_{n+1}) - \boldsymbol{q}(x, t_n))\mathrm{d}x + \int_{t_n}^{t_{n+1}} [\boldsymbol{f}(\boldsymbol{q})]_{x_{i-1/2}}^{x_{i+1/2}}\,\mathrm{d}t = 0. \tag{2.3}$$

We define the cell-averaged value of $\boldsymbol{q}$ at time t_n:

$$\boldsymbol{Q}_i^n = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \boldsymbol{q}(x, t_n)\mathrm{d}x, \tag{2.4}$$

and a time-averaged flux

$$\boldsymbol{F}^n_{i\pm1/2} = \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} \boldsymbol{f}(\boldsymbol{q}(x_{i\pm1/2}, t))\mathrm{d}t. \tag{2.5}$$

We can develop an explicit time-marching algorithm by rearranging Eq. (2.3) and introducing the time- and grid-averaged variables

$$\boldsymbol{Q}^{n+1}_i = \boldsymbol{Q}^n_i - \frac{\Delta t}{\Delta x}\left(\boldsymbol{F}^n_{i+1/2} - \boldsymbol{F}^n_{i-1/2}\right). \tag{2.6}$$

2.2 Godunov's method for linear systems

Godunov's method has been a major achievement in the field of hyperbolic equations, which has opened up the way to modern finite-volume techniques. It consists of three steps: reconstructing, evolving, and averaging:

1. *Reconstruction.* We assume that we can approximate the solution $\boldsymbol{q}(x,t)$ by a piecewise constant function $\tilde{\boldsymbol{q}}^n_i(x,t_n) = \boldsymbol{Q}^n_i$ for $x \in C_i = (x_{i-1/2}, x_{i+1/2})$: Note that to second order, we have

$$\boldsymbol{Q}^n_i = \boldsymbol{q}(x_i,t_n) + \frac{\Delta x}{6}\partial_x\boldsymbol{q}(x_i,t_n) + \frac{\Delta x^2}{24}\partial_{xx}\boldsymbol{q}(x_i,t_n).$$

   AlthGodunov's method is a first-order accurate scheme. We can use higher-order reconstructions of the approximate the function $\boldsymbol{q}(x,t)$ (e.g., a piecewise linear function with a nonzero slope in each grid cell).

2. *Evolution.* Using Eq. (2.6) or another method, we look at how the solution $\tilde{\boldsymbol{q}}^{n+1}_i$ evolves from its state at time t_n. This step amounts to solve Riemann problems at each cell boundary $x_{i\pm1/2}$.

3. *Averaging.* We average this function over each grid cell

$$\boldsymbol{Q}^{n+1}_i = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \tilde{\boldsymbol{q}}(x,t_{n+1})\mathrm{d}x, \tag{2.7}$$

   We can piece together the Riemann solutions provided that the waves from two adjacent interfaces have not started to interact. This condition is usually met when the *Courant-Friedrichs-Lewy* (CFL) condition is satisfied (no wave passes through more than one grid cell within Δt):

$$\frac{s_{max}\Delta t}{\Delta x} \leq 1, \tag{2.8}$$

   where s_{max} represents the largest wave speed.

Godunov's method was initially used to solve the Euler equations in gas dynamics. The flux $\boldsymbol{F}^n_{i+1/2}$ was determined from the exact solution to the Riemann problem for the Euler equations. Approximate Riemann solvers are today used because they are faster. Godunov's method is robust and stable when the CFL condition is met. When approximate solvers are used, this may not be the case, and thus special care has to be paid to robustness and stability. Moreover, Godunov's method tends to smear out solutions near discontinuities. By using limiters, approximate Riemann solvers deal more efficiently with discontinuities. They also build numerical solutions as linear combinations of travelling discontinuities—they do not use rarefaction waves, which are therefore approximated as discontinuities. Transonic waves[1] may need more care.

[1] see Fig. 2.4 for a quick definition.

2.3 Wave decomposition for linear systems

2.3.1 Introductive example

In Clawpack, we will use a variant of Godunov's method based on the wave decomposition seen in Chapter 1. There are other strategies such as *flux differencing* (Toro, 2001; LeVeque, 2002; Guinot, 2010). The advantage of wave decomposition over other approaches is that it can also be applied to non-conservative equations.

Let illustrate how Clawpack proceeds with the flux estimation by considering a problem of dimension $m = 3$. Let us assume that we have three different eigenvalues such that $\lambda_1 < 0 < \lambda_2 < \lambda_3$. As shown by Fig. 2.2, from the node $x_{i-1/2}$ emerge three characteristics $x_{i-1/2} + \lambda_i t$, which will create three discontinuities in $\tilde{\boldsymbol{q}}$ at time t_{n+1}. Recall that in Chapter 1, we learned from Eq. (1.28) that the initial jump in the Riemann problem at $x_{i-1/2}$ can be decomposed into three waves

$$\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1} = \sum_{k=1}^{m} \boldsymbol{W}_{k,i-1/2}, \tag{2.9}$$

where $\boldsymbol{W}_{k,i-1/2}$ is related to the right eigenvectors $\alpha_{k,i-1/2}\boldsymbol{w}_{k,i-1/2}$. As seen in Fig. 2.2, the first wave $\boldsymbol{W}_{1,i-1/2}$ will not modify the value of the solution at time t_{n+1}, but the two other waves will do. For instance, the second wave will modify the value of $\boldsymbol{q}$ over a fraction of the grid cell $\lambda_2 \Delta t/\Delta x$ by the amount

$$-\lambda_2 \frac{\Delta t}{\Delta x} \boldsymbol{W}_{2,i-1/2}$$

relative to the initial value $\boldsymbol{Q}_i$.

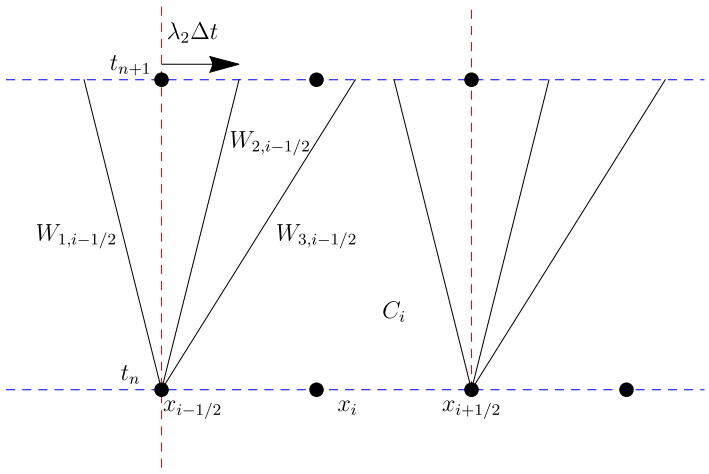


Figure 2.2 Wave structure for the node $x_{i-1/2}$.

2.3.2 General formulation

If we repeat the reasoning seen in the previous section for all waves, we obtain

$$\boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n - \frac{\Delta t}{\Delta x} \left(\lambda_2 \boldsymbol{W}_{2,i-1/2} + \lambda_3 \boldsymbol{W}_{3,i-1/2} + \lambda_1 \boldsymbol{W}_{1,i+1/2}. \right), \tag{2.10}$$

This can be readily generalised to arbitrary hyperbolic systems. Let us introduce the notation

$$\lambda^+ = \max(\lambda, 0) \text{ and } \lambda^- = \min(\lambda, 0). \tag{2.11}$$

The updated value $\boldsymbol{Q}_i^{n+1}$ is then

$$\boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n - \frac{\Delta t}{\Delta x}\left(\sum_{k=1}^{m} \lambda_k^+ \boldsymbol{W}_{k,i-1/2} + \sum_{k=1}^{m} \lambda_k^- \boldsymbol{W}_{k,i+1/2}.\right), \tag{2.12}$$

The cell average depends on the right-going waves from $x_{i-1/2}$ and left-going waves from $x_{i+1/2}$.

LeVeque (2002) introduced a shorthand notation

$$\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2} = \sum_{k=1}^{m} \lambda_k^+ \boldsymbol{W}_{k,i-1/2}, \tag{2.13}$$

$$\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2} = \sum_{k=1}^{m} \lambda_k^- \boldsymbol{W}_{k,i+1/2}, \tag{2.14}$$

which are interpreted as *fluctuations*: $\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2}$ represents the effect of all right-going waves from $x_{i-1/2}$ (where there is a discontinuity $\Delta \boldsymbol{Q}_{i-1/2} = \boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}$) on the cell average at time t_{n+1}. This formulation that holds for linear problems will be generalized to nonlinear problems.

2.3.3 *Interface flux*

Note that that the interface between two cells, the interface value can be written (see Eq. (1.35)):

$$\boldsymbol{Q}_{i-1/2} = \boldsymbol{Q}_{i-1} + \sum_{\lambda_k<0} \boldsymbol{W}_{k,i-1/2}. \tag{2.15}$$

We then deduce that for a linear system, the flux at the interface is:

$$\boldsymbol{F}_{i-1/2}^n = \boldsymbol{f}(\boldsymbol{Q}_{i-1/2}) = \boldsymbol{A} \cdot \boldsymbol{Q}_{i-1/2} = \boldsymbol{A} \cdot \boldsymbol{Q}_{i-1} + \sum_{\lambda_k<0} \boldsymbol{A} \cdot \boldsymbol{W}_{k,i-1/2}. \tag{2.16}$$

As $\boldsymbol{W}_k$ is an eigenvector of $\boldsymbol{A}$, we can rearrange the terms

$$\boldsymbol{F}_{i-1/2}^n = \boldsymbol{A} \cdot \boldsymbol{Q}_{i-1} + \sum_{k=1}^{m} \lambda_k^- \boldsymbol{W}_{k,i-1/2} = \boldsymbol{A} \cdot \boldsymbol{Q}_{i-1} + \boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i-1/2}. \tag{2.17}$$

This expression will be generalised to nonlinear systems (which, once linearised, involve only shock waves), for which we will assume that

$$\boldsymbol{F}_{i-1/2}^n = \boldsymbol{f}(\boldsymbol{Q}_{i-1}) + \boldsymbol{A}^- \Delta \boldsymbol{Q}_{i-1/2}. \tag{2.18}$$

or equivalently:

$$\boldsymbol{F}_{i-1/2}^n = \boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{A}^+ \Delta \boldsymbol{Q}_{i-1/2}. \tag{2.19}$$

Can we proceed differently for nonlinear systems? For a nonlinear problem, the theoretical expression of the flux is more complicated. Integrating the hyperbolic equation (2.1) over $[-\delta x,\ 0] \times [0, \delta t]$ (see Fig. 2.3) gives

$$\int_{-\delta x}^{0} \boldsymbol{q}(x,\ \delta t)\mathrm{d}x = \int_{-\delta x}^{0} \boldsymbol{q}(x,\ 0)\mathrm{d}x + \int_{0}^{\delta t} \boldsymbol{f}(\boldsymbol{q}(-\delta x,\ t))\mathrm{d}t - \int_{0}^{\delta t} \boldsymbol{f}(\boldsymbol{q}(0,\ t))\mathrm{d}t,$$

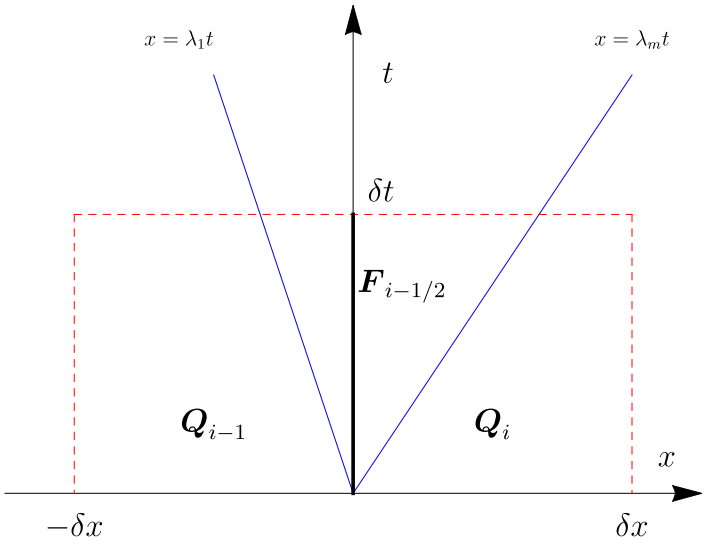


Figure 2.3 Calculating the interface flux $\boldsymbol{F}_{i-1/2}$.

and if δx is chosen such that it lies in the domain controlled by the initial conditions $\boldsymbol{Q}_{i-1}$ or $\boldsymbol{Q}_i$, then we can rearrange the terms

$$\int_{-\delta x}^{0} \boldsymbol{q}(x,\ \delta t)\mathrm{d}x = \delta x \boldsymbol{Q}_{i-1} + \delta t \boldsymbol{f}(\boldsymbol{Q}_{i-1}) - \delta t \boldsymbol{F}_{i-1/2}.$$

This gives us the relation:

$$\boldsymbol{F}_{i-1/2} = \boldsymbol{f}(\boldsymbol{Q}_{i-1}) + \frac{\delta x}{\delta t}\boldsymbol{Q}_{i-1} - \frac{1}{\delta t}\int_{-\delta x}^{0} \boldsymbol{Q}(x,\ \delta t)\mathrm{d}x. \tag{2.20}$$

The relation leads to no formal result, but it can be exploited to provide approximate solvers such as the HLL solver (Toro, 2019).

2.4 Approximate Riemann solvers for nonlinear problems

Earlier in this chapter, we have seen that a general time-marching algorithm to solve the hyperbolic equation (2.1) is given by Eq. (2.6):

$$\boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n - \frac{\Delta t}{\Delta x}\left(\boldsymbol{F}_{i+1/2}^n - \boldsymbol{F}_{i-1/2}^n\right).$$

At each interface $x_{i-1/2}$, the flux function is given by

$$\boldsymbol{F}_{i-1/2}^n = \boldsymbol{f}(\boldsymbol{Q}_{i-1/2}^n),$$

where $\boldsymbol{Q}_{i-1/2}^n$ is the value of $\boldsymbol{Q}$ obtained along the ray $x = x_{i-1/2}$. It depends on the values $\boldsymbol{Q}_i^n$ and $\boldsymbol{Q}_i^n$ of either side of the interface. In the absence of a source terme, $\boldsymbol{Q}_i^n$ remains constant along this ray.

2.4.1 Scalar problems

Scalar Riemann problem are associated with five possible wave configurations (see Fig. 2.4):

(a) Left-going shock wave: $Q^n_{i-1/2} = Q^n_i$.

(b) Left-going rarefaction wave: $Q^n_{i-1/2} = Q^n_i$.

(c) Transonic[2] rarefaction wave: $Q^n_{i-1/2} = q_s(Q^n_i, Q^n_i)$. This is the only case for which we cannot set the $Q_{i-1/2}$ value. Further calculations are needed to evaluate the value q_s. The unknown value q_s satisfies

$$Q^n_{i-1} < q_s < Q^n_i,$$

and this is associated with the vertical ray, its characteristic speed is zero. Therefore, q_s is the solution to

$$f'(q_s) = 0. \tag{2.21}$$

(d) Right-going rarefaction wave: $Q^n_{i-1/2} = Q^n_{i-1}$.

(e) Right-going shock wave: $Q^n_{i-1/2} = Q^n_{i-1}$.

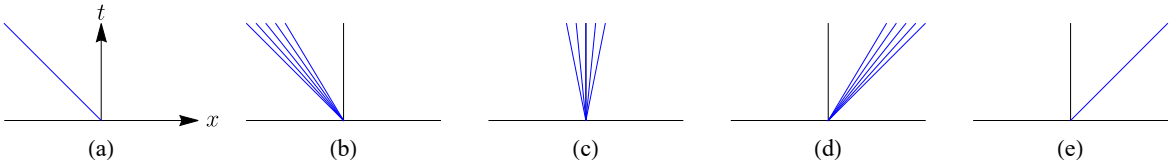


Figure 2.4 The five possible solutions to a scalar Riemann problem: (a) left-going shock wave; (b) left-going rarefaction wave; (c) transonic rarefaction wave; (d) right-going rarefaction wave; and (e) right-going shock wave.

For a convex scalar flux, we can summarise all these possibilities

$$F^n_{i-1/2} = \begin{cases} f(Q^n_{i-1}) & \text{if } Q^n_{i-1} > q_s \text{ and } s > 0 \\ f(Q^n_i) & \text{if } Q^n_i < q_s \text{ and } s < 0 \\ f(q_s) & \text{if } Q^n_{i-1} < q_s < Q^n_i, \end{cases} \tag{2.22}$$

where the shock speed s is given by:

$$s = \frac{f(Q^n_i) - f(Q^n_{i-1})}{Q^n_i - Q^n_{i-1}}.$$

A more compact way used in Clawpack is given

$$F^n_{i-1/2} = \begin{cases} \min\limits_{Q^n_{i-1} \le q \le Q^n_i} f(q) & \text{if } Q^n_{i-1} \le Q^n_i \\ \max\limits_{Q^n_i \le q \le Q^n_{i-1}} f(q) & \text{if } Q^n_{i-1} \ge Q^n_i \end{cases} \tag{2.23}$$

[2]It is called transonic because it moves with velocity 0. In gas dynamics, this happens when one of the eigenvalues $u \pm c$ (c: sound speed) takes the value 0, thus when the fluid moves at the same speed as sound. In Fig. 2.4(c) the fluid is accelerated from a subsonic velocity to a supersonic one through a rarefaction wave.

The Lax entropy condition is an extra condition imposed to shock waves solution for them to be physically admissible. A shock wave must dissipate energy, not create energy (or from a thermodynamical standpoint, entropy increases through a shock, and does not decrease). A shock wave satisfies the Lax entropy condition if its speed lies between bounds fixed by the initial data

$$f'(q_l) > s > f'(q_r). \tag{2.24}$$

For a scalar problem, the time-marching algorithm to solve the hyperbolic equation (2.1) is given by this variant of Eq. (2.6):

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - F_{i-1/2}^n\right),$$

or equivalently

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2}^n - f(Q_i) - (F_{i-1/2}^n - f(Q_i))\right).$$

We can make an analogy with the formulation for linear equations, which emphasizes the role of fluctuations (see § 2.3). We put the equation above in a form consistent with the LeVeque's notation:

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(A^+\Delta Q_{i-1/2} + A^-\Delta Q_{i+1/2}\right),$$

where the fluctuations $A^\pm\Delta Q_{i\pm1/2}$ are defined by

$$A^+\Delta Q_{i-1/2} = f(Q_i^n) - f(Q_{i-1/2}^n),$$
$$A^-\Delta Q_{i+1/2} = f(Q_{i+1/2}^n) - f(Q_i^n).$$

High-resolution techniques involve defining the wave $W_{i-1/2}$ and speed $s_{i-1/2}$ associated with the Riemann problem:

$$W_{i-1/2} = Q_i - Q_{i-1},$$
$$s_{i-1/2} = \begin{cases} \dfrac{f(Q_i^n) - f(Q_{i-1}^n)}{Q_i^n - Q_{i-1}^n} & \text{if } Q_i \neq Q_{i-1}, \\ f'(Q_i^n) & \text{if } Q_i = Q_{i-1}. \end{cases}$$

When the Riemann solution is a shock wave, the speed chosen is the one given by the Rankine-Hugoniot equation. When it is a rarefaction wave, the speed chose provides a proper estimate of the actual wave speed, and the wave behaviour can be approximated by a shock wave even the latter would not satisfy the entropy condition (2.24). The big advantage is that we can treat all waves as shock waves regardless of their actual nature. When the solution is not a transonic wave, we can also express the fluctuations as:

$$A^+\Delta Q_{i-1/2} = s_{i-1/2}^+ W_{i-1/2}, \tag{2.25}$$
$$A^-\Delta Q_{i+1/2} = s_{i+1/2}^- W_{i+1/2}, \tag{2.26}$$

where $s^+ = \max(s, 0)$ and $s^- = \min(s, 0)$. Equation (2.25) is used in Clawpack for solving scalar problems.

When the Riemann solution consists of a transonic rarefaction wave, the fluctuation terms $A^\pm\Delta Q_{i\pm1/2}$ need to be corrected using an *entropy fix*. In Clawpack, the wave W and speed s are first computed, and from them, we determine the fluctuations using Eq. (2.25). If $f'(Q_{i-1}) < 0 < f'(Q_i)$, then the fluctuations in Eq. (2.25) are replaced by one of the equations (for the interface from which the transonic wave originates):

$$A^+\Delta Q_{i-1/2} = f(Q_i^n) - f(q_s), \tag{2.27}$$
$$A^-\Delta Q_{i+1/2} = f(q_s) - f(Q_i^n). \tag{2.28}$$

Although this approach based on an entropy fix is unnecessary for scalar problems, it is easy to generalize to nonlinear systems of hyperbolic equations, for which there is no easy way to determine the rarefaction wave structure exactly.

