Chapter 7

THE SOFTWARE PACKAGE CLAWMAN

In the previous chapter, the details for a new class of high-resolution wave propagation algorithms for conservation laws on curved manifolds were presented. These methods have been shown to accurately capture the solution of systems consisting of both scalar and vector equations on various manifolds. One feature of the methods developed in the preceding chapters is that they are quite general. Specifically, the methods were not developed for a particular conservation law on a particular manifold. In fact, they can be applied to a wide variety of hyperbolic systems on general manifolds.

In this chapter, we present an implementation of these algorithms in the software package CLAWMAN (**C**onservation **LAW**s on **MAN**ifolds), a freely available collection of Fortran 90 routines. The implementation in CLAWMAN incorporates the generality of the methods developed in the previous chapters of this dissertation, making it useful to a broad class of users. The goal of this chapter is to give first-time users of the package enough information to be able to begin running, and modifying the examples provided with the code.

The CLAWMAN package is an extension of the CLAWPACK (**C**onservation **LAW**s **PACK**age) software discussed in [41], [42], and [46]. The CLAWPACK software is also freely available and can be found on the web at,

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

The documentation found on the website for CLAWPACK will be helpful for users unfamiliar with the basic implementation of high-resolution wave propagation algorithms.

7.1 Obtaining CLAWMAN

The software package CLAWMAN is freely available on the web at,

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

First-time users can go to the link that says, "download software", and choose the root directory called

`/clawman/2d`.

This link will transfer the entire contents of the two-dimensional version of CLAWMAN, including some examples described below. The file received is a directory archive that is gzipped for compression. The `.tar.gz` file should be placed in any directory that we will denote by `/install`. A good choice for `/install` is a user's home directory, e.g., `/install` = `/home/dbale`. The file can be unzipped, and the directory structure restored by executing the following commands:

```
>gunzip clawman.tar,

>tar -xvf clawman.tar,
```

where the UNIX or LINUX prompt has been denoted by `>`. Once these commands have been executed, `/install` will contain a directory called `/clawman`. In `/clawman`, there are directories for implementations in various numbers of spatial dimensions. The relevant directory for the methods developed in this thesis is `/2d`. The directory `/2d` contains the entire two-dimensional implementation discussed in this thesis.

Once the file has been downloaded, unzipped, and the directory structure restored, the subdirectory structure underneath `/2d` is organized as follows:

dir: `/modules` – Define and store elements of the code. These modules take the place of common blocks in the Fortran 77 version of CLAWPACK.

- module: `init_module.f90` – Defines and stores initialization parameters, vectors and arrays relevant to the code. The elements of this module are made available to the routines in CLAWMAN using the command `use init_module`. The elements of this module are discussed in Section 7.2.1.
- module: `grid_module.f90` – Defines and stores grid related parameters, vectors, and arrays. Metric information is defined here. The elements of this module are made available to routines by including the statement `use grid_module`.
- module: `step_module.f90` – Defines and stores parameters for each time step. For example, this module stores the current time step if the evolution is using a variable time step.
- module: `slice_module.f90` – Defines parameters and vectors relevant for each one-dimensional slice used to solve Riemann problems. Elements of this module are made available to the routines by including the statement `use slice_module`.

dir: `/cm_claw` – The directory containing the main time-stepping routines.

- file: `cm_claw2.f90` – The main stepping routine.
- file: `cm_step2.f90` Evolves the conservation law over one time step. This routine calls `cm_flux2.f90`.
- file: `cm_flux2.f90` – Parallel transports and orthonormalizes data before calling the Riemann solver.
- file: `init_params.f90` – Initializes code parameters discussed below.
- file: `init_slice.f90` – Allocates the memory for one-dimensional slices used during computation.
- file: `set_slice.f90` – Sets the solution and metric data along the one-dimensional slices on which Riemann problems are solved.

dir: /grid Contains the routines that set the coordinate grid, set the coordinate flux, and perform other operations related to the coordinates. The routines that perform parallel propagation and orthonormalization are in this directory.

