Part 3: Project Overview and Details

Mathematisches Institut

August 14, 2017

Goal of the Project

Implement a two-dimensional finite volume solver for the ideal MHD equations on Cartesian grids. Apply said solver to standard benchmark problem(s) from the MHD community.

Slide and Program Structure

- Provide specific details regarding the projects in the handout
- By design the projects build on one another and there will be less coding as you progress
- The program structure presented is intended for ease of implementation and is, thus, not optimized
- For example, fluxes are often computed twice which could be removed with a more clever implementation strategy

Project 1: Mesh and Data Structures

- Assume periodic boundary conditions on the interval $[a, b]$
- Divide interval into N uniform, non-overlapping cells with $\Delta x = \frac{b-a}{N}$
- Store average solution at cell centers

$$x_{\mathtt{i}} = a + \frac{\Delta x}{2} + (\mathtt{i} - 1)\Delta x \tag{1}$$

- Store the solution of conserved variables in a 2D array: `u(i,k)`
- Cell index: $\mathtt{i} = 1, \dots, N$
- Equation index: $\mathtt{k} = 1, \dots, 8$

Reminder: The vector of conserved variables contains

$$\vec{u} = (\varrho, \varrho u, \varrho v, \varrho w, E, B_1, B_2, B_3)^T \tag{2}$$

Project 1: x-Flux

```
getAdvectiveFluxX
Input: solution state: u(i,:)

   From the solution in a given cell compute the eight components
   of the flux function, f, for the 1D ideal MHD system

Output: x-flux vector: f
```

Reminder: The vector flux for the 1D MHD equations is

$$\vec{f} = \begin{bmatrix} \varrho u \\ \varrho u^2 + p + \frac{1}{2}\|\vec{B}\|^2 - B_1^2 \\ \varrho uv - B_1 B_2 \\ \varrho uw - B_1 B_3 \\ u\left(E + p + \frac{1}{2}\|\vec{B}\|^2\right) - B_1(\vec{v}\cdot\vec{B}) \\ 0 \\ uB_2 - vB_1 \\ uB_3 - wB_1 \end{bmatrix} \tag{3}$$

Hint: Computation of the flux requires the primitive variables

Project 1: Maximum Eigenvalue in x-Direction

```
                          getMaxEigenvalueX
Input: solution state: u(i,:)

   Given a certain solution state, u(i,:), compute the absolute value
   of the two fast magnetoacoustic eigenvalues and take the maximum.

Output: maximum eigenvalue: lambda_x
```

Reminder: The fast magnetoacoustic eigenvalues in the x-direction are

$$\lambda^x_{\pm f} = u \pm c_f \quad \text{with} \tag{4}$$

$$c_f = \sqrt{\frac{1}{2}(\tilde{a}^2 + \tilde{b}^2) + \frac{1}{2}\sqrt{(\tilde{a}^2 + \tilde{b}^2)^2 - 4\tilde{a}^2\tilde{b}_1^2}} \tag{5}$$

with the conventional notation

$$\vec{\tilde{b}} = \frac{\vec{B}}{\sqrt{\varrho}}, \quad \tilde{b}^2 = \tilde{b}_1^2 + \tilde{b}_2^2 + \tilde{b}_3^2, \quad \tilde{a}^2 = \frac{p\gamma}{\varrho} \tag{6}$$

Project 1: Riemann Solver in x-Direction

```
                                   RiemannFluxX
Input: two solution states: u_L(i,:), u_R(i,:)

   Use the local Lax-Friedrichs Riemann solver to compute the numerical
   flux at a cell interface. This numerical flux must compute the
   advective flux at each solution state as well as the maximum
   eigenvalue at the interface, i.e.,
                     max{lambda_L, lambda_R}

Output: numerical flux: F_LLF
```

Reminder: The form of the local Lax-Friedrichs Riemann solver is

$$\vec{F}^{n,LLF}_{i+\frac{1}{2}} = \frac{1}{2}\left(\vec{F}_{i+1} + \vec{F}_i\right) - \frac{\lambda^x_{i+\frac{1}{2},max}}{2}\left(\vec{U}_{i+1} - \vec{U}_i\right) \qquad (7)$$

Project 1: Time Step Calculation

```
computeTimestep
Input: CFL number, delta_x, solution array: u(:,:)

   Loop through each cell and compute the maximum eigenvalue. Take
   the maximum of these values over all cells to obtain the global term
   for lambda_max. Then compute the largest stable time step delta_t.

Output: time step: delta_t
```