2.4.2 Systems of equations

The method used for scalar problems can be generalised to systems of hyperbolic equations. The crux lies in the determination of the interface value $\boldsymbol{Q}^n_{i-1/2}$. This value is usually one of the intermediate states that connect the left and right states through a series of shock and rarefaction waves. When $\boldsymbol{Q}^n_{i-1/2}$ is part of a transonic rarefaction wave, additional work is required to determine the wave structure.

The computational approach to solving the nonlinear Riemann problem is the same as the one taken for linear problems. When dealing with Godunov's equation (2.6), Clawpack still uses the wave-propagation form

$$\begin{aligned} \boldsymbol{Q}^{n+1}_i &= \boldsymbol{Q}^n_i - \frac{\Delta t}{\Delta x}\left(\boldsymbol{F}^n_{i+1/2} - \boldsymbol{F}^n_{i-1/2}\right), \\ &= \boldsymbol{Q}^n_i - \frac{\Delta t}{\Delta x}\left(\boldsymbol{A}^+\Delta\boldsymbol{Q}^n_{i-1/2} + \boldsymbol{A}^-\Delta\boldsymbol{Q}^n_{i+1/2}\right), \end{aligned} \tag{2.29}$$

where the fluctuations are defined by generalizing the linear case (see § 2.3.3):

$$\boldsymbol{A}^-\Delta\boldsymbol{Q}^n_{i+1/2} = \boldsymbol{f}(\boldsymbol{Q}^n_{i+1/2}) - \boldsymbol{f}(\boldsymbol{Q}^n_i), \tag{2.30}$$

$$\boldsymbol{A}^+\Delta\boldsymbol{Q}^n_{i-1/2} = \boldsymbol{f}(\boldsymbol{Q}^n_i) - \boldsymbol{f}(\boldsymbol{Q}^n_{i-1/2}), \tag{2.31}$$

These definitions are useful when the solution to the Riemann problem is a transonic wave. When the solution is a shock or rarefaction wave, the fluctuations can be approximated by considering that in the close vicinity of the initial state the solution behaves like a shock wave, and like in the linear case, the fluctuations are given by:

$$\boldsymbol{A}^-\Delta\boldsymbol{Q}^n_{i+1/2} = \sum_{k=1}^{M_w} s^-_{k,i+1/2}\boldsymbol{W}^n_{k,i+1/2}, \tag{2.32}$$

$$\boldsymbol{A}^+\Delta\boldsymbol{Q}^n_{i-1/2} = \sum_{k=1}^{M_w} s^+_{k,i-1/2}\boldsymbol{W}^n_{k,i-1/2}, \tag{2.33}$$

where M_w is the number of waves (usually $M_w = m$), $s^- = \min(0, s)$ and $s^+ = \max(0, s)$.

The computation cost is high if we use exact Riemann solvers. A variety of approximate Riemann solvers have been proposed to reduce this cost (Toro, 2001; LeVeque, 2002).

Linearised solvers

Nonlinear equations

$$\frac{\partial}{\partial t}\boldsymbol{q} + \frac{\partial}{\partial x}\boldsymbol{f}(\boldsymbol{q}) = 0$$

can be linearised when the initial values $\boldsymbol{q}_l$ and $\boldsymbol{q}_r$ are sufficiently close to each other, and put in the linear form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \hat{\boldsymbol{A}} \cdot \frac{\partial \boldsymbol{q}}{\partial x} = 0,$$

where the constant matrix $\hat{\boldsymbol{A}}$ is an approximation of $\boldsymbol{f}'(\boldsymbol{q})$ for $\boldsymbol{q} \approx \boldsymbol{q}_l \approx \boldsymbol{q}_r$. The Roe function (studied later) is an example of linearised solvers (Roe, 1981).

Two-wave solvers

Several approximate solvers are based on the idea that the Riemann solution can be approximated by picking up two of the m waves, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, and defining an intermediate state $\boldsymbol{Q}_*$ such that $\boldsymbol{W}_1 = \boldsymbol{Q}_* - \boldsymbol{Q}_l$ and $\boldsymbol{W}_2 = \boldsymbol{Q}_r - \boldsymbol{Q}_*$. The Rankine-Hugoniot implies that $\boldsymbol{f}(\boldsymbol{Q}_l) - \boldsymbol{f}(\boldsymbol{Q}_*) = s_1\boldsymbol{W}_1$ and $\boldsymbol{f}(\boldsymbol{Q}_r) - \boldsymbol{f}(\boldsymbol{Q}_*) = s_2\boldsymbol{W}_2$. By adding these two equations, we end up with a system of m equations

$$\boldsymbol{f}(\boldsymbol{Q}_r) - \boldsymbol{f}(\boldsymbol{Q}_l) = s_1\boldsymbol{W}_1 + s_2\boldsymbol{W}_2,$$

which gives

$$\boldsymbol{Q}_* = \frac{\boldsymbol{f}(\boldsymbol{Q}_r) - \boldsymbol{f}(\boldsymbol{Q}_l) - s_2\boldsymbol{Q}_r + s_1\boldsymbol{Q}_l}{s_1 - s_2}.$$

The various solvers proposed so far differ by the choice of the speeds s_1 and s_2 along with the waves $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. Lax-Friedrichs and Harten-Lax-van Leer (HLL) are classic solvers.

The advantage of HLL solvers is that they usually do not need an entropy fix to compute transonic rarefaction waves. As they involve only two waves (thereby ignoring all other waves), they may lead to poorer resolutions for systems made of $m > 2$ equations (Toro, 2019).

2.5 Roe solver

The Roe solver linearises the governing equation (2.1):

$$\frac{\partial \boldsymbol{q}}{\partial t} + \hat{\boldsymbol{A}}\frac{\partial \boldsymbol{q}}{\partial x} = 0.$$

The matrix $\hat{\boldsymbol{A}}$ is constructed so that it approximates $\boldsymbol{f}'(\boldsymbol{q})$ in the neighbourhood of $\boldsymbol{Q}_i$ and $\boldsymbol{Q}_{i-1}$ and satisfies the conditions

1. Continuity condition:
$$\hat{\boldsymbol{A}}_{i-1/2} \to \boldsymbol{f}'(\boldsymbol{q}) \text{ when } \boldsymbol{Q}_{i-1}, \boldsymbol{Q}_i \to \boldsymbol{q}.$$

2. Hyperbolicity: $\hat{\boldsymbol{A}}_{i-1/2}$ is diagonisable, with m right eigenvectors $\hat{\boldsymbol{w}}_{k,i-1/2}$ associated with eigenvalues $s_{k,i-1/2} = \lambda_{k,i-1/2}$.

3. Roe linearisation. This third property states that if $\boldsymbol{Q}_{i-1}$ and $\boldsymbol{Q}_i$ are connected by a single wave $\boldsymbol{W} = \boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}$ in the original Riemann problem, then $\boldsymbol{W}$ should also be an eigenvector of $\hat{\boldsymbol{A}}_{i-1/2}$:
$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \hat{\boldsymbol{A}}_{i-1/2} \cdot (\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}) = s(\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}),$$
where s is the wave speed. Formally, the matrix $\hat{\boldsymbol{A}}_{i-1/2}$ can be determined by integrating the Jacobian over a straight-line path $\boldsymbol{q}(\xi) = \boldsymbol{Q}_{i-1} + \xi\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1})$
$$\hat{\boldsymbol{A}}_{i-1/2} = \int_0^1 \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{q}(\xi))}{\mathrm{d}\boldsymbol{q}}\mathrm{d}\xi$$
There is no guarantee that the resulting matrix is diagonisable with m real eigenvalues and that it takes an analytical form. By making a change of variable, Roe (1981) showed that this difficulty can often be overcome (LeVeque, 2002, see pp. 317-323).

An alternative choice to Roe's linearisation is to set a particular value, for instance

$$\hat{\boldsymbol{A}}_{i-1/2} = \frac{1}{2}(\boldsymbol{f}'(\boldsymbol{Q}_{i-1}) + \boldsymbol{f}'(\boldsymbol{Q}_i)).$$

2.6 Two-wave solver: HLL solver

The idea underpinning the HLL solver's derivation is that the solution to the Riemann problem consists of two shock waves separating an intermediate state from the left and right initial states. The speeds s_1 and s_2 of these waves are given by the Rankine-Hugoniot equation

$$\boldsymbol{f}(\boldsymbol{Q}_{i-1}) - \boldsymbol{f}(\boldsymbol{Q}_*) = s_1(\boldsymbol{Q}_{i-1} - \boldsymbol{Q}_*), \tag{2.34}$$

$$\boldsymbol{f}(\boldsymbol{Q}_*) - \boldsymbol{f}(\boldsymbol{Q}_i) = s_2(\boldsymbol{Q}_* - \boldsymbol{Q}_i) \tag{2.35}$$

Solving Eqs. (2.34) and (2.35) for $\boldsymbol{Q}_*$ and $\boldsymbol{F}_* = \boldsymbol{f}(\boldsymbol{Q}_*)$ gives

$$\boldsymbol{Q}_* = \frac{s_2\boldsymbol{Q}_i - s_1\boldsymbol{Q}_{i-1}}{s_2 - s_1} + \frac{\boldsymbol{F}_{i-1} - \boldsymbol{F}_i}{s_2 - s_1} \tag{2.36}$$

$$\boldsymbol{F}_* = \frac{s_2\boldsymbol{F}_{i-1} - s_1\boldsymbol{F}_i}{s_2 - s_1} - s_2 s_2 \frac{\boldsymbol{Q}_{i-1} - \boldsymbol{Q}_i}{s_2 - s_1}, \tag{2.37}$$

with $\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Q}_i)$. For the Harten–Lax–van-Leer (HLL) solver, the speeds are defined as the lower and upper bounds of all characteristic speeds:

$$s_{1,i-1/2} = \min_{1\le k\le m} \left(\min(\lambda_{k,i-1}, \hat{\lambda}_{k,i-1/2})\right), \tag{2.38}$$

$$s_{2,i-1/2} = \min_{1\le k\le m} \left(\min(\lambda_{k,i}, \hat{\lambda}_{k,i-1/2})\right), \tag{2.39}$$

where $\lambda_{k,i}$ is the kth eigenvalue of the Jacobian $\boldsymbol{f}'(\boldsymbol{Q}_i)$ and $\hat{\lambda}_{k,i-1/2}$ is kth eigenvalue of the Roe matrix (linearised Jacobian).

2.7 Alternative: the f-wave method

An alternative approach to the wave decomposition is to first split the jump in $\boldsymbol{f}$ into f-waves:

$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \sum_{k=1}^{m_w} \boldsymbol{Z}_{k,i-1/2}, \tag{2.40}$$

moving at speeds $s_{k,i-1/2}$, then express the fluctuations in terms of the f-waves. This method is useful to study the second-order accuracy of wave-propagation methods or in the context of spatially-varying flux functions $\boldsymbol{f}(\boldsymbol{x},\ \boldsymbol{q})$ (LeVeque, 2002, see § 15.5). It also guarantees that approximate Riemann solvers are conservative.

When dealing with a linear or linearised problems, we can decompose $\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1})$ as a linear combination of the right eigenvectors $\hat{\boldsymbol{w}}_{k,i-1/2}$ of the linearised matrix $\hat{\boldsymbol{A}}_{i-1/2}$:

$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \sum_{k=1}^{m_w} \beta_{k,i-1/2}\hat{\boldsymbol{w}}_{k,i-1/2}, \tag{2.41}$$

where the coefficient vector $\boldsymbol{\beta}_{i-1/2}$ is the solution to the linear system (2.41):

$$\boldsymbol{\beta}_{i-1/2} = \boldsymbol{R}^{-1} \cdot (\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1})) = \boldsymbol{L} \cdot (\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1})). \tag{2.42}$$

The f-waves are then

$$\boldsymbol{Z}_{k,i-1/2} = \beta_{k,i-1/2}\hat{\boldsymbol{w}}_{k,i-1/2}. \tag{2.43}$$

These f-waves are related to the waves $\boldsymbol{W}_{k,i-1/2}$ when the wave speeds are nonzero

$$\boldsymbol{W}_{k,i-1/2} = \frac{\boldsymbol{Z}_{k,i-1/2}}{s_{k,i-1/2}}, \tag{2.44}$$

and the fluctuations are

$$\boldsymbol{A}^{-}_{i-1/2}\Delta\boldsymbol{Q}_{i-1/2} = \sum_{k:\ s_k<0} \boldsymbol{Z}_{k,i-1/2}, \tag{2.45}$$

$$\boldsymbol{A}^{+}_{i-1/2}\Delta\boldsymbol{Q}_{i-1/2} = \sum_{k:\ s_k>0} \boldsymbol{Z}_{k,i-1/2}. \tag{2.46}$$

2.8 High-resolutions methods

High-resolutions methods aim to increase the approximation accuracy when evaluating $\boldsymbol{Q}_i^{n+1}$ from $\boldsymbol{Q}_i^n$ and avoid the occurrence of large fluctuations near discontinuities. They take the form

$$\boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n - \frac{\Delta t}{\Delta x}\left(\boldsymbol{A}^{-}\Delta\boldsymbol{Q}^n_{i+1/2} + \boldsymbol{A}^{+}\Delta\boldsymbol{Q}^n_{i-1/2}\right) - \frac{\Delta t}{\Delta x}\left(\hat{\boldsymbol{F}}_{i+1/2} - \hat{\boldsymbol{F}}_{i-1/2}\right), \tag{2.47}$$

where $\hat{\boldsymbol{F}}_{i+1/2}$ is the flux correction

$$\hat{\boldsymbol{F}}_{i+1/2} = \frac{1}{2}\sum_{k=1}^{M_w} |s_{k,i-1/2}|\left(1 - \frac{\Delta t}{\Delta x}|s_{k,i-1/2}|\right)\tilde{\boldsymbol{W}}_{k,i-1/2} \tag{2.48}$$

where $\tilde{\boldsymbol{W}}_{k,i-1/2}$ is the limited version of the kth wave $\boldsymbol{W}_{k,i-1/2}$ obtained by comparing this wave with the jump $\boldsymbol{W}_{k,I-1/2}$ in the upwind direction ($I = i-1$ is $s_{k,i-1/2} > 0$ and $I = i+1$ is $s_{k,i-1/2} < 0$) (LeVeque, 2002, see Chaps. 6 and 15).

2.9 Implementation in Clawpack

Clawpack is a Fortran-based library developed to solve hyperbolic partial differential equations in the form

$$\kappa\frac{\partial}{\partial t}\boldsymbol{q} + \nabla\cdot\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{S}, \tag{2.49}$$

where κ is the capacity function (or constant), $\boldsymbol{q}$ the unknown, $\boldsymbol{f}$ the flux function, and $\boldsymbol{S}$ the source term.

2.9.1 Clawpack installation

Linux is best suited to run the Clawpack library. Installation requires a few additional libraries (see www.clawpack.org/prereqs.html)

- Compiler: gfortran (available from most linux distributions) or ifort (which needs a license, free for academic activities).
- Python: version 3, scipy, numpy, pip, and git

- It can be useful to install anaconda. This environment makes it possible to manage the python packages and offers several functionalities like jupyter (a system of Python-based notebooks that can be read by a web browser), spyder (a scientific environment written for Python), R, and julia. Jupyter notebooks available from github can be read locally on the computer or via a web interface such as nbviewer.jupyter.org/.

Following the procedure with pip (see www.clawpack.org/installing.html) is the easy way to install Clawpack.

Finally, it is necessary to edit the `.bashrc` file by providing the required environment variables

```
export CLAW=$HOME/clawpack-v5.7.0
export FC=gfortran
```

2.9.2 Legacy Clawpack

In its original form developed by Randall Leveque, Clawpack has been based on a set of Fortran 77 routines (LeVeque, 2002).

The main programme was originally located in the file driver.f. This file allocated storage for the arrays used by Clawpack. This is done now automatically, and the user does not need to fill this file. This programme then calls claw1ez, which reads the file claw.data created by the python script setrun.py (it can be created by typing make .data).

The initial condition is contained in the file qinit.f. We should define the cell average value for the entire domaine, but for a continuous function, this average value is the value taken by f at x_i (cell midpoint).

The initial conditions are processed in the file bc1.f. The type of boundary conditions is prescribed in the file claw.data.

The Riemann solver is contained in the file rp1.f. The idea is to decompose any discontinuity into a set of waves $\boldsymbol{W}_k$:

$$\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1} = \sum_{k=1}^{m} \boldsymbol{W}_k$$

avec m is the wave number (which is usually equal to the dimension of the system). For Godunov's method, the value $\boldsymbol{W}_i$ is updated as follows:

$$\boldsymbol{Q}_i^{n+1} = \boldsymbol{Q}_i^n - \frac{\Delta t}{\Delta x}(\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2} + \boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2}),$$

where we distinguish between the left-going wave (coming from the right endpoint $x_{i+1/2}$) :

$$\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i+1/2} = \sum_k \min(\lambda_{i+1/2}^k, 0) \boldsymbol{W}_{k,i+1/2},$$

and the right-going wave

$$\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2} = \sum_k \max(\lambda_{i-1/2}^k, 0) \boldsymbol{W}_{k,i-1/2},$$

The left-going wave is zero if $\lambda_{k,i+1/2} > 0$ (because this the wave moves from right to left) and the right-going wave is zero if $\lambda_{k,i-1/2} < 0$. The Riemann solver needs two input data: the two arrays `ql` and `qr` related to the left and right states. High-resolution methods require further information. Note that for the Riemann solver at the interface $x_{i-1/2}$, we use the following notation for referring to cells

$i-1$ and i: `qr(i-1,:)`$= \boldsymbol{Q}_{r,i-1/2}$ and `ql(i1,:)`$= \boldsymbol{Q}_{l,i-1/2}$, and in this notation, left and right refer to the left and right of the cell i or $i-1$, and not what happens relative to the interface.

The solver provides

- the functions amdq (literally "a minus delta q", which is the vector $\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2}$) and apdq (vecteur $\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i+1/2}$),
- wave (the vawe $\boldsymbol{W}_{k,i-1/2}$), and
- s (the eigenvalue $\lambda_{k,i-1/2}$).

Caveat. Note[3] that the Riemann problem at the interface $x_{i-1/2}$ between cells i–1 and i has data:

- Left state: q^R_{i-1} = qr(:, i-1).
- Right state: q^L_{i-1} =ql(:,i).

This notation is confusing since in the solver direction we use q_l to denote the left state and q_r to denote the right state in specifying Riemann data.

There are many other routines, which are not always required. They are called by the main driver by default, but do not return anything. Among the most important:

- setprob.f: the routine claw1ez calls setprob.f before each execution, which makes it possible to initialize some parameters.
- setaux.f: the routine claw1ez calls the routine setaux.f before each execution to initialize the auxiliary variables (for instance, bed topography).
- b4step1.f: the routine claw1 calls the routine bc4step1.f before each step to perform additional tasks.
- src1.f: if the equation involves a source term, this file is used to correct the solution to the homogenous equation.

2.9.3 Pyclaw

Pyclaw is a python package that offers a convenient framework for pre- and post-processing information, interfacing and running Clawpack or Sharpclaw (Ketcheson *et al.*, 2012; Mandli *et al.*, 2016). It can call Fortran or Python routines. Interfaced with PyWENO and PETSc, Pyclaw provides extended functionality in terms of parallel computing (Ketcheson *et al.*, 2012).

[3] See www.clawpack.org/riemann.html.

