5

Introduction to the CLAWPACK Software

The basic class of finite volume methods developed in this book has been implemented in the software package CLAWPACK. This allows these algorithms to be applied to a wide variety of hyperbolic systems simply by providing the appropriate Riemann solver, along with initial data and boundary conditions. The high-resolution methods introduced in Chapter 6 are implemented, but the simple first-order Godunov method of Chapter 4 is obtained as a special case by setting the input parameters appropriately. (Specifically, set `method(2)=1` as described below.) In this chapter an overview of the software is given along with examples of its application to simple problems of advection and acoustics.

The software includes more advanced features that will be introduced later in the book, and can solve linear and nonlinear problems in one, two, and three space dimensions, as well as allowing the specification of capacity functions introduced in Section 2.4 (see Section 6.16) and source terms (see Chapter 17). CLAWPACK is used throughout the book to illustrate the implementation and behavior of various algorithms and their application on different physical systems. Nearly all the computational results presented have been obtained using CLAWPACK with programs that can be downloaded to reproduce these results or investigate the problems further. These samples also provide templates that can be adapted to solve other problems. See Section 1.5 for details on how to access webpages for each example.

Only the one-dimensional software is introduced here. More extensive documentation, including discussion of the multidimensional software and adaptive mesh refinement capabilities, can be downloaded from the webpage

```
http://www.amath.washington.edu/~claw/
```

See also the papers [283], [257] for more details about the multidimensional algorithms, and [32] for a discussion of some features of the adaptive mesh refinement code.

5.1 Basic Framework

In one space dimension, the CLAWPACK routine `claw1` (or the simplified version `claw1ez`) can be used to solve a system of equations of the form

$$\kappa(x)q_t + f(q)_x = \psi(q, x, t), \tag{5.1}$$

where $q = q(x,t) \in \mathbb{R}^m$. The standard case of a homogeneous conservation law has $\kappa \equiv 1$ and $\psi \equiv 0$,

$$q_t + f(q)_x = 0. \tag{5.2}$$

The flux function $f(q)$ can also depend explicitly on x and t as well as on q. Hyperbolic systems that are not in conservation form, e.g.,

$$q_t + A(x,t)q_x = 0, \tag{5.3}$$

can also be solved.

The basic requirement on the homogeneous system is that it be hyperbolic in the sense that a Riemann solver can be specified that, for any two states Q_{i-1} and Q_i, returns a set of M_w waves $\mathcal{W}^p_{i-1/2}$ and speeds $s^p_{i-1/2}$ satisfying

$$\sum_{p=1}^{M_w} \mathcal{W}^p_{i-1/2} = Q_i - Q_{i-1} \equiv \Delta Q_{i-1/2}.$$

The Riemann solver must also return a left-going fluctuation $\mathcal{A}^-\Delta Q_{i-1/2}$ and a right-going fluctuation $\mathcal{A}^+\Delta Q_{i-1/2}$. In the standard conservative case (5.2) these should satisfy

$$\mathcal{A}^-\Delta Q_{i-1/2} + \mathcal{A}^+\Delta Q_{i-1/2} = f(Q_i) - f(Q_{i-1}) \tag{5.4}$$

and the fluctuations then define a *flux-difference splitting* as described in Section 4.13. Typically

$$\mathcal{A}^-\Delta Q_{i-1/2} = \sum_p \left(s^p_{i-1/2}\right)^- \mathcal{W}^p_{i-1/2}, \qquad \mathcal{A}^+\Delta Q_{i-1/2} = \sum_p \left(s^p_{i-1/2}\right)^+ \mathcal{W}^p_{i-1/2}, \tag{5.5}$$

where $s^- = \min(s,0)$ and $s^+ = \max(s,0)$. In the nonconservative case (5.3), there is no flux function $f(q)$, and the constraint (5.4) need not be satisfied.

Only the fluctuations are used for the first-order Godunov method, which is implemented in the form introduced in Section 4.12,

$$Q^{n+1}_i = Q^n_i - \frac{\Delta t}{\Delta x}\left(\mathcal{A}^+\Delta Q_{i-1/2} + \mathcal{A}^-\Delta Q_{i+1/2}\right), \tag{5.6}$$

assuming $\kappa \equiv 1$.