- file: `set_grid.f90` – Sets the grid according to the defined coordinates. The process by which the coordinates and geometry is set is outlined in Section 7.3.
- file: `set_flux.f90` – Computes the spatially varying flux by calling `fluxfunct.f90` to compute $\boldsymbol{f}(\boldsymbol{q})$, and then multiplies that result by $\sqrt{h}$.
- file: `parallel_trans.f90` – Performs parallel transport on vector components of $\boldsymbol{q}$ from the cell center to the cell edge. This process is described in Section 7.4.
- file: `parallel_trans_inv.f90` – Performs the inverse parallel transport from the cell edge to the cell center.
- file: `ortho.f90` – Performs the orthonormalization of vector components of $\boldsymbol{q}$ at the cell edge.
- file: `plot_grid.f90` – Outputs the location of each grid cell for plotting purposes. This file can be modified to print either cell-centered values, or cell-edge values.
- file: `plot_aux.f90` – Outputs the components of the auxiliary array in each grid cell.

dir: /manifolds – Contains three subdirectories `/euclidean`, `/riemannian`, and `/numerical` described below.

- dir: /euclidean – Contains a directory for each coordinate system that covers $\mathscr{M} = \mathbb{R}^2$. For example, one subdirectory found here is `/polar` which contains the necessary files to describe the polar basis. A general directory, like `/polar`, will be denoted by `/geometry`.
  - dir: /geometry – A general directory containing the routines that set the geometry for a given basis.
    - func: `hfunct.f90` – Computes the components of the metric tensor $[h_{ij}]$.
    - func: `chr.f90` – (optional routine) Computes the Christoffel symbols Γ^i_{jk}.
    - func: `dchr.f90` – (optional routine) computes the derivatives of the Christoffel symbols $\mathcal{G}^i_{jkl} = \Gamma^i_{jk,l}$.
- dir: /riemannian – Contains a directory for each Riemannian manifold. Once again, these directories contain the functions that define h_{ij}, Γ^i_{jk} and $\mathcal{G}^i_{jkl}$.
  - dir: /geometry – A general directory containing the routines that set the geometry for a given basis.
    - func: `hfunct.f90` – Computes the components of the metric tensor $[h_{ij}]$.
    - func: `chr.f90` – (optional routine) Computes the Christoffel symbols Γ^i_{jk}.
    - func: `dchr.f90` – (optional routine) computes the derivatives of the Christoffel symbols $\mathcal{G}^i_{jkl} = \Gamma^i_{jk,l}$.

dir: $\boxed{\texttt{/numerical}}$ – The reason **chr.f90** and **dchr.f90** are optional in the user-provided directories is that they are also provided here. In this directory Γ^i_{jk} and $\mathcal{G}^i_{jkl}$ are approximated with finite differences.

func: $\boxed{\texttt{chr.f90}}$ – Computes a finite difference approximation to the Christoffel symbols Γ^i_{jk}.

func: $\boxed{\texttt{dchr.f90}}$ – Computes a finite difference approximation to the derivatives of the Christoffel symbols $\mathcal{G}^i_{jkl} = \Gamma^i_{jk,l}$.

dir: $\boxed{\texttt{/systems}}$ – Contains a directory for each system of equations that have been implemented. For example, in the distribution version there are two directories, **/advection** and **/scalarfield**. The typical system directory will be referred to as **/system**

dir: $\boxed{\texttt{/system}}$ –

dir: $\boxed{\texttt{/fluxfunct}}$ – Contains files pertaining to the coordinate flux.

file: $\boxed{\texttt{flux2.f90}}$ – A subroutine that returns the coordinate flux $\boldsymbol{f}(\boldsymbol{q})$.

dir: $\boxed{\texttt{/rp}}$ – Contains Riemann solvers.

