Ay190 – Worksheet 15
Daniel DeFelippis
Date: March 7, 2014

1D SPH Code

1

The program we are running here does "Smoothed Particle Hydrodynamics." This method centers on using a smoothing kernel to deal with a large amount of gas particles. The field of particles (really the field of a property of the particles like density) is convolved with this smoothing kernel:

$$F_{smooth}(r) = \int F(r)W(r - r', h)dr'$$

where W is the smoothing kernel, and h is the characteristic length scale of the smoothing. This kernel will be equal to 0 for values of r' for, in this case, particles farther than 2 scale lengths away from a given particle. Particles between 0 and 1 scale lengths away, and between 1 and 2 scale lengths away, have two different values for the smoothing kernel, as shown by the code below.

```
invh = 1.0/h
u = r*invh
alphaD = twothirds * invh
if u >= 0.0 and u < 1.0:
    return alphaD*(1. - 1.5*u**2 + 0.75*u**3)
elif u < 2.0:
    return alphaD*(0.25*(2.-u)**3)
else:
    return 0.0
```

Here, since we have discrete particles, the integral above is really a sum:

$$F_s(r) \approx \sum_j \frac{m_j}{\rho_j} F_j W(r - r_j, h)$$

where m and ρ are the mass and density at the location of particle j.

Now, for computing any fields, we need to know how many particles are nearer than twice the scale length from a given particle, which the function "neighbor_search_1D" does. Given that, the functions "smoothing_kernel" (code copied above) and "dsmoothing_kernel_rij" compute the kernel and the grad of the kernel, which we will need for computing other things.

For example, the smoothed density at the location of particle i is given by

$$\rho_i \approx \sum_j \frac{m_j}{W}(r - r_j, h)$$

. which we can calculate using the functions listed above, since we have inital values of densities from which we can get masses. THe function "get_density" does this.

One other variable we need to calculate is the artificial viscosity, which increases the entropy of the system during shocks. Each particle will have an artificial viscosity with each of its neighbors, and the function "get_artificial_viscosity" does this.

There are two main equations which this system uses to evolve (the other variables evolve according to those two equations): the momentum equaition and energy equations:

$$\frac{dv_i}{dt} = -\sum_{j=1}^{n} m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \prod_{ij} \right) \nabla W_{ij}(h)$$
$$\frac{d\epsilon_i}{dt} = \frac{1}{2}\sum_{j=1}^{n} m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \prod_{ij} \right) v_{ij} \nabla W_{ij}(h)$$

where $\prod_{ij}$ is the artificial viscosity term and P is pressure (calculated from equation of state using density and energy). The function "get_accels" uses a nested for loop to calculate $\frac{dv_i}{dt}$ for all particles i:

```
accels[i] -= m[jj]*(  p[i]/rho[i]**2 \
                    + p[jj]/rho[jj]**2 \
                    + av[i,jj]) \
                    *dsmoothing_kernel_rij(rij,h)
```

and the function "get_energyRHS" uses the same process to calculate $\frac{d\epsilon_i}{dt}$ for all particles i:

```
epsrhs[i] += 0.5*m[jj]*(  p[i]/rho[i]**2 \
                        + p[jj]/rho[jj]**2 \
                        + av[i,jj]) \
                        *vij*dsmoothing_kernel_rij(rij,h)
```

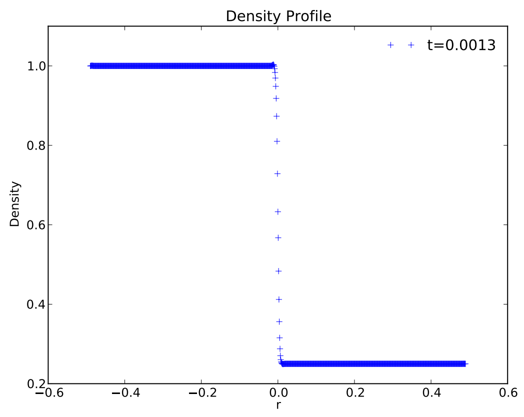
The actual program first sets the initial densities, energies, and velocities. The densities are different for the left and right side, as are the energies, and the velocities are all 0. From these, it calculates the pressures and speeds of sound. Then it runs the neighbor search function so it can calculate the smoothed density. After initializing the timestep, the program finally starts evolving the system.

For each iteration, it calculates artificial viscosities, then accelerations (RHS of momentum equations). The accelerations are used to update the velocities and half step velocities, and then the positions. It also calculates the new RHS of the energy equation, which is used to update the energy.

