Ay190 – Worksheet 16
John Pharo
Date: March 12, 2014
Fiddlers on the Roof: Mee Chatarin Wongurailertkun, Cutter Coryell

Problem 1

This hydrodynamics code is based around a mydata object which stores most of the relevant hydrodynamic data of the system. It stores lists of cell centers, conserved variables, and pressures. The mydata object has several methods defined on it for the purposes of evolving the system in time.

These include functions to set up the 1-D grid and the initial values of the system. In the main program, these are the first methods called, immediately after variables are set with various initial conditions. Passing these variables to these methods sets up the initial state of the system.

Next are a variety of helper methods for switching between primitive and conserved variables, applying the boundary conditions, and implementing various reconstruction methods. These will all be called by the final helper methods, hlle and calc-rhs, which actually update the system.

In the main code, once the mydata object is initialized and the grid setup, the program enters a while loop, which will iterate until the specified end time is reached. In the while loop, the program updates the plot of the system, and then calculates the dynamic time step. After that, the program calculates the new rhs, intermediate step, and boundaries, and then uses these to update the mydata object. Finally, it updates the time, and the loop resumes.

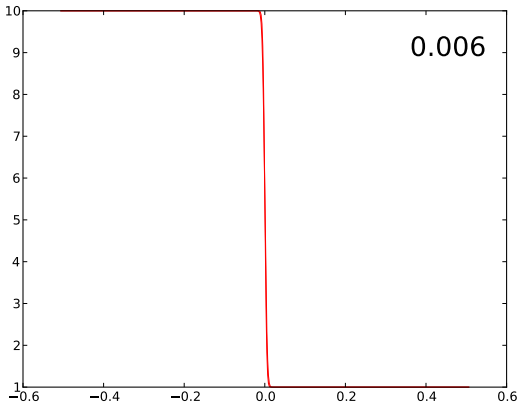

Figure 1: This is the system at t=0.01 s, so essentially in its initial state.

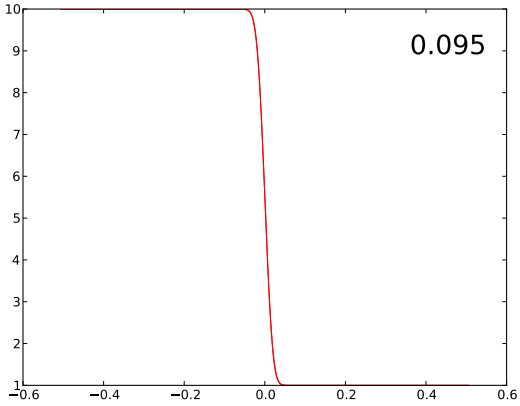

Figure 2: This is the system at t=0.1 s.

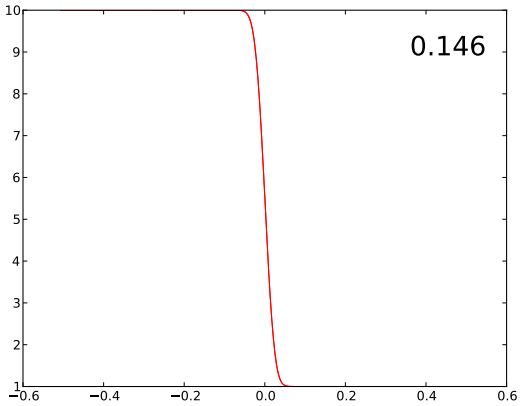

Figure 3: This is the system at t=0.15 s.

Problem 2

See figures 4 through 9.

Problem 4

Using for loops for the code that I wrote, running hlle 10 times takes 0.29532289505 s, but by using numpy arrays, it takes only 0.233140945435 s. Since I only replaced the for loops in the code I changed, there are still for loops in hlle, which, if replaced, could potentially decrease the time needed even more.

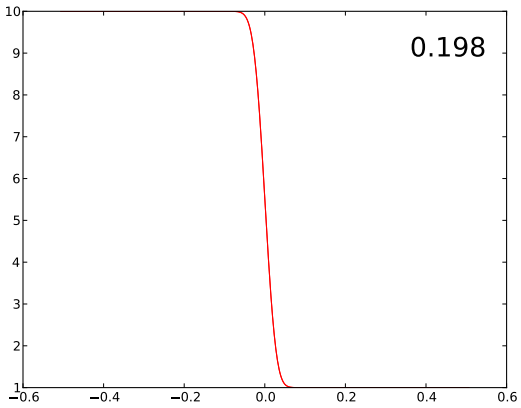

Figure 4: This is the system at t=0.2 s.

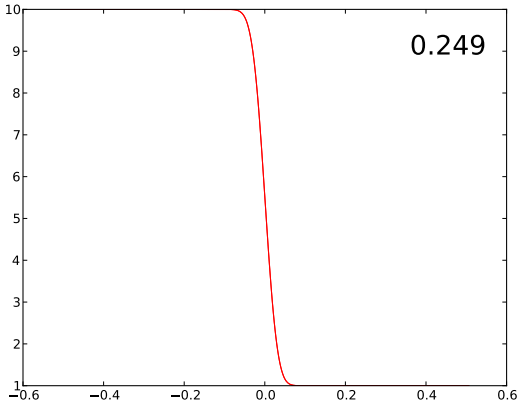

Figure 5: This is the system at t=0.25 s.