The Riemann solver must be supplied by the user in the form of a subroutine `rp1`, as described below. Typically the Riemann solver first computes waves and speeds and then uses these to compute $\mathcal{A}^+\Delta Q_{i-1/2}$ and $\mathcal{A}^-\Delta Q_{i-1/2}$ internally in the Riemann solver. The waves and speeds must also be returned by the Riemann solver in order to use the high-resolution methods described in Chapter 6. These methods take the form

$$Q^{n+1}_i = Q^n_i - \frac{\Delta t}{\Delta x}\left(\mathcal{A}^+\Delta Q_{i-1/2} + \mathcal{A}^-\Delta Q_{i+1/2}\right) - \frac{\Delta t}{\Delta x}\left(\tilde{F}_{i+1/2} - \tilde{F}_{i-1/2}\right), \tag{5.7}$$

where

$$\tilde{F}_{i-1/2} = \frac{1}{2}\sum_{p=1}^{m} \left|s^p_{i-1/2}\right| \left(1 - \frac{\Delta t}{\Delta x}\left|s^p_{i-1/2}\right|\right) \tilde{\mathcal{W}}^p_{i-1/2}. \tag{5.8}$$

Here $\widetilde{\mathcal{W}}^p_{i-1/2}$ represents a limited version of the wave $\mathcal{W}^p_{i-1/2}$, obtained by comparing $\mathcal{W}^p_{i-1/2}$ to $\mathcal{W}^p_{i-3/2}$ if $s^p > 0$ or to $\mathcal{W}^p_{i+1/2}$ if $s^p < 0$.

When a capacity function $\kappa(x)$ is present, the Godunov method becomes

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\kappa_i \, \Delta x}\left(\mathcal{A}^+ \Delta Q_{i-1/2} + \mathcal{A}^- \Delta Q_{i+1/2}\right), \tag{5.9}$$

See Section 6.16 for discussion of this algorithm and its extension to the high-resolution method.

If the equation has a source term, a routine `src1` must also be supplied that solves the source-term equation $q_t = \psi(q, \kappa)$ over a time step. A fractional-step method is used to couple this with the homogeneous solution, as described in Chapter 17. Boundary conditions are imposed by setting values in ghost cells each time step, as described in Chapter 7. A few standard boundary conditions are implemented in a library routine, but this can be modified to impose other conditions; see Section 5.4.4.

5.2 Obtaining CLAWPACK

The latest version of CLAWPACK can be downloaded from the web, at

```
http://www.amath.washington.edu/~claw/
```

Go to "download software" and select the portion you wish to obtain. At a minimum, you will need

```
claw/clawpack
```

If you plan to use Matlab to plot results, some useful scripts are in

```
claw/matlab
```

Other plotting packages can also be used, but you will have to figure out how to properly read in the solution produced by CLAWPACK.

The basic CLAWPACK directories `1d`, `2d`, and `3d` each contain one or two examples in directories such as

```
claw/1d/example1
```

that illustrate the basic use of CLAWPACK. The directory

```
claw/book
```

contains drivers and data for all the examples presented in this book. You can download this entire directory or selectively download specific examples as you need them. Some other applications of CLAWPACK can be found in

```
claw/applications
```

5.3 Getting Started

The discussion here assumes you are using the Unix (or Linux) operating system. The Unix prompt is denoted by `unix>`.

5.3.1 Creating the Directories

The files you download will be gzipped `tar` files. Before installing any of CLAWPACK, you should create a directory named `<path>/claw` where the pathname `<path>` depends on where you want these files to reside and the local naming conventions on your computer. You should download any CLAWPACK or `claw/book` files to this directory. After downloading any file of the form `name.tar.gz`, execute the following commands:

```
unix> gunzip name.tar.gz
unix> tar -xvf name.tar
```

This will create the appropriate subdirectories within `<path>/claw`.

