CentPack version 1.0
User's Guide

Jorge Balbás
University of Michigan, Ann Arbor

Eitan Tadmor
University of Maryland, College Park

July 19, 2006

http://www.cscamm.umd.edu/centpack/

Contents

License and Limitations

Acknowledgements

The authors would like to thank Chris Anderson (UCLA Mathematics) for contributing the original source code for the class of multidimensional arrays used in this package, and Nick Kisseberth (former CSCAMM Computer System Administrator) for his help in organazing and simplifying the source code and its installation process.

Feedback and Citing CentPack

If you successfully use CentPack, please:

1. Let us know: send us email at centpack@cscamm.umd.edu with a summary of your use of CentPack. Your contribution may be useful for us and others working on hyperbolic conservation laws and related problems.
2. Let others know: cite its authors and the package in any published work for which you used CentPack. You can use this BibTeX entry for citing CentPack:

```
@misc {CentPack,
    AUTHOR = {Balb{\'a}s, Jorge and Tadmor, Eitan},
     TITLE = {{\sc CentPack}},
HOWPUBLISHED={online at http://www.cscamm.umd.edu/centpack/software},
     MONTH = {July},
      YEAR = {2006},
      NOTE = {Available at: http://www.cscamm.umd.edu/centpack/software},
}
```

1 Introduction

1.1 What is CentPack?

CentPack is a collection of freely distributed `C++` libraries that implement several *black-box* type central schemes for one- and two-dimensional hyperbolic conservation laws,

$$u_t + f(u)_x = 0 \qquad \text{and} \qquad u_t + f(u)_x + g(u)_y = 0, \tag{1.1}$$

respectively.

Only a minimal number of auxiliary functions containing model and problem specific information (i.e., fluxes, initial and boundary conditions, output variables, etc.) are required from the user (examples are provided), and few (if any) modifications of the core source files of CentPack should be needed to solve most problems that accept the above formulations, (1.1), the source code can be easily modified and adapted to the taste and requirements of its user.

CentPack is part of a larger effort, CentralStation, dedicated to the development and application of central schemes to a wide range of scientific problems. For more information about central schemes and their implementation visit

http://www.cscamm.umd.edu/centpack/

1.2 Download

CentPack is freely distributed, precompiled binaries for some operating systems are available for download at:

http://www.cscamm.umd.edu/centpack/software/

A full source code distribution is also available for registered users. Both distributions –precompiled binaries and full source code alike– include additional source files that allow users to compile and run sample applications.

2 Software

2.1 CentPack `C++` Libraries

The core of this software consists of seven `C++` libraries and their corresponding header files. The first one, libarray.a, implement the class of multidimensional dynamic data structures (arrays) used to store the solution variables. The other six libraries, `libcentpack_Xd_YDZ.a`, implement different versions of non-oscillatory central schemes according to the following notation

`X` = number of space dimensions: `X = 1 or 2`,

`YD` = formulation: `Y = F` for Fully-Discrete or `Y = S` for Semi-Discrete, and

`Z` = order of accuracy of the scheme: 2 for 2nd order, and 3 for 3rd.

For example, the library `libcentpack_1d_SD2.a` implements a one-dimensional, semi-discrete, 2nd order central solver.

Remarks:

1. The source code distribution contains the source and make files needed to create this libraries, not the libraries themselves. For a detailed description of each subroutine, please refer to the `.cc` file describing the subroutine.
2. The number of source files compiled under each library depends on the dimensions, formulation, and order of accuracy of the specific central scheme implemented by that library. For a list of the functions in each library, please refer to appendix B.

3. The fully-discrete solvers provided in this package are only 2nd order.
4. This package (or previous test versions of it) has been compiled, tested and validated in a variety of UNIX / UNIX-like platforms (e.g., Solaris, Linux, IBM SP/2, Mac OS/X) for a variety of test problems corresponding to different models (e.g., Euler equations of gas dynamics and Ideal MHD equations). The compilation and execution instructions provided below and the included Makefiles are linux-based (GNU's `g++` compiler), but they should work on other platforms with minor or no modifications.