CHAPTER 3

Examples

3.1 Acoustic waves

3.1.1 Governing equation

When linearised, the acoustic wave equation takes the form

$$\frac{\partial p}{\partial t} + K\frac{\partial u}{\partial x} = 0,$$
$$\frac{\partial u}{\partial t} + \frac{1}{\varrho}\frac{\partial p}{\partial x} = 0,$$

where K is the bulk modulus, ϱ the density, $u(x,\ t)$ and $p(x,\ t)$ the velocity and pressure. We define the speed of sound as $c = \sqrt{K/\varrho}$ and impedance $Z = \sqrt{K\varrho}$. In a tensorial form, the acoustic wave equation is:

$$\frac{\partial \boldsymbol{q}}{\partial t} + \boldsymbol{A}\cdot\frac{\partial \boldsymbol{q}}{\partial x},\ \text{with}\ \boldsymbol{q} = \begin{pmatrix} p \\ u \end{pmatrix}\ \text{and}\ \boldsymbol{A} = \begin{pmatrix} O & K \\ \varrho^{-1} & 0 \end{pmatrix}.$$

We define the right and left eigenvector matrices $\boldsymbol{R}$ and $\boldsymbol{L}$

$$\boldsymbol{R} = \begin{pmatrix} -Z & Z \\ 1 & 1 \end{pmatrix}\ \text{et}\ \boldsymbol{L} = \frac{1}{2}\begin{pmatrix} Z^{-1} & 1 \\ Z & 1 \end{pmatrix}$$

and the eigenvalue matrix $\boldsymbol{\Lambda}$

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda^1 & 0 \\ 0 & \lambda^2 \end{pmatrix}\ \text{avec}\ \lambda^1 = -c\ \text{et}\ \lambda^2 = +c.$$

We diagonalize the matrix $\boldsymbol{A}$

$$\boldsymbol{A} = \boldsymbol{R}\cdot\boldsymbol{\Lambda}\cdot\boldsymbol{L}.$$

By introducing the Riemann variables

$$\boldsymbol{r} = \boldsymbol{L}\cdot\boldsymbol{q}$$

we want to solve

$$\frac{\partial \boldsymbol{r}}{\partial t} + \boldsymbol{\Lambda}\cdot\frac{\partial \boldsymbol{r}}{\partial x} = 0,$$

subject to the initial conditions

$$r_i = \begin{cases} r_{i,l}\ \text{if}\ x < 0 \\ r_{i,r}\ \text{if}\ x > 0 \end{cases}$$

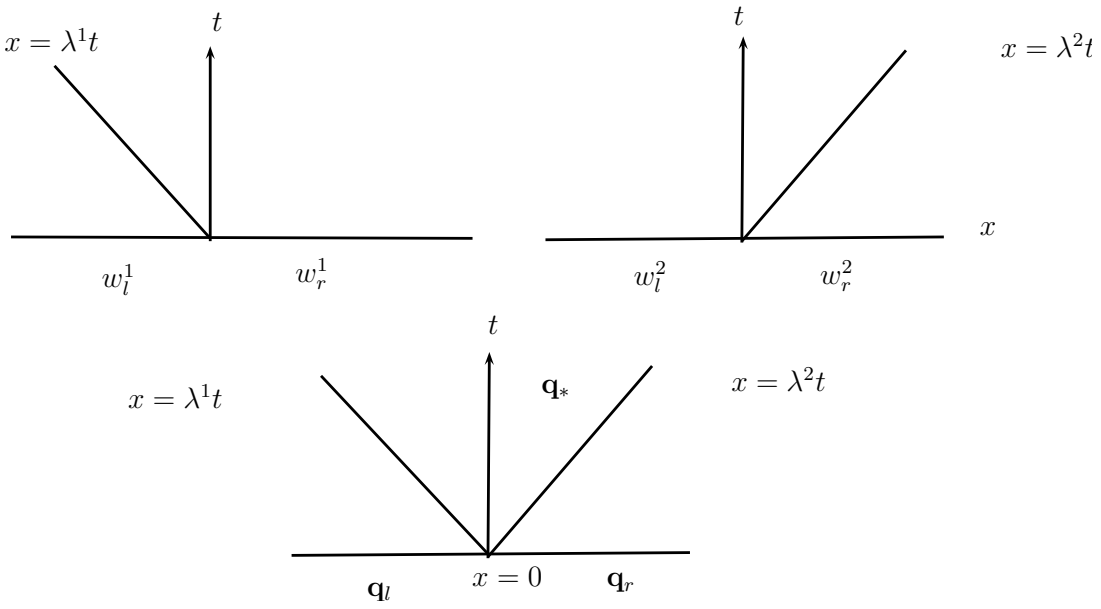


Figure 3.1 Solution to the Riemann problem.

In a Riemann problem, the left and right states can be connected using the right eigenvectors:

$$\boldsymbol{q}_r - \boldsymbol{q}_l = \alpha^1 \boldsymbol{w}^1 + \alpha^2 \boldsymbol{w}^2 = \boldsymbol{R} \cdot \boldsymbol{\alpha},$$

thus

$$\boldsymbol{\alpha} = \boldsymbol{R}^{-1} \cdot (\boldsymbol{q}_r - \boldsymbol{q}_l) = \boldsymbol{L} \cdot (\boldsymbol{q}_r - \boldsymbol{q}_l),$$

which leads to:

$$\alpha_1 = \frac{1}{2}\left(-\frac{p_r - p_l}{Z} + u_r - u_l\right),$$

$$\alpha_2 = \frac{1}{2}\left(\frac{p_r - p_l}{Z} + u_r - u_l\right),$$

The jump from $\boldsymbol{q}_l$ to $\boldsymbol{q}_*$ is $\boldsymbol{W}^1 = \alpha^1 \boldsymbol{r}^1$ while the jump from $\boldsymbol{q}_*$ to $\boldsymbol{q}_r$ is $\boldsymbol{W}^2 = \alpha^2 \boldsymbol{r}^2$.

3.1.2 Implementation

In classic Clawpack, the algorithm for the solver is quite simple.

```
1  ! ==================================================
2  subroutine rp1(maxm,meqn,mwaves,maux,mbc,mx,ql,qr,auxl,auxr,wave,s,amdq,
       apdq)
3  ! ==================================================
4
5      implicit double precision (a-h,o-z)
6
7      dimension wave(meqn, mwaves, 1-mbc:maxm+mbc)
8      dimension    s(mwaves,1-mbc:maxm+mbc)
9      dimension   ql(meqn, 1-mbc:maxm+mbc)
10     dimension   qr(meqn, 1-mbc:maxm+mbc)
11     dimension apdq(meqn, 1-mbc:maxm+mbc)
12     dimension amdq(meqn, 1-mbc:maxm+mbc)
13
```

```
!     local arrays
!     ------------
    dimension delta(2)

!     # density, bulk modulus, and sound speed, and impedance of medium:
!     # (should be set in setprob.f)
    common /cparam/ rho,bulk,cc,zz

!     # find a1 and a2, the coefficients of the 2 eigenvectors:
    do 20 i = 2-mbc, mx+mbc
        delta(1) = ql(1,i) - qr(1,i-1)
        delta(2) = ql(2,i) - qr(2,i-1)
        a1 = (-delta(1) + zz*delta(2)) / (2.d0*zz)
        a2 =  (delta(1) + zz*delta(2)) / (2.d0*zz)

    !        # Compute the waves.

        wave(1,1,i) = -a1*zz
        wave(2,1,i) = a1
        s(1,i) = -cc

        wave(1,2,i) = a2*zz
        wave(2,2,i) = a2
        s(2,i) = cc

    20 END DO


!     # compute the leftgoing and rightgoing flux differences:
!     # Note s(1,i) < 0   and   s(2,i) > 0.

    do 220 m=1,meqn
        do 220 i = 2-mbc, mx+mbc
            amdq(m,i) = s(1,i)*wave(m,1,i)
            apdq(m,i) = s(2,i)*wave(m,2,i)
    220 END DO

    return
    end subroutine rp1
```

3.1.3 Implementation in Pyclaw

Here is an example of notebook setting up a solver for the acoustic wave equations.

```
%matplotlib inline

from numpy import sqrt, exp, cos
from clawpack import riemann
from clawpack import pyclaw

def setup(outdir='./_output',  output_style=1):

    riemann_solver = riemann.acoustics_1D_py.acoustics_1D
    solver = pyclaw.ClawSolver1D(riemann_solver)
    solver.limiters = pyclaw.limiters.tvd.MC
    solver.kernel_language = 'Python'

```

```
    x = pyclaw.Dimension(0.0, 1.0, 100, name='x')
    domain = pyclaw.Domain(x)
    num_eqn = 2
    state = pyclaw.State(domain, num_eqn)

    solver.bc_lower[0] = pyclaw.BC.periodic
    solver.bc_upper[0] = pyclaw.BC.periodic

    rho = 1.0   # Material density
    bulk = 1.0  # Material bulk modulus

    state.problem_data['rho'] = rho
    state.problem_data['bulk'] = bulk
    state.problem_data['zz'] = sqrt(rho*bulk)   # Impedance
    state.problem_data['cc'] = sqrt(bulk/rho)   # Sound speed

    xc = domain.grid.x.centers
    beta = 100
    gamma = 0
    x0 = 0.75
    state.q[0, :] = exp(-beta * (xc-x0)**2) * cos(gamma * (xc - x0))
    state.q[1, :] = 0.0

    solver.dt_initial = domain.grid.delta[0] / state.problem_data['cc'] *
   0.1

    claw = pyclaw.Controller()
    claw.solution = pyclaw.Solution(state, domain)
    claw.solver = solver
    claw.outdir = outdir
    claw.output_style = output_style
    output_style = 1
    claw.tfinal = 1.0
    claw.num_output_times = 10
    claw.keep_copy = True
    claw.setplot = setplot

    return claw


def setplot(plotdata):
    """
    Specify what is to be plotted at each frame.
    Input:  plotdata, an instance of visclaw.data.ClawPlotData.
    Output: a modified version of plotdata.
    """
    plotdata.clearfigures()  # clear any old figures,axes,items data

    # Figure for pressure
    plotfigure = plotdata.new_plotfigure(name='Pressure', figno=1)

    # Set up for axes in this figure:
    plotaxes = plotfigure.new_plotaxes()
    plotaxes.axescmd = 'subplot(211)'
    plotaxes.ylimits = [-0.2, 1.0]
    plotaxes.title = 'Pressure'

    # Set up for item on these axes:
```

```
    plotitem = plotaxes.new_plotitem(plot_type='1d_plot')
    plotitem.plot_var = 0
    plotitem.plotstyle = '-o'
    plotitem.color = 'b'
    plotitem.kwargs = {'linewidth': 2, 'markersize': 5}

    # Set up for axes in this figure:
    plotaxes = plotfigure.new_plotaxes()
    plotaxes.axescmd = 'subplot(212)'
    plotaxes.xlimits = 'auto'
    plotaxes.ylimits = [-0.5, 1.1]
    plotaxes.title = 'Velocity'

    # Set up for item on these axes:
    plotitem = plotaxes.new_plotitem(plot_type='1d_plot')
    plotitem.plot_var = 1
    plotitem.plotstyle = '-'
    plotitem.color = 'b'
    plotitem.kwargs = {'linewidth': 3, 'markersize': 5}

    return plotdata
```

To run the script and plot one result here (frame 10) using setplot, the following can be done:

```
claw = setup()
claw.run()

from clawpack.visclaw import data
from clawpack.visclaw import frametools
plotdata = data.ClawPlotData()
plotdata.setplot = setplot
claw.plotdata = frametools.call_setplot(setplot,plotdata)

frame = claw.load_frame(10)
f=claw.plot_frame(frame)
```

```
We can also plot a frame directly

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

frame = claw.frames[5]
w = frame.q[0,:]
x = frame.state.grid.c_centers
x = x[0]

plt.plot(x, w)
```

Here is how Pyclow has encoded the solver of the Riemann problem.

```
def acoustics_1D(q_l,q_r,aux_l,aux_r,problem_data):
    r"""
    Basic 1d acoustics riemann solver, with interleaved arrays

    *problem_data* is expected to contain -
     - *zz* - (float) Impedence
     - *cc* - (float) Speed of sound

```

```
    See :ref:`pyclaw_rp` for more details.

    :Version: 1.0 (2009-02-03)
    """
    import numpy as np

    # Convenience
    num_rp = np.size(q_l,1)

    # Return values
    wave = np.empty( (num_eqn, num_waves, num_rp) )
    s = np.empty( (num_waves, num_rp) )
    amdq = np.empty( (num_eqn, num_rp) )
    apdq = np.empty( (num_eqn, num_rp) )

    # Local values
    delta = np.empty(np.shape(q_l))

    delta = q_r - q_l
    a1 = (-delta[0,:] + problem_data['zz']*delta[1,:]) / (2.0 *
problem_data['zz'])
    a2 = (delta[0,:] + problem_data['zz']*delta[1,:]) / (2.0 * problem_data
['zz'])

    # Compute the waves
    # 1-Wave
    wave[0,0,:] = -a1 * problem_data['zz']
    wave[1,0,:] = a1
    s[0,:] = -problem_data['cc']

    # 2-Wave
    wave[0,1,:] = a2 * problem_data['zz']
    wave[1,1,:] = a2
    s[1,:] = problem_data['cc']

    # Compute the left going and right going fluctuations
    for m in range(num_eqn):
        amdq[m,:] = s[0,:] * wave[m,0,:]
        apdq[m,:] = s[1,:] * wave[m,1,:]

    return wave, s, amdq, apdq
```

CHAPTER 4

Burger's equations

4.1 Theory

Let us consider the Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, \tag{4.1}$$

or in the flux form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \text{ where } f(u) = \frac{u^2}{2}.$$

The solution to the Riemann problem has two types of solution:

- *rarefaction wave*: $u = U(\xi)$ with $\xi = x/t$. Substituting this form into Eq. (4.1)

$$U(\xi) = \xi.$$

- *shock wave*: The Rankine-Hugoniot equation tells us that the shock moves at speed:

$$\dot{s} = \frac{[\![f(u)]\!]}{[\![u]\!]}.$$

The solution to the Riemann problem depends on the sign of $u_r - u_l$:

- If $u_r > u_l$, we have a rarefaction wave separating two constant states. The characteristic curves separating $U(\xi)$ from u_l and u_r are, respectively, $x = u_l t$ and $x = u_r t$.
- If $u_r < u_l$, we have a shock wave moving at speed

$$\dot{s} = \frac{1}{2}(u_r + u_l)$$

4.2 Approximate solvers

In Clawpack, the Riemann solver expresses the idea the Q_i values evolve because of fluctuations:

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(\mathcal{A}^+ \Delta Q_{i-1/2} + \mathcal{A}^- \Delta Q_{i+1/2}),$$

where the fluctuations are

$$\mathcal{A}^+\Delta Q_{i-1/2} = f(Q_i) - f(Q^{\downarrow}_{i-1/2}),$$
$$\mathcal{A}^-\Delta Q_{i+1/2} = f(Q^{\downarrow}_{i+1/2}) - f(Q_i),$$

where $Q_{i\pm1/2}$ represents the value advected along the characteristic coming from the $x_{i\pm1/2}$ interface. When the solution to the Riemann problem is not a transonic wave, the idea is to approximate this solution as a shock wave (even though it is a rarefaction wave). The show propagates the wave $\mathcal{W}$ at a speed s:

$$\mathcal{W}_{i-1/2} = Q_i - Q_{i-1},$$
$$s_{i-1/2} = \frac{f(Q_i) - f(Q_{i-1})}{Q_i - Q_{i-1}},$$

for $Q_i \neq Q_{i-1}$. We then deduce

$$\mathcal{A}^+\Delta Q_{i-1/2} = s_{i-1/2}\mathcal{W}_{i-1/2},$$
$$\mathcal{A}^-\Delta Q_{i-1/2} = s_{i-1/2}\mathcal{W}_{i-1/2},$$

When the solution to the Riemann problem is a transonic wave, we use the definition of the fluctuations

$$\mathcal{A}^+\Delta Q_{i-1/2} = f(Q_i) - f(q_s),$$
$$\mathcal{A}^-\Delta Q_{i-1/2} = f(Q_s) - f(Q_{i-1}),$$

where q_s is the value such as $f'(q_s) = 0$ (vertical characteristic corresponding to $x - x_{i-1/2} = 0 \cdot t$). For the Burgers equation we have $q_s = 0$.

To summarize, we express the functions `amdp`, `apdp`, `s`, and `W`:

$$\mathcal{A}^+\Delta U_{i-1/2} = s_{i-1/2}\mathcal{W}_{i-1/2},$$
$$\mathcal{A}^-\Delta U_{i-1/2} = s_{i-1/2}\mathcal{W}_{i-1/2},$$

with

$$\mathcal{W}_{i-1/2} = U_i - U_{i-1},$$
$$s_{i-1/2} = \frac{1}{2}(U_i + U_{i-1}),$$

but if $U_{i-1} < 0$ et $U_i > 0$ then

$$\mathcal{A}^+\Delta U_{i-1/2} = \frac{1}{2}U_i^2,$$
$$\mathcal{A}^-\Delta U_{i-1/2} = -\frac{1}{2}U_{i-1}^2,$$

In Clawpack, the treatment of the transonic wave is called an *entropy fix*, and its use in the Riemann solver is indicated through the Boolean variable `efix`.

The Roe solver involves linearising Burger's equation (4.1):

$$\frac{\partial q}{\partial t} + \hat{q}\frac{\partial q}{\partial x} = 0, \tag{4.2}$$

where $\hat{q}$ is the intermediate state

$$\hat{q} = \frac{q_l + q_r}{2}$$

to be consistent with the Rankine-Hugoniot equation. This solver is equivalent to the one described above.

4.2.1 Implementation in Clawpack

```
c
c
      efix = .true.    !# Compute correct flux for transonic rarefactions
c
      do 30 i=2-mbc,mx+mbc
c
c        # Compute the wave and speed
c
         wave(i,1,1) = ql(i,1) - qr(i-1,1)
         s(i,1) = 0.5d0 * (qr(i-1,1) + ql(i,1))
c
c
c        # compute left-going and right-going flux differences:
c        ------------------------------------------------------
c
         amdq(i,1) = dmin1(s(i,1), 0.d0) * wave(i,1,1)
         apdq(i,1) = dmax1(s(i,1), 0.d0) * wave(i,1,1)
c
         if (efix) then
c           # entropy fix for transonic rarefactions:
            if (qr(i-1,1).lt.0.d0 .and. ql(i,1).gt.0.d0) then
               amdq(i,1) = - 0.5d0 * qr(i-1,1)**2
               apdq(i,1) =   0.5d0 * ql(i,1)**2
               endif
            endif
   30    continue
```

4.2.2 Implementation in Pyclaw

```
def burgers_1D(q_l,q_r,aux_l,aux_r,problem_data):
    r"""
    Riemann solver for Burgers equation in 1d
    *problem_data* should contain -
     - *efix* - (bool) Whether a entropy fix should be used, if not present
    ,
       false is assumed
    """

    num_rp = q_l.shape[1]
    # Output arrays
    wave = np.empty( (num_eqn, num_waves, num_rp) )
    s = np.empty( (num_waves, num_rp) )
    amdq = np.empty( (num_eqn, num_rp) )
    apdq = np.empty( (num_eqn, num_rp) )

    # Basic solve
    wave[0,:,:] = q_r - q_l
    s[0,:] = 0.5 * (q_r[0,:] + q_l[0,:])

    s_index = np.zeros((2,num_rp))
    s_index[0,:] = s[0,:]
    amdq[0,:] = np.min(s_index,axis=0) * wave[0,0,:]
    apdq[0,:] = np.max(s_index,axis=0) * wave[0,0,:]

```

```
25      # Compute entropy fix
26      if problem_data['efix']:
27          transonic = (q_l[0,:] < 0.0) * (q_r[0,:] > 0.0)
28          amdq[0,transonic] = -0.5 * q_l[0,transonic]**2
29          apdq[0,transonic] = 0.5 * q_r[0,transonic]**2
30
31      return wave, s, amdq, apdq
```

In this routine, `ql` is the left initial condition. It is a $p \times N$ array ($p = 1$ the problem dimension, and N the number of cells). So, `num_rp = q_l.shape[1]` gives N. First, the `amdp`, `apdp`, `s`, and `W` are initialised, then `s` and `W` are defined. Finally, the fluctuations are defined using the numpy function `numpy.min`, which provides the minimum value: absolute, for each column (with the `axis=0` option), or for each row (with the `axis=1` option). For instance, the lines

```
import numpy as np
x=np.array([[1,4],[2,5],[3,-2]])
np.min(x,axis=None)
np.min(x,axis=0)
np.min(x,axis=1)
```

provide the values: -2, `array([ 1, -2])` and `array([ 1,  2, -2])`, respectively.

