ISSS-10: Tutorial 2D MHD code

Jürgen Dreher
Ruhr-Universität Bochum, Germany

Contents:

1. Code description
2. Exercises
3. How to...

Code description

Overview

The code integrates hyperbolic conservation laws of type

$$\partial_t \mathbf{u} = -\partial_x \mathbf{F}_x(\mathbf{u}) - \partial_y \mathbf{F}_y(\mathbf{u}) \tag{1}$$

For the 2-dim MHD equations, the conserved variables and flux components are, in dimensionless form,

$$\mathbf{u} = (\rho,\ \rho v_x,\ \rho v_y,\ B_x,\ B_y,\ e)^T\ ,$$

$$\mathbf{F}_x = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + p + p_m - B_x^2 \\ \rho v_x v_y - B_x B_y \\ 0 \\ v_x B_y - B_x v_y \\ (e + p + p_m)\, v_x - B_x (v_x B_x + v_y B_y) \end{pmatrix}$$

$$\mathbf{F}_y = \begin{pmatrix} \rho v_y \\ \rho v_y v_x - B_y B_x \\ \rho v_y^2 + p + p_m - B_y^2 \\ v_y B_x - B_y v_x \\ 0 \\ (e + p + p_m)\, v_y - B_y (v_x B_x + v_y B_y) \end{pmatrix}$$

with magnetic pressure $p_m := \frac{1}{2}(B_x^2 + B_y^2)$ and total energy density $e = p/(\gamma - 1) + \rho(v_x^2 + v_y^2)/2 + p_m$.

Note: The code uses variables `ux` and `uy` for the momentum density components ρv_x and ρv_y.

System (1) is discretized with finite volumes (grid cells) of size $\Delta x \times \Delta y$ and the cell averages $\mathbf{u}_{i,j}$ are advanced with a semi-discrete scheme

$$\frac{d}{dt}\mathbf{u}_{i,\,j} = -\frac{1}{\Delta x}(H_{x,\,i+1/2,\,j} - H_{x,\,i-1/2,\,j}) - \frac{1}{\Delta y}(H_{y,\,i,\,j+1/2} - H_{y,\,i,\,j-1/2}) \tag{2}$$

Numerial fluxes $H_{x,y}$ at the cell interfaces are computed with a central scheme (Kurganov & Levy, 2000):

$$\mathbf{H}_{x,\ i+1/2,\ j} = \frac{\mathbf{F}_x(\mathbf{u}^{x+}_{i+1/2,\ j}) + \mathbf{F}_x(\mathbf{u}^{x-}_{i+1/2,\ j})}{2} - \frac{a_{i+1/2,\ j}}{2}\left[\mathbf{u}^{x,+}_{i+1/2,\ j} - \mathbf{u}^{x,-}_{i+1/2,\ j}\right] \tag{3}$$

Here, $\mathbf{u}^{x,\pm}_{i+1/2,\ j}$ are left/right-side reconstructed values at the upper (in x-direction) call interface, and

$$a_{j+1/2} = \max\left[\sigma(\mathbf{u}^{+}_{i+1/2,\ j}), \sigma(\mathbf{u}^{-}_{i+1/2,\ j})\right]$$

is the maximum wave speed estimated from these reconstruced values (similar for the y-direction).

System (2) is integrated with a SSP 3rd order Runge Kutta scheme.

Execution sequence

Global variables and functions are declared in file `global.h`, values of the parameters (computational domain: `xmin, xmax` etc., #grid cells: `nx, ny`, #integration steps: `nStep`, time step size: `dt` etc.) can be changed in `global.cc`.