2.2 Auxiliary Files – User Specified, Examples Provided

To construct a CentPack solver, users need to specify a number of subroutines to be compiled and linked to the libraries `libcentpack_Xd_YDZ.a`, and an `input` file with some parameters to define the solution domain, computational mesh, simulation time, etc. Several examples defining different hyperbolic models and corresponding input parameters are provided with all CentPack packages. Although these subroutines need not to be defined in separate files, and the user may choose to define them all in a single file, we find more convenient to do it in separate files. Following this source code structure, below, we provide a description of these `C++` source files (we use the name of the `.cc` file defining the function, not the function header).

2.2.1 Model Specific Files

1. `fluxx.cc` – defines the function $f(u)$ in equation (1.1).
2. `fluxy.cc` – defines the function $g(u)$ in equation (1.1). THIS FUNCTION IS ONLY REQUIRED FOR 2d SOLVERS!
3. `spectral_radii.cc` – estimates the maximum speed of propagation of the components of u in each space dimension. FOR 1d SOLVERS THE NAME OF THIS FUNCTION IS `spectral_radius.cc`.
4. `writeout.cc` – defines the output variables as one- or two-dimensional arrays and a rule to extract them from the multi-dimensional array holding the solution u of equation (1.1), the variables are extracted and written to files adequately numbered for further manipulation and analysis of results (e.g., plotting, error analysis, etc.).

2.2.2 Problem Specific Files

1. `main_Xd_YDZ.cc` – we use this file to call CentPack's main routine, `centpack_main_Xd_YDZ()`. However, the user may choose to make the call to `centpack_main_Xd_YDZ()` from any `C++` main should he/she be interested in evolving the solution of a hyperbolic PDE only as part of a larger problem.
2. `initial_conditions.cc` – defines a rule to define the solution mesh and initialize the array holding the solution u. The function is called by CentPack's main routine before initiating the evolution.
3. `boundary_conditions.cc` – boundary conditions are implemented by applying them to four "ghost" rows and columns (two around each boundary of the solution domain) in the solution array u. This function provides with a rule to fill those rows/columns according to the problem boundary conditions. This allows for the computation of the solution over the computational domain to be carried out without having to alter CentPack's core files when changing the boundary conditions of the problem.
4. `input` – a short file containing the required input required by CentPack to compute the solution (i.e., number of grid cells along each dimension, CFL restriction, final time of simulation, and other model/problem specific information); the value for each input variable is written on a separate line of the file in the following order (those with additional information in parenthesis, should be omitted in input files for other solvers):

`x_init`	–	left end-point of domain in x-direction
`x_final`	–	right end-point of domain in x-direction
`y_init`	–	left end-point of domain in y-direction (2d solvers only)
`y_final`	–	right end-point of domain in y-direction (2d solvers only)
`J`	–	number of grid cells in x-direction
`K`	–	number of grid cells in y-direction (2d solvers only)
`L`	–	number of components of u
`gamma`	–	parameter specifying constant ratio of specific heats
`t_final`	–	final time of simulation
`dt_out`	–	time interval at which intermediate output is desired
`cfl`	–	time step restriction
`alpha`	–	parameter for second order reconstruction, $1 \leq$ `alpha` ≤ 2 (still required as input for 3rd order solvers, value in that case is irrelevant)
`B1`	–	free parameter (from MHD magnetic field, only required for 1d solvers)

2.3 Header Files

For each of the six CentPack libraries, `libcentpack_Xd_YDZ.a`, distributed in this package, there is a corresponding header file, `centpack_Xd_YDZ.h`, containing the prototypes of each of the core routines needed to compile the library, and the prototypes of the auxiliary routines needed to build a central solver with that library. These header files are in the directory `CP-1.0/include/`.

