MATH 6640 Project: 1D Ideal Magnetohydrodynamics

Tyler Tippens

1 Introduction

In this project I've attempted to write a solver for the one-dimensional magnetohydrodynamic (MHD) equations. This is a system of 7 coupled PDEs which describe the motion of a magnetized fluid under the influence of a magnetic field. MHD is a difficult problem numerically: in general, the equations are not fully hyperbolic, as the four characteristic wave modes (entropy, Alfvén, slow, and fast waves) can be degenerate. However, in the 1-D case we do not have to consider this problem.

The PDS system can be written in terms of conserved quantities and their fluxes:

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial f(\vec{U})}{\partial x} = 0 \qquad ,$$

with the vector $\vec{U}$ being the system of coupled equations,

$$\frac{\partial}{\partial t}\begin{pmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ B_y \\ B_x \\ e \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} \rho u_x \\ \rho u_x^2 + p^* - B_x^2 \\ \rho u_x u_y - B_x B_y \\ \rho u_x u_z - B_x B_z \\ B_y u_x - B_x u_y \\ B_z u_x - B_x u_z \\ (e+p^*)u_x - B_x(\vec{B}\cdot\vec{u}) \end{pmatrix} = 0 \qquad .$$

Here ρ is the density, $\vec{u}$ is the velocity, $\vec{B}$ is the magnetic field, and e is the energy density. The total pressure $p^* = p + \frac{1}{2}\left|\vec{B}\right|^2$ is the fluid pressure p plus the magnetic pressure. The fluid

pressure is given by the equation of state:

$$p = (\gamma - 1)\left(e - \frac{1}{2}\rho|\vec{u}|^2 - \frac{1}{2}\left|\vec{B}\right|^2\right) \quad ,$$

with the ratio of specific heats $\gamma = 2$ is fixed.

MHD solvers must also enforce $\nabla \cdot \vec{B} = 0$, a condition which is not present in pure fluid dynamics and which is not one of the MHD equations integrated by the solver. This is usually handled in one of two ways: a divergence cleaner, which removes numerical divergence of the magnetic field after each time step of the integration; or by using a staggered mesh, where the magnetic field values are defined on cell boundaries rather than at cell centers. However, again this problem is avoided here by considering only the 1-D problem, since the field component B_x in the simulation axis is constant and thus the magnetic field is inherently divergence-free.

In the following section I will describe the numerical scheme I used. Then I will present and discuss my results. My code is attached at the end of the report.

2 Numerical Scheme

I have implemented the non-oscillatory central finite volume scheme described in Balbás et al. [2004]. The scheme involves reconstruction of the cell-centered values from the known cell averages using a piecewise polynomial, followed by a predictor-corrector step to find the cell averages of a mesh staggered by one half-cell (i.e., the averages located at the edges of the un-staggered mesh) in the next time step. In practice, the scheme is described by equations 2.19-2.21, 2.26, and 2.28 of Balbás et al. [2004]. What follows is a description of this scheme as implemented.

The cell average $\overline{w}_j^n$ of a mesh cell at index j is time-stepped to time level $n+1$ to find the cell average staggered by half a mesh cell, index $j+\frac{1}{2}$, such that the solution will alternate between the original and staggered mesh. When calculating on the staggered mesh, all offset cell indices $j+1$ in the following equations are replaced by $j-1$ in order to calculate back to the original mesh.

The solution on the staggered mesh at the next time level is given by

$$\overline{w}_{j+\frac{1}{2}}^{n+1} = \frac{1}{2}\left(\overline{w}_j^n + \overline{w}_{j+1}^n\right) + \frac{1}{8}\left(w'_j - w'_{j+1}\right) - \frac{\Delta t}{\Delta x}\left(f_{j+1}^{n+\frac{1}{2}} - f_j^{n+\frac{1}{2}}\right) \quad .$$

The numerical derivative w'_j is approximated by the `MinMod` function:

$$w'_j = \texttt{MinMod}\left[\alpha(\overline{w}_{j+1} - \overline{w}_j), \frac{1}{2}(\overline{w}_{j+1} - \overline{w}_{j-1}), \alpha(\overline{w}_j - \overline{w}_{j-1})\right] \quad ,$$

where I have selected $\alpha = 1.4$ following Balbás et al. [2004]. Here `MinMod` denotes the function

$$\texttt{MinMod}(a,b,c) = \begin{cases} \text{sign}(a)\min(|a|,|b|,|c|) & \text{sign}(a) = \text{sign}(b) = \text{sign}(c) \\ 0 & \text{otherwise} \end{cases} \quad .$$

The flux $f_j^{n+\frac{1}{2}} = f(w_j^{n+\frac{1}{2}})$, with

$$w_j^{n+\frac{1}{2}} = \overline{w}_j^n - \frac{\Delta t}{2\Delta x} f'(w_j^n) \quad .$$

The derivative f' is again approximated by the `MultiMod` function, eliminating the need to evaluate the Jacobian matrix or its eigenvalues and eigenvectors for this method.