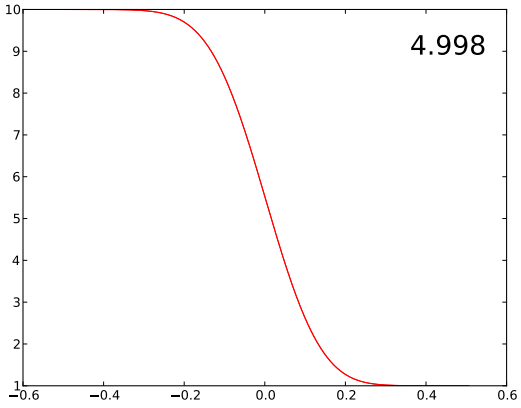

Figure 6: This is the system after a much greater time has passed, at t=5.0 s.

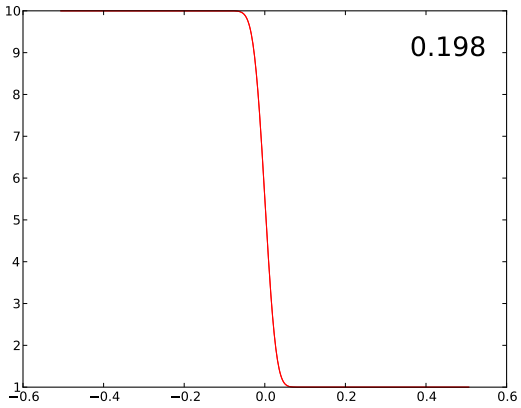

Figure 7: This is the system at t=0.2 s, using the piecewise constant method of reconstruction. This appears to produce the smoothest solution, and it spreads to a greater width in the same amount of time.

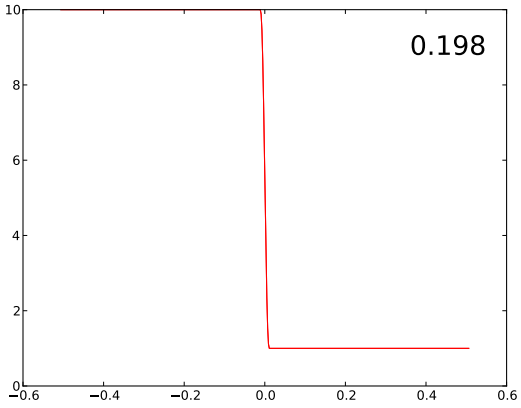

Figure 8: This is the system at t=0.2 s, using the TVD-MC2 method of reconstruction. This is the least smooth solution, appearing pretty choppy and also mostly vertical; the shock didn't spread very far.

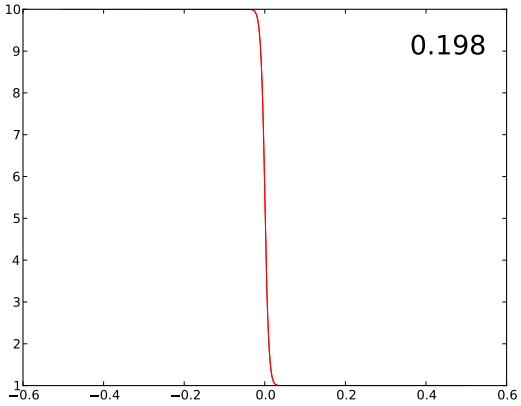

Figure 9: This is the system at t=0.2 s, using the TVD-minmod method of reconstruction.

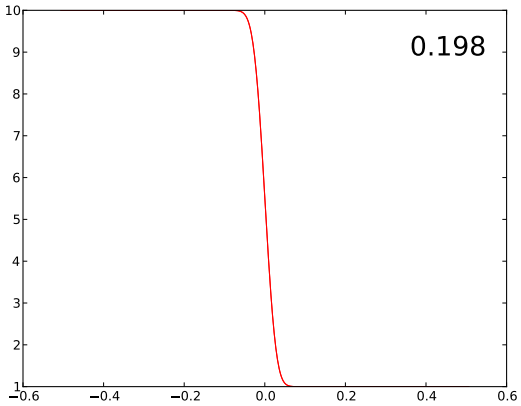

Figure 10: This is the system at t=0.2 s, using for loops.

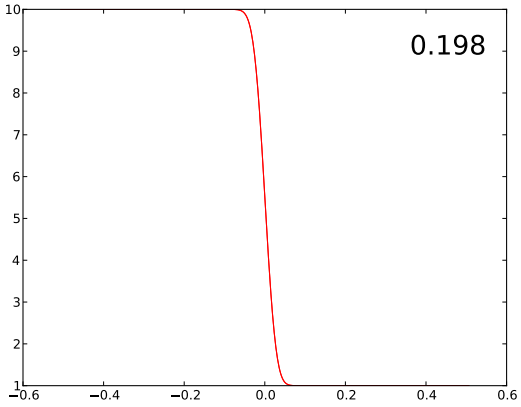

Figure 11: This is the system at t=0.2 s, using numpy arrays. Note that it is identical to the previous solution, though the method changed.