A WORD OF CAUTION: while these header files may help the user understand the structure of the code and how to modify it to best fit his/her needs, we recommend not to modify the functions' headers unless: (*i*) it is required to make CentPack work for a different type of problem (i.e., non-homogeneous hyperbolic models, incorporating CentPack into a solver for a larger problem, etc.), and (*ii*) the user knows what he/she is doing.

2.4 Makefiles

CentPack is distributed with a number of Makefiles that allow the user to compile and install the libraries described above as well as several examples. When executed, and according to the additional passed with the `make` command, the Makefile at the top CentPack directory, calls recursively to other make files in the `src/` and `src/Xd/samples/` (`X = 1,2`) subdirectories. These Makefiles assume `g++` as the `C++` compiler, if you wish to compile CentPack with a different `C++` compiler, makes sure you replace `g++` for your compiler name wherever it appears in the Makefiles. The Makefiles files can be easily modified in order for users to compile their own central solvers. For a complete list of the options to pass with the `make` command, see appendix C.

2.5 Multi-dimensional Arrays

CentPack uses its own class of multidimensional arrays to store and access the solution (and other intermediate and associated) variables. The array indexing starts at zero, i.e., A(0) stores the first element of the one-dimensional array A, and A(k) its (k+1)st element. The arrays class do not define any vector and or matrix arithmetic operations, CentPack does not require them, but the user can certainly add those if needed with relative ease. To add such capabilities, first define the prototype of the function(s) implementing the arithmetic operation(s) to the file `CP-1.0/include/doublearrayXd.h` (`X =1,2,3`), then add the function to the corresponding `CP-1.0/src/common/doublearrayXd.cc`.

2.6 Installation

2.6.1 Installing CentPack Libraries – Source Code Distribution Only

Remarks and Notation:

1. The following instructions assume that the user specified files have been created (or the provided examples are used).
2. The `>` symbol below represents the command line prompt of the command shell.

If you downloaded pre-compiled binaries, you can skip this section. If you downloaded CentPack's source code:

1. Move or copy the downloaded archive CP-1-0.tar.gz to your home directory (or anywhere else in your hard drive where you have write permission), and cd there

   ```
   > cp CP-1-0.tar.gz $HOME/CP-1-0.tar.gz
   > cd $HOME
   ```

2. Extract the contents of the archive (a directory called `CP-1.0`) and `cd` to `CP-1.0`

   ```
   > tar -zxf CP-1-0.tar.gz
   > cd CP-1.0
   ```

3. To compile and install CentPack libraries, type

   ```
   > make libs
   ```

   This will create seven libraries in the subdirectory `CP-1.0/lib/`, one with the definition of the array classes (data structures) used by CentPack, and six with the implementation of each one of the central solvers implemented by this version of CentPack. You can verify this by typing

   ```
   > ls lib/
   ```

   If everything went well, you should see a file called `liabarray.a`, and six other that follow the notation `libcentpack_Xd_YDZ.a`, with `X, Y`, and `Z`, taking on the values specified above.

2.6.2 How to Compile ALL examples

To compile all the examples distributed with CentPack, make sure you are in CP-1.0, then type

```
> make samples
```

This will create create a new directory, `CP-1.0/samples/`, containing six subdirectories consisting of the name of the example, followed by a key of the form `_Xd_YDZ` denoting the central solver used to compile the example as specified above (i.e., `burguers_1d_SD3`, `euler_2d_FD2`, etc.) These directories contain everything needed to run the examples.

2.6.3 How to compile a single example

Alternatively, individual examples can be compiled by typing

```
> make NAME_OF_EXAMPLE
```

with `NAME_OF_EXAMPLE` replaced by one of the following:

`burgers_1d_SD3`, `euler_1d_SD2`, `MHD_1d_FD2`, `euler_2d_FD2`, `MHD_2d_SD2`, and `scalar_2d_SD3`.

For example, the command

```
> make burgers_1d_SD3
```

will create the directory `CP-1.0/samples/burgers_1d_SD3/` with all the files needed to solve Burgers' equation and directories to write the output to.