5.3.2 Environment variables for the path

You should now set the environment variable `CLAW` in Unix so that the proper files can be found:

```
unix> setenv CLAW <path>/claw
```

You might want to put this line in your `.cshrc` file so it will automatically be executed when you log in or create a new window. Now you can refer to `$CLAW/clawpack/1d`, for example, and reach the correct directory.

5.3.3 Compiling the code

Go to the directory `claw/clawpack/1d/example1`. There is a file in this directory named `compile`, which should be executable so that you can type

```
unix> compile
```

This should invoke `f77` to compile all the necessary files and create an executable called `xclaw`. To run the program, type

```
unix> xclaw
```

and the program should run, producing output files that start with `fort.` In particular, `fort.q0000` contains the initial data, and `fort.q0001` the solution at the first output time. The file `fort.info` has some information about the performance of CLAWPACK.

5.3.4 Makefiles

The `compile` file simply compiles all of the routines needed to run CLAWPACK on this example. This is simple, but if you make one small change in one routine, then everything has to be recompiled. Instead it is generally easier to use a `Makefile`, which specifies what set of object files (ending with `.o`) are needed to make the executable, and which Fortran files (ending with `.f`) are needed to make the object files. If a Fortran file is changed, then it is only necessary to recompile this one rather than everything. This is done simply by typing

```
unix> make
```

A complication arises in that the example1 directory only contains a few of the necessary Fortran files, the ones specific to this particular problem. All the standard CLAWPACK files are in the directory claw/clawpack/1d/lib. You should first go into that directory and type make to create the object files for these library routines. This only needs to be done once if these files are never changed. Now go to the example1 directory and also type make. Again an executable named xclaw should be created. See the comments at the start of the Makefile for some other options.

5.3.5 Matlab Graphics

If you wish to use Matlab to view the results, you should download the directory claw/matlab and then set the environment variable

```
unix> setenv MATLABPATH ".:\$CLAW/matlab"
```

before starting Matlab, in order to add this directory to your Matlab search path. This directory contains the plotting routines plotclaw1.m and plotclaw2.m for plotting results in one and two dimensions respectively.

With Matlab running in the example1 directory, type

```
Matlab> plotclaw1
```

to see the results of this computation. You should see a pulse advecting to the right with velocity 1, and wrapping around due to the periodic boundary conditions applied in this example.

5.4 Using CLAWPACK – a Guide through example1

The program in claw/clawpack/1d/example1 solves the advection equation

$$q_t + uq_x = 0$$

with constant velocity $u = 1$ and initial data consisting of a Gaussian hump

$$q(x, 0) = \exp(-\beta(x - 0.3)^2). \tag{5.10}$$

The parameters $u = 1$ and $\beta = 200$ are specified in the file setprob.data. These values are read in by the routine setprob.f described in Section 5.5.

5.4.1 The Main Program (driver.f)

The main program is located in the file driver.f. It simply allocates storage for the arrays needed in CLAWPACK and then calls claw1ez, described below. Several parameters are set and used to declare these arrays. The proper values of these parameters depends on the particular problem. They are:

maxmx: The maximum number of grid cells to be used. (The actual number mx is later read in from the input file claw1ez.data and must satisfy mx $\leq$ maxmx.)

meqn: The number of equations in the hyperbolic system, e.g., meqn = 1 for a scalar equation, meqn = 2 for the acoustics equations (2.50), etc.

`mwaves`: The number of waves produced in each Riemann solution, called M_w in the text. Often `mwaves = meqn`, but not always.

`mbc`: The number of ghost cells used for implementing boundary conditions, as described in Chapter 7. Setting `mbc = 2` is sufficient unless changes have been made to the CLAWPACK software resulting in a larger stencil.

`mwork`: A work array of dimension `mwork` is used internally by CLAWPACK for various purposes. How much space is required depends on the other parameters:

```
mwork ≥ (maxmx + 2*mbc) * (2 + 4*meqn + mwaves + meqn*mwaves)
```

If the value of `mwork` is set too small, CLAWPACK will halt with an error message telling how much space is required.

`maux`: The number of "auxiliary" variables needed for information specifying the problem, which is used in declaring the dimensions of the array `aux` (see below).