3 Results & Analysis

I tested my code using the same initial and boundary conditions as described by Balbás et al. [2004] in order to compare my results to theirs. This is the Brio-Wu shock tube, a standard problem for 1-D MHD testing. The vector $\vec{U}$ on the mesh $0 \le x \le 1$ is initialized to

$$\vec{U} = \begin{cases} (1,0,0,0,1,0,0.5) & 0 \le x < 0.5 \\ (0.125,0,0,0,-1,0,0.5) & 0.5 \le x \le 1 \end{cases} \quad ,$$

with Dirichlet boundary conditions at $x = 0, 1$. We choose $B_x = 0.75$ and $\gamma = 2$. I ran the solver on a mesh with 800 cells and a time step $\Delta t = \Delta x/50$.

Figure 1 shows a snapshot of various quantities fairly early into the simulation, at time $t \approx 0.06$. This is slightly less than one third of the time level shown in Figure 3 of Balbás et al. [2004]. Unfortunately my code becomes unstable shortly after this time, and as the solution develops it begins diverging. I suspect this is most likely a result of my constant time step becoming insufficient as wave modes develop in the shock tube and propagate out from the discontinuity at $x = 0.5$. The code runs very fast, so there is much room for improving the time step. Ideally I would implement an adaptive time step as done by Balbás et al. [2004] to ensure