Remark: if you chose to compile a single example, the command '`> make samples`' won't do anything after that example has been created. To compile all examples after one has already been created, you will have to do it one by one, or first get rid of the directory `CP-1.0/samples/`, i.e., make sure you are at `CP-1.0/`, then type '`> rm -r samples`'. **CAUTION**: be careful not to delete the directories `CP-1.0/src/Xd/samples/`

2.7 What Can you Do with CentPack?

CentPack is highly adaptable, it provides users with a robust collection of central solvers ready to use for nonlinear hyperbolic conservation laws, (1.1). In addition, the full source distribution, provides registered users with access to the core routines that implement the schemes and the possibility of changing them to handle other hyperbolic models (e.g., models with source terms, viscosity terms, etc.) and/or incorporate CentPack to solvers of models containing that include hyperbolic equations as part of a larger model (e.g., collisionless plasmas).

Before modifying the code, however, we strongly recommend users to experiment with CentPack's original distribution, the examples included with it, and new examples that can be written in the form (1.1). The following instructions will guide users through such process.

2.7.1 Running a CentPack Example

Running one of CentPack's examples (assuming `burgers_1d_SD3` as the example to be run):

1. After compiling the example(s), go into the sample directory you wish to run. Make sure you are in `CP-1.0`, then type

   ```
   > cd samples/burgers_1d_SD3/
   ```

   Alternatively, if you compiled other examples, you can browse `CP-1.0/samples/` and decide which example you want to run, then move to its simulation directory, for example

   ```
   > ls samples/ > cd samples/euler_1d_SD2/
   ```

2. Once in the directory of the example you wish to run, type its name, in our example,

   ```
   > ./burgers_1d_SD3
   ```

   Note that each simulation directory contains only one executable whose name is quite obvious (i.e., `burgers_1d_SD3`, `euler_2d_FD2`, `MHD_2d_SD2`, etc.), the only other contents are the file named `input`, and few directories ending in `_files/`, where CentPack's output will be written.

   This command will start running a CentPack's solver. CentPack will read the data in the `input` file, set the initial conditions, and evolve them according to the hyperbolic conservation law, equation (1.1). After each time iteration, CentPack will write a line to the screen that will help monitor the progress of the simulation. This line contains the following information:

`run`	–	present simulation time
`dt`	–	time step used in the last time iteration
`dt_cpu`	–	CPU time employed in last time iteration
`t`	–	total CPU time thus far
`odd/even`	–	(fully-discrete schemes only) indicates whether the last time step was an even or odd one

   In addition to the monitor line, the solver will check whether it is time to write an output (as specified by the parameter `dt_out` specified in the file `input`) and if that is the case, write it to the output directoties.

   Alternatively, you can run CentPack's example by typing

   ```
   > ./burgers_1d_SD3 > monitor &
   ```

   This will re-direct the monitoring information to the file monitor.

   Depending on the user privileges your machine grants you, the performance of the solver can be improved by typing

   ```
   > nice ./burgers_1d_SD3 > monitor &
   ```

When the simulation is completed, in addition to the output files, a file named `run_info.txt` containing some simulation information, will be generated. The information written to this file is minimal, i.e., size of last time step used, total CPU time, mesh size, and CFL number. This information may be useful to remind the user what parameters he or she used to run a simulation anytime after it was done (e.g., when writing a paper containing thosee results).

2.7.2 Create and Run a New CentPack Example

This section describes how to create a new CentPack solver for a problem that can be expressed in the form (1.1). The steps outlined below are, in our opinion, the fastest and easiest way for creating such solvers, however, users may prefer to do it differently.

1. Go to the directory containing the existing CentPack examples:

   ```
   > cd $HOME/CP-1.0/src/Xd/samples
   ```

   where `X=1,2` denotes the space dimensions.