Three arrays are declared in `driver.f`:

`q(1-mbc:maxmx+mbc, meqn)`: This array holds the approximation Q_i^n (a vector with `meqn` components) at each time t_n. The value of i ranges from 1 to `mx` where `mx <= maxmx` is set at run time from the input file. The additional ghost cells numbered `(1-mbc):0` and `(mx+1):(mx+mbc)` are used in setting boundary conditions.

`work(mwork)`: Used as work space.

`aux(1-mbc:maxmx+mbc, maux)`: Used for auxiliary variables if `maux > 0`. For example, in a variable-coefficient advection problem the velocity in the ith cell might be stored in `aux(i,1)`. See Section 5.6 for an example and more discussion. If `maux = 0`, then there are no auxiliary variables, and `aux` can simply be declared as a scalar or not declared at all, since this array will not be referenced.

5.4.2 *The Initial Conditions* (`qinit.f`)

The subroutine `qinit.f` sets the initial data in the array `q`. For a system with `meqn` components, `q(i,m)` should be initialized to a cell average of the `m`th component in the `i`th grid cell. If the data is given by a smooth function, then it may be simplest to evaluate this function just at the center of the cell, which agrees with the cell average to $\mathcal{O}((\Delta x)^2)$. The left edge of the cell is at `xlower + (i-1)*dx`, and the right edge is at `xlower + i*dx`. It is only necessary to set values in cells `i = 1:mx`, not in the ghost cells. The values of `xlower`, `dx`, and `mx` are passed into `qinit.f`, having been set in `claw1ez`.

5.4.3 *The* `claw1ez` *Routine*

The main program `driver.f` sets up array storage and then calls the subroutine `claw1ez`, which is located in `claw/clawpack/1d/lib`, along with other standard CLAWPACK subroutines described below. The `claw1ez` routine provides an easy way to use CLAWPACK that should suffice for many applications. It reads input data from a file `claw1ez.data`, which is assumed to be in a standard form described below. It also makes other assumptions about what the user is providing and what type of output is desired. After checking the

inputs for consistency, claw1ez calls the CLAWPACK routine claw1 repeatedly to produce the solution at each desired output time.

The claw1 routine (located in claw/clawpack/1d/lib/claw1.f) is much more general and can be called directly by the user if more flexibility is needed. See the documentation in the source code for this routine.

5.4.4 Boundary Conditions

Boundary conditions must be set in every time step, and claw1 calls a subroutine bc1 in every step to accomplish this. The manner in which this is done is described in detail in Chapter 7. For many problems the choice of boundary conditions provided in the default routine claw/clawpack/1d/lib/bc1.f will be sufficient. For other boundary conditions the user must provide an appropriate routine. This can be done by copying the bc1.f routine to the application directory and modifying it to insert the appropriate boundary conditions at the points indicated.

When using claw1ez, the claw1ez.data file contains parameters specifying what boundary condition is to be used at each boundary (see Section 5.4.6, where the mthbc array is described).

5.4.5 The Riemann Solver

The file claw/clawpack/1d/example1/rp1ad.f contains the Riemann solver, a subroutine that should be named rp1 if claw1ez is used. (More generally the name of the subroutine can be passed as an argument to claw1.) The Riemann solver is the crucial user-supplied routine that specifies the hyperbolic equation being solved. The input data consists of two arrays ql and qr. The value ql(i,:) is the value Q_i^L at the left edge of the ith cell, while qr(i,:) is the value Q_i^R at the right edge of the ith cell, as indicated in Figure 5.1. Normally ql = qr and both values agree with Q_i^n, the cell average. More flexibility is allowed because in some applications, or in adapting CLAWPACK to implement different algorithms, it is useful to allow different values at each edge. For example, we might want to define a piecewise linear function within the grid cell as illustrated in Figure 5.1 and