4.2.3 Examples

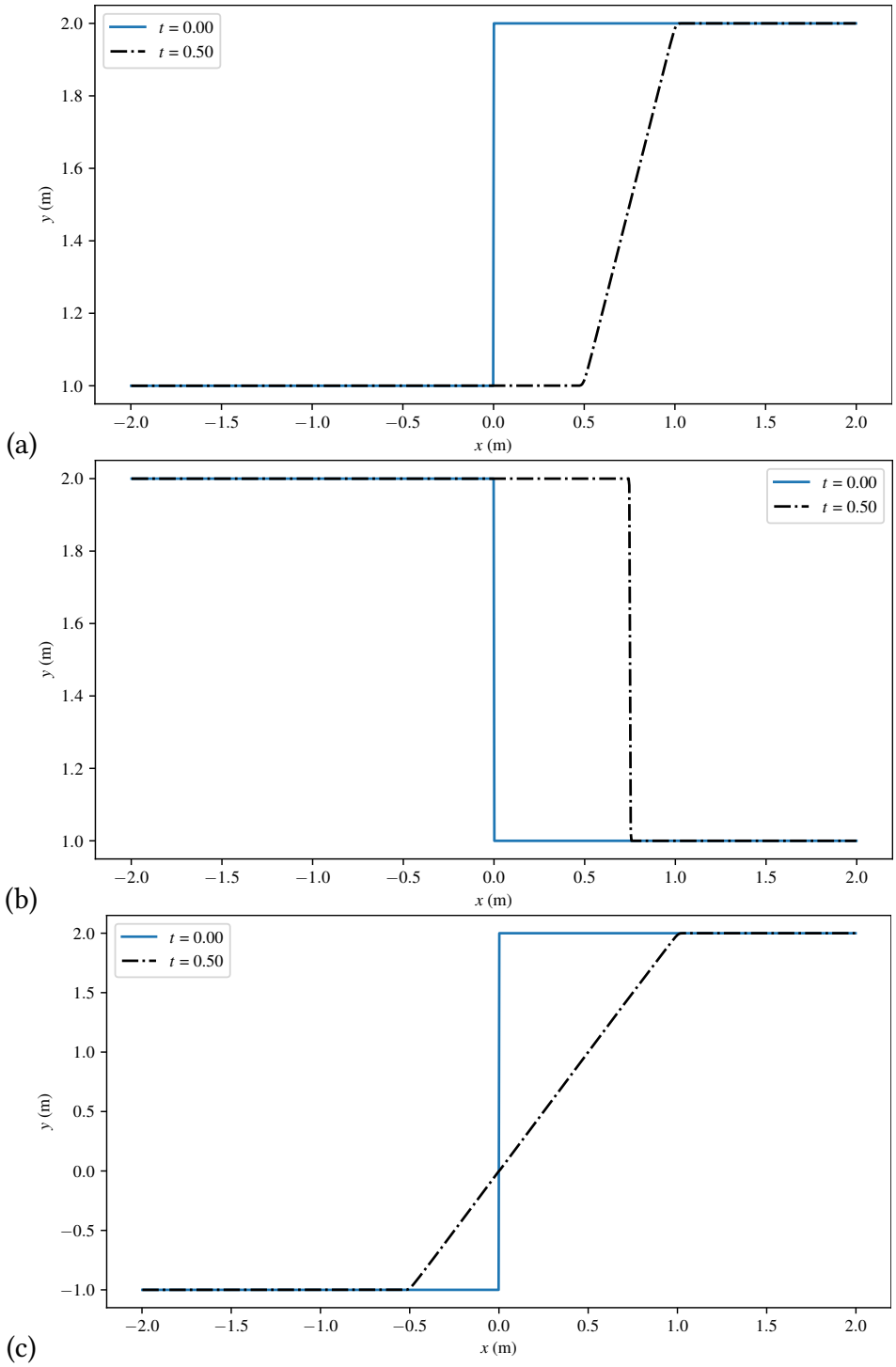


Figure 4.1 Solutions to the Riemann problem: (a) rarefaction wave; (b) shock wave; (c) transonic rarefaction wave.

4.3 Nonlinear advection equation with a source term

4.3.1 Theoretical considerations

Let us consider a rainfall of intensity I over a sloping bed inclined at α (see Fig. 4.2). There are two possible runoff mechanisms: superficial or hyporheic flow. For both cases, we assume that the flow depth is $h(x,t)$ and velocity $u(x,t)$ related to h: $u = ah^b$, where a and b are two coefficients: $a = C\sqrt{\alpha}$ et $b = 1/2$ if one considers runoff with a Chézy friction C.

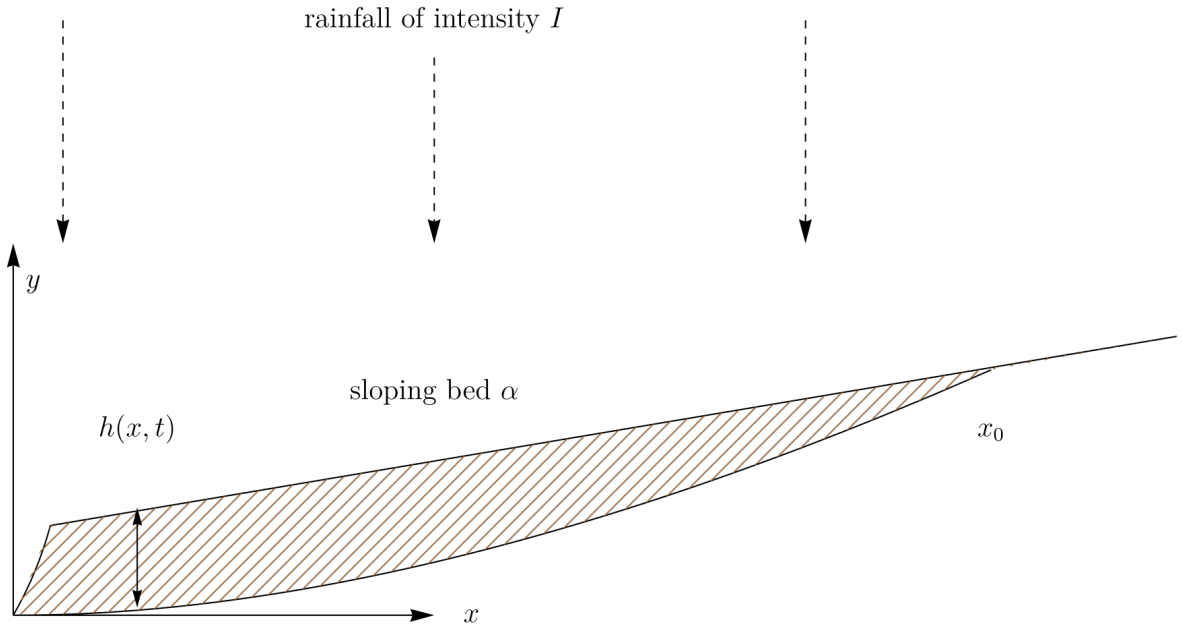


Figure 4.2 Flow generated by a rainfall.

The governing equation is given by mass conservation:

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} = I. \tag{4.3}$$

As we have $u = ah^b$, we obtain:

$$\frac{\partial h}{\partial t} + c(h)\frac{\partial h}{\partial x} = I \text{ with } c(h) = a(b+1)h^b.$$

or in a characteristic form:

$$\frac{\mathrm{d}h}{\mathrm{d}t} = I \text{ along } \frac{\mathrm{d}x}{\mathrm{d}t} = c(h) = a(b+1)h^b,$$

and we assume that initially the flow depth is zero (dry bed: $h(x,0) = 0$) and no water comes from upstream of x_0 ($h(x_0,\, t) = 0$). The solution to the characteristic equation is $h = It$ along the characteristic curve:

$$x = \int a(b+1)h^b \mathrm{d}t + x_1 = aI^b t^{1+b} + x_1, \tag{4.4}$$

where x_1 is constante of integration (such that at $x = x_1$, we have $h = 0$). This implies for any x ($0 \leq x \leq x_0$, with the frame used in Fig. 4.2, $x_0 = L_0$), we have:

- a linear growth $h(x,\,t) = It$ until time t_∞ such that $aI^b t_\infty^{1+b} = x_0 - x$;
- a stationary state for:

$$h(x,\,t) = h_\infty(x) = \left(\frac{I(x_0 - x)}{a}\right)^{1/(1+b)}.$$

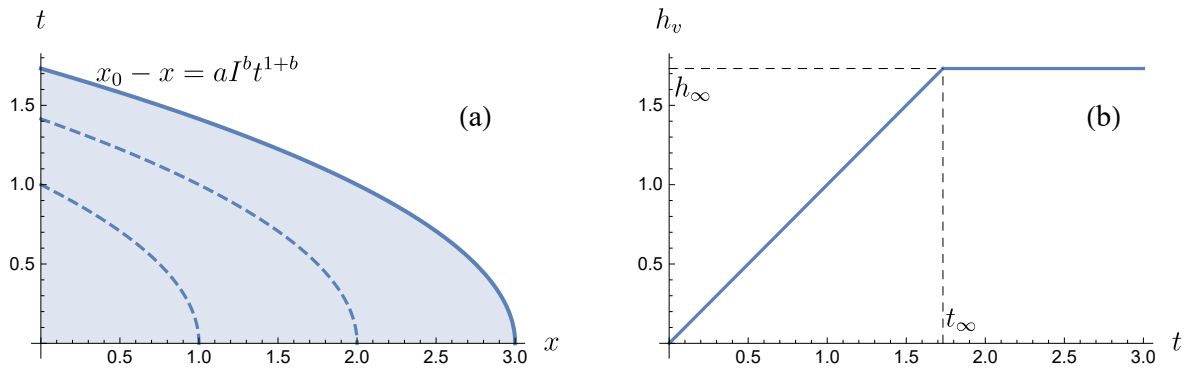


Figure 4.3 (a) characteristic curves (4.4). The thick line represents $x = aI^b t^{1+b}$, the path of a fluid parcel emitted from x_0. The coloured area represents the domain controlled by the initial condition $h = 0$ for which we observe a linear growth $h(x,\,t) = It$. Above the curve $x = aI^b t^{1+b}$, the depth is constant and equal to $h_\infty(x)$. (b) Flow depth variation at $x = 0$. Computation for arbitrary values $a = 1$ 1/s $b = 1$, $I = 1$ m/s, and $x_0 = 3$ m.

4.3.2 Numerical implementation

```
import numpy as np
import matplotlib.pyplot as plt
import os
from clawpack import riemann
plt.ioff()

#!/usr/bin/env python
# encoding: utf-8

r"""
Burgers' equation
"""
def source_term(solver, state, dt):
    i  = state.problem_data['i']
    h  = state.q[0, :]
    # Update to momentum
    state.q[0, :] +=   dt * i

def inlet_bc(state,dim,t,qbc,auxbc,num_ghost):
    "inlet boundary conditions"
    qbc[0, :num_ghost] = 0.

def b4step(solver,state):
    h  = state.q[0,:]
    t  = state.t
```

```
    hf = h[-1]
    front.append([t,hf])


def setup(use_petsc=0,kernel_language='Fortran',outdir='./_output',
    solver_type='classic'):

    if use_petsc:
        import clawpack.petclaw as pyclaw
    else:
        from clawpack import pyclaw

    if kernel_language == 'Python':
        riemann_solver = riemann.burgers_1D_py.burgers_1D
    elif kernel_language == 'Fortran':
        riemann_solver = riemann.burgers_1D

    if solver_type=='sharpclaw':
        solver = pyclaw.SharpClawSolver1D(riemann_solver)
    else:
        solver = pyclaw.ClawSolver1D(riemann_solver)
        solver.limiters = pyclaw.limiters.tvd.vanleer
    solver.kernel_language = kernel_language

    solver.bc_lower[0] = pyclaw.BC.custom
    solver.user_bc_lower = inlet_bc
    solver.bc_upper[0] = pyclaw.BC.extrap
    solver.step_source = source_term
    solver.before_step = b4step

    x = pyclaw.Dimension(0.0,10.0,1000,name='x')
    domain = pyclaw.Domain(x)
    num_eqn = 1
    state = pyclaw.State(domain,num_eqn)
    xc = state.grid.x.centers
    state.q[0,:] = 0.
    state.problem_data['efix']=True
    state.problem_data['i'] = 1

    claw = pyclaw.Controller()
    claw.tfinal = 10
    claw.num_output_times = 20
    claw.solution = pyclaw.Solution(state,domain)
    claw.solver = solver
    claw.outdir = outdir
    claw.setplot = setplot
    claw.keep_copy = True

    return claw

def setplot(plotdata):
    """
    Plot solution using VisClaw.
    """
    plotdata.clearfigures()  # clear any old figures,axes,items data
    # Figure for q[0]
    plotfigure = plotdata.new_plotfigure(name='q[0]', figno=0)

```

```
    # Set up for axes in this figure:
    plotaxes = plotfigure.new_plotaxes()
    plotaxes.xlimits = 'auto'
    plotaxes.ylimits = [-1., 2.]
    plotaxes.title = 'q[0]'
    # Set up for item on these axes:
    plotitem = plotaxes.new_plotitem(plot_type='1d')
    plotitem.plot_var = 0
    plotitem.plotstyle = '-o'
    plotitem.color = 'b'

    return plotdata
```

```
front = []
claw = setup()
claw.run()

ind=5
ind2=20
delta_t=claw.tfinal/claw.num_output_times

fig = plt.figure(figsize=(8,4))
left, bottom, width, heigth = 0.2, 0.2, 0.8, 0.8
ax = fig.add_axes((left ,bottom, width, heigth ))
ax.ylimits = [0,0.1]
frame = claw.frames[ind]
h = frame.q[0,:]
frame = claw.frames[ind2]
h2 = frame.q[0,:]

x = frame.state.grid.x.centers
ax.plot(x,h,label='t =  {:.2f}'.format(ind*delta_t))
ax.plot(x,h2, 'k-.',label='t  {:.2f}'.format(ind2*delta_t))

ax.set_xlabel(r'$x$ (m)')
ax.set_ylabel(r'$y$ (m)')
ax.legend()
plt.show()
```

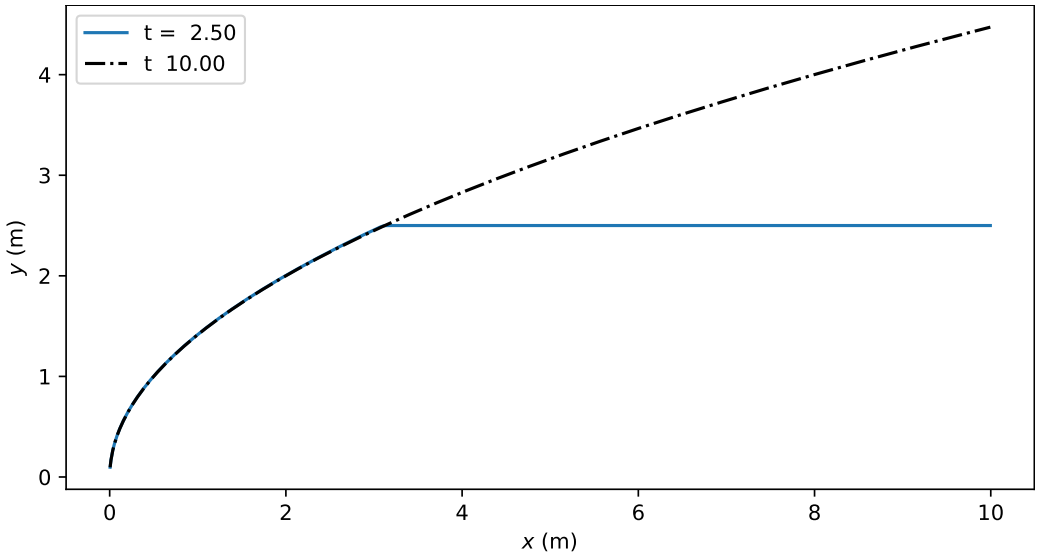


Figure 4.4 Numerical solutions at time $t = 2.5$ s and $t = 10$ s.

CHAPTER 5

Shallow water equations

5.1 Theory

The shallow water equations (also called the Saint-Venant) equations consists of the mass and momentum balance equation for a depth-averaged water flow. In this chapter, we consider the simplest case, in which the bottom is horizonal and exerts no resistance, and the flow is one-directional. In this case, the conservative form of the governing equations comprises the mass balance equation

$$\frac{\partial h}{\partial t} + \frac{\partial q}{\partial x} = 0, \tag{5.1}$$

where h denotes the flow depth, $q = hu$ is the flow rate, and u the depth-averaged velocity. The second equation is the momentum balance equation

$$\frac{\partial q}{\partial t} + \frac{\partial hu^2}{\partial x} + gh\frac{\partial h}{\partial x} = 0, \tag{5.2}$$

where g is gravitational acceleration, and the unknowns are q and h. The non-conservative form is useful when the solution is smooth:

$$\frac{\partial h}{\partial t} + \frac{\partial q}{\partial x} = 0, \tag{5.3}$$

where h denotes the flow depth, $q = hu$ is the flow rate, and u the depth-averaged velocity. The second equation is the momentum balance equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + g\frac{\partial h}{\partial x} = 0. \tag{5.4}$$

In a matrix form, Eqs. (5.1)-(5.2) takes the form:

$$\frac{\partial}{\partial t}\boldsymbol{Q} + \frac{\partial}{\partial x}\boldsymbol{f}(\boldsymbol{Q}) = 0, \tag{5.5}$$

where

$$\boldsymbol{f} = \begin{pmatrix} q \\ q^2/h + gh^2/2 \end{pmatrix} \text{ and } \boldsymbol{Q} = \begin{pmatrix} h \\ q \end{pmatrix}. \tag{5.6}$$

The Jacobian is

$$\boldsymbol{f}' = \begin{pmatrix} 0 & 1 \\ -q^2/h^2 + gh & 2q/h \end{pmatrix}, \tag{5.7}$$

whose eigenvalues are

$$\lambda_1 = u - \sqrt{gh} \text{ and } \lambda_2 = u + \sqrt{gh}, \tag{5.8}$$

associated with the right eigenvectors:

$$\boldsymbol{w}_1 = \begin{pmatrix} 1 \\ u - \sqrt{gh} \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} 1 \\ u + \sqrt{gh} \end{pmatrix}. \tag{5.9}$$

5.1.1 Dam break solution

Let us consider the dam break problem on a wet bed (that is, the initial flow depth is nonzero everywhere):

$$h(x,\ 0) = \begin{cases} h_l \text{ for } x < 0, \\ h_r \text{ for } x > 0, \end{cases} \tag{5.10}$$

and $u(x,\ 0)$ everywhere, and we assume that $h_l > h_l$. The solution's structure is shown by Fig. 5.3. There is an intermediate state $\boldsymbol{Q} = (h_*,\ q_*)$ separated from the left initial state $\boldsymbol{Q}_l$ by a rarefaction wave, and from the right initial state $\boldsymbol{Q}_r$ by a shock wave.

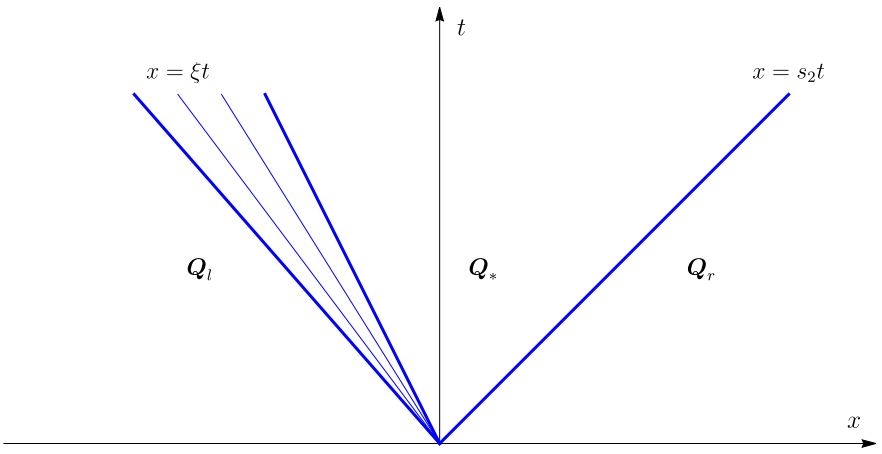


Figure 5.1 dambreak with an intermediate state separated from the left initial state by a rarefaction wave, and from the right initial state by a shock wave.