Reminder: The time step for an explicit method is dictated by the CFL condition

$$\Delta t = \mathtt{CFL}\frac{\Delta x}{\lambda_{max}} \tag{8}$$

Project 1: Explicit Euler Time Integration

```
                         integrateInTime
Input: solution array at current time: u_n(:,:), delta_x, delta_t

   Loop through each interface in the domain and compute the Riemann
   flux, F_LLF. Note that special care must be taken at the left
   interface of the first cell and the right interface of the last cell
   to incorporate the boundary condition. Once all numerical fluxes
   are computed, update the solution with the finite volume scheme.
   For reduced storage overwrite the current solution with the new one.

Output: updated solution array at next time: u_np1(:,:)
```

Reminder: The finite volume update in each cell has the form

$$\vec{U}_i^{n+1} = \vec{U}_i^n - \frac{\Delta t}{\Delta x}\left(\vec{F}_{i+\frac{1}{2}}^{n,LLF} - \vec{F}_{i-\frac{1}{2}}^{n,LLF}\right) \tag{9}$$

Hint: To include periodic boundary conditions you can use a “ghost” cell to get the necessary left/right state needed at the edges of the domain

Project 1: Driver Routine

```
                         finiteVolumeMain
Input: nothing

   This is the main driver of the finite volume scheme. Quantities
   such as a, b, N, the final time, and gamma are defined here.
   Also allocates the memory to store the solution (u=zeros(N,8)).
   It will contain a time loop to integrate from t=0 up to t=T.
   First, initialize the solution array u. Next, compute the time step
   and then integrate with forward Euler. Increment in time and
   the loop repeats. Can use MATLAB ‘‘plot’’ function for 1D
   data and the ‘‘drawnow’’ commands to make movies.

Output: data to the screen, plots, etc.
```

Hint: Ensure that the time loop reaches the final time T exactly. The program should not overshoot the desired value.

Project 1: Code Validation

- Important to validate implementation with a convergence test
- Ensures one obtains the expected order of the scheme and that there are no bugs
- Use an analytical solution of the PDEs such that we can compare the error between the true solution and the approximation
- Increase resolution and the errors should decrease
- Full description of the convergence test and the necessary error analysis is available in the handout

Project 1: Application to a MHD Shocktube

- Use the code to run a simulation of the compound shock test described in the handout
- Run with simulation for different numbers of cells to see how the features are captured

Project 2: Strategy for 2D Extension

- Extension of the one-dimensional implementation to 2D is straightforward
- Due in large part to the assumption that a uniform, Cartesian grid of cells discretizes the domain
- Re-use as much debugged code as we can, e.g. `getAdvectiveFluxX, RiemannFluxX`
- Implement local Lax-Friedrichs flux in the y-direction as well

Project 2: Mesh and Data Structures

- Assume periodic boundary conditions for the domain $[a, b]^2$
- Divide domain into N^2 uniform, square cells so $\Delta x = \Delta y = \frac{b-a}{N}$
- Store average solution at cell centers

$$(x_{\mathtt{i}}, y_{\mathtt{j}}) = \left(a + \frac{\Delta x}{2} + (\mathtt{i} - 1)\Delta x, b + \frac{\Delta y}{2} + (\mathtt{j} - 1)\Delta y\right)$$

- Store the solution of conserved variables in a 3D array: `u(i,j,k)`
- Cell indices: $\mathtt{i}, \mathtt{j} = 1, \ldots, N$
- Equation index: $\mathtt{k} = 1, \ldots, 8$

Project 2: y-Flux

```
getAdvectiveFluxY
Input: solution state: u(i,j,:)

   For a given cell in the domain compute the eight components of the
   flux function, g, in the ideal MHD system.

Output: y-flux vector: g
```

Reminder: The vector y-flux, $\vec{g}$, for the ideal MHD equations is

$$\vec{g} = \begin{bmatrix} \varrho v \\ \varrho uv - B_1 B_2 \\ \varrho v^2 + p + \frac{1}{2}\|\vec{B}\|^2 - B_2^2 \\ \varrho vw - B_2 B_3 \\ v\left(E + p + \frac{1}{2}\|\vec{B}\|^2\right) - B_2(\vec{v}\cdot\vec{B}) \\ vB_1 - uB_2 \\ 0 \\ vB_3 - wB_2 \end{bmatrix} \tag{10}$$

Hint: Computation of the flux requires the primitive variables

Project 2: Maximum Eigenvalue in y-Direction

```
                         getMaxEigenvalueY
Input: solution state: u(i,j,:)

   Given a certain solution state, u(i,j,:), compute the absolute value
   of the two fast magnetoacoustic eigenvalues and take the maximum.

Output: maximum eigenvalue: lambda_y
```