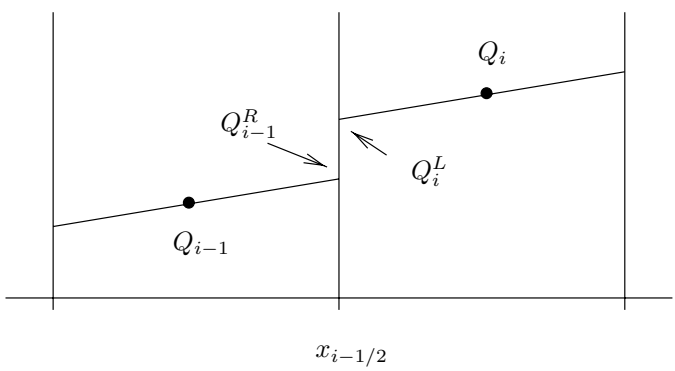

Fig. 5.1. The states used in solving the Riemann problem at the interface $x_{i-1/2}$.

then solve the Riemann problems between these values. This approach to high-resolution methods is discussed in Section 10.1.

Note that the Riemann problem at the interface $x_{i-1/2}$ between cells $i-1$ and i has data

$$\begin{aligned} &\text{left state:} && Q^R_{i-1} = \texttt{qr(i}-1,\texttt{:)}, \\ &\text{right state:} && Q^L_i = \texttt{ql(i,:)}. \end{aligned} \tag{5.11}$$

This notation is potentially confusing in that normally we use q_l to denote the left state and q_r to denote the right state in specifying Riemann data. The routine `rp1` must solve the Riemann problem for each value of `i`, and return the following:

`amdq(i,1:meqn)`, the vector $\mathcal{A}^-\Delta Q_{i-1/2}$ containing the left-going fluctuation as described in Section 4.12.

`apdq(i,1:meqn)`, the vector $\mathcal{A}^+\Delta Q_{i-1/2}$ containing the right-going fluctuation as described in Section 4.12.

`wave(i,1:meqn,p)`, the vector $\mathcal{W}^p_{i-1/2}$ representing the jump in q across the pth wave in the Riemann solution at $x_{i-1/2}$, for $p = 1, 2, \ldots,$ `mwaves`. (In the code `mw` is typically used in place of p.)

`s(i,p)`, the wave speed $s^p_{i-1/2}$ for each wave.

For Godunov's method, only the fluctuations `amdq` and `apdq` are actually used, and the update formula (5.6) is employed. The waves and speeds are only used for high-resolution correction terms as described in Chapter 6.

For the advection equation, the Riemann solver in `example1` returns

$$\begin{aligned} \texttt{wave(i, 1, 1)} &= \texttt{ql(i)} - \texttt{qr(i} - 1\texttt{)}, \\ \texttt{s(i, 1)} &= u, \\ \texttt{amdq(i, 1)} &= \min(u, 0)*\texttt{wave(i, 1, 1)}, \\ \texttt{apdq(i, 1)} &= \max(u, 0)*\texttt{wave(i, 1, 1)}. \end{aligned}$$

Sample Riemann solvers for a variety of other applications can be found in `claw/book` and `claw/applications`. Often these can be used directly rather than writing a new Riemann solver.

5.4.6 The Input File `claw1ez.data`

The `claw1ez` routine reads data from a file named `claw1ez.data`. Figure 5.2 shows the file from `example1`. Typically one value is read from each line of this file. Any text following this value on each line is not read and is there simply as documentation. The values read are:

`mx`: The number of grid cells for this computation. (Must have `mx < maxmx`, where `maxmx` is set in `driver.f`.)

`nout`: Number of output times at which the solution should be written out.

`outstyle`: There are three possible ways to specify the output times. This parameter selects the desired manner to specify the times, and affects what is required next.