2. Create a new directory to place the new auxiliary files:

   ```
   > mkdir new_example
   ```

3. Copy the files of any existing example (with the same number of space dimensions) to the newly created directory, assuming `MHD_OT` is the example we copy from:

   ```
   > cp MHD_OT/*.cc new_example
   > cp MHD_OT/Makefile new_exampe/
   > cp MHD_OT/input new_example
   ```

4. Edit the files `flux_x.cc` and `flux_y.cc` you just copied so that they define the fluxes $f(u)$ and $g(u)$ as for your model, the file `spectral_radii.cc` so that it returns (an estimate) of the largest eigen values of the Jacobians $\partial f/\partial u$ and $\partial g/\partial u$, and the file `writeout.cc` so that it outputs the variables you are interested on.

5. Two lines of the file `Makefile` also need to be modified: in the first line, set `APPNAME` to the name of your application, and in the line `DIRS`, change the names of the output directories so that the function `writeout.cc` can find them and write the output files to them.

6. Move back to CentPack's root directory, `$HOME/CP-1.0/`, and add the following two lines to the file `$HOME/CP-1.0/Makefile`:

   ```
   name_of_new_example:
         (cd src/Xd/samples/new_example; make samples)
   ```

   where `X` = 1 or 2, and `new_example` is the name of the directory you created to place the auxiliary files. MAKE SURE THE SECOND LINE, THE ONE IN PARENTHESIS, IS TABBED.

7. Now, you can build the new example by typing, from CentPack's root directory, `$HOME/CP-1.0/`,

   ```
   > make name_of_new_example
   ```

   Now you can run the example as explained above.

2.7.3 Modifying CentPack Libraries

While there is little guidance we can provide as to what modifications need to be done to make CentPack work for particular examples, we list here some good practices to adopt when making modifications to the core files, namely:

1. If you use CentPack's regular distribution to solve hyperbolic conservation laws, (**??**), without any additional source terms, constraints, or additional non-hyperbolic equations, leave that installation intact and download a new copy of CentPack's source code (possibly renaming the original copy so no conflicts arise).
2. It is not a bad idea to start by modifying the core files of a second-order scheme, those in the directories `CP-1.0/src/Xd/YD2` (`X =` 1 or 2, `Y = F` or `S`), the number of core files is smaller.
3. Make sure that whatever modifications you make of the source files are reflected in the function prototypes of the corresponding header file `CP-1.0/include/libcentpack_Xd_YDZ`
4. If new core files are added to the solver, those should be included either in the file `CP-1.0/src/Makefile`, or in a new one you can write to compile only the new library plus the array library.

3 CentPack's Output

The parameter `dt_out`, provided in the file `input`, will determine at what approximate time interval output will be produced during the simulation. CentPack's output is produced by the function `writeout.cc` (several examples of this function are provided), which specifies the names of the variables to be output and a rule to extract and/or calculate them from the array holding the solution. Note that this output variables are not necessarily the ones evolved by CentPack, e.g., when solving Euler's equations of gas dynamics, one may want to output the pressure and not the conserved energy.

In our example, the output produced by CentPack is written to two subdirectories of `burgers_1d_SD3`: `u_files/` and `t_files/`. The first contains files with the numerical solution of equation (1.1), `u_0`, `u_1`, `u_2`, etc. (with `u_0` corresponding to the initial conditions). The directory `t_files/` contains the files `t_0`, `t_1`, `t_2`, etc., each holding a single value indicating the simulation time at which the corresponding `u_n` file was written, `n = 0,1,2, ...`

The output files can be easily loaded into MATLAB © or similar software applications for scientific computing (e.g., Scilab © or Octave ©), plotted, and manipulated with those applications for further analysis of the results.

The output function does not write the mesh coordinates, x and y. These are not time dependent (i.e., there is no need to output them every single time the output function is called) and can be easily generated by any of the applications mentioned above.

The animations in the various examples we present at CentPack's website have been generated in three steps:

1. A MATLAB © script is used to upload the solution files one by one, plot the corresponding frame, and output it in some graphics format (the format `.png` in this case).
2. Using the `mogrify` command of the graphics suit ImageMagik © (included in most UNIX distributions), the files are converted into `.gif` format.
3. The command `convert` with the option `-adjoin` if ImageMagik © is used to sequence the `.gif` frames to build an animated `.gif` file.

