Introduction

Fortran 90 Shallow Water Solver/ Toy code

- Solves 1D shallow water equations using finite volume methods.
- Uses **Roe's Approximate Riemann Solver** for flux computation.
- Uses **HLL Approximate Riemann Solver** for flux computation.
- Uses **DOT Approximate Riemann Solver** for flux computation.
- Implements **Euler** for time integration.

Governing Equations

1D Shallow Water Equations:

$$\frac{\partial}{\partial t}\begin{bmatrix} h \\ hu \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix} = 0 \tag{1}$$

- h: Water height (stored in `Q(1,:)`)
- hu: Momentum (stored in `Q(2,:)`)
- Velocity: $u = \frac{hu}{h}$
- Gravity: g

Code Structure

Main Components:

- **main.f90**: Driver program controlling execution.
- **initial_cond.f90**: Sets up initial conditions.
- **initial_cond.f90**: Sets up the topography.
- **grid_x.f90**: Defines computational grid.
- **flux_*.f90**: Computes fluxes using * solver.
- **euler.f90**: Implements Euler time stepping.
- **bound.f90**: Implements boundary conditions.
- **postpro.f90**: Stores the solution to files.

Run and Execute

3 Folders:

- src: The sources.
- input: Input files. Input.dat: input parameters defined by the user.
- output: To be created. Empty in the beginning. Contains numerical results in the end.
- src: The sources

To compile:

- **Makefile**: In a terminal. make clean and then make

To run:

- **./my_1dsolver**

Euler Method:

$$Q^{n+1} = Q^n - \frac{\Delta t}{\Delta x}(F_{i+1/2} - F_{i-1/2}) \tag{2}$$

Approximate Riemann Solver:

- Inputs: Conservative Q and non conservative variables PV
- Output: Fluxes computed at each computational face.

Reflective and Open Boundaries:

- **Reflective**: $Q(1,0) = Q(1,1)$, $Q(2,0) = -Q(2,1)$....
- **Open**: Extrapolation of state variables.

Code Execution Flow

1. Read input variables, initialize grid and initial conditions.
   - Compute primitive variables
   - Print in files the initial conditions.
   - Compute dt based on CFL condition.
2. Time-stepping loop:
   - Compute primitive variables.
   - Compute fluxes using chosen solver.
   - Update the solution for the next time step.
   - Enforce boundary conditions.
   - Copy the solution to keep in the next time step.
   - Compute dt based on CFL condition.
3. Post-processing: Save the results for plotting.

To do

- **Implement an SSP RK2 scheme in time**
- **Implement the second order MUSCL reconstruction**
- **Implement the minmode and MC limiters**
- **Obtain the rate of convergence for the space discretization in using a manufactured solution**