file: $\boxed{\texttt{rpn2.f90}}$ – A subroutine that solves the normal Riemann problems and returns $\boldsymbol{\mathcal{A}}^{\pm}\boldsymbol{\Delta Q}$, and each f-wave $\boldsymbol{\mathcal{Z}}^p$. If orthonormalization is being used, this should be a Cartesian-based solver.

file: $\boxed{\texttt{rpt2.f90}}$ – A subroutine that solves the transverse Riemann problems and returns the transverse fluctuations.

dir: $\boxed{\texttt{/execution}}$ – Contains routines defined by the user.

file: $\boxed{\texttt{main.f90}}$ – The main subroutine of the code. This subroutine calls all initialization and time-stepping routines.

file: $\boxed{\texttt{user\_module.f90}}$ – Defines and stores all user-defined parameters. Any elements needed for a specific application should be defined in this module. These elements are made available to other routines by the statement `use user_module`.

file: $\boxed{\texttt{init\_user\_params.f90}}$ – A user-defined routine that reads the file **init.data** to set the elements defined and stored in the **user_module**.

file: $\boxed{\texttt{ic.f90}}$ – The subroutine that sets initial conditions.

file: $\boxed{\texttt{param.data}}$ – Input file for the initialization parameters defined in Section 7.2.1. An example file is shown in Table 7.1.

file: $\boxed{\texttt{init.data}}$ – Input file for the user-defined parameters discussed in Section 7.2.2. An example of this file is shown in Table 7.2.

dir: $\boxed{\texttt{/bc}}$ – Contains routines that set boundary conditions. Boundary conditions specific to a particular application can be kept in the **/execution** directory.

file: $\boxed{\texttt{bc2ex0.f90}}$ – A subroutine that fills the ghost cells with zero-order extrapolated data. This sets outflow boundary conditions.

Table 7.1: A typical `param.data` file used to define initialization parameters described in Section 7.2.1.

```
4, 4, 1                       ...  meqn, mwaves, mvec
3                             ...  mvec_v(1:mvec)
1, 2, 2, 0, 0, 0, 2, 2        ...  method(1:8)
4, 4, 4, 4                    ...  mthlim(1:mwaves)
2, 2, 2, 2                    ...  bnd(1:4)
1.0, 0.9                      ...  cfl_v(1:2)
0.0, 1.0, 0.0, 1.0            ...  (x10,x1f),(x20,x2f)
2, 50, 50                     ...  mbc, mx1, mx2
0                             ...  mrstrt
1.0, 10                       ...  tend, mout
999999                        ...  nstep_v(1)
0.005                         ...  dt_v(1)
1,1,2                         ...  mmetric, trans, morder
```

dir: `/limiters` – The limiting process involves the comparison of f-waves at different locations in $\mathscr{M}$. Theoretically, parallel transport should be used to make this comparison for vector components of q. However, extensive tests with exponential curvature demonstrated that this was not necessary.

file: `cm_limiter.f90` – A subroutine that performs the limiting process on f-waves.

func: `cm_philim.f90` – A function that defines the particular limiter being used.

dir: `/misc` – Contains the subroutines that perform a restart in the case that a computation is not starting from an initial condition. The routines in this directory are simply Fortran 90 versions of those found in CLAWPACK.

dir: `/matlab` – Contains Matlab routines (`.m` files) that enable plotting of the results. For more about the visualization of the output see [41].

There are also examples available in the current directory structure. For example, the two dimensional advection equation in polar coordinates, and the Riemannian surface example, both discussed in Chapter 6. With the directory structure in mind, we now discuss the necessary parameters for the code.

7.2 Code parameters

Parameters used in CLAWMAN are separated into two sets. The first set are *initialization* parameters and are necessary to initialize the constants, vectors, and arrays used by the code during the computation. For example, these parameters set the number of equations being solved, whether correction fluxes and transverse waves will be included in the update, the final time for the computation, and other parameters described in Section 7.2.1. The initialization parameters are defined in a file called `param.data` that resides in each `/execution` directory described above. The routine `init_params.f90` is the first routine called from `main.f90` and reads these parameters into the memory module set up by `init_module.f90`. Each routine that has the statement `use init_module` will have access to the elements of the initialization parameters.