The intermediate state satisfies the Rankine-Hugoniot condition:

$$s(\boldsymbol{Q}_* - \boldsymbol{Q}_r) = \boldsymbol{f}(\boldsymbol{Q}_*) - \boldsymbol{f}(\boldsymbol{Q}_r), \tag{5.11}$$

which implies that the shock speed is

$$s = \frac{q_* - q_r}{h_* - h_r} = u_* \mp \sqrt{g h_r \frac{h_r + h_*}{2h_*}},$$

and the flow rare depends on the initial rate (which is 0 here) and depth on the right:

$$q_* = q_r + (h_* - h_r)\left(u_r \pm \sqrt{g h_r \left(1 + \frac{h_* - h_r}{h_r}\right)\left(1 + \frac{h_* - h_r}{2h_r}\right)}\right), \tag{5.12}$$

which can be transformed into

$$u_* = u_r \pm (h_* - h_r)\sqrt{\frac{g}{2}\left(\frac{1}{h_r} + \frac{1}{h_*}\right)}. \tag{5.13}$$

The intermediate state has also to be compatible with a rarefaction wave solution. A rarefaction wave is a similarity solution $\boldsymbol{Q}(\xi)$ with $\xi = x/t$. Substituting this form into the hyperbolic system (5.5) gives:

$$-\xi \boldsymbol{Q}'(\xi) + \boldsymbol{f}'(\boldsymbol{Q}) \cdot \boldsymbol{Q}'(\xi) = 0,$$

which shows that $\boldsymbol{Q}'(\xi)$ is a right eigenvector of the Jacobian matrix $\boldsymbol{f}'$, and thus there exists a scalar coefficient $\alpha(\xi)$ such that

$$\boldsymbol{Q}'(\xi) = \alpha(\xi)\boldsymbol{w}_k,$$

with $k = 1, 2$, Let us assume that $\alpha = 1$ and $k = 1$ (that is, we are looking for the 1-rarefaction wave), then we have to solve

$$\boldsymbol{Q}'(\xi) = \begin{pmatrix} h' \\ q' \end{pmatrix} = \begin{pmatrix} 1 \\ u - \sqrt{gh} \end{pmatrix}.$$

We deduce by setting the first constant of integration to zero:

$$h(\xi) = \xi,$$

$$q' = \frac{q}{\xi} - \sqrt{g\xi} \Rightarrow q(\xi) = a\xi - 2\xi\sqrt{g\xi},$$

where a is constant of integration. As we have $h = \xi$, this means that we also have $q(h) = ah - 2h\sqrt{gh}$ We impose that the intermediate state lies on the rarefaction wave, and thus

$$q_* = ah_* - 2h_*\sqrt{gh_*} \Rightarrow a = u_* + 2h_*\sqrt{gh_*}.$$

The 1-rarefaction is thus the curve

$$q(h) = hu_* + 2h(\sqrt{gh_*} - \sqrt{gh}). \tag{5.14}$$

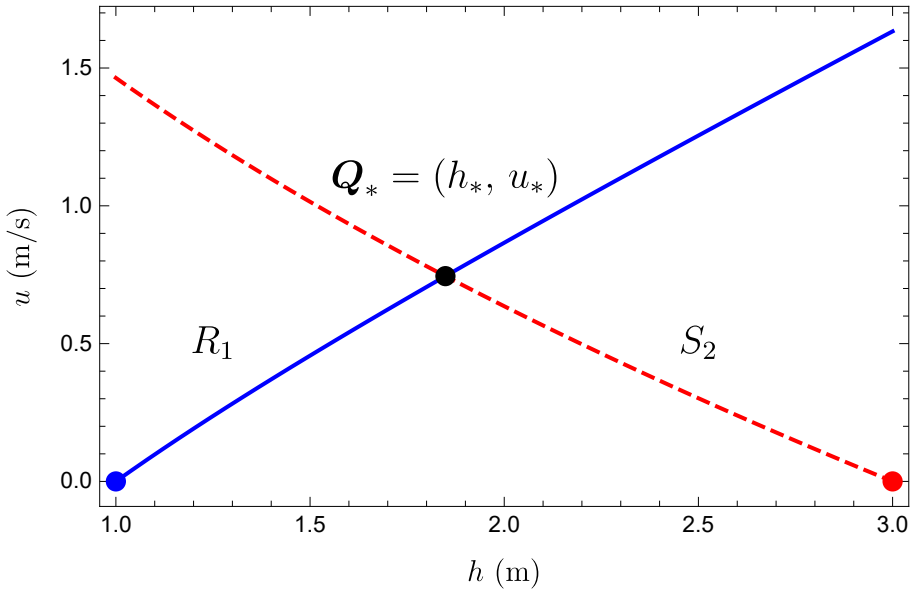


Figure 5.2 Phase plane for $h_l = 3$ and $h_r = 1$ (with $g = 1$ m/s^2).

5.2 Approximate solver: the Roe solver

5.2.1 Derivation

The idea underpinning the Roe solver's derivation is a change of variable. The linearised flux matrix has been defined in § 2.5

$$\hat{\boldsymbol{A}}_{i-1/2} = \int_0^1 \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{q}(\xi))}{\mathrm{d}\boldsymbol{q}}\mathrm{d}\xi. \tag{5.15}$$

We make the following change of variable

$$\boldsymbol{z} = \frac{\boldsymbol{q}}{\sqrt{h}} = \begin{pmatrix} \sqrt{h} \\ u\sqrt{h} \end{pmatrix}, \tag{5.16}$$

or by inversion

$$\boldsymbol{q} = \begin{pmatrix} z_1^2 \\ z_1 z_2 \end{pmatrix} \Rightarrow \frac{\partial \boldsymbol{q}}{\partial \boldsymbol{z}} = \begin{pmatrix} 2z_1 & 0 \\ z_2 & z_1 \end{pmatrix}. \tag{5.17}$$

The flux function and its Jacobian are

$$\boldsymbol{f} = \begin{pmatrix} z_1 z_2 \\ z_2^2 + \frac{1}{2} g z_1^4 \end{pmatrix} \Rightarrow \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{z}} = \begin{pmatrix} z_2 & z_1 \\ 2g z_1^3 & 2z_2 \end{pmatrix}.$$

Using the change of variable, we now integrate

$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \int_0^1 \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\xi}(\boldsymbol{z})\mathrm{d}\xi$$

along the straight line

$$\boldsymbol{z} = \boldsymbol{Z}_{i-1} + (\boldsymbol{Z}_i - \boldsymbol{Z}_{i-1})\xi.$$

As $\boldsymbol{z}' = \boldsymbol{Z}_i - \boldsymbol{Z}_{i-1}$, we have

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) &= \int_0^1 \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{z}}(\boldsymbol{z}) \frac{\mathrm{d}\boldsymbol{z}}{\mathrm{d}\xi}\mathrm{d}\xi, \\ &= (\boldsymbol{Z}_i - \boldsymbol{Z}_{i-1}) \int_0^1 \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{z}}(\boldsymbol{z})\mathrm{d}\xi, \\ &= \begin{pmatrix} \bar{Z}_2 & \bar{Z}_1 \\ 2g\bar{Z}_1\bar{h} & 2\bar{Z}_2 \end{pmatrix} \cdot (\boldsymbol{Z}_i - \boldsymbol{Z}_{i-1}) \end{aligned}$$

where

$$\bar{Z}_k = \frac{1}{2}(Z_{k,i-1} + Z_{k,i}) \text{ and } \bar{h} = \frac{1}{2}(h_{i-1} + h_i).$$

Now we have to return to the original variables by linking $\boldsymbol{Z}$ and $\boldsymbol{Q}$. By integrating Eq. (5.16), we get

$$\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1} = \int_{i-1}^{i} \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{z}}\mathrm{d}\boldsymbol{z} = \int_0^1 \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{z}}\boldsymbol{z}'\mathrm{d}\xi = \begin{pmatrix} 2\bar{Z}_1 & 0 \\ \bar{Z}_2 & \bar{Z}_1 \end{pmatrix} \cdot (\boldsymbol{Z}_i - \boldsymbol{Z}_{i-1}).$$

We eventually find:

$$\hat{\boldsymbol{A}}_{i-1/2} = \begin{pmatrix} \bar{Z}_2 & \bar{Z}_1 \\ 2g\bar{Z}_1\bar{h} & 2\bar{Z}_2 \end{pmatrix} \cdot \begin{pmatrix} 2\bar{Z}_1 & 0 \\ \bar{Z}_2 & \bar{Z}_1 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 \\ g\bar{h} - (\bar{Z}_2/Z_1)^2 & 2\bar{Z}_2/Z_1 \end{pmatrix},$$

and after returning to the original variables

$$\hat{\boldsymbol{A}}_{i-1/2} = \begin{pmatrix} 1 & 1 \\ -\hat{u} + g\bar{h} & 2\hat{u} \end{pmatrix} \text{ where } \hat{u} = \frac{\sqrt{h_{i-1}}u_{i-1} + \sqrt{h_i}u_i}{\sqrt{h_{i-1}} + \sqrt{h_i}}. \tag{5.18}$$

The Roe matrix is thus the Jacobian matrix $\boldsymbol{f}'(\boldsymbol{q})$ evaluated at the intermediate state $\boldsymbol{q}) = (\bar{h},\ \bar{h}\hat{u})$. It has the eigenvalues

$$\lambda_1 = \hat{u} - \sqrt{g\bar{h}} \text{ and } \lambda_2 = \hat{u} + \sqrt{g\bar{h}},$$

associated with the right eigenvectors:

$$\boldsymbol{w}_1 = \begin{pmatrix} 1 \\ \hat{u} - \sqrt{g\bar{h}} \end{pmatrix} \text{ and } \boldsymbol{w}_2 = \begin{pmatrix} 1 \\ \hat{u} + \sqrt{g\bar{h}} \end{pmatrix}. \tag{5.19}$$

We can decompose the initial jump $\boldsymbol{Q}_i - \boldsymbol{Q}_{i-1}$ as

$$\Delta \boldsymbol{Q} = \boldsymbol{Q}_i - \boldsymbol{Q}_{i-1} = \boldsymbol{R} \cdot \boldsymbol{\alpha}, \tag{5.20}$$

where $\boldsymbol{R} = [\boldsymbol{w}_1,\ \boldsymbol{w}_2]$ is the right-eigenvector matrix. We then deduce the $\boldsymbol{\alpha}$ coefficients by inverting the matrix $\boldsymbol{R}$

$$\boldsymbol{\alpha} = \boldsymbol{R}^{-1} \cdot \Delta \boldsymbol{Q} = \frac{1}{2\hat{c}} \begin{pmatrix} (\hat{u} + \hat{c})\Delta Q_1 - \Delta Q_2 \\ (-\hat{u} + \hat{c})\Delta Q_1 + \Delta Q_2 \end{pmatrix} \tag{5.21}$$

where $\hat{c} = \sqrt{g\bar{h}}$ and $\Delta \boldsymbol{Q} = (\Delta Q_1,\ \Delta Q_2)$.

5.2.2 Wave form

To summarize the results, we need the following equations to write the Roe solver's algorithm:

- The velocities associated with the intermediate state

$$\hat{u} = \frac{q_{i-1}/\sqrt{h_{i-1}} + q_i/\sqrt{h_i}}{\sqrt{h_{i-1}} + \sqrt{h_i}} \text{ and } \bar{c} = \sqrt{\frac{1}{2}(\sqrt{h_{i-1}} + \sqrt{h_i})}. \tag{5.22}$$

- The waves $\boldsymbol{W}_k$:

$$\boldsymbol{W}_k = \alpha_k \boldsymbol{w}_k,\ k = 1,\ 2 \tag{5.23}$$

  where α_k are the components of the $\boldsymbol{\alpha}$ vector given by Eq. (6.11) and $\boldsymbol{w}_k$ are the right eigenvectors of the Roe matrix given by Eq. (5.19).

- the characteristic speeds

$$s_1 = \hat{u} - \bar{c} \text{ and } s_2 = \hat{u} + \bar{c}. \tag{5.24}$$

- The fluctuations are

$$\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2} = \sum_{k=1}^{2} \min(\lambda_{i-1/2}^k, 0) \boldsymbol{W}_{k,i-1/2},$$

$$\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2} = \sum_{k=1}^{2} \max(\lambda_{i-1/2}^k, 0) \boldsymbol{W}_{k,i-1/2},$$

  which gives in the present context:

  - if $s_k > 0$, then `amdq(m,i) = s*wave`.
  - if $s_k < 0$, then `apdq(m,i) = s*wave`.

5.2.3 Implementation

Here is how the Roe solver is implemented in Clawpack (with the entropy fix to compute transonic wave).

```
1 subroutine rp1(maxmx,num_eqn,num_waves,num_aux,num_ghost,num_cells, &
2                ql,qr,auxl,auxr,wave,s,amdq,apdq)
3
4 ! waves: 2
5 ! equations: 2
6
7 ! Conserved quantities:
```

```
!       1 depth
!       2 momentum

! See http://www.clawpack.org/riemann.html for a detailed explanation
! of the Riemann solver API.

    implicit none

    integer, intent(in) :: maxmx, num_eqn, num_waves, num_aux, num_ghost, &
                           num_cells
    real(kind=8), intent(in), dimension(num_eqn,1-num_ghost:maxmx+num_ghost
    ) :: ql, qr
    real(kind=8), intent(in), dimension(num_aux,1-num_ghost:maxmx+num_ghost
    ) :: auxl, auxr
    real(kind=8), intent(out) :: s(num_waves, 1-num_ghost:maxmx+num_ghost)
    real(kind=8), intent(out) :: wave(num_eqn, num_waves, 1-num_ghost:maxmx
    +num_ghost)
    real(kind=8), intent(out), dimension(num_eqn,1-num_ghost:maxmx+
    num_ghost) :: amdq,apdq

    ! local variables:
    real(kind=8) :: a1,a2,ubar,cbar,s0,s1,s2,s3,hr1,uhr1,hl2,uhl2,sfract,df
    real(kind=8) :: delta(2)
    integer :: i,m,mw

    logical :: efix

    data efix /.true./    !# Use entropy fix for transonic rarefactions

    ! Gravity constant set in setprob.f or the shallow1D.py file
    real(kind=8) :: grav
    common /cparam/ grav

    ! Main loop of the Riemann solver.
    do 30 i=2-num_ghost,num_cells+num_ghost


        ! compute  Roe-averaged quantities:
        ubar = (qr(2,i-1)/dsqrt(qr(1,i-1)) + ql(2,i)/dsqrt(ql(1,i)))/ &
               ( dsqrt(qr(1,i-1)) + dsqrt(ql(1,i)) )
        cbar=dsqrt(0.5d0*grav*(qr(1,i-1) + ql(1,i)))

        ! delta(1)=h(i)-h(i-1) and  delta(2)=hu(i)-hu(i-1)
        delta(1) = ql(1,i) - qr(1,i-1)
        delta(2) = ql(2,i) - qr(2,i-1)

        ! Compute coeffs in the evector expansion of delta(1),delta(2)
        a1 = 0.5d0*(-delta(2) + (ubar + cbar) * delta(1))/cbar
        a2 = 0.5d0*( delta(2) - (ubar - cbar) * delta(1))/cbar

        ! Finally, compute the waves.
        wave(1,1,i) = a1
        wave(2,1,i) = a1*(ubar - cbar)
        s(1,i) = ubar - cbar

        wave(1,2,i) = a2
        wave(2,2,i) = a2*(ubar + cbar)
        s(2,i) = ubar + cbar
```

```
62
63      30 enddo
64
65      ! Compute fluctuations amdq and apdq
66      ! ----------------------------------
67
68      if (efix) go to 110
69
70      ! No entropy fix
71      ! ---------------------------------------------
72      ! amdq = SUM s*wave   over left-going waves
73      ! apdq = SUM s*wave   over right-going waves
74
75      do m=1,2
76          do i=2-num_ghost, num_cells+num_ghost
77              amdq(m,i) = 0.d0
78              apdq(m,i) = 0.d0
79              do mw=1,num_waves
80                  if (s(mw,i) < 0.d0) then
81                      amdq(m,i) = amdq(m,i) + s(mw,i)*wave(m,mw,i)
82                  else
83                      apdq(m,i) = apdq(m,i) + s(mw,i)*wave(m,mw,i)
84                  endif
85              enddo
86          enddo
87      enddo
88
89      ! with no entropy fix we are done...
90      return
91
92
93      ! ---------------------------------------------
94
95      110 continue
96
97      ! With entropy fix
98      ! ------------------
99
100     ! compute flux differences amdq and apdq.
101     ! First compute amdq as sum of s*wave for left going waves.
102     ! Incorporate entropy fix by adding a modified fraction of wave
103     ! if s should change sign.
104
105     do 200 i=2-num_ghost,num_cells+num_ghost
106
107         ! -------------------------------------------------------
108         ! check 1-wave:
109         ! ---------------
110
111         ! u-c in left state (cell i-1)
112         s0 = qr(2,i-1)/qr(1,i-1) - dsqrt(grav*qr(1,i-1))
113
114         ! check for fully supersonic case:
115         if (s0 >= 0.d0 .and. s(1,i) > 0.d0)  then
116             ! everything is right-going
117             do m=1,2
118                 amdq(m,i) = 0.d0
119                 enddo
```

```
            go to 200
        endif

        ! u-c to right of 1-wave
        hr1  = qr(1,i-1) + wave(1,1,i)
        uhr1 = qr(2,i-1) + wave(2,1,i)
        s1 =  uhr1/hr1 - dsqrt(grav*hr1)

        if (s0 < 0.d0 .and. s1 > 0.d0) then
            ! transonic rarefaction in the 1-wave
            sfract = s0 * (s1-s(1,i)) / (s1-s0)
        else if (s(1,i) < 0.d0) then
            ! 1-wave is leftgoing
            sfract = s(1,i)
        else
            ! 1-wave is rightgoing
            sfract = 0.d0   !# this shouldn't happen since s0 < 0
        endif

        do m=1,2
            amdq(m,i) = sfract*wave(m,1,i)
            enddo

        ! -----------------------------------------------------
        ! check 2-wave:
        ! ---------------
        ! u+c in right state  (cell i)
        s3 = ql(2,i)/ql(1,i) + dsqrt(grav*ql(1,i))

        ! u+c to left of 2-wave
        hl2  = ql(1,i) - wave(1,2,i)
        uhl2 = ql(2,i) - wave(2,2,i)
        s2 = uhl2/hl2 + dsqrt(grav*hl2)

        if (s2 < 0.d0 .and. s3 > 0.d0) then
            ! transonic rarefaction in the 2-wave
            sfract = s2 * (s3-s(2,i)) / (s3-s2)
        else if (s(2,i) < 0.d0) then
            ! 2-wave is leftgoing
            sfract = s(2,i)
        else
            ! 2-wave is rightgoing
            go to 200
        endif

        do m=1,2
            amdq(m,i) = amdq(m,i) + sfract*wave(m,2,i)
            enddo

    200 enddo


    ! compute the rightgoing flux differences:
    ! df = SUM s*wave   is the total flux difference and apdq = df - amdq

    do m=1,2
        do i = 2-num_ghost, num_cells+num_ghost
            df = 0.d0
```

```
            do mw=1,num_waves
                df = df + s(mw,i)*wave(m,mw,i)
                enddo
            apdq(m,i) = df - amdq(m,i)
            enddo
        enddo

    return

    end subroutine rp1
```

Here is how the Roe solver is implemented in Pyclaw

```
def shallow_roe_1D(q_l, q_r, aux_l, aux_r, problem_data):
    r"""
    Roe shallow water solver in 1d::
    """
    # Array shapes
    num_rp = q_l.shape[1]

    # Output arrays
    wave = np.empty( (num_eqn, num_waves, num_rp) )
    s = np.zeros( (num_waves, num_rp) )
    amdq = np.zeros( (num_eqn, num_rp) )
    apdq = np.zeros( (num_eqn, num_rp) )

    # Compute roe-averaged quantities
    ubar = ( (q_l[1,:]/np.sqrt(q_l[0,:]) + q_r[1,:]/np.sqrt(q_r[0,:])) /
             (np.sqrt(q_l[0,:]) + np.sqrt(q_r[0,:])) )
    cbar = np.sqrt(0.5 * problem_data['grav'] * (q_l[0,:] + q_r[0,:]))

    # Compute Flux structure
    delta = q_r - q_l
    a1 = 0.5 * (-delta[1,:] + (ubar + cbar) * delta[0,:]) / cbar
    a2 = 0.5 * ( delta[1,:] - (ubar - cbar) * delta[0,:]) / cbar

    # Compute each family of waves
    wave[0,0,:] = a1
    wave[1,0,:] = a1 * (ubar - cbar)
    s[0,:] = ubar - cbar

    wave[0,1,:] = a2
    wave[1,1,:] = a2 * (ubar + cbar)
    s[1,:] = ubar + cbar

    s_index = np.zeros((2,num_rp))
    for m in range(num_eqn):
        for mw in range(num_waves):
            s_index[0,:] = s[mw,:]
            amdq[m,:] += np.min(s_index,axis=0) * wave[m,mw,:]
            apdq[m,:] += np.max(s_index,axis=0) * wave[m,mw,:]

    return wave, s, amdq, apdq
```

5.2.4 Sonic entropy fix

When the solution to the Riemann problem is a transonic wave, the Roe approximate solution may be incorrect. In that case, there is an intermediate state $\boldsymbol{Q}_*$ between the left and right states $\boldsymbol{Q}_{i-1}$ and $\boldsymbol{Q}_i$, and the associated speeds are

$$\lambda_{1,i-1} = u_{i-1} - \sqrt{gh_{i-1}},\ \lambda_{1,*} = u_* - \sqrt{g\hat{h}_*}$$

and

$$\lambda_{2,*} = u_* + \sqrt{g\hat{h}_*},\ \lambda_{2,i} = u_i + \sqrt{g\hat{h}_i}.$$

When $\lambda_{1,i-1} < 0 < \lambda_{1,*}$ (resp. $\lambda_{2,*} < 0 < \lambda_{2,i}$), we can suspect that the 1-wave (resp. the 2-wave) is a transonic rarefaction wave. By using the analytical expression for a centred rarefaction wave (LeVeque, 2002, see, § 13.8.5), we can deduce the interface values

$$h_{i-1/2} = \frac{1}{9g}\left(u_{i-1} + 2\sqrt{gh_{i-1}}\right)^2, \tag{5.25}$$

$$u_{i-1/2} = u_{i-1} + 2\left(\sqrt{gh_{i-1}} - \sqrt{gh_{i-1/2}}\right) \tag{5.26}$$

The flux fluctuations are computed using Eqs. (2.32) and (2.33).