1. Execution starts in function `main`, file `main.cc`. Fields are allocated.
2. Initial values for the fields are set (file `initFiels.cc`).
3. `nStep` integration steps are made (function `doStep` in file `timeStep.cc`). Each consists of 3 Euler-type substeps. In each substep,
   - the $\mathbf{u}$-values are reconstruced to the cell interfaces with a MinMod-limiter or CWENO method (file `reconstruct.cc`)
   - the numerical interface fluxes H_x and H_y are composed from the physical flux $\mathbf{F}$ and diffusive fluxes according to Eq. (3) in (function `calcFlux` in file `flux.cc`).
     The flux function $\mathbf{F}$ itself, i.e. the equations that shall be integrated, is defined in the functions `fluxFunctionX/Y`.
   - the fluxes are applied to $\mathbf{u}$ in `timeStep.cc`
   - boundary conditions for two ghost cell layers are set (function `boundaryCond` in file `boundary.cc`).

   Inbetween, the data is written to vtk files in subdirectory `data` (file `vtkout.cc`) every `outputStep` steps.

Note: The code is **not** written with best runtime performance in mind. It can easily be improved in that respect.

Exercises

1. One-dimensional linear Alfvén wave

The code is set up to simulate a smooth 1-dim Alfvén wave travelling along the x-direction with small amplitude. Initial conditions are

$$\rho_0 = const., \quad p_0 = const., \quad v_{x0} = 0, \quad v_{y0} = A\sin(kx)$$
$$B_{x0} = const., \quad B_{y0} = A\sqrt{\rho_0}\sin(kx)$$

a) Compile and run the code (see the *How to* section for details).

Load the data into VisIt and inspect the wave components v_x and B_x: Verify that the propagation speed is the Alfén speed $c_A = B_{x0}/\sqrt{\rho}$.
Initial values can be modified in `initFields.cc` (recompile afterwards).
When changing parameters, be careful with the stability criterion $\Delta t < \Delta x/c_A$.

b) Repeat the computation with the minMod-reconstruction instead of the CWENO-reconstruction by (un-)commenting the corresponding lines in function `recoMinus`, file `reconstruct.cc`.
How does this affect the wave amplitude and shape during the simulation?

Hint: In VisIt, you can plot a cut through the data with the line-tool: In the plot window toolbar, activate the "Lineout mode" by clicking the graph symbol. Then, press "shift" and drag the mouse horizontally through the plot window.

2. Orszag-Tang vortex

A widely used test case for the formation of a shock from smooth initial conditions is the two-dimensional "Orszag-Tang vortex" with $\gamma = 5/3$ and initial conditions

$$\rho_0 = \gamma^2, \quad p_0 = \gamma, \quad v_{x0} = -\sin y, \quad v_{y0} = \sin x, \qquad B_{x0} = -\sin y, \quad B_{y0} = \sin(2x)$$

a) Run this test on the domain $(x, y) \in [0, 2\pi] \times [0, 2\pi]$, first with 100 x 100 cells. Integrate up to time $t = 3$ with $\Delta t = 10^{-2}$, output every 20 steps.

Change between minMod- and CWENO-reconstruction and compare e.g. ρ in the final stage.

Hint: You can create different data sets by changing the `outFileStem` in `global.cc`. You can also save the plots from within VisIt with "File $\rightarrow$ Set save options" and "File $\rightarrow$ Save window".

b) Increase the resolution to 288^2 cells, reduce `dt` and increase `nStep` accordingly. Compare with the paper by Balbas & Tadmor.

3. Shock-cloud interaction

Try this:

- domain size $[-0.5, 0.8] \times [-0.5, 0.5]$ with 100 x 100 cells
- copy file `initFields.cc.ex3` to `initFields.cc` for the initial conditions. They describe a shock at $x = -0.4$ and a high-density plasma cloud around $x = y = 0$.
- use fixed boundaries in x-direction by commenting-out the call of `boundaryX()` in `boundary.cc`
- do 1000 time steps with `dt = .02 * fmin(dx, dy)` and output every 20 steps.