Elements of the other set are called *user* parameters and are used to set application specific data. For example, any parameters needed for initial conditions or boundary conditions would be considered user parameters since they would change for each application. These parameters are defined in a file called `init.data` that also resides in each `/ execution` directory. They are read by the routine `init_user_params.f90` that is called directly after initialization parameters are read. The user parameters, and data structures that they help to define are stored in a memory module set up by `user_module.f90`. Just like the initialization parameters, these data structures are available to any routine that includes the statement `use user_module`.

7.2.1 Initialization parameters

Table 7.1 shows a typical `param.data` file used by `init_params.f90` to define the parameters and data structures needed by the code. The parameters shown in the figure are similar to those needed in the CLAWPACK software, but there are some differences that need to be discussed. The parameters needed in the CLAWMAN software are the following:

`meqn`: The number of equations in the system being solved. For the classical or relativistic Euler equations this parameter is set to `meqn` $= n_{eq} + 2$.

`mwaves`: The number of f-waves that that are used to solve Riemann problems. This is almost always set to `mwaves` = `meqn`.

`mvec`: The number of vectors in $\boldsymbol{q}$. For example, the wave equation and the Euler equations have `mvec` $= 1$.

`mvec_v`: A vector with `mvec` components. Each component of `mvec_v` defines the element in $\boldsymbol{q}$ that holds the first component of a vector. For example, the Euler equations have `mvec` $= 1$ and that vector is the momentum $\vec{s}$ with tangent components s^i. The state vector is defined to be $\boldsymbol{q} = (\rho, s^i, e)$, so that `mvec_v(1)` $= 2$. This information is used to apply parallel transport and orthonormalization operations on vector components of $\boldsymbol{q}$.

`method_v`: There are a few differences between these parameters and those in `method` for CLAWPACK. These differences are described in the following:

`method v(2)` – A parameter that determines whether the correction fluxes (**??**) and the source corrections (**??**) are included in the update.

- `method_v(2)` = 1 – No correction fluxes or source corrections are included in the update.
- `method_v(2)` = 2 – Correction fluxes, but no source corrections are included in the update.
- `method_v(2)` = 3 – Both correction fluxes and source corrections are included in the update.

`method v(3)` – A parameter that determines whether transverse propagation is included in the update.

- `method_v(3)` = 0 – No transverse information is used in the update.
- `method_v(3)` = 1 – Only first order transverse information is used in the update.
- `method_v(3)` = 2 – Correction fluxes are included in the transverse decomposition.
- `method_v(3)` = 3 – Both correction fluxes and source corrections are included in transverse decomposition.

`method_v(5)` – This parameter is defined the same as it is in CLAWPACK. It determines whether a split method will be used for a source term. In CLAWMAN, geometric sources are handled by parallel transport so that `method_v(5)` = 0, unless there is another source present.

`method_v(8)` – This is a parameter that determines how many components of the auxiliary array will be output. If `method_v(8)` $\neq 0$, then the components are output to files with names like `aux.001`.

`mthlim`: The particular limiter that will be applied to each f-wave.

`bnd` These parameters can be used to set different boundary conditions along each grid boundary.

`cfl_v`: Maximum allowed CFL, and desired CFL.

`x10`: The coordinates grid is the rectangle [`x10`, `x1f`] × [`x20`, `x2f`].

`mbc`: The number of ghost cells along each boundary.

`mx1`,`mx2`: The number of grid cells in each coordinate direction.

`mrstrt`: Set to zero to begin the computation from the initial conditions. Otherwise, the run is a restart from previously evolved data.

`tend`: Final time of the computation

`mout`: Number of times at which the code will output results.

`nstep_v`: Holds the maximum number of time steps allowed.

Table 7.2: A typical `init.data` file used to define user parameters.

```
-0.5,0.15,2.0,2.0      ...  alf,omega,x0,y0
0.2,0.15,0.11          ...  a,delta,xloc
```

`dt_v`: Holds the initial time step.