Reminder: The fast magnetoacoustic eigenvalues in the y-direction are

$$\lambda^{y}_{\pm f} = v \pm c_f \quad \text{with} \tag{11}$$

$$c_f = \sqrt{\frac{1}{2}(\tilde{a}^2 + \tilde{b}^2) + \frac{1}{2}\sqrt{(\tilde{a}^2 + \tilde{b}^2)^2 - 4\tilde{a}^2\tilde{b}_2^2}} \tag{12}$$

note c_f is different from that defined on a previous slide

Project 2: Riemann Solver in y-Direction

```
                         RiemannFluxY
Input: two solution states: u_L(i,j,:), u_R(i,j,:)

   Use the local Lax-Friedrichs numerical flux in the y-direction
   at a cell interface. This numerical flux must compute the g
   advective flux at each solution state as well as the maximum
   eigenvalue at the interface.

Output: numerical flux: G_LLF
```

Reminder: The form of the local Lax-Friedrichs Riemann solver is

$$\vec{G}^{n,LLF}_{i,j+\frac{1}{2}} = \frac{1}{2}\left(\vec{G}_{i,j+1} + \vec{G}_{i,j}\right) - \frac{\lambda^{y}_{j+\frac{1}{2},max}}{2}\left(\vec{U}_{i,j+1} - \vec{U}_{i,j}\right) \tag{13}$$

Project 2: Compute Time Step

```
computeTimestep
Input: CFL number, delta_x, delta_y, solution array: u(:,:,:)

   Loop through each cell and compute the global maximum eigenvalue in
   both x and y directions. The CFL condition in 2D incorporates the
   two wave speeds. Then compute the largest stable time step delta_t.

Output: time step: delta_t
```

Reminder: The two-dimensional CFL condition is given by

$$\Delta t = \mathrm{CFL} \cdot \min\left\{ \frac{\Delta x}{\lambda^x_{max}}, \frac{\Delta y}{\lambda^y_{max}} \right\} \tag{14}$$

Project 2: Explicit Euler Time Integration

```
                                integrateInTime
Input: solution array at current time: u_n(:,:,:), delta_x, delta_y,
       delta_t

    Loop through each interface in the domain and compute the Riemann
    fluxes, F_LLF and G_LLF. Note that, again, special care must be
    taken at the cells on the boundary of the domain. Once all numerical
    fluxes are computed, update the solution with the finite volume
    scheme. For reduced storage overwrite the current solution with
    the new one.

Output: updated solution array at next time: u_np1(:,:,:)
```

Reminder: The 2D finite volume update in each cell has the form

$$\vec{U}_{i,j}^{n+1} = \vec{U}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\vec{F}_{i+\frac{1}{2},j}^{n,LLF} - \vec{F}_{i-\frac{1}{2},j}^{n,LLF}\right) - \frac{\Delta t}{\Delta y}\left(\vec{G}_{i,j+\frac{1}{2}}^{n,LLF} - \vec{G}_{i,j-\frac{1}{2}}^{n,LLF}\right) \quad (15)$$

Hint: Use “ghost” cells at the boundaries to include periodic boundary conditions

Project 2: Driver and Validation

- Finite volume driver routine remains largely unchanged
- Only difference, modulo changing “plot” calls to “surf” calls, is to increase the size of the solution array from 2D to 3D

- Re-run a convergence test to validate the 2D implementation of the finite volume solver
- Details of the analytical solution and error analysis are provided in the handout

Project 3: Extension to GLM Divergence Cleaning

- Next address the errors in the divergence-free constraint introduced by the numerical scheme
- Already selected the GLM divergence correction method
- Augment the system with a ninth equation and couple a correction variable into the magnetic field equations
- Store the solution of conserved variables in a 3D array: `u(i,j,k)`
- Cell index: $\mathtt{i,j} = 1, \ldots, N$
- Equation index: $\mathtt{k} = 1, \ldots, 9$

Reminder: The vector of conserved variables **now** contains

$$\vec{u} = (\varrho, \varrho u, \varrho v, \varrho w, E, B_1, B_2, B_3, \psi)^T \tag{16}$$

Project 3: x-GLM-Flux

```
getAdvectiveFluxXGLM
Input: solution state: u(i,j,:)

   From the solution in a given cell compute the nine components of
   the x-GLM-flux function, f_GLM.

Output: x-GLM-flux vector: f_GLM
```

Reminder: The vector x-GLM-flux is

$$\vec{f}_{GLM} = \begin{bmatrix} \varrho u \\ \varrho u^2 + p + \frac{1}{2}\|\vec{B}\|^2 - B_1^2 \\ \varrho uv - B_1 B_2 \\ \varrho uw - B_1 B_3 \\ u\left(E + p + \frac{1}{2}\|\vec{B}\|^2\right) - B_1(\vec{v}\cdot\vec{B}) \\ \psi \\ uB_2 - vB_1 \\ uB_3 - wB_1 \\ c_h^2 B_1 \end{bmatrix} \tag{17}$$