We plan to provide sample scripts and more detailed instructions ;to generate animations as time allows.

A Central Schemes

In this section we provide a brief description of the nonoscillatory central schemes implemented in this version of CentPack. For further information, we refer the user to the references listed below, and to the repository of publications on central schemes available at CentPack's website:

http://www.cscamm.umd.edu/centpack/publications/

A.1 One-dimensional Central Schemes

The solutions of equation (1.1) are characterized by the onset and propagation of discontinuities as time evolves. *High-resolution* central schemes based on the *staggered* evolution of the cell averages of u, over *small* grid cells of size Δx,

$$\bar{u}(x,t) := \frac{1}{\Delta x}\int_{x-\Delta x/2}^{x+\Delta x/2} u(\xi,t)\,d\xi, \tag{A.1}$$

avoid any detailed information of the *eigen* structure of the Jacobian matrices of $f(u)$ and $g(u)$, providing a simple and efficient approach to accurately locate and resolve these discontinuities.

A.1.1 Fully-discrete Second-order Schemes

In [7], Nessyahu-Tadmor introduced a second-order extension (NT) of the first-order Lax-Friedrichs (LxF) scheme, [4]. The resulting family of high-resolution schemes reduces the numerical viscosity of LxF while increasing the accuracy of the computed solution. The main two ingredients of this scheme are:

1. a non-oscillatory piecewise polynomial reconstruction of point values from their cell averages; followed by
2. staggered evolution of the reconstructed cell averages – implemented with simple quadrature formulae (e.g., midpoint, trapezoidal, and Simpson's rules).

Starting with the cell averages $\{\bar{u}_j^n\}$ at time t^n, the fully-discrete NT scheme realizes the new staggered cell averages, $\{\bar{u}_{j+\frac{1}{2}}^{n+1}\}$, at time $t^{n+1} = t^n + \Delta t$, with the following *predictor-corrector* formulation (see figure A.1.1(a)):

$$u_j^{n+\frac{1}{2}} \;=\; \bar{u}_j^n \;-\; \lambda f_j', \tag{A.2}$$

$$\bar{u}_{j+\frac{1}{2}}^{n+1} \;=\; \frac{1}{2}\,(\bar{u}_j^n + \bar{u}_{j+1}^n) \;+\; \frac{1}{8}(u_j' - u_{j+1}') \;-\; \lambda\left[f\big(u_j^{n+\frac{1}{2}}\big) + f\big(u_{j+1}^{n+\frac{1}{2}}\big)\right]. \tag{A.3}$$

The time step Δt is determined according to the CFL limitation,

$$\frac{\Delta t}{\Delta x}\,\max\left\{\rho\Big(\frac{\partial f}{\partial u}\Big)\right\} \le \frac{1}{2}, \tag{A.4}$$

where λ stands for the mesh ratio $\Delta t/\Delta x$, and f_j' and u_j' represent a first-order approximations of the spatial derivatives of f and u respectively. In particular, CentPack approximates these derivatives using a MinMod limiter, [5]. That is,

$$u_j' = \mathrm{MinMod}\Big(\alpha\,(\bar{u}_j^n - \bar{u}_{j-1}^n), \frac{1}{2}\,(\bar{u}_{j+1}^n - \bar{u}_{j-1}^n), \alpha\,(\bar{u}_{j+1}^n - \bar{u}_j^n)\Big), \quad 1 \le \alpha < 2, \tag{A.5}$$

and similarly for f, where

$$\mathrm{MinMod}(x_1, x_2, x_3, \dots) = \begin{cases} \min_j(x_j) & \text{if } \; x_j > 0 \;\; \forall j \\ \max_j(x_j) & \text{if } \; x_j < 0 \;\; \forall j \\ 0 & \text{otherwise;} \end{cases} \tag{A.6}$$