`mmetric`: If set to zero, there will be no transformation to an orthonormal frame at each cell interface. Otherwise, the transformation occurs and it is assumed the Riemann solver has no geometry associated with it.

`trans`: If set to zero, there will be no parallel propagation of vector components. In this case, the parameter `method(5)` $\neq 0$ so that the source is handled by a split method. Otherwise, parallel transport handles the geometric source.

`morder`: Determines the order at which parallel transport is carried out. Since the matrices that carry out parallel transport are only computed once at the beginning of the computation, efficiency is not significantly diminished by setting `morder` $= 2$.

7.2.2 User parameters

In Section 6.10 the method developed in this thesis was applied to the scalar field equation on a manifold defined by the surface $b\,(x^1, x^2)$. The calculations presented in that section were carried out using CLAWMAN, and in Section 7.5, the implementation of this particular system will be discussed. Recall from Section 6.10 that the surface had the form

$$b(x,y) = \alpha\ \exp -\frac{\left((x-x_0)^2 + (y-y_0)^2\right)^{1/2}}{\omega^2}, \tag{7.1}$$

and the initial condition was a right-propagating pressure pulse that had the functional form

$$\boldsymbol{q}(t=0,x,y) = \begin{cases} a\ \cos\frac{\pi\,(x-x_l)}{\delta} + 1.0) & \text{if } |x-x_l| < \delta \\ 0 & \text{if } |x-x_l| > \delta \end{cases}. \tag{7.2}$$

Initializing the user parameters for these functions is done in the file `init.data`. Table 7.2 shows the particular file that initializes the necessary parameters for this example. The parameters used are as follows:

– Surface:

`alf`: Magnitude of the Gaussian dip in the surface that defines the manifold. Denoted by α in the surface definition (7.1).

`omeg`: Denoted by ω in (7.1).

Table 7.3: A typical function `hfunct.f90` that is used to set the metric coefficients h_{11}, h_{12}, and h_{22}.

```
DOUBLE PRECISION FUNCTION hfunct(mcomp,x1,x2)
 USE user_module
 IMPLICIT NONE
 DOUBLE PRECISION x1,x2,b,bx,by
 INTEGER mcomp
 b = alf*dEXP(-((x1-x0)**2 + (x2-y0)**2)/(omega*omega))
 bx = -b*( (2.d0*(x1-x0))/(omega*omega) )
 by = -b*( (2.d0*(x2-y0))/(omega*omega) )
 go to (11,12,13) mcomp
11 CONTINUE
 hfunct = 1.d0 + bx**2
 RETURN
12 CONTINUE
 hfunct = bx*by
 RETURN
13 CONTINUE
 hfunct = 1.d0 + by**2
 RETURN
END FUNCTION hfunct
```

`x0`: x-location of the center of the Gaussian surface.

`y0`: y-location of the center of the Gaussian surface.

– Initial Conditions:

`a`: Height of the initial pressure pulse denoted by a in the initial condition (7.2).

`delta`: Denoted by δ in (7.2).

`xloc`: center of the initial pressure pulse along the x-direction. This parameter is denoted by x_l in (7.2).

Any other user-defined parameters or data structures that are needed would be added to `init.data`, stored in the module created by `user_module.f90`, and read in by `init_user_params.f90`.

7.3 Setting the geometry on $\mathscr{M}$

The geometry for a specific manifold is set in CLAWMAN by defining the metric tensor, $[h_{ij}]$, the Christoffel symbols, Γ^i_{jk}, and their derivatives $\mathcal{G}^i_{jkl}$. These quantities are used in `set_grid.f90` to define geometric quantities including the volume element $\sqrt{h}$, the matrices $\left[P^i_k\right]$ that represent discrete parallel propagation operators, and the transformation matrices $\left[O^i_k\right]$ that perform the orthonormalization at each cell edge. The components h_{ij}, Γ^i_{jk}, and $\mathcal{G}^i_{jkl}$ are set in the functions `hfunct.f90`, `chr.f90`, and `dchr.f90`, respectively. These functions should be placed in the directory denoted by `/geometry` in Section 7.1. Note that the definition of the geometry for a particular manifold is kept separate from application directories so that one manifold can be used for multiple systems without having to redefine the geometry.

