MATH-UA 230 Final Project

ISAAC DRACHMAN

New York University

iwd204@nyu.edu

May 20, 2016

Abstract

This write-up details my final project for MATH-UA 230: Fluid Dynamics for Spring 2016. I implement a hydrodynamics code in python. The code simulates two instances of the 1D Shock Tube experiment. The program outputs a gif which animates plots of pressure, density, and velocity along the tube. Relevant source code is contained within the file `grid.py`.

I. PROJECT BASICS

The primary task of the simulation is to numerically solve the Euler Equation. This is a simplified version of the Navier-Stokes equation which details a fluid's conservation of momentum. The simplification is that of high Reynold's number, where inertial forces outweigh the viscous dissipation of energy. Without the viscous component of Navier-Stokes, the Euler equation can be more easily computed using local numerical PDE methods.

The source code in `grid.py` implements two classes: `Grid1D` and a subclass `ShockTube`. `Grid1D` structures a discretized one dimensional surface using equidistant points. A single data structure, denoted `U`, tracks conserved quantities at these discrete points. These quantities are mass (ρ), momentum (ρv), and total energy (E). All fields, including pressure p, velocity v, internal energy e, and others, and the fluxes of conserved quantities are deduced from `U`. At each step of the simulation, the code computes the fluxes at grid half-points (using HLL) and accordingly advances `U` one unit of time using a first-order PDE method. This follows the procedure detailed in the class handout.

The code makes use of the python library numpy to create `U` and other relevant data structures out of the library's fast array type. The HLL method and in fact the entire simulation step process is done in a vectorized fashion on these arrays. This allows not only for greater speed but for better generalization to larger arrays (more resolution) as well.

II. 1D SHOCK TUBE

This experiment is detailed in the project handout distributed in class. A one-dimensional tube, along the x-axis, is filled with fluid. The left half of the tube contains fluid of high pressure and high density while the right half of the tube contains fluid of low pressure and low density. The initial velocity is zero everywhere in the tube. The simulation produces plots of pressure, density, and velocity along the tube as time progresses. This allows for visualization of the resulting shock and refraction waves. The first instance uses the initial values ($p_L = 1$, $\rho_L = 1$, $p_R = 0.125$, $\rho_R = 0.1$). The second instance uses the initial values ($p_L = 100$, $\rho_L = 10$, $p_R = 1$, $\rho_R = 1$). The reason that the latter instance may be considered "harder" is that it induces a much more severe discontinuity. It therefore requires that the simulation use a much smaller time increment to ensure the Courant-Friedrichs-Lewy condition is maintained.

To run this experiment, first make sure all relevant libraries and dependencies are installed (these are listed in `grid.py`.) Then you may run `$ python grid.py` for the first "easy" instance and `$ python grid.py hard` for the second instance. The output (gifs) have been provided. The first instance produces `shocktube-easy.gif` and the second produces `shocktube-hard.gif`. The easy shock tube runs in a couple minutes while the hard shock tube may take several minutes as the time resolution is 10x higher.

III. Discussion

In both instances of the simulation we can see a shock wave of high pressure and density moving to the right with a refractory wave of lower pressure and density on the left. Once the right-moving wave reaches the boundary, the right side's pressure and density flatten. However, the refractory wave continues to drop the left side's pressure and density towards the flat levels of the right side.

The code currently visualizes the simulation by plotting pressure, density, and velocity in separate plots along the tube. Alternatively, pressure and density shown on the same plot by plotting pressure and using a color map for density. Both methods may use the matplotlib library.

As mentioned above, the "harder" instance of the shock tube requires a much smaller time step. The code uses one-tenth the delta-time. The Courant-Friedrichs-Lewy condition which relates time and space increments could still be maintained for the "harder" instance by scaling down the time step but scaling up the space step (using less points on the grid.) It would be interesting to further consider the numerical stability of these shock tube experiments.