4. Linear advection test

This test demonstrates that discontinuities will diffuse to a number of grid cells: Modify the equations to realize the simple advection equation

$$\partial_t \rho = -\partial_x(a_x \rho) - \partial_x(a_y \rho)$$

with $a_x = a_y = 1$ for ρ and $\partial_t = 0$ for all other quantities:

- the functions `speedX` and `speedY` should return the speeds $a_x = a_y = 1$:

```
static real speedX(real* fields)
// fields contain the reconstructed values at cell interface...
{
   return 1.;
}
```

- set

```
fluxes[Rho] = fields[Rho];
```

  and all other fluxes to 0 in functions `fluxFunctionX/Y`.

a) advect a circular step in a domain$[-\pi, \pi] \times [-\pi, \pi]$. Initialize ρ with

```
rho(i, j) = ( sqr(x-M_PI) + sqr(y-M_PI) < 4 ) ? 1. : 0.;
```

(this is C slang and means "`if ( sqr() ... ) < 4 then rho(i, j) = 1; else rho(i, j) = 0. ;`")
Set all other quantities with 0. Use 50^2 cells and move the step once through the domain.
Compare results for the two different reconstructions. What about the extrema?

b) Repeat with higher resolution.

How to ...

- **...compile & run the code:**
  To compile, unpack the code, change to the code directory. Type "`make`" at the terminal prompt and watch out for compilation errors. If you make changes to individual files, just type "`make`" again. The executable file is "`mhd2d`".

  To run the code, just type "`mhd2d`" at the terminal prompt. It will (hopefully) print out a message, run without errors and produce vtk data files in the subdirectory `data`.

  You can also combine both by typing "`make && ./mhd2d`".

- **...look at the data (using VisIt):**

  - start VisIt, e.g. in a terminal: type "`visit`".
  - select the "`Auto update`" checkbox.
  - select "File → Open file...", browse into the `data` directory and open the "MHD_.vtk" database.
  - then, select "Plots → Pseudocolor → rho" to display, e.g., the mass density rho. You can select other variables for display under the "Variables" menu.
  - cycle through the time steps using the time slider and/or the forward/backward buttons.

  The data sets contain the conserved quantities, e.g. the momentum density $u_x = \rho v_x$. To inspect the velocity itself, create an "expression" in visit:

  - select "Controls → Expressions → New"
  - name it e.g. "`vx`"
  - select "scalar mesh variable",
  - write the definition "`ux / rho`"
  - click "apply".

  You can also define a "vector mesh variable", e.g. "`v`" with " { `ux / rho, uy / rho` } " as definition. It can be plotted with "Plots → Vector → v".

- **...change simulation parameters (grid resolution, time step etc.):**
  Change the parameters values in file `global.cc`, **save the file**, "make" the code and run it again.

- **...change the initial data:**
  Make the changes in file `initFields.cc` and re-compile with "`make`".

- **...change the defining equations:**
  Initially, the 2-dim. MHD equations are defined as fluxes $\mathbf{F}$ in file `flux.cc`, functions "`fluxFunctionX/Y`". These operate on the reconstructed values of the conserved quantities.
  To implement other equations that can be formulated as conservation laws, program the correspondig flux function and adjust the speed estimate in functions "`speedX/Y`".

- **...add more dependent variables:**
  You can add more fields to the system vector $\mathbf{u}$:

  - in `global.h`, insert an additional identifier into the enum, e.g. to add u_z and B_z,

    ```
    enum { Rho=0, Ux, Uy, Uz, Bx, By, Bz, E, N_FIELDS};
    ```

    You may also introduce references to the new fields in `global.h`,

    ```
    extern Array& uz;
    extern Array& bz;
    ```

    and initialize them in `global.cc`:

    ```
    Array& uz = fields[Uz];
    Array& bz = fields[Bz];
    ```

  - set initial values for the new fields in `initFields.cc`,
  - add the correspondig expression for the fluxes in functions `fluxFunctionX/Y`, file `flux.cc` (i.e. the "equations" for the new variables). Also adjust the speed estimates in `speedX/Y`.
  - to write the fields out to the vtk files, add approproate `writeArray`-lines in `vtkout.cc`.