A typical file defining the function `hfunct.f90` is shown in Table 7.3. In this file, components are computed for the two-dimensional metric that describes the surface defined by equation (7.1). This file was used for the implementation of the examples in Section 6.10. The function in Table 7.3 only specifies three of the four components for $\overset{\leftrightarrow}{h}$. This is due to the fact that symmetry of the metric tensor requires that $h_{21} = h_{12}$.

The file `chr.f90` defines the functional form of the Christoffel symbols Γ^i_{jk}. The indices that label each symbol can take on values from 1 to n_d, resulting in a total of n_d^3 Christoffel symbols. However, not all n_d^3 symbols are independent. Using equation (5.61) which defines Christoffel symbols in terms of the metric components, together with the symmetry of the metric, we see that $\Gamma^i_{jk} = \Gamma^i_{kj}$. This symmetry reduces the number of Christoffel symbols that must be set in `chr.f90` to $n_d^2(n_d+1)/2$. In two dimensions, this is a reduction from eight to six, but for three dimensions, the reduction is from twenty seven symbols to sixteen.

The derivatives of the Christoffel symbols are denoted by $\mathcal{G}^i_{jkl}$ and are set in the function defined by `dchr.f90`. Notice that there are n_d^4 possible components, but the symmetry of the Christoffel symbols reduces the number of independent components to $n_d^3(n_d+1)/2$. A further reduction in the number of needed components comes from the fact that parallel transport only occurs along coordinate lines. The necessary components are shown in equation (6.60), where the last two indices of $\mathcal{G}^i_{jkl}$ are always the same. The resulting number of independent components that need to be set in `dchr.f90` is $n_d^2(n_d^2 - n_d + 2)/2$. In two dimensions this reduction is from sixteen to eight components, but in three dimensions it is a reduction from eighty one to thirty six components.

A feature of CLAWMAN that was mentioned in Section 7.1, but should be mentioned again is the fact that `chr.f90` and `dchr.f90` are optional routines. Since Γ^i_{jk} and $\mathcal{G}^i_{jkl}$ can be defined in terms of the derivatives of metric components, the user-defined function in `hfunct.f90` can be used to generate finite difference approximations for Christoffel symbols and their derivatives. Such finite differences have been implemented in `chr.f90` and `dchr.f90` that reside in the `/numerical` directory described in Section 7.1. These routines can be used in place of user-defined functions discussed above. This is a useful feature for metrics with complicated components.

7.4 Solving the Riemann problems

As we saw in Chapter 6, the algorithm solves Riemann problems by incorporating geometric source terms using parallel transport. The algorithm also orthonormalizes tangent components at each cell interface where the Riemann problems are solved. Parallel transport

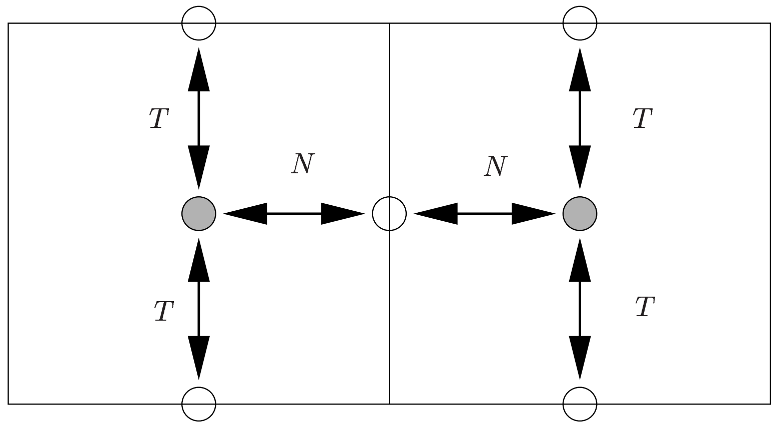

Figure 7.1: Parallel transport to both the edge at which the normal Riemann problem is solved, and the four edges at which transverse Riemann problems are solved.

