AM 260: Computational Fluid Dynamics Final

Dante Buhl

January 6, 2026

Assignment Description:

1. Sod Shock tube, $N_x = 128$, HLL, $t_f = 0.2$, with FOG and PLM+minmod and PPM+minmod
2. Rarefraction, $N_x = 128$, PLM with minmod, $t_f = 0.15$ for HLL and ROE
3. Blast2, $N_x = 128$ with PPM and ROE, $t = 0.038$ for MC,Minmod,VL, compare to FOG + ROE
4. Shu-Osher PLM/MC/HLL $N_x = 32, 64, 128$ $t_f = 1.8$, compare to FOG+HLL
5. Shu-Osher w PLM/vanLeer/Roe, $N_x = 128$ with CFL $= 0.2, 0.4, 0.6, 0.8, 1.0, 1.4$, $t_f = 1.8$.

Notes on Dongwooks Code:
General Notes on code structure:

- the V variables in the code (gr_V) is responsible for the primitive variables at any time, i.e. the actual values at those points.
- the gr_xCoords array stores the cell centered discretization (Page 2 of the assignment PDF, eq. 5)
- the sim_shockLoc variable is the shock interface for the IC, and may be updated later in the code (relevant for the IC)
- the U variables in the code (gr_U) is responsible for the conservative variables at any time.
- the length of V at any given time is stored in NUMB_VAR, while the length of U at any given time is NSYS_VAR
- the 'small pressure' variable is the lower bound for the primitive pressure upon evalutation.
- the sim_order variables stores the order of the solver, i.e. order 1 is a 1st order solver such as the FOG method.

THINGS TO DO:

- X write the Average State script
- X primitive right eigenvectors (pg. 130 Lect Note)
- X prim and cons left eigenvector (pg. 130 & 131 Lect Note)
- X slope limiters (mc & vanLeers)
- X Characteristic Limiting for PLM (pg. 156 Char Limit)
- X ROE section for PLM
- X PPM method (pg. 157-162)
- X ROE riemann solver (pg. 140-147)

- X BC for Periodic and Shu-Osher problem
- X DEBUG PPM sovler (need to fix the shapes of several arrays)
- Implement all test cases (X, 2, 3, 4, 5)
- write report (can be very bad as long as code and results are decent)

driver_euler1d.f90: the driver file which calls all other files as needed. The file begins by initializing a grid and simulation configuration (specifically with the grid_init and sim_init subroutines). This file calls the following subroutines (and subsequent files), calls:

- grid_init
- sim_init
- io_writeOutput
- cfl
- soln_ReconEvoleAvg
- soln_update
- bc_apply

modules:

- sim_data.f90
- grid_data.f90
- io.f90
- bc.f90
- eos.f90

grid_init.f90: this file is responsible for creating, allocating, and storring important parameters for the grid discretization. Reads number of interior cells, number of ghost cells, and domain bounds from a file called slug.init. Allocates the following arrays. gr_U, gr_V, gr_W, gr_VL, gr_vR, gr_flux, gr_eigval, gr_leigvc, gr_reigvc. calls:

- read_initFileInt

modules:

- grid_data.f90
- read_initFile.f90

read_initFile.f90: a module file specifically designed for reading in relevant simulation data. It has a subroutine for each specific datatype. Each of these subroutines is designed to parse data from a string similar to a json routine. The data is taken in as a character array and then indexed by a keyword. Then the value at that index is returned. subroutines:

- read_initFileInt
- read_initFileReal
- read_initFileBool
- read_initFileChar

grid_data.f90: a grid_data module, specifically which hosts the initialization of relevant grid discretization arrays (most of which are allocated in the grid_init.f90 routine).
sim_init.f90: an initialization file which reads in relevant simulation data from the slug.init file. Specifically this reads from the slug init file, the order of the simulation, the number of timesteps, the cfl number, the max time, the interval time, the riemann solver, the slope limiter, the 'name', the char limitter, the Left and Right Density/Velocity/Pressure values, specific heat, shock location, 'small pressure', boundary condition type, and frequency of io outputs. calls:

- read_initFileInt
- read_initFileReal
- read_initFileChar
- read_initFileBool
- sim_initBlock

modules:

- sim_data.f90
- read_initFile.f90

sim_initBlock.f90: initializes values into the gr_V, gr_xCoord arrays. calls:

- prim2cons

modules:

- sim_data.f90
- grid_data.f90
- primconsflux.f90

sim_data.f90: a module file responsible for initializing simulation data, mostly for IC, but some is relevant later in the simulation)
primconsflux.f90: A mopdule file responsible for containing subroutines to switch between different data types i.e. primitive, conservative, and flux. Note that the cons2flux routine is not yet written yet.
subroutines:

- prim2cons
- cons2prim
- prim2flux
- cons2flux

calls:

- eos_cell

modules:

- grid_data.f90
- sim_data.f90
- eos.f90

eos.f90: A module file which is responsible for returning the pressure at any given moment from the other primitive vars. eos_cell assumes that the density, internal energy, and game are already known, while eos_all computes the pressure from the conservative variables by first computing the primitive values and then computing the pressure. subroutines:

- eos_all
- eos_cell

io.f90: a module file specifically for subroutines related to IO. Note that the sim_name variable is responsible for naming the output file, i.e. slug_'sim_name'.dat. This subroutine only prints a specific timestep at each call. subroutines:

- io_writeOutput

cfl.f90: a subroutine which returns the computed timestep according to the cfl safety factor computation. It should be noted that it returns dt = SF*dx/maxSpeed. Here maxSpeed is computed using max(v_x) + cs, where cs is the computed local speed of sound subroutines:

- cfl

soln_ReconEvoleAvg.f90: a dispatcher routine which calls subsequent reconstruction routines. Also stores left and right states for the conservative variables. calls:

- soln_reconstruct
- soln_getFlux

modules:

- grid_data.f90
- sim_data.f90

soln_reconstruct.f90: another dispatcher routine which calls a subsequent solver FOG/PLM/PPM depending on the 'sim_order'. calls:

- soln_FOG
- soln_PLM
- soln_PPM

modules:

- grid_data.f90
- sim_data.f90

soln_FOG.f90: updates the left and right states (gr_vL and gr_vR) according to the FOG method.
soln_PLM.f90: updates the left and right states according to the PLM method, note that all primitive vars after besides Density, Velocity, and Pressure are updaterd using the FOG method. Note that two sections of this routine need to be implemented, the characteristic limitting section and the 'ROE' riemann solver. This does not use the conservative eigenvectors. calls:

- eigenvalues
- left_eigenvectors
- right_eigenvectors
- minmod

- vanLeer
- mc

modules:

- grid_data.f90
- sim_data.f90
- slopeLimiter.f90
- eigensystem.f90

slopeLimiter.f90: a module file which contains the subroutines for computing each of the slope limiters used in this assignment. Note that the mc and vanLeer slope limiters are note yet implemented. subroutines:

- minmod
- mc
- vanLeer

eigensystem.f90: a module file for the eigen subroutines. Note that the primitive right eigenvector computation is not implemented, nor are the conservative and primitative left eigenvector computations. subroutines:

- eigenvalues
- left_eigenvectors
- right_eigenvectors

modules:

- grid_data.f90

soln_PPM.f90: updates the left and right states according to the PPM method (this has not yet been implemented). calls:
modules:

- grid_data.f90
- sim_data.f90
- slopeLimiter.f90
- eigensystem.f90

soln_getFlux.f90: a dispatcher routine which calls either the HLL or ROE routines to obtain the fluxes (gr_flux) from the primative left and right states. calls:

- hll
- roe

modules:

- grid_data.f90
- sim_data.f90

hll.f90: a subroutine which returns the fluxes computed at each intereior point according to the hll method. calls:

- prim2flux

- prim2cons

modules:

- grid_data.f90
- primconsflux.f90

roe.f90: a subroutine which returns the fluxes computed at each interior point according to the roe method. Note that this is not yet fully implemented. calls:

- averageState
- eigenvalues
- left_eigenvectors
- right_eigenvectors
- prim2flux
- prim2cons

modules:

- grid_data.f90
- primconsflux.f90
- eigensystem.f90

soln_update.f90: this updates the conservative variables using U(i+1) = U(i) - dt/dx(Flux_R - Flux_L). Then the primitive vars are updated after this computation. calls:

- prim2cons
- cons2prim

modules:

- grid_data.f90
- primconsflux.f90

bc.f90: a module file responsible for all boundary condition subroutines. The Shu-Osher BC and Periodic BCs still need to be implemented subroutines:

- bc_apply
- bc_outflow
- bc_reflect
- bc_periodic
- bc_user

modules:

- grid_data
- sim_data

averageState.f90: a subroutine which computes the average of the left and right values