`outstyle = 1`: The next line contains a single value `tfinal`. The computation should proceed to this time, and the `nout` outputs will be at times `t0 + (tfinal - t0)/nout`, where the initial time `t0` is set below.

```
50              mx          = cells in x direction

11              nout        = number of output times to print results
1               outstyle    = style of specifying output times
2.2d0           tfinal      = final time

0.1d0           dtv(1)      = initial dt (used in all steps if method(1)=0)
1.0d99          dtv(2)      = max allowable dt
1.0d0           cflv(1)     = max allowable Courant number
0.9d0           cflv(2)     = desired Courant number
  500           nv(1)       = max number of time steps per call to claw1

1               method(1)   = 1 for variable dt
2               method(2)   = order
0               method(3)   = not used in one dimension
1               method(4)   = verbosity of output
0               method(5)   = source term splitting
0               method(6)   = mcapa
0               method(7)   = maux (should agree with parameter in driver)

1               meqn        = number of equations in hyperbolic system
1               mwaves      = number of waves in each Riemann solution
3               mthlim(mw)  = limiter for each wave  (mw=1,mwaves)

0.d0            t0          = initial time
0.0d0           xlower      = left edge of computational domain
1.0d0           xupper      = right edge of computational domain

2               mbc         = number of ghost cells at each boundary
2               mthbc(1)    = type of boundary conditions at left
2               mthbc(2)    = type of boundary conditions at right
```

Fig. 5.2. A typical `claw1ez.data` file, from `claw/clawpack/1d/example1` for advection.

`outstyle = 2`: The next line(s) contain a list of `nout` times at which the outputs are desired. The computation will end when the last of these times is reached.

`outstyle = 3`: The next line contains two values

```
nstepout, nsteps
```

A total of `nsteps` time steps will be taken, with output after every `nstepout` time steps. The value of `nout` is ignored. This is most useful if you want to insure that time steps of maximum length are always taken with a desired Courant number. With the other output options, the time steps are adjusted to hit the desired times exactly. This option is also useful for debugging if you want to force the solution to be output every time step, by setting `nstepout = 1`.

`dtv(1)`: The initial value of Δt used in the first time step. If `method(1) = 0` below, then fixed-size time steps are used and this is the value of Δt in all steps. In this case Δt must divide the time increment between all requested outputs an integer number of times.

`dtv(2)`: The maximum time step Δt to be allowed in any step (in the case where `method(1) = 1` and variable Δt is used). Variable time steps are normally chosen

based on the Courant number, and this parameter can then simply be set to some very large value so that it has no effect. For some problems, however, it may be necessary to restrict the time step to a smaller value based on other considerations, e.g., the behavior of source terms in the equations.

`cflv(1)`: The maximum Courant number to be allowed. The Courant number is calculated after all the Riemann problems have been solved by determining the maximum wave speed seen. If the Courant number is no larger than `cflv(1)`, then this step is accepted. If the Courant number is larger, then:

`method(1)=0`: (fixed time steps), the calculation aborts.

`method(1)=1`: (variable time steps), the step is rejected and a smaller time step is taken.

Usually `cflv(1) = 1` can be used.

`cflv(2)`: The desired Courant number for this computation. Used only if `method(1)=1` (variable time steps). In each time step, the next time increment Δt is based on the maximum wave speed found in solving all Riemann problems in the *previous* time step. If the wave speeds do not change very much, then this will lead to roughly the desired Courant number. It's typically best to take `cflv(2)` to be slightly smaller than `cflv(1)`, say `cflv(2) = 0.9`.

`nv(1)`: The maximum number of time steps allowed in any single call to `claw1`. This is provided as a precaution to avoid too lengthy runs.

`method(1)`: Tells whether fixed or variable size time steps are to be used.

`method(1) = 0`: A fixed time step of size `dtv(1)` will be used in all steps.

`method(1) = 1`: CLAWPACK will automatically select the time step as described above, based on the desired Courant number.

`method(2)`: The order of the method.

`method(2) = 1`: The first-order Godunov's method described in Chapter 4 is used.

`method(2) = 2`: High-resolution correction terms are also used, as described in Chapter 6.

`method(3)`: This parameter is not used in one space dimension. In two and three dimensions it is used to further specify which high-order correction terms are applied.

`method(4)`: This controls the amount of output printed by `claw1` on the screen as CLAWPACK progresses.

`method(4) = 0`: Information is printed only when output files are created.

`method(4) = 1`: Every time step the value Δt and Courant number are reported.

`method(5)`: Tells whether there is a source term in the equation. If so, then a fractional-step method is used as described in Chapter 17. Time steps on the homogeneous hyperbolic equation are alternated with time steps on the source term. The solution operator for the source terms must be provided by the user in the routine `src1.f`.