and orthonormalization are carried out by the matrices `pmat` and `omat`, respectively. These matrices are initialized in `setgrid.f90`, and stored in the `grid_module`. The routines that utilize these matrices to perform the parallel transport, and orthonormalization are called `parallel_trans.f90` and `ortho.f90`, respectively. Figure 7.1 shows the process by which both normal and transverse Riemann problems are solved. The transport paths taken by vector components in a normal Riemann problem are denoted with an "N", and the paths taken for a transverse Riemann problem are denoted with a "T".

7.4.1 Normal Riemann problems

The paths denoted with an "N" in Figure 7.1 depict the process of solving a normal Riemann problem. These steps are listed in the following:

1. Use `parallel_trans.f90` to transport both the left and right cell-centered fluxes to the cell edge. This sets up the modified fluxes (6.62) and (6.63) from Chapter 6.

2. Use `ortho.f90` to orthonormalize tangent components in the modified fluxes.

3. Solve an orthonormal (e.g. Cartesian) Riemann problem that generates fluctuations based on f-waves.

4. Use `ortho.f90` to transform the vector components of fluctuations and f-waves to the coordinate basis at the cell edge.

5. Use `parallel_trans_inv.f90` to transport both the fluctuations and the f-waves to cell centers where they are used in the update (6.72).

Once the normal Riemann problems have been solved, the cell centered fluctuations are used in transverse Riemann problems to propagate information in a multidimensional way.

7.4.2 Transverse Riemann problems

Figure 7.1 also has paths denoted with a "T". These paths depict how vector components of fluctuations are transported to solve transverse Riemann problems. The figure shows that both the left-going and right-going fluctuations are used to compute transverse information at the four contiguous interfaces in the transverse direction. The process by which transverse Riemann problems are solved at one of these interfaces is the following:

1. Use `parallel_trans.f90` to transport the cell-centered fluctuation returned from the normal Riemann problem to the interface.

2. Use `ortho.f90` to orthonormalize tangent components in the modified fluctuations.

3. Solve an orthonormal (e.g. Cartesian) transverse Riemann problem based on this modified data.

4. Use `ortho.f90` to transform tangent components of the transverse fluctuations back to the coordinate basis.

5. Finally, Use `parallel_trans_inv.f90` to transport the transverse fluctuations back to the cell center where they are used to modify the correction fluxes.

The parallel transport of fluctuations returned by normal Riemann solvers are expensive. The process described above occurs four times for each normal Riemann problem. Tests have shown that, though it makes sense to implement parallel propagation for these fluctuations, it is not necessary. Numerical results are only slightly improved by the use of parallel transport of the transverse terms.

7.5 Environment variables

In order to compile and run the examples provided with CLAWMAN, environment variables must be set to make sure that the appropriate files can be found. The necessary environment variables include:

`$INSTALL`: Points to the installation directory denoted by `/install`.

`$CLAWMAN2d`: Points to `$INSTALL/clawman/2d`.

`$EUCLIDEAN`: Points to `$CLAWMAN2d/manifolds/euclidean`.

`$RIEMANNIAN`: Points to `$CLAWMAN2d/manifolds/riemannian`.

`$NUMERICAL`: Points to `$CLAWMAN2d/manifolds/numerical`.

`$SYSTEMS`: Points to `$CLAWMAN2d/systems`.

These environment variables are used in `compile` files in the application's main directory denoted above by `/execution`. Once the proper environment variables are set, the code will compile to the executable file `a.out`. The code can be run with the command

```
>a.out.
```

Input parameters are read from both `param.data` and `init.data` in the same directory.