5.3 HLLE solver

5.3.1 Principle

The HLL method is a two-wave solver that considers that the solution to the Riemann problem consists of two shock waves separating the intermediate state $\boldsymbol{Q}_*$ $\boldsymbol{Q}_i$ from the left and right initial states $\boldsymbol{Q}_{i-1}$ and $\boldsymbol{Q}_i$. In § 2.6, we have seen that this intermediate state and the associated flux can be determined by solving the Rankine-Hugoniot equations for the discontinuities across the two shock waves. Here we provide another proof based on volume integrals.

Integrating the shallow water equations (5.5) over the domain $[x_1,\ x_2] \times [0, \Delta t]$ in the $x-t$ plane (see Fig. 2.3) gives

$$\int_{x_1}^{x_2} \boldsymbol{q}(x,\ \Delta t)\mathrm{d}x = \int_{x_1}^{x_2} \boldsymbol{q}(x,\ 0)\mathrm{d}x + \int_0^{\Delta t} \boldsymbol{f}(\boldsymbol{q}(x_1,\ t))\mathrm{d}t - \int_0^{\Delta t} \boldsymbol{f}(\boldsymbol{q}(x_2,\ t))\mathrm{d}t,$$

where $x_1 = s_1\Delta t$ and $x_2 = s_2\Delta t$. Since $\boldsymbol{q}(x,\ 0)$ is fixed by the initial conditions, we deduce

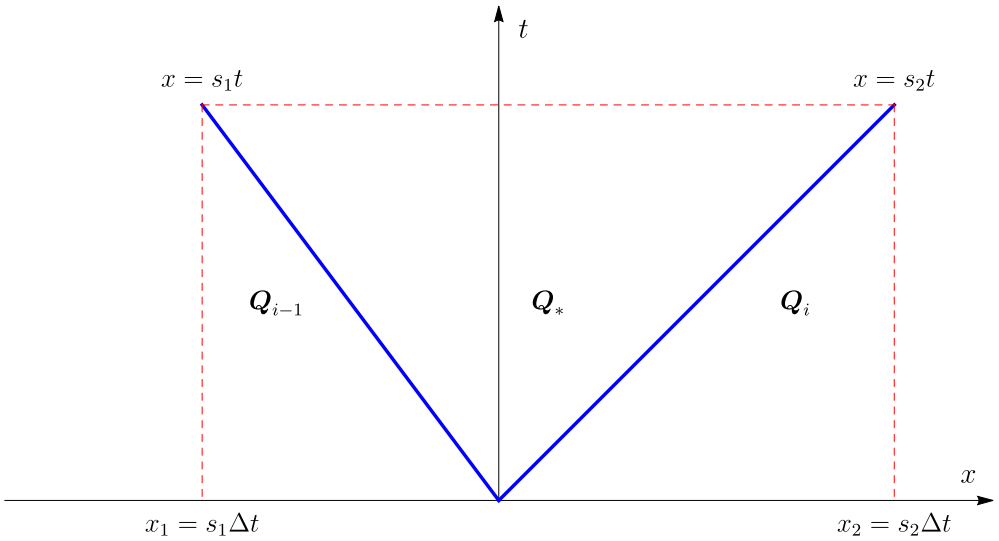
$$\boldsymbol{Q}_* = \frac{1}{x_2 - x_1}\int_{x_1}^{x_2} \boldsymbol{q}(x,\ \Delta t)\mathrm{d}x = \frac{\boldsymbol{Q}_i x_2 - \boldsymbol{Q}_{i-1}x_1}{x_2 - x_1} - \frac{\Delta t}{x_2 - x_1}(\boldsymbol{F}_i - \boldsymbol{F}_{i-1}),$$

where

$$\boldsymbol{F}_{i-1} = \frac{1}{\Delta t}\int_0^{\Delta t} \boldsymbol{f}(\boldsymbol{q}(x_1,\ t))\mathrm{d}t \text{ and } \boldsymbol{F}_i = \frac{1}{\Delta t}\int_0^{\Delta t} \boldsymbol{f}(\boldsymbol{Q}(x_2,\ t))\mathrm{d}t.$$

We find that the intermediate state is:

$$\boldsymbol{Q}_* = \frac{\boldsymbol{Q}_i s_2 - \boldsymbol{Q}_{i-1}s_1}{s_2 - s_1} - \frac{\boldsymbol{F}_i - \boldsymbol{F}_{i-1}}{s_2 - s_1}. \tag{5.27}$$



In § 2.3.3, we derived the general expression (2.20) for computing the interface flux from the left and right flux

$$\boldsymbol{F}_{i-1/2} = \boldsymbol{f}(\boldsymbol{Q}_{i-1}) + \frac{\delta x}{\delta t}\boldsymbol{Q}_{i-1} - \frac{1}{\delta t}\int_{-\delta x}^{0} \boldsymbol{Q}(x,\ \delta t)\mathrm{d}x.$$

which gives us when we take $\delta x = -x_2 = -s_2 \Delta t$:

$$\boldsymbol{F}_* = \frac{s_2 \boldsymbol{F}_{i-1} - s_1 \boldsymbol{F}_i}{s_2 - s_1} - s_2 s_2 \frac{\boldsymbol{Q}_{i-1} - \boldsymbol{Q}_i}{s_2 - s_1}. \tag{5.28}$$

This expression of the flux holds when the two shock waves fan out on either side of $x = 0$. In that case, the interface flux is defined as $\boldsymbol{F}_{i-1/2} = \boldsymbol{F}_*$. If both shock waves go to the right (i.e., if $s_1 > 0$) then $\boldsymbol{F}_{i-1/2} = \boldsymbol{F}_{i-1}$. In the opposite case, then $\boldsymbol{F}_{i-1/2} = \boldsymbol{F}_i$:

$$\boldsymbol{F}_{i-1/2} = \begin{cases} \boldsymbol{F}_{i-1} \text{ if } s_1 > 0, \\ \boldsymbol{F}_* \text{ if } s_1 \geq 0 \geq s_2, \\ \boldsymbol{F}_i \text{ if } s_2 < 0, \end{cases} \tag{5.29}$$

The last problem to be settled is the determination of the shock speed s_1 and s_2. We use the suggest of Einfeldt, which explains why the solver is called HLLE. Let us first consider the 1-wave. If this wave is a rarefaction wave, its speeds ranges from $\lambda_1(\boldsymbol{Q}_{i-1})$ to $\lambda_1(\boldsymbol{Q}_i) = u_{i-1} - \sqrt{gh_{i-1}}$; we select the minimum value $\lambda_1(\boldsymbol{Q}_{i-1})$. If it is a shock, its speed can be estimated using the Roe matrix (5.18): $s_1 = \hat{u} - \hat{c}$. As we do not know whether the 1-wave is a shock or rarefaction wave, we take the lower bound:

$$s_1 = \min(u_{i-1} - \sqrt{gh_{i-1}},\ \hat{u} - \hat{c}). \tag{5.30}$$

The same applies for the 2-wave. We define s_2 as the upper bound

$$s_2 = \max(u_i + \sqrt{gh_i},\ \hat{u} + \hat{c}). \tag{5.31}$$

In short, we compute the Roe averages, and deduce the shock speeds (5.30) and (5.31). The intermediate state is given by Eq (5.27). The waves are

$$\boldsymbol{W}_1 = \boldsymbol{Q}_* - \boldsymbol{Q}_{i-1} \text{ and } \boldsymbol{W}_2 = \boldsymbol{Q}_i - \boldsymbol{Q}_*. \tag{5.32}$$

The fluctuations are then

$$\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i-1/2} = s_1 \boldsymbol{W}_1$$
$$\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2} = s_1 \boldsymbol{W}_2.$$

5.3.2 Implementation in Pyclaw

```
def shallow_hll_1D(q_l,q_r,aux_l,aux_r,problem_data):
    r"""
    HLL shallow water solver ::


        W_1 = Q_hat - Q_l    s_1 = min(u_l-c_l,u_l+c_l,lambda_roe_1,
    lambda_roe_2)
        W_2 = Q_r - Q_hat    s_2 = max(u_r-c_r,u_r+c_r,lambda_roe_1,
    lambda_roe_2)

        Q_hat = ( f(q_r) - f(q_l) - s_2 * q_r + s_1 * q_l ) / (s_1 - s_2)

    *problem_data* should contain:
     - *g* - (float) Gravitational constant

    :Version: 1.0 (2009-02-05)
    """
    # Array shapes
    num_rp = q_l.shape[1]
    num_eqn = 2
    num_waves = 2

    # Output arrays
    wave = np.empty( (num_eqn, num_waves, num_rp) )
    s = np.empty( (num_waves, num_rp) )
    amdq = np.zeros( (num_eqn, num_rp) )
    apdq = np.zeros( (num_eqn, num_rp) )

    # Compute Roe and right and left speeds
    ubar = ( (q_l[1,:]/np.sqrt(q_l[0,:]) + q_r[1,:]/np.sqrt(q_r[0,:])) /
        (np.sqrt(q_l[0,:]) + np.sqrt(q_r[0,:])) )
    cbar = np.sqrt(0.5 * problem_data['grav'] * (q_l[0,:] + q_r[0,:]))
    u_r = q_r[1,:] / q_r[0,:]
    c_r = np.sqrt(problem_data['grav'] * q_r[0,:])
    u_l = q_l[1,:] / q_l[0,:]
    c_l = np.sqrt(problem_data['grav'] * q_l[0,:])

    # Compute Einfeldt speeds
    s_index = np.empty((4,num_rp))
    s_index[0,:] = ubar+cbar
    s_index[1,:] = ubar-cbar
    s_index[2,:] = u_l + c_l
    s_index[3,:] = u_l - c_l
    s[0,:] = np.min(s_index,axis=0)
    s_index[2,:] = u_r + c_r
    s_index[3,:] = u_r - c_r
    s[1,:] = np.max(s_index,axis=0)

    # Compute middle state
    q_hat = np.empty((2,num_rp))
    q_hat[0,:] = ((q_r[1,:] - q_l[1,:] - s[1,:] * q_r[0,:]
                            + s[0,:] * q_l[0,:]) / (s[0,:] - s[1,:]))
    q_hat[1,:] = ((q_r[1,:]**2/q_r[0,:] + 0.5 * problem_data['grav'] * q_r
    [0,:]**2
                - (q_l[1,:]**2/q_l[0,:] + 0.5 * problem_data['grav'] * q_l
    [0,:]**2)
```

```
                - s[1,:] * q_r[1,:] + s[0,:] * q_l[1,:]) / (s[0,:] - s
    [1,:]))

    # Compute each family of waves
    wave[:,0,:] = q_hat - q_l
    wave[:,1,:] = q_r - q_hat

    # Compute variations
    s_index = np.zeros((2,num_rp))
    for m in range(num_eqn):
        for mw in range(num_waves):
            s_index[0,:] = s[mw,:]
            amdq[m,:] += np.min(s_index,axis=0) * wave[m,mw,:]
            apdq[m,:] += np.max(s_index,axis=0) * wave[m,mw,:]

    return wave, s, amdq, apdq
```

5.4 F-wave formulation

5.4.1 Principle

The f-wave method consists of decomposing the flux jump into f-waves

$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \sum_{k=1}^{m_w} \boldsymbol{Z}_{k,i-1/2},$$

where the f-wave $\boldsymbol{Z}_{k,i-1/2}$ can be related to the right eigenvector $\hat{\boldsymbol{w}}_{k,i-1/2}$ of the Roe matrix:

$$\boldsymbol{Z}_{k,i-1/2} = \beta_{k,i-1/2}\hat{\boldsymbol{w}}_{k,i-1/2}$$

where the coefficient $\beta_{k,i-1/2}$ is the linear solution (see § 2.7):

$$\boldsymbol{\beta}_{i-1/2} = \boldsymbol{L} \cdot (\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1})).$$

with $\boldsymbol{L} = \boldsymbol{R}^{-1}$. We find that

$$\boldsymbol{\beta}_{i-1/2} = \frac{1}{2\hat{c}} \begin{pmatrix} \Phi_l - \phi_r + (\hat{u} + \hat{c})(q_r - q_l) \\ \Phi_r - \phi_l - (\hat{u} - \hat{c})(q_r - q_l) \end{pmatrix}, \tag{5.33}$$

where Φ is the shorthand notation: $\Phi = u^2 h + gh^2/2$. The f-waves are then

$$\boldsymbol{Z}_{1,i-1/2} = \boldsymbol{\beta}_{1,i-1/2}\boldsymbol{w}_1 = \frac{\Phi_l - \Phi_r + (\hat{u} + \hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u} - \hat{c} \end{pmatrix} \tag{5.34}$$

and

$$\boldsymbol{Z}_{2,i-1/2} = \boldsymbol{\beta}_{2,i-1/2}\boldsymbol{w}_2 = \frac{\Phi_r - \Phi_l - (\hat{u} - \hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u} + \hat{c} \end{pmatrix}. \tag{5.35}$$

5.4.2 Implementation in Pyclaw

```
def shallow_water_fwave_1d(q_l, q_r, aux_l, aux_r, problem_data):
    r"""Shallow water Riemann solver using fwaves

    *problem_data* should contain:
     - *grav* - (float) Gravitational constant
     - *dry_tolerance* - (float) Set velocities to zero if h is below this
       tolerance.
    """

    g = problem_data['grav']
    dry_tolerance = problem_data['dry_tolerance']


    num_rp = q_l.shape[1]
    num_eqn = 2
    num_waves = 2

    # initializing f-waves
    fwave = np.empty( (num_eqn, num_waves, num_rp) )
    # right eigenvectors
    r1 = np.empty( (num_waves, num_rp) )
    r2 = np.empty( (num_waves, num_rp) )
    # initializing fluctuations and shock speeds
    amdq = np.zeros( (num_eqn, num_rp) )
    apdq = np.zeros( (num_eqn, num_rp) )
    s = np.empty( (num_waves, num_rp) )


    # Extract state
    hl = q_l[0, :]
    ql = q_l[1, :]
    ul = np.where(hl > dry_tolerance, ql/hl , 0.0)
    hr = q_r[0, :]
    qr = q_r[1, :]
    ur = np.where(hr > dry_tolerance, qr/hr, 0.0)

    phi_l = hl * ul**2 + 0.5 * g * hl**2
    phi_r = hr * ur**2 + 0.5 * g * hr**2
    h_bar = 0.5 * (hr + hl)

    # Speeds
    u_hat = (np.sqrt(hl) * ul + np.sqrt(hr) * ur) / (np.sqrt(hl) + np.sqrt(
    hr) )
    c_hat = np.sqrt(g * h_bar)
    lambda1 = u_hat - c_hat
    lambda2 = u_hat + c_hat


    beta1 = (phi_l - phi_r +lambda2*(qr-ql))/2/c_hat
    beta2 = (phi_r - phi_l -lambda1*(qr-ql))/2/c_hat



    r1[0, :] = 1.
    r1[1, :] = u_hat - c_hat
    r2[0, :] = 1.
    r2[1, :] = u_hat + c_hat

```

```
    s[0,:] = u_hat - c_hat
    s[1,:] = u_hat + c_hat

    # 1st f-wave
    fwave[0,0,:] = beta1*r1[0,:]
    fwave[1,0,:] = beta1*r1[1,:]
    # 2nd f-wave
    fwave[0,1,:] = beta2*r2[0,:]
    fwave[1,1,:] = beta2*r2[1,:]

    for m in range(num_eqn):
        for mw in range(num_waves):
            amdq[m, :] += (s[mw, :] < 0.0) * fwave[m, mw, :]
            apdq[m, :] += (s[mw, :] > 0.0) * fwave[m, mw, :]

            amdq[m, :] += (s[mw, :] == 0.0) * fwave[m, mw, :] * 0.5
            apdq[m, :] += (s[mw, :] == 0.0) * fwave[m, mw, :] * 0.5

    return fwave, s, amdq, apdq
```

5.5 Example: dam break

We consider a dam break problem with the following initial conditions: $h_l = 10$ m et $u_l = 0$ for $x \leq 0$, and $h_l = 0.5$ m et $u_l = 0$ for $x > 0$. We compare the three solvers: Roe (with or without the entropy fix), the HLLE solver, and the f-wave formulation. Figures 5.3 and 5.4 show the comparison.

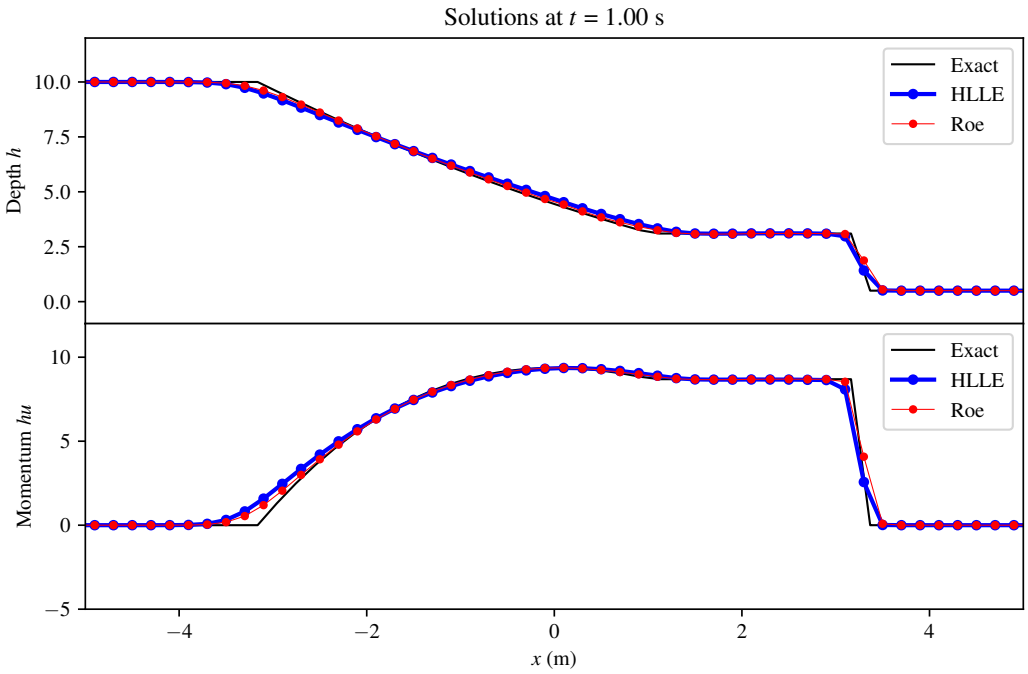


Figure 5.3 Comparison between the analytical solution, the Roe (with entropy fix) and HLLE solution.