`method(5) = 0`: There is no source term. In this case the default routine `claw/clawpack/1d/lib/src1.f` can be used, which does nothing, and in fact this routine will never be called.

`method(5) = 1`: A source term is specified in `src1.f`, and the first-order (Godunov) fractional-step method should be used.

`method(5) = 2`: A source term is specified in `src1.f`, and a Strang splitting is used. The Godunov splitting is generally recommended rather than the Strang splitting, for reasons discussed in Chapter 17.

method(6): Tells whether there is a "capacity function" in the equation, as introduced in Section 2.4.

method(6) = 0: No capacity function; $\kappa \equiv 1$ in (2.27).

method(6) = mcapa > 0: There is a capacity function, and the value of κ in the ith cell is given by aux(i,mcapa), i.e., the mcapa component of the aux array is used to store this function. In this case *capacity-form differencing* is used, as described in Section 6.16.

method(7): Tells whether there are any auxiliary variables stored in an aux array.

method(7) = 0: No auxiliary variables. In this case the array aux is not referenced and can be a dummy variable.

method(7) = maux > 0: There is an aux array with maux components. In this case the array must be properly declared in driver.f.

Note that we must always have maux $\geq$ mcapa. The value of method(7) specified here must agree with the value of maux set in driver.f.

meqn: The number of equations in the hyperbolic system. This is also set in driver.f and the two should agree.

mwaves: The number of waves in each Riemann solution. This is often equal to meqn but need not be. This is also set in driver.f, and the two should agree.

mthlim(1:mwaves): The limiter to be applied in each wave family as described in Chapter 6. Several different limiters are provided in CLAWPACK [see (6.39)]:

mthlim(mw) = 0: No limiter (Lax–Wendroff)

mthlim(mw) = 1: Minmod

mthlim(mw) = 2: Superbee

mthlim(mw) = 3: van Leer

mthlim(mw) = 4: MC (monotonized centered)

Other limiters can be added by modifying the routine claw/clawpack/1d/lib/philim.f, which is called by claw/clawpack/1d/lib/limiter.f.

t0: The initial time.

xlower: The left edge of the computational domain.

xupper: The right edge of the computational domain.

mbc: The number of ghost cells used for setting boundary conditions. Usually mbc = 2 is used. See Chapter 7.

mthbc(1): The type of boundary condition to be imposed at the left boundary. See Chapter 7 for more description of these and how they are implemented. The following values are recognized:

mthbc(1) = 0: The user will specify a boundary condition. In this case you must copy the file claw/clawpack/1d/lib/bc1.f to your application directory and modify it to insert the proper boundary conditions in the location indicated.

mthbc(1) = 1: Zero-order extrapolation.

mthbc(1) = 2: Periodic boundary conditions. In this case you must also set mthbc(2) = 2.

mthbc(1) = 3: Solid wall boundary conditions. This set of boundary conditions only makes sense for certain systems of equations; see Section 7.3.3.

mthbc(2): The type of boundary condition to be imposed at the right boundary. The same values are recognized as described above.

5.5 Other User-Supplied Routines and Files

Several other routines may be provided by the user but are not required. In each case there is a default version provided in the library `claw/clawpack/1d/lib` that does nothing but `return`. If you wish to provide a version, copy the library version to the application directory, modify it as required, and also modify the `Makefile` to point to the modified version rather than to the library version.

`setprob.f` The `claw1ez` routine always calls `setprob` at the beginning of execution. The user can provide a subroutine that sets any problem-specific parameters or does other initialization. For the advection problem solved in `example1`, this is used to set the advection velocity u. This value is stored in a common block so that it can be passed into the Riemann solver, where it is required. The parameter `beta` is also set and passed into the routine `qinit.f` for use in setting the initial data according to (5.10). When `claw1ez` is used, a `setprob` subroutine must always be provided. If there is nothing to be done, the default subroutine `claw/clawpack/1d/lib/setprob.f` can be used, which does nothing but `return`.