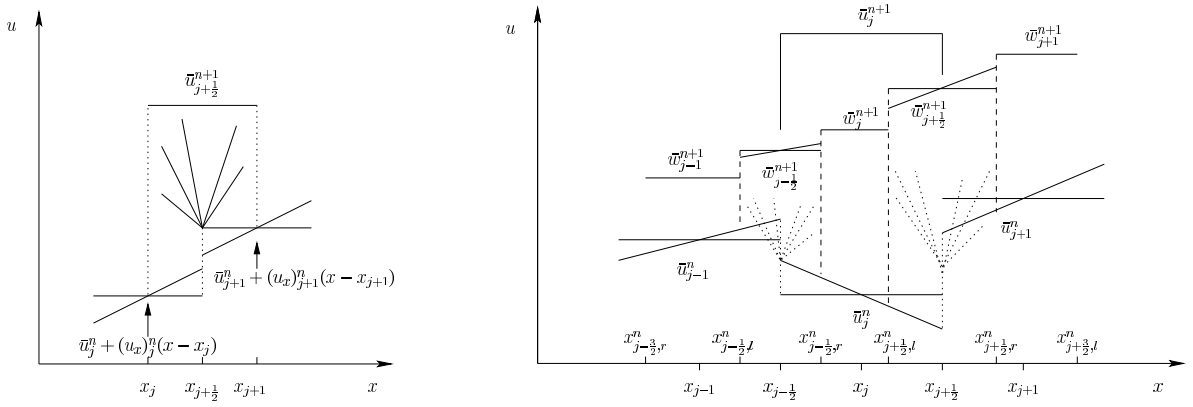

Figure 1: One-dimensional central schemes: (a) staggered evolution, (b) modified central differencing

A.1.2 Semi-discrete Second- and Third-order Schemes

The fully-discrete NT scheme does not enjoy an analogous semi-discrete formulation. A modification introduced by Kurganov and Tadmor in [3], in which the information provided by the maximum local speed of propagation, $a_{j\pm\frac{1}{2}}(t)$, allows us to distinguish between the regions where the solution remains smooth, and those reached by the propagating discontinuities (see figure A.1.1(b)). This modification, leads to the more vesatile, non-staggered, semi-discrete formulation

$$\frac{d}{dt}\bar{u}_j(t) = \lim_{\Delta t\to 0}\frac{\bar{u}^{n+1}_j - \bar{u}^n_j}{\Delta t} = -\frac{H_{j+\frac{1}{2}}(t) - H_{j-\frac{1}{2}}(t)}{\Delta x}, \tag{A.7}$$

where

$$H_{j\pm\frac{1}{2}}(t) = \frac{f\big(u^+_{j\pm\frac{1}{2}}(t)\big) + f\big(u^-_{j\pm\frac{1}{2}}(t)\big)}{2} - \frac{a_{j\pm\frac{1}{2}}(t)}{2}\,(u^+_{j\pm\frac{1}{2}}(t) - u^-_{j\pm\frac{1}{2}}(t)). \tag{A.8}$$

Here, the interface point values, $u^+_{j+\frac{1}{2}}$ and $u^-_{j+\frac{1}{2}}$, are recovered by some piecewise non-oscillatory polynomial reconstruction, $\{p_j(x)\}_j$,

$$u^-_{j+\frac{1}{2}} := p_j(x_{j+\frac{1}{2}}) \qquad \text{and} \qquad u^+_{j+\frac{1}{2}} := p_{j+1}(x_{j+\frac{1}{2}}), \tag{A.9}$$

and the evolution dictated by (A.7) can be carried out by the ODE solver of our choice (e.g., Runge-Kutta, Adams-Bashford, etc.)