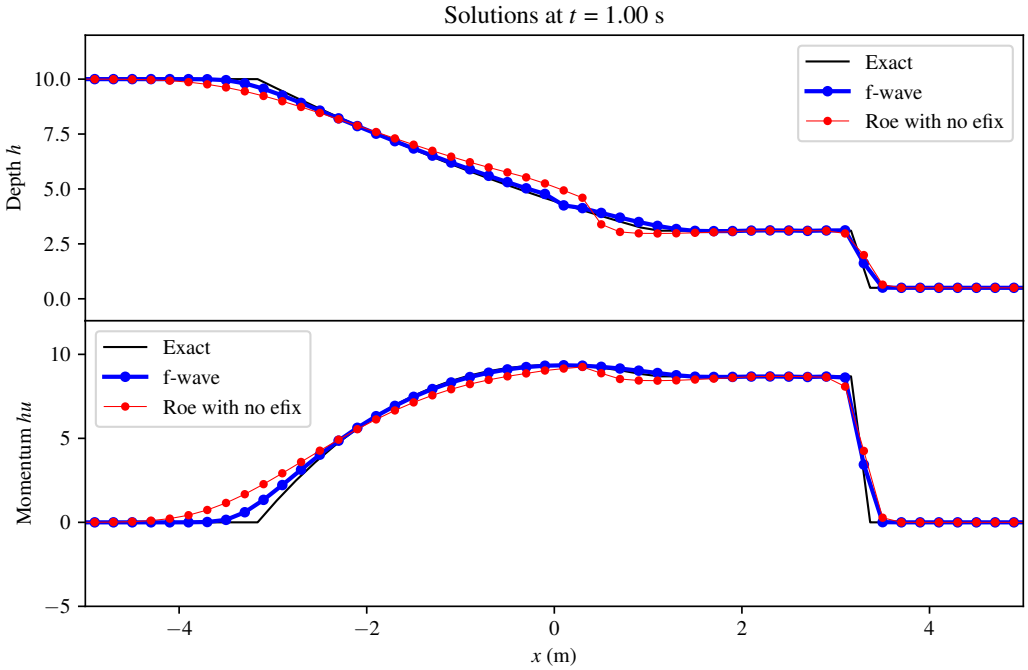
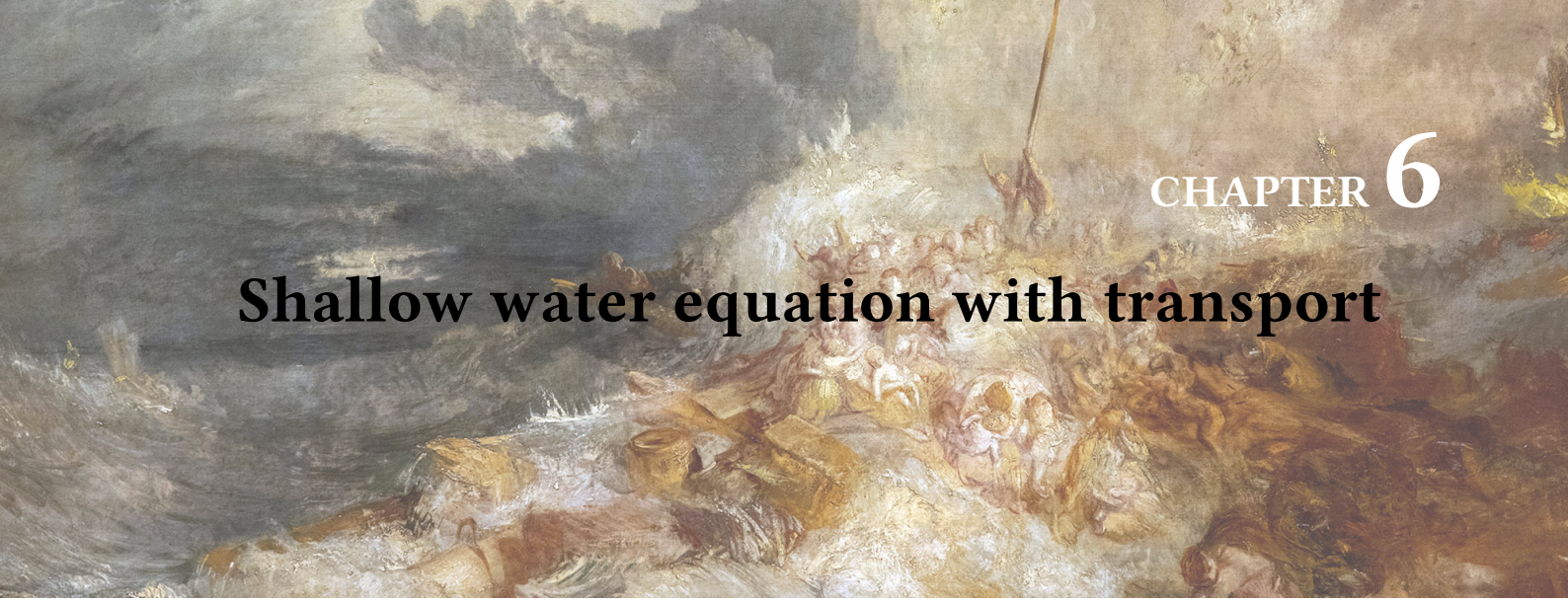


Figure 5.4 Comparison between the analytical solution, the Roe (with no entropy fix) and f-wave solution.



CHAPTER 6

Shallow water equation with transport

6.1 Theory

Let us consider the shallow water equations seen in Chap. 5, supplemented with an equation representing the advection of a scalar quantity $\phi(x,\ t)$ (for instance, the concentration of a pollutant that does not interplay with the water flow):

$$\frac{\partial h}{\partial t} + \frac{\partial q}{\partial x} = 0, \tag{6.1}$$

$$\frac{\partial q}{\partial t} + \frac{\partial hu^2}{\partial x} + gh\frac{\partial h}{\partial x} = 0, \tag{6.2}$$

$$\frac{\partial h\phi}{\partial t} + \frac{\partial q\phi}{\partial x} = 0, \tag{6.3}$$

where h denotes the flow depth, $q = hu$ is the flow rate, and u the depth-averaged velocity. where g is gravitational acceleration, and the unknowns are q, h and ϕ. In a matrix form, Eqs. (6.1)-(6.3) takes the form:

$$\frac{\partial}{\partial t}\boldsymbol{Q} + \frac{\partial}{\partial x}\boldsymbol{f}(\boldsymbol{Q}) = 0, \tag{6.4}$$

where

$$\boldsymbol{f} = \begin{pmatrix} q \\ q^2/h + gh^2/2 \\ q\phi \end{pmatrix} \text{ and } \boldsymbol{Q} = \begin{pmatrix} h \\ q \\ h\phi \end{pmatrix}. \tag{6.5}$$

The Jacobian is

$$\boldsymbol{f}' = \begin{pmatrix} 0 & 1 & 0 \\ -u^2 + gh & 2u & 0 \\ -u\phi & \phi & u \end{pmatrix}, \tag{6.6}$$

whose eigenvalues are

$$\lambda_1 = u - \sqrt{gh}, \lambda_2 = u \text{ and } \lambda_3 = u + \sqrt{gh}, \tag{6.7}$$

associated with the right eigenvectors:

$$\boldsymbol{w}_1 = \begin{pmatrix} 1 \\ u - \sqrt{gh} \\ \phi \end{pmatrix}, \ \boldsymbol{w}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ and } \boldsymbol{w}_3 = \begin{pmatrix} 1 \\ u + \sqrt{gh} \\ \phi \end{pmatrix}. \tag{6.8}$$

The scalar quantity ϕ is decoupled from the water flow, and its speed depends only on the water flow velocity: $\lambda_2 = u$. The associated field is said to be *linearly degenerate* because $\nabla\lambda_2 \cdot \boldsymbol{w}_2 = 0$. This gives rise to *contact discontinuities*: when ϕ experiences a shock, there is a discontinuity in u, and thus the characteristic speeds are equal on either side of the shock waves (the characteristic curves are parallel to the shock cuves). The condition $\nabla\lambda_2 \cdot \boldsymbol{w}_2 = 0$ means that the eigenvalue is unchanged when we move along the integral curve $\boldsymbol{w}_2(\zeta)$.

6.2 Roe solver

6.2.1 Derivation

The Roe solver is close to the version derived in Chap. 5. The only difference lies in the adding of a third wave. We need the following equations to write the Roe solver's algorithm:

- The velocities associated with the intermediate state

$$\hat{u} = \frac{q_{i-1}/\sqrt{h_{i-1}} + q_i/\sqrt{h_i}}{\sqrt{h_{i-1}} + \sqrt{h_i}} \text{ and } \bar{c} = \sqrt{\frac{1}{2}(h_{i-1} + h_i)}. \tag{6.9}$$

- The waves $\boldsymbol{W}_k$:

$$\boldsymbol{W}_k = \alpha_k \boldsymbol{w}_k, \ k = 1, \ 3 \tag{6.10}$$

where α_k are the components of the $\boldsymbol{\alpha}$ vector obtained by inverting the matrix $\boldsymbol{R}$

$$\boldsymbol{\alpha} = \boldsymbol{R}^{-1} \cdot \Delta \boldsymbol{Q} = \frac{1}{2\hat{c}} \begin{pmatrix} (\hat{u} + \hat{c})\Delta Q_1 - \Delta Q_2 \\ \Delta Q_3 - \phi \Delta Q_1 \\ (-\hat{u} + \hat{c})\Delta Q_1 + \Delta Q_2 \end{pmatrix} \tag{6.11}$$

where $\hat{c} = \sqrt{g\bar{h}}$ and $\Delta \boldsymbol{Q} = (\Delta Q_1, \ \Delta Q_2, , \ \Delta Q_3)$. For the second wave, we impose that there is no jump ΔQ_1 associated with the contact discontinuity, and thus we impose

$$\alpha_2 = \Delta Q_3.$$

- the characteristic speeds

$$s_1 = \hat{u} - \bar{c}, \ s_2 = \hat{u} \text{ and } s_3 = \hat{u} + \bar{c}. \tag{6.12}$$

- The fluctuations are

$$\boldsymbol{A}^+ \cdot \Delta \boldsymbol{Q}_{i-1/2} = \sum_{k=1}^{3} \min(\lambda^k_{i-1/2}, 0) \boldsymbol{W}_{k,i-1/2},$$

$$\boldsymbol{A}^- \cdot \Delta \boldsymbol{Q}_{i+1/2} = \sum_{k=1}^{3} \max(\lambda^k_{i-1/2}, 0) \boldsymbol{W}_{k,i-1/2},$$

which gives in the present context:

  - if $s_k > 0$, then `amdq(m,i) = s*wave`.
  - if $s_k < 0$, then `apdq(m,i) = s*wave`.

6.2.2 Implementation in Clawpack

```
subroutine rp1(maxmx,num_eqn,num_waves,num_aux,num_ghost,num_cells, &
               ql,qr,auxl,auxr,wave,s,amdq,apdq)

! Solve Riemann problems for the 1D shallow water equations
! with an additional passively advected tracer:
!    (h)_t + (u h)_x = 0
!    (uh)_t + ( uuh + .5*gh^2 )_x = 0
```

```
!    c_t + uc_x = 0
! using Roe's approximate Riemann solver with entropy fix for
! transonic rarefractions.

! waves: 3
! equations: 3

! Conserved quantities:
!       1 depth
!       2 momentum
!       3 tracer

! See http://www.clawpack.org/riemann.html for a detailed explanation
! of the Riemann solver API.

    implicit none

    integer, intent(in) :: maxmx, num_eqn, num_waves, num_aux, num_ghost, &
                           num_cells
    real(kind=8), intent(in), dimension(num_eqn,1-num_ghost:maxmx+num_ghost
    ) :: ql, qr
    real(kind=8), intent(in), dimension(num_aux,1-num_ghost:maxmx+num_ghost
    ) :: auxl, auxr
    real(kind=8), intent(out) :: s(num_waves, 1-num_ghost:maxmx+num_ghost)
    real(kind=8), intent(out) :: wave(num_eqn, num_waves, 1-num_ghost:maxmx
    +num_ghost)
    real(kind=8), intent(out), dimension(num_eqn,1-num_ghost:maxmx+
    num_ghost) :: amdq,apdq

    ! local variables:
    real(kind=8) :: a1,a2,ubar,cbar,s0,s1,s2,s3,hr1,uhr1,hl2,uhl2,sfract,df
    real(kind=8) :: delta(2)
    integer :: i,m,mw
    logical :: efix

    data efix /.true./    ! Use entropy fix for transonic rarefactions

    ! Gravity constant set in setprob.f or the shallow1D.py file
    real(kind=8) :: grav
    common /cparam/ grav

    ! Main loop of the Riemann solver.
    do 30 i=2-num_ghost,num_cells+num_ghost

        ! compute  Roe-averaged quantities:
        ubar = (qr(2,i-1)/dsqrt(qr(1,i-1)) + ql(2,i)/dsqrt(ql(1,i)))/ &
               ( dsqrt(qr(1,i-1)) + dsqrt(ql(1,i)) )
        cbar=dsqrt(0.5d0*grav*(qr(1,i-1) + ql(1,i)))

        ! delta(1)=h(i)-h(i-1) and  delta(2)=hu(i)-hu(i-1)
        delta(1) = ql(1,i) - qr(1,i-1)
        delta(2) = ql(2,i) - qr(2,i-1)

        ! Compute coeffs in the evector expansion of delta(1),delta(2)
        a1 = 0.5d0*(-delta(2) + (ubar + cbar) * delta(1))/cbar
        a2 = 0.5d0*( delta(2) - (ubar - cbar) * delta(1))/cbar

        ! Finally, compute the waves.
```

```
        wave(1,1,i) = a1
        wave(2,1,i) = a1*(ubar - cbar)
        wave(3,1,i) = 0.d0
        s(1,i) = ubar - cbar

        wave(1,2,i) = a2
        wave(2,2,i) = a2*(ubar + cbar)
        wave(3,2,i) = 0.d0
        s(2,i) = ubar + cbar

        wave(1,3,i) = 0.d0
        wave(2,3,i) = 0.d0
        wave(3,3,i) = ql(3,i) - qr(3,i-1)
        s(3,i) = ubar

    30 enddo

    ! Compute fluctuations amdq and apdq
    ! ----------------------------------

    if (efix) go to 110

    ! No entropy fix
    ! ----------------------------------------------
    ! amdq = SUM s*wave   over left-going waves
    ! apdq = SUM s*wave   over right-going waves

    do m=1,num_waves
        do i=2-num_ghost, num_cells+num_ghost
            amdq(m,i) = 0.d0
            apdq(m,i) = 0.d0
            do mw=1,num_waves
                if (s(mw,i) < 0.d0) then
                    amdq(m,i) = amdq(m,i) + s(mw,i)*wave(m,mw,i)
                else
                    apdq(m,i) = apdq(m,i) + s(mw,i)*wave(m,mw,i)
                endif
            enddo
        enddo
    enddo

    ! with no entropy fix we are done...
    return

    ! ----------------------------------------------
    110 continue

    ! compute the rightgoing flux differences:
    ! df = SUM s*wave   is the total flux difference and apdq = df - amdq

    do i = 2-num_ghost, num_cells+num_ghost
        do m=1,2
            df = 0.d0
            do mw=1,2
                df = df + s(mw,i)*wave(m,mw,i)
                enddo
            apdq(m,i) = df - amdq(m,i)
            enddo
```

```
120
121         ! tracer (which is in non-conservation form)
122         if (s(3,i) < 0) then
123             amdq(m,i) = amdq(m,i) + s(3,i)*wave(m,3,i)
124           else
125             apdq(m,i) = apdq(m,i) + s(3,i)*wave(m,3,i)
126           endif
127
128         enddo
129
130     return
131
132 end subroutine rp1
```

6.2.3 Implementation in Pyclaw

```
1  def shallow_roe_1D(q_l, q_r, aux_l, aux_r, problem_data):
2      r"""
3      Roe shallow water solver in 1d
4      """
5
6      # Array shapes
7      num_rp    = q_l.shape[1]
8      num_eqn   = 3
9      num_waves = 3
10
11     g = problem_data['grav']
12
13     # Output arrays
14     wave = np.empty( (num_eqn, num_waves, num_rp) )
15     s = np.zeros( (num_waves, num_rp) )
16     amdq = np.zeros( (num_eqn, num_rp) )
17     apdq = np.zeros( (num_eqn, num_rp) )
18
19     # Compute roe-averaged quantities
20     ubar = ( (q_l[1,:]/np.sqrt(q_l[0,:]) + q_r[1,:]/np.sqrt(q_r[0,:])) /
21              (np.sqrt(q_l[0,:]) + np.sqrt(q_r[0,:])) )
22     cbar = np.sqrt(0.5 * g * (q_l[0,:] + q_r[0,:]))
23
24     # Compute Flux structure
25     delta  = q_r - q_l
26     delta1 = q_r[0,:] - q_l[0,:]
27     delta2 = q_r[1,:] - q_l[1,:]
28     alpha1 = 0.5 * (-delta2 + (ubar + cbar) * delta1) / cbar
29     alpha2 = 0.5 * ( delta2 - (ubar - cbar) * delta1) / cbar
30
31     # Compute each family of waves
32     wave[0,0,:] = alpha1
33     wave[1,0,:] = alpha1 * (ubar - cbar)
34     wave[2,0,:] = 0.
35     s[0,:]      = ubar - cbar
36
37     wave[0,2,:] = alpha2
38     wave[1,2,:] = alpha2 * (ubar + cbar)
39     wave[2,2,:] = 0.
40     s[2,:]      = ubar + cbar
```

```
41
42      wave[0,1,:] = 0.
43      wave[1,1,:] = 0.
44      wave[2,1,:] = q_r[2,:] - q_l[2,:]
45      s[1,:]      = ubar
46
47      s_index = np.zeros((3,num_rp))
48      for m in range(num_eqn):
49              for mw in range(num_waves):
50                  s_index[0,:] = s[mw,:]
51                  amdq[m,:] += np.min(s_index,axis=0) * wave[m,mw,:]
52                  apdq[m,:] += np.max(s_index,axis=0) * wave[m,mw,:]
53
54
55      return wave, s, amdq, apdq
```

6.3 HLLC Solver

6.3.1 Principle

The HLLC solver is an extension of the HLL scheme proposed by Eleuterio Toro (Toro, 2001) to cope with the existence of a contact discontinuity. The HLL solver defines an intermediate state separating the left and right initial states. The HLLC introduces two distinct intermediate states split by the second characteristic $x = \lambda_2 t$ (see Fig. 6.1). The fluxes associated with the two intermediate states are defined using the Rankine-Hugoniot equation:

$$\boldsymbol{F}_{*,l} - \boldsymbol{F}_l = s_1(\boldsymbol{Q}_{*,l} - \boldsymbol{Q}_l), \tag{6.13}$$

$$\boldsymbol{F}_{*,r} - \boldsymbol{F}_r = s_3(\boldsymbol{Q}_{*,r} - \boldsymbol{Q}_r) \tag{6.14}$$

where $\boldsymbol{F}_{*,l} = \boldsymbol{f}(\boldsymbol{Q}_{*,l})$ and $\boldsymbol{F}_{*,r} = \boldsymbol{f}(\boldsymbol{Q}_{*,r})$.

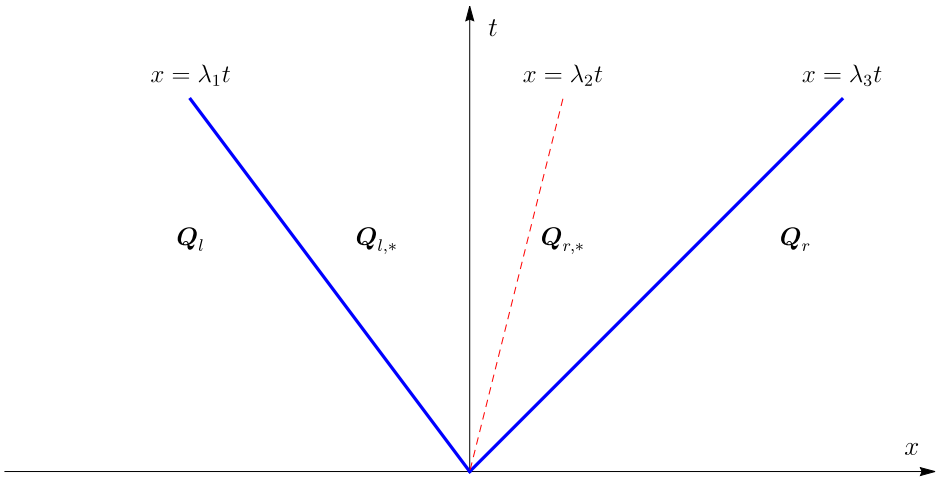


Figure 6.1 The three waves separating the left and right initial states.

In the absence of the advection equation (6.3), there would be only one intermediate state. As tracer advection does not interplay with water flow, we impose that the first two components (those associated with the water flow) of $\boldsymbol{F}_{*,r}$ and $\boldsymbol{F}_{*,l}$ are identical:

$$F_{*,l,1} = F_{*,r,1} = \frac{\lambda_3 F_{l,1} - \lambda_1 F_{r,1}}{\lambda_3 - \lambda_1} - \lambda_1 \lambda_3 \frac{\lambda_3 - \lambda_1}{h_l - h_r}, \tag{6.15}$$

$$F_{*,l,2} = F_{*,r,2} = \frac{\lambda_3 F_{l,2} - \lambda_1 F_{r,2}}{\lambda_3 - \lambda_1} - \lambda_1 \lambda_3 \frac{\lambda_3 - \lambda_1}{h_l u_l - h_r u_r}, \tag{6.16}$$

where $F_1 = hu$ and $F_2 = hu^2 + gh^2/2$. For the third component, we impose that there is no jump across the 1- and 3-characteristics. The only jump in $hu\phi$ is across $x = \lambda_2 t$. Because the third component $hu\phi$ is the product of ϕ and the first $\boldsymbol{F}$ component hu, then we can write

$$F_{*,l,3} = F_{*,l,1}\phi_l, \tag{6.17}$$

$$F_{*,r,3} = F_{*,r,2}\phi_r. \tag{6.18}$$

The flux at the interface $x = 0$ is thus $F_{*,l,3}$ if $\lambda_2 > 0$, and $F_{*,r,3}$ if $\lambda_2 > 0$.