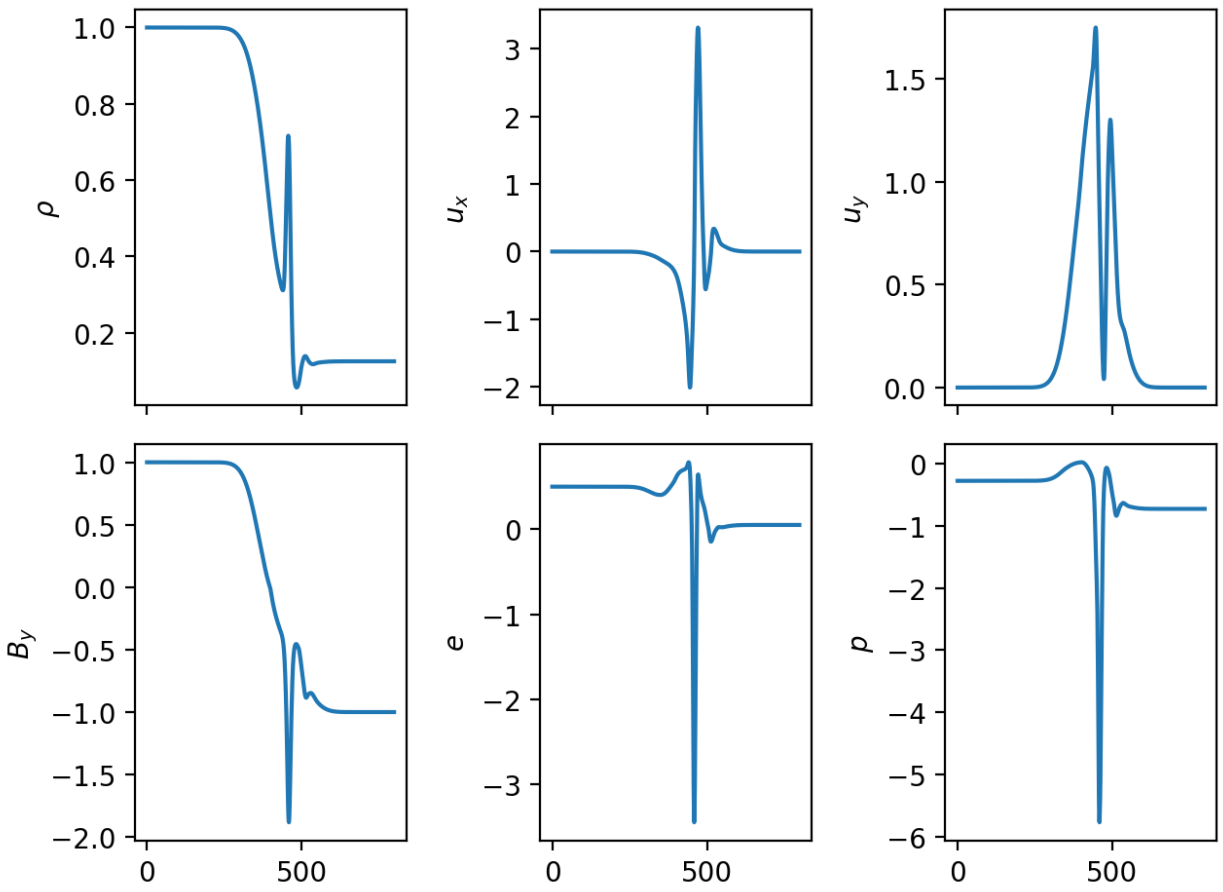

Figure 1: Left-to-right, top-to-bottom: density, x velocity, y velocity, y magnetic field component, energy density, and pressure. Quantities are plotted for $t \approx 0.06$. The horizontal axis shows $j = 0, 1, 2, \ldots, 799$.

the CFL condition is always met:

$$\Delta t = 0.4 \frac{\Delta x}{\max_k |a_k(U)|} \quad ,$$

where $a_k(U)$ are the eigenvalues of the Jacobian matrix of the fluxes $f(U)$. However, I've run out of time to continue the project and so have not implemented the calculation of this matrix and its eigenvalues, nor tracked down any potential implementation issues that may still be lurking.

Overall, this was a challenging problem. The methods for solving the MHD equations are complicated and universally multi-faceted. After reading quite a few papers the method that I chose to implement was the simplest that worked for the 1-D problem, as it did not require calculation, eigen-decomposition, or inversion of any large matrices. The necessity for a divergence-free solution ($\nabla \cdot \vec{B} = 0$) in 2- and 3-D significantly increases the complexity and

compute cost of the relevant schemes, and the Jacobian for a system of 7 coupled equations is highly non-trivial to find. Furthermore, large matrix inversions are required for many methods, or are avoided using still more complicated schemes.

References

Jorge Balbás, Eitan Tadmor, and Cheng-Chin Wu. Non-oscillatory central schemes for one- and two-dimensional mhd equations: I. *Journal of Computational Physics*, 201(1): 261–285, 2004. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2004.05.020. URL https://www.sciencedirect.com/science/article/pii/S0021999104002293.