CentPack libraries include second- and third-order semi-discrete solvers. In the case of the third-order scheme, the reconstruction is implemented by the third-order CWENO reconstruction, [2,6]

$$p_j(x) \;=\; w_{\rm L}P_{\rm L}(x) \;+\; w_{\rm C}P_{\rm C}(x) \;+\; w_{\rm R}P_{\rm R}(x), \tag{A.10}$$

where $P_{\rm L}(x)$, and $P_{\rm R}(x)$ are linear polynomials formed with the pair of cell averages $\bar{u}^n_{j-1}$ and $\bar{u}^n_j$, and $\bar{u}^n_j$ and $\bar{u}^n_{j+1}$ respectively, $P_{\rm C}(x)$ a parabola formed with $\bar{u}^n_{j-1}$, $\bar{u}^n_j$, and $\bar{u}^n_{j+1}$, and the nonlinear weights, $w_{\rm L}$, $w_{\rm C}$, and $w_{\rm R}$ are calculated so as to guarantee the non-oscillatory behavior of the reconstruction. And the third-order SSP Runge-Kutta solver, [1]

$$\begin{aligned} u^{(1)} &= u^{(0)} + \Delta t\, C[u^{(0)}], \\ u^{(2)} &= u^{(1)} + \frac{\Delta t}{4}\,(-3C[u^{(0)}] + C[u^{(1)}]), \\ u^{n+1} &:= u^{(3)} = u^{(2)} + \frac{\Delta t}{12}\,(-C[u^{(0)}] - C[u^{(1)}] + 8\; C[u^{(2)}]), \end{aligned} \tag{A.11}$$

is used for the evolution step, where

$$C[w(t)] = -\frac{H_{j+\frac{1}{2}}(w(t)) - H_{j-\frac{1}{2}}(w(t))}{\Delta x}. \tag{A.12}$$

CentPack's second-order semi-discrete solver, calculates the interface values $u^{\pm}_{j+\frac{1}{2}}$ via the linear reconstruction $p_j(x) = \bar{u}^n_j + u'_j \frac{x-x_j}{\Delta x}$, where u' is calculated as in (A.5), and evolves (A.7) using

$$\begin{aligned} u^{(1)} &= u^{(0)} + \Delta t C[u^{(0)}] \\ u^{(1)} &= u^{(0)} + \frac{\Delta t}{2}(C[u^{(1)}] - C[u^{(0)}]) \end{aligned} \tag{A.13}$$

A.2 Two-dimensional Central Schemes

A.2.1 Fully-discrete Second-order Schemes

A.2.2 Semi-discrete Second- and Third-order Schemes

B List of Files within Each CentPack Library

C Makefile Options

References

[1] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor, *Strong stability-preserving high-order time discretization methods*, SIAM Rev. **43** (2001), no. 1, 89–112 (electronic). MR **2002f:**65132

[2] Alexander Kurganov and Doron Levy, *A third-order semidiscrete central scheme for conservation laws and convection-diffusion equations*, SIAM J. Sci. Comput. **22** (2000), no. 4, 1461–1488 (electronic). MR **2001j:**65127

[3] Alexander Kurganov and Eitan Tadmor, *New high-resolution central schemes for nonlinear conservation laws and convection-diffusion equations*, J. Comput. Phys. **160** (2000), no. 1, 241–282. MR **2001d:**65135

[4] Peter D. Lax, *Weak solutions of nonlinear hyperbolic equations and their numerical computation*, Comm. Pure Appl. Math. **7** (1954), 159–193. MR 16,524g

[5] Bram van Leer, *Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method [J. Comput. Phys.* **32** *(1979), no. 1, 101–136]*, J. Comput. Phys. **135** (1997), no. 2, 227–248. MR 1 486 274

[6] Doron Levy, Gabriella Puppo, and Giovanni Russo, *Central WENO schemes for hyperbolic systems of conservation laws*, M2AN Math. Model. Numer. Anal. **33** (1999), no. 3, 547–571. MR **2000f:**65079

[7] Haim Nessyahu and Eitan Tadmor, *Nonoscillatory central differencing for hyperbolic conservation laws*, J. Comput. Phys. **87** (1990), no. 2, 408–463. MR **91i:**65157