`setaux.f` The `claw1ez` routine calls a subroutine `setaux` before the first call to `claw1`. This routine should set the array `aux` to contain any necessary data used in specifying the problem. For the example in `example1` no `aux` array is used (`maux` = 0 in `driver.f`) and the default subroutine `claw/clawpack/1d/lib/setaux.f` is specified in the `Makefile`.

`b4step1.f` Within `claw1` there is a call to a routine `b4step1` before each call to `step1` (the CLAWPACK routine that actually takes a single time step). The user can supply a routine `b4step1` in place of the default routine `claw/clawpack/1d/lib/b4step1.f` in order to perform additional tasks that might be required each time step. One example might be to modify the `aux` array values each time step, as described in Section 5.6.

`src1.f` If the equation includes a source term ψ as in (5.1), then a routine `src1` must be provided in place of the default routine `claw/clawpack/1d/lib/src1.f`. This routine must solve the equation $q_t = \psi$ over one time step. Often this requires solving an ordinary differential equation in each grid cell. In some cases a partial differential equation must be solved, for example if diffusive terms are included with $\psi = q_{xx}$, then the diffusion equation must be solved over one time step.

5.6 Auxiliary Arrays and `setaux.f`

The array `q(i,1:meqn)` contains the finite-volume solution in the ith grid cell. Often other arrays defined over the grid are required to specify the problem in the first place. For example, in a variable-coefficient advection problem

$$q_t + u(x)q_x = 0$$

the Riemann solution at any cell interface $x_{i-1/2}$ depends on the velocities u_{i-1} and u_i. The `aux` array can be used to store these values and pass them into the Riemann solver. In the advection example we need only one auxiliary variable, so `maux = 1` and we store the velocity u_i in `aux(i,1)`. See Chapter 9 for more discussion of variable-coefficient problems.

Of course one could hard-wire the specific function $u(x)$ into the Riemann solver or pass it in using a common block, but the use of the auxiliary arrays gives a uniform treatment of such data arrays. This is useful in particular when adaptive mesh refinement is applied, in which case there are many different q grids covering different portions of the computational domain and it is very convenient to have an associated aux array corresponding to each.

The claw1ez routine always calls a subroutine setaux before beginning the computation. This routine, normally stored in setaux.f, should set the values of all auxiliary arrays. If maux = 0 then the default routine claw/clawpack/1d/lib/setaux.f can be used, which does nothing. See Section 5.6 for an example of the use of auxiliary arrays.

In some problems the values stored in the aux arrays must be time-dependent, for example in an advection equation of the form $q_t + u(x,t)q_x = 0$. The routine setaux is called only once at the beginning of the computation and cannot be used to modify values later. The user can supply a routine b4step1 in place of the default routine claw/clawpack/1d/lib/b4step1.f in order to modify the aux array values each time step.

5.7 An Acoustics Example

The directory claw/clawpack/1d/example2 contains a sample code for the constant-coefficient acoustics equations (2.50). The values of the density and bulk modulus are set in setprob.f (where they are read in from a data file setprob.data). In this routine the sound speed and impedance are also computed and passed to the Riemann solver in a common block. The Riemann solver uses the formulas (3.31) to obtain α^1 and α^2, and then the waves are $\mathcal{W}^p = \alpha^p r^p$. The boundary conditions are set for a reflecting wall at $x = -1$ and nonreflecting outflow at $x = 1$.

Exercises

The best way to do these exercises, or more generally to use CLAWPACK on a new problem, is to copy an existing directory to a new directory with a unique name and modify the routines in that new directory.

5.1. The example in claw/clawpack/1d/example2 has method(2)=2 set in claw1ez.data, and hence uses a high-resolution method. Set method(2)=1 to use the upwind method instead, and compare the computed results.

5.2. Modify the data from Exercise 5.1 to take time steps for which the Courant number is 1.1. (You must change both cflv(1) and cflv(2) in claw1ez.data.) Observe that the upwind method is unstable in this case.

5.3. The initial data in claw/clawpack/1d/example2/qinit.f has $q^2(x,0) = 0$ and hence the initial pressure pulse splits into left going and right going pulses. Modify $q^2(x,0)$ so that the initial pulse is purely left going.