An estimate of the wave speed λ_2 is (Toro, 2001):

$$\lambda_2 = \frac{\lambda_1 h_r (u_r - \lambda_3) - \lambda_3 h_l (u_l - \lambda_1)}{h_r (u_r - \lambda_3) - h_l (u_l - \lambda_1)}. \tag{6.19}$$

We consider three waves

$$\boldsymbol{W}_1 = \boldsymbol{Q}_{l,*} - \boldsymbol{Q}_l,\ \boldsymbol{W}_2 = \boldsymbol{Q}_{r,*} - \boldsymbol{Q}_{l,*} \text{ and } \boldsymbol{W}_3 = \boldsymbol{Q}_r - \boldsymbol{Q}_{r,*}. \tag{6.20}$$

In § 2.6 and 5.3.1, we have shown that the intermediate state for the water flow is:

$$\boldsymbol{Q}_*^\dagger = \frac{s_3 \boldsymbol{Q}_r^\dagger - s_1 \boldsymbol{Q}_l^\dagger}{s_3 - s_1} - \frac{\boldsymbol{F}_r^\dagger - \boldsymbol{F}_l^\dagger}{s_3 - s_1},$$

where $\boldsymbol{Q}_*^\dagger = (h,\ hu)$ and $\boldsymbol{F}^\dagger = (q,\ \Phi)$ (with $q = hu$ and $\Phi = hu^2 + gh^2/2$) are the first two components of $\boldsymbol{Q}$ and $\boldsymbol{F}$. We thus have

$$h_* = \frac{s_3 h_r - s_1 h_l}{s_3 - s_1} - \frac{s_3 q_r - s_1 q_l}{s_3 - s_1},$$

and

$$q_* = (hu)_* = \frac{s_3 q_r - s_1 q_l}{s_3 - s_1} - \frac{s_3 \Phi_r - s_1 \Phi_l}{s_3 - s_1}.$$

We then deduce:

$$\boldsymbol{W}_1^\dagger = \begin{pmatrix} h_* - h_l \\ q_* - q_l \end{pmatrix},\ \boldsymbol{W}_2^\dagger = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ and } \boldsymbol{W}_3^\dagger = \begin{pmatrix} h_r - h_* \\ q_r - q_* \end{pmatrix}.$$

For the third component, we have

$$W_{1,3} = 0,\ W_{2,3} = \phi_r - \phi_l, \text{ and } W_{1,3} = 0.$$

6.3.2 Implementation in Pyclaw

```
def shallow_hllc_1D(q_l,q_r,aux_l,aux_r,problem_data):
    r"""
    HLLC shallow water solver ::
    """
    # Array shapes
    num_rp    = q_l.shape[1]
    num_eqn   = 3
    num_waves = 3

    g = problem_data['grav']

    # Output arrays
    wave = np.empty( (num_eqn, num_waves, num_rp) )
    s    = np.empty( (num_waves, num_rp) )
    amdq = np.zeros( (num_eqn, num_rp) )
    apdq = np.zeros( (num_eqn, num_rp) )

    h_l  = q_l[0,:]
    h_r  = q_r[0,:]
    hu_l = q_l[1,:]
    hu_r = q_r[1,:]
    u_r = hu_r/h_r
    c_r = np.sqrt(g * h_r)
    u_l = hu_l/h_l
    c_l = np.sqrt(g * h_l)
    Phi_l = u_l**2*h_l+0.5*g*h_l**2
    Phi_r = u_r**2*h_r+0.5*g*h_r**2

    # Compute Roe and right and left speeds
    u_hat = (hu_l/np.sqrt(h_l) + hu_r/np.sqrt(h_r))/(np.sqrt(h_l) + np.sqrt
    (h_r))
    c_hat = np.sqrt(0.5 * g * (h_r + h_l))

    # Compute Einfeldt speeds
    s_index = np.empty((2,num_rp))
    s_index[0,:] = u_hat - c_hat
    s_index[1,:] = u_l - c_l
    s[0,:] = np.min(s_index,axis=0)
    s_index[0,:] = u_r + c_r
    s_index[1,:] = u_hat + c_hat
    s[2,:] = np.max(s_index,axis=0)

    lambda_1 = u_hat - c_hat
    lambda_3 = u_hat + c_hat
    u_toro = (lambda_1*h_r*(u_r-lambda_3) - lambda_3*h_l*(u_l-lambda_1)    )
        \
              /(h_r*(u_r-lambda_3) - h_l*(u_l-lambda_1))
    s[1,:] = u_hat

    # Compute middle state
    h_star  = (h_r * s[2,:] - h_l * s[0,:]-(hu_r-hu_l))/(s[2,:]-s[0,:])
    hu_star = (hu_r * s[2,:] - hu_l * s[0,:]-(Phi_r-Phi_l))/(s[2,:]-s[0,:])

    # Compute each family of waves
    wave[0,0,:] = h_star - h_l
    wave[1,0,:] = hu_star - hu_l
```

```
    wave[2,0,:] = 0.
    wave[0,1,:] = 0.
    wave[1,1,:] = 0.
    wave[2,1,:] = q_r[2,:]-q_l[2,:]
    wave[0,2,:] = h_r - h_star
    wave[1,2,:] = hu_r - hu_star
    wave[2,2,:] = 0.

    # Compute variations
    s_index = np.zeros((3,num_rp))
    for m in range(num_eqn):
        for mw in range(num_waves):
            s_index[0,:] = s[mw,:]
            amdq[m,:] += np.min(s_index,axis=0) * wave[m,mw,:]
            apdq[m,:] += np.max(s_index,axis=0) * wave[m,mw,:]

    return wave, s, amdq, apdq
```

6.4 F-wave formulation

6.4.1 Principle

The f-wave method consists of decomposing the jump in the flux (6.5) into three f-waves

$$\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1}) = \sum_{k=1}^{3} \boldsymbol{Z}_{k,i-1/2},$$

where the f-wave $\boldsymbol{Z}_{k,i-1/2}$ can be related to the right eigenvector $\hat{\boldsymbol{w}}_{k,i-1/2}$ of the Roe matrix:

$$\boldsymbol{Z}_{k,i-1/2} = \beta_{k,i-1/2}\hat{\boldsymbol{w}}_{k,i-1/2}$$

where the coefficient $\beta_{k,i-1/2}$ is the linear solution (see § 2.7):

$$\boldsymbol{\beta}_{i-1/2} = \boldsymbol{L} \cdot (\boldsymbol{f}(\boldsymbol{Q}_i) - \boldsymbol{f}(\boldsymbol{Q}_{i-1})).$$

with $\boldsymbol{L} = \boldsymbol{R}^{-1}$. We find that:

$$\boldsymbol{\beta}_{i-1/2} = \frac{1}{2\hat{c}} \begin{pmatrix} \Phi_l - \Phi_r + (\hat{u} + \hat{c})(q_r - q_l) \\ 2\hat{c}(\phi_r q_r - \phi_l q_l + \phi(q_r - q_l)) \\ \Phi_r - \Phi_l - (\hat{u} - \hat{c})(q_r - q_l) \end{pmatrix}, \tag{6.21}$$

where $\Phi = hu^2 + gh^2/2$. The f-waves are then:

$$\boldsymbol{Z}_{1,i-1/2} = \boldsymbol{\beta}_{1,i-1/2}\boldsymbol{w}_1 = \frac{\Phi_l - \Phi_r + (\hat{u} + \hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u} - \hat{c} \\ \phi \end{pmatrix},$$

$$\boldsymbol{Z}_{2,i-1/2} = \boldsymbol{\beta}_{2,i-1/2}\boldsymbol{w}_2 = (\phi_r q_r - \phi_l q_l - \phi(q_r - q_l)) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

and

$$\boldsymbol{Z}_{3,i-1/2} = \boldsymbol{\beta}_{3,i-1/2}\boldsymbol{w}_3 = \frac{\Phi_r - \Phi_l - (\hat{u} - \hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u} + \hat{c} \\ \phi \end{pmatrix}.$$

As for the Roe solver, we assume that there is no jump in ϕ for the 1- and 3- shock waves while for the 2-wave, there is no jump in h and hu (and so $q_r = q_l = \hat{q} = \hat{u}\bar{h}$), and so the correct f-waves are:

$$\boldsymbol{Z}_{1,i-1/2} = \boldsymbol{\beta}_{1,i-1/2}\boldsymbol{w}_1 = \frac{\Phi_l - \Phi_r + (\hat{u}+\hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u}-\hat{c} \\ 0 \end{pmatrix}, \tag{6.22}$$

$$\boldsymbol{Z}_{2,i-1/2} = \boldsymbol{\beta}_{2,i-1/2}\boldsymbol{w}_2 = (\phi_r - \phi_l)\hat{q} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{6.23}$$

and

$$\boldsymbol{Z}_{3,i-1/2} = \boldsymbol{\beta}_{3,i-1/2}\boldsymbol{w}_3 = \frac{\Phi_r - \Phi_l - (\hat{u}-\hat{c})(q_r - q_l)}{2\hat{c}} \begin{pmatrix} 1 \\ \hat{u}+\hat{c} \\ 0 \end{pmatrix}. \tag{6.24}$$

6.4.2 Implementation in Pyclaw

```
def shallow_hllc_fwave_1d(q_l, q_r, aux_l, aux_r, problem_data):
    r"""Shallow water Riemann solver using fwaves
    """

    g             = problem_data['grav']
    dry_tolerance = problem_data['dry_tolerance']

    num_rp    = q_l.shape[1]
    num_eqn   = 3
    num_waves = 3

    # Initializing arrays
    fwave = np.empty( (num_eqn, num_waves, num_rp) )
    s     = np.empty( (num_waves, num_rp) )
    amdq  = np.zeros( (num_eqn, num_rp) )
    apdq  = np.zeros( (num_eqn, num_rp) )
    r1    = np.zeros( (num_waves, num_rp) )
    r2    = np.zeros( (num_waves, num_rp) )
    r3    = np.zeros( (num_waves, num_rp) )

    # Extract state
    h_l  = q_l[0, :]
    h_r  = q_r[0, :]
    hu_l = q_l[1, :]
    hu_r = q_r[1, :]
    u_l  = np.where(h_l > dry_tolerance, hu_l / h_l, 0.0)
    u_r  = np.where(h_r > dry_tolerance, hu_r / h_r, 0.0)

    # Flux and Roe depth
    phi_l = h_l * u_l**2 + 0.5 * g * h_l**2
    phi_r = h_r * u_r**2 + 0.5 * g * h_r**2
    h_bar = 0.5 * (h_l + h_r)

    # Speeds
    u_hat = (np.sqrt(h_l)*u_l + np.sqrt(h_r)*u_r)/ (np.sqrt(h_l) + np.sqrt(
   h_r))
    c_hat = np.sqrt(g * h_bar)

```

```
    s[0, :] = np.amin(np.vstack((u_l - np.sqrt(g * h_l), u_hat - c_hat)),
axis=0)
    s[1, :] = u_hat
    s[2, :] = np.amax(np.vstack((u_r + np.sqrt(g * h_r), u_hat + c_hat)),
axis=0)

    beta1 = (phi_l - phi_r + (u_hat+c_hat)*(hu_r-hu_l))/2./c_hat
    beta2 = (q_r[2, :]- q_l[2, :])*u_hat
    beta3 = (phi_r - phi_l - (u_hat-c_hat)*(hu_r-hu_l))/2./c_hat

    r1[0, :] = 1.
    r1[1, :] = u_hat - c_hat
    r1[2, :] = 0.

    r2[0, :] = 0.
    r2[1, :] = 0.
    r2[2, :] = 1.

    r3[0, :] = 1.
    r3[1, :] = u_hat + c_hat
    r3[2, :] = 0.

    fwave[0, 0, :] = beta1 * r1[0, :]
    fwave[1, 0, :] = beta1 * r1[1, :]
    fwave[2, 0, :] = beta1 * r1[2, :]

    fwave[0, 1, :] = beta2 * r2[0, :]
    fwave[1, 1, :] = beta2 * r2[1, :]
    fwave[2, 1, :] = beta2 * r2[2, :]

    fwave[0, 2, :] = beta3 * r3[0, :]
    fwave[1, 2, :] = beta3 * r3[1, :]
    fwave[2, 2, :] = beta3 * r3[2, :]

    for m in range(num_eqn):
        for mw in range(num_waves):
            amdq[m, :] += (s[mw, :] < 0.0) * fwave[m, mw, :]
            apdq[m, :] += (s[mw, :] > 0.0) * fwave[m, mw, :]

    return fwave, s, amdq, apdq
```

6.5 Example: dam break

We consider a dam break problem with the following initial conditions: $h_l = 3$ m et $u_l = 0$ for $x \leq 0$, and $h_l = 1$ m et $u_l = 0$ for $x > 0$. We compare the two solvers: Roe (with no entropy fix) and the f-wave formulation of the HLLC solver. Figures 6.2 shows the comparison.

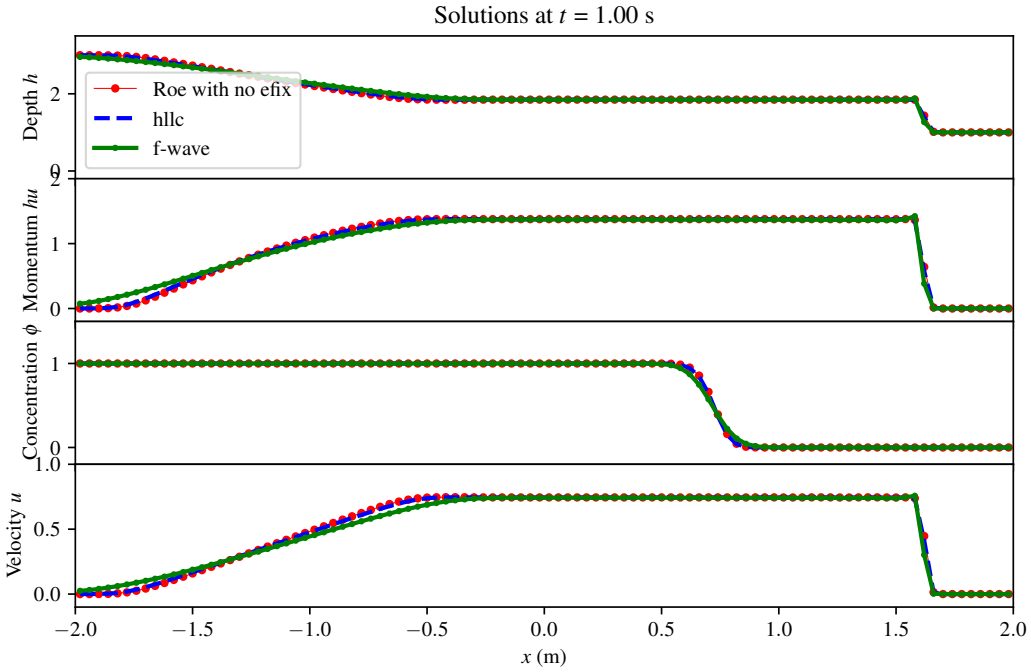
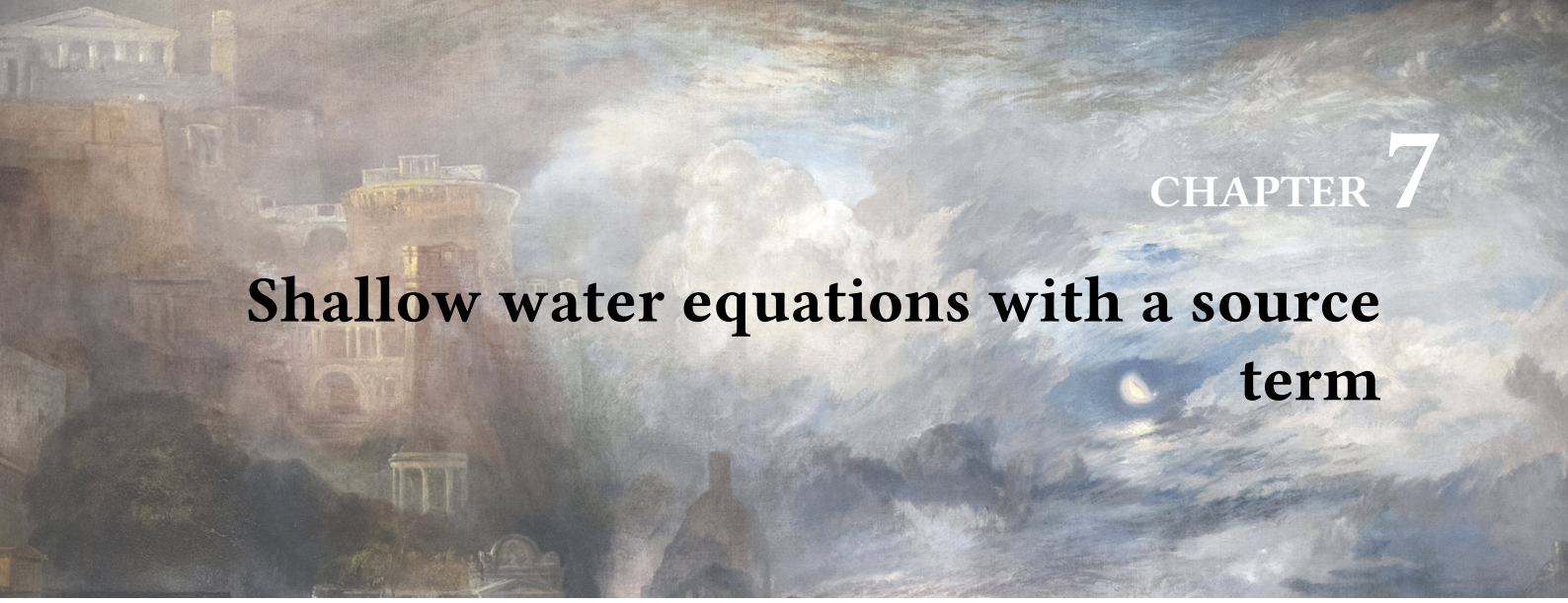


Figure 6.2 Comparison of the three methods: Roe solver, HLLC, and the f-wave variant of the HLLC algorithm. Computations done with $g = 1$ m/s^2.



CHAPTER 7

Shallow water equations with a source term

7.1 Theory

7.1.1 *flow resistance*

In an one-dimensional fixed Cartesian frame, the Saint-Venant equations take the tensorial form

$$\frac{\partial}{\partial t}\boldsymbol{Q} + \nabla \boldsymbol{f}(\boldsymbol{Q}) = \boldsymbol{S}, \tag{7.1}$$

where $\boldsymbol{Q} = (h, hu)$ is the unknown, and $\boldsymbol{S} = (0, S)$ is the source term. The computation strategy involves first solving the homogenous problem (LeVeque, 2002) :

$$\frac{\partial}{\partial t}\boldsymbol{Q} + \nabla \boldsymbol{f}(\boldsymbol{Q}) = 0, \tag{7.2}$$

then correcting the solution by taking the effect of the source term on the momentum $q = hu$:

$$\varrho \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q} = S(\boldsymbol{Q}), \tag{7.3}$$

where $S(\boldsymbol{Q})$ takes the following form if we consider a flow experiencing flow resistance:

$$S(\boldsymbol{U}) = -\frac{\varrho g}{K^2 h^{1/3}}|u|u, \tag{7.4}$$

$$= -\frac{\varrho g}{K^2 h^{7/3}}|q|q, \tag{7.5}$$

where K is the Manning-Strickler coefficient.

Let us assume that we have computed the solution $\boldsymbol{q}_*$ to the homogenous equation (7.2), and we are now seeking the solution at time $k+1$. Using a semi-implicit discretization of (7.3) leads to

$$q^{k+1} = q^* - \mathrm{d}t\frac{g}{K^2 h^{7/3}}|q^*|q^{k+1}, \tag{7.6}$$

$$q^* = q^{k+1}\left(1 + \frac{g\mathrm{d}t}{K^2 h^{7/3}}|q^*|\right), \tag{7.7}$$

$$q^{k+1} = \frac{q^*}{1 + \mathrm{d}t\dfrac{g}{K^2 h^{7/3}}|q^*|}. \tag{7.8}$$

Guinot, V. 2010 *Wave Propagation in Fluids—Models and Numerical Techniques*. Hoboken: John Wiley & Sons.

Ketcheson, D. I., LeVeque, R. J. & del Razo, M. 2020 *Riemann Problems and Jupyter Solutions*. Philadelphia: Society for Industrial and Applied Mathematics.

Ketcheson, D. I., Mandli, K., Ahmadia, A. J., Alghamdi, A., De Luna, M. Q., Parsani, M., Knepley, M. G. & Emmett, M. 2012 PyClaw: Accessible, extensible, scalable tools for wave propagation problems. *SIAM Journal on Scientific Computing* **34** (4), C210–C231.

LeVeque, R. 1992 *Numerical Methods for Conservation Laws*. Basel: Birkhäuser.

LeVeque, R. 2002 *Finite Volume Methods for Hyperbolic Problems*. Cambridge: Cambridge University Press.

Mandli, K. T., Ahmadia, A. J., Berger, M., Calhoun, D., George, D. L., Hadjimichael, Y., Ketcheson, D. I., Lemoine, G. I. & LeVeque, R. J. 2016 Clawpack: building an open source ecosystem for solving hyperbolic PDEs. *PeerJ Computer Science* **2**, e68.

Roe, P. 1981 Approximate Riemann solvers, parameters vectors, and difference schemes. *J. Comput. Phys.* **44**, 357–372.

Toro, E. 2001 *Shock-Capturing Methods for Free-Surface Shallow Flows*. Chichester: Wiley.

Toro, E. F. 2019 The HLLC Riemann solver. *Shock Waves* **29**, 1065–1082.