Hint: Computation of the flux requires the primitive variables

Project 3: y-GLM-Flux

```
getAdvectiveFluxYGLM
Input: solution state: u(i,j,:)

   From the solution in a given cell compute the nine components of
   the y-GLM-flux function, g_GLM.

Output: y-GLM-flux vector: g_GLM
```

Reminder: The vector y-GLM-flux is

$$\vec{g}_{GLM} = \begin{bmatrix} \varrho v \\ \varrho u v - B_1 B_2 \\ \varrho v^2 + p + \frac{1}{2}\|\vec{B}\|^2 - B_2^2 \\ \varrho v w - B_2 B_3 \\ v\left(E + p + \frac{1}{2}\|\vec{B}\|^2\right) - B_2(\vec{v}\cdot\vec{B}) \\ vB_1 - uB_2 \\ \psi \\ vB_3 - wB_2 \\ c_h^2 B_2 \end{bmatrix} \tag{18}$$

Hint: Computation of the flux requires the primitive variables

Project 3: Explicit Euler GLM Time Integration

```
integrateInTime
Input: solution array at current time: u_n(:,:,:), delta_x, delta_y,
       delta_t

   Loop through each interface in the domain and compute the Riemann
   fluxes, F_LLF and G_LLF for the GLM MHD equations. Note that, again,
   special care must be taken at the cells on the boundary of the
   domain. The GLM source term is also incorperated into the
   approximation. Once all numerical fluxes are computed, update the
   solution with the finite volume scheme. For reduced storage
   overwrite the current solution with the new one.

Output: updated solution array at next time: u_np1(:,:,:)
```

Reminder: The 2D GLM update in each cell has the form

$$\vec{U}_{i,j}^{n+1} = \vec{U}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\vec{F}_{i+\frac{1}{2},j}^{n,GLM-LLF} - \vec{F}_{i-\frac{1}{2},j}^{n,GLM-LLF}\right) - \frac{\Delta t}{\Delta y}\left(\vec{G}_{i,j+\frac{1}{2}}^{n,GLM-LLF} - \vec{G}_{i,j-\frac{1}{2}}^{n,GLM-LLF}\right) + \Delta t\vec{S} \quad (19)$$

where

$$\vec{S} = (0,0,0,0,0,0,0,0,-(c_h^2/c_p^2)\psi)^T, \quad c_h = \lambda_{max}, \; c_p = 0.18 \quad (20)$$

Project 4: Blast Wave with and without Magnetic Fields

- Alter the initialization routine to include the initial condition for the blast wave from the handout
- Run the simulation with no magnetic fields (Euler) and with a strong background magnetic field in B_1
- Run the simulation with and without divergence cleaning to examine the effect
- How does the introduction of magnetic fields influence the flow?

Bonus Project: Reconstruction to Second Order

- This takes the functioning first order code and alters it to be second order in space and time
- It is of interest to compare results computed with first and second order (but the same number of degrees of freedom)
- Apply to advanced test cases: Orzag-Tang, magnetic rotor

Bonus Project: Linear Reconstruction with `minmod` Limiter

```
reconstructSolution
Input: solution array: u_n(:,:,:)

   Loop over all cells and perform a linear reconstruction of the
   solution in each cell. This will result in four values of the
   reconstructed solution to be used at each of the four interfaces of
   the cell. Thus, the return argument of this routine will have the
   size u_rec(N,N,4,9).

Output: reconstructed solution array: u_rec(:,:,:,:)
```

Hint: There is an excellent discussion of linear reconstruction and the `minmod` limiter on pages 18 and 19 of the DMV article available on the website.

Bonus Project: Second order in Time

Hint: Alter the time integration method from explicit Euler to Heun's method. For a ODE $y'(t) = f(y(t), t)$ the explicit Euler is

$$y^{n+1} = y^n + \Delta t\, f(y^n, t^n). \tag{21}$$

The second order Heun method reads as

$$\begin{aligned} \tilde{y} &= y^n + \Delta t\, f(y^n, t^n) \\ y^{n+1} &= y^n + \frac{\Delta t}{2}(f(y^n, t^n) + f(\tilde{y}, t^n)) \end{aligned} \tag{22}$$

Bonus Project: Validation and Applications

- Re-run the convergence test to ensure you obtain the correct spatial and temporal accuracy
- Re-run the MHD blast wave and compare the results with the first order method
- Alter the initialization routine to include two new test cases: the Orszag-Tang vortex and the Magnetic Rotor