4 Code

The code is written in Julia and follows in its entirety. Figure 1 was produced in Python and simply plots the quantities output by the code below.

```
#########################################################
####################  Functions  ########################
#########################################################

# write the mesh to file for plotting
function write_mesh(mesh::Array{Float64, 2}, Bx, gamma, tl)
    filename = "data/mhd_TL$tl.out"

    open(filename, "w") do file
        for j in eachindex(mesh[1, :])
            write(file, cell_to_text(mesh[:, j], Bx, gamma))
            write(file, '\n')
        end
    end
end

# write a cell to text
function cell_to_text(cell::Array{Float64, 1}, Bx, gamma)
    rho, rho_vx, rho_vy, rho_vz, By, Bz, e = cell
```

```
    pressure = (gamma - 1)*(e - 0.5*sum([v^2 / rho for v in [rho_vx, rho_vy, rho_v
                0.5*sum([b^2 for b in [Bx, By, Bz]]))
    return "$(rho),$(rho_vx / rho),$(rho_vy / rho),$(rho_vz / rho),$(By),$(Bz),$(e
end

# minmod function for approximating derivatives
function minmod(alpha, wjm, wj, wjp)
    if sign(wjm) == sign(wj) && sign(wj) == sign(wjp)
        delminus = wj - wjm
        delplus = wjp - wj
        delave = 0.5 * (delminus + delplus)

        return sign(wj) * min(abs(delminus), abs(delplus), abs(delave))
    else
        return 0
    end
end

# collect fluxes for all cells
function gather_fluxes(mesh::Array{Float64, 2}, Bx, gamma)
    fluxes = similar(mesh)
    for j in eachindex(mesh[1, :])
        fluxes[:, j] .= gather_fluxes(mesh[:, j], Bx, gamma)
    end
    return fluxes
end

# calculate fluxes for one secc
function gather_fluxes(cell::Array{Float64, 1}, Bx, gamma)
    # variables
    rho, rho_vx, rho_vy, rho_vz, By, Bz, e = cell
    vx = rho_vx / rho
    vy = rho_vy / rho
    vz = rho_vz / rho

    Bmag2 = Bx^2 + By^2 + Bz^2
```

```
    vmag2 = vx^2 + vy^2 + vz^2

    # equation of state
    p = (gamma - 1)*(e - 0.5*rho*vmag2 - 0.5*Bmag2)

    pstar = p + 0.5*Bmag2

    # fluxes                                              # quantity
    f1 = rho_vx                                           # rho
    f2 = rho_vx^2 / rho + pstar - Bx^2                    # rho_vx
    f3 = rho_vx * vy - Bx * By                            # rho_vy
    f4 = rho_vx * vz - Bx * Bz                            # rho_vz
    f5 = By * vx - Bx * vy                                # By
    f6 = Bz * vx - Bx * vz                                # Bz
    f7 = (e + pstar)*vx - Bx*(Bx*vx + By*vy + Bz*vz)      # energy

    return Array{Float64}([f1, f2, f3, f4, f5, f6, f7])
end

############################################################
############################################################

function run()
    # parameters
    nx = 800
    alpha = 1.4
    Bx = 0.75
    gamma = 2

    dx = 1/nx
    dt = 1/(100*nx)

    T_MAX = 0.2
    TL_MAX = div(T_MAX, dt)
    numeq = 7
```

```
# derived
r = dt / dx
rhalf = r / 2

# set initial and boundary conditions
# first dimension is un/staggered
mesh = Array{Float64}(undef, 2, numeq, nx)
stag = 0
# Brio-Wu shock tube
e_init = 1 / (gamma - 1) - 0.5
mesh[1, :, 1:(div(nx,2))] .= [1., 0., 0., 0., 1., 0., e_init]
mesh[1, :, (div(nx,2)+1):nx] .= [0.125, 0., 0., 0., -1., 0., e_init / 10]

mesh[2, :, :] .= copy(mesh[1, :, :])

# misc
t = 0
tl = 0
update_partial = Array{Float64, 1}(undef, numeq)
wjnhalf_arr = Array{Float64, 1}(undef, numeq)
wjplusnhalf_arr = similar(wjnhalf_arr)

while t <= T_MAX

    flxs = gather_fluxes(mesh[stag+1, :, :], Bx, gamma)

    for j in 3:nx-2

        for eq = 1:numeq
            w = mesh[stag+1, eq, :]
            f = flxs[eq, :]
            wprime     = minmod(alpha, w[j-1], w[j], w[j+1])
            wprimeplus = minmod(alpha, w[j-(2*stag)], w[j-(2*stag)+1],
                                w[j-(2*stag)+2])

            wjnhalf_arr[eq] = w[j] - rhalf*minmod(alpha, f[j-1], f[j], f[j+1])
```

```
            wjplusnhalf_arr[eq] = w[j-(2*stag)+1] - rhalf * minmod(alpha,
                                    f[j-(2*stag)],   f[j-(2*stag)+1],
                                    f[j-(2*stag)+2])

            update_partial[eq] = 0.5*(w[j] + w[j-(2*stag)+1])
                                    + (-1*stag)*(wprime - wprimeplus) / 8
        end

        fjnhalf     = gather_fluxes(wjnhalf_arr,     Bx, gamma)
        fjplusnhalf = gather_fluxes(wjplusnhalf_arr, Bx, gamma)

        if stag == 0
            mesh[2, :, j] .= update_partial + r*(fjplusnhalf - fjnhalf)
        else
            mesh[1, :, j] .= update_partial - r*(fjplusnhalf - fjnhalf)
        end

    end

    if tl % div(TL_MAX, 10) == 0
        print("t = $tl/$TL_MAX\n")
    end

    if sum(isnan.(mesh)) > 0
        print("NaN at time $t\n")
        return
    end

    if tl % div(TL_MAX, 30) == 0
        write_mesh(mesh[stag+1, :, :], Bx, gamma, tl)
    end

    # next time step we're back on the other grid
    if stag == 1
        stag = 0
    else
```

```
            stag = 1
        end
        t += dt
        tl += 1
    end
end

run()
```