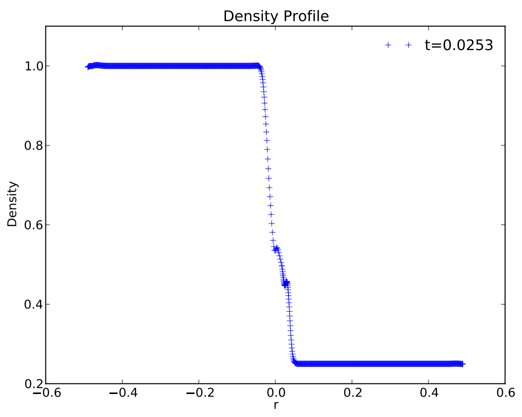
After all this, it updates all of the variables initialized above, updates the plot, and increases the time (i.e. goes to the next iteration which is dt later than the current one).

Plots of every 20 iterations are shown in figures 1 to 5 below, including the first iteration. (i.e., 1, 20, 40, ... , 160).

We see the expected zones: the first and fifth are constant with the initial densities as the value, and there is a rarefaction zone as well as clear contact and shock close-to-vertical lines.

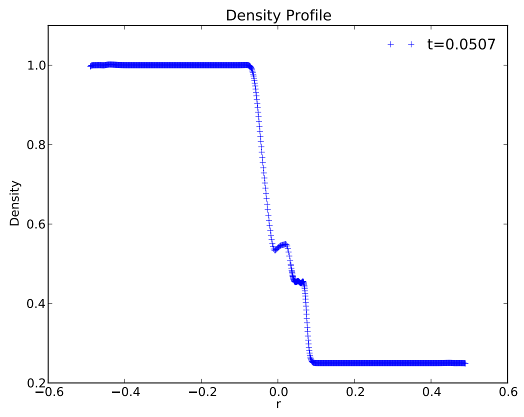


Density profile: Iteration 1

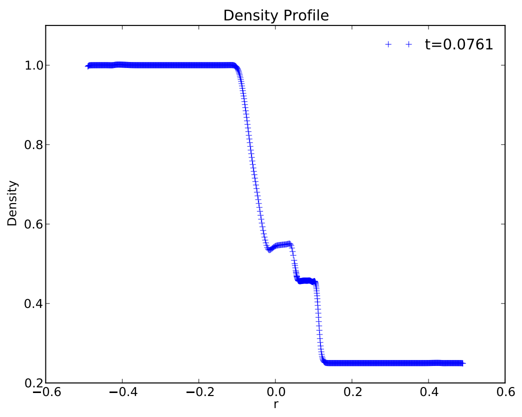


Iteration 20

Figure 1



Iteration 40

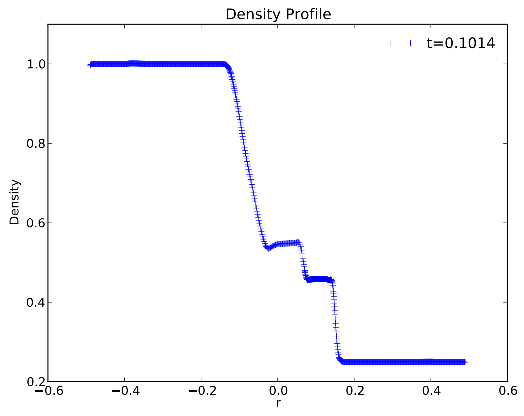


Iteration 60

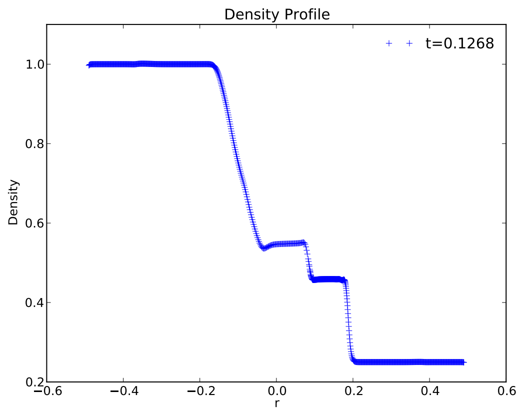
Figure 2

2

We now use the Riemann solver written in Fortran. We can plot the density profile of the Riemann solver at $t = 0.2$ (after editing the desired time and position values) with the density profile of the SPH code at the same time, as well as other variables which the program calculates: velocity and pressure. As expected, the plots in figures 6, 7, and 8 demonstrate that the SPH solver matches the exact solver very well. There are notable differences though. As shown in the density profile (figure 6), the corners of the SPH solution look "overcurved" compared to the sharp corners of the exact solution. Additionally, what are supposed to be vertical lines at around $x = 0.15$ and $x = 0.3$ are slanted in the SPH solution. In the velocity profile (figure 7), there is a small bump at the end of the rarefaction zone that is present in the SPH solution but not the exact solution. In the pressure profile (figure 8), there is what appears to be an instability in the SPH solution at around $x = 0.15$, which is the location of the contact. It isn't in the exact solution, which suggests that the SPH solver isn't the best at calculating the pressures at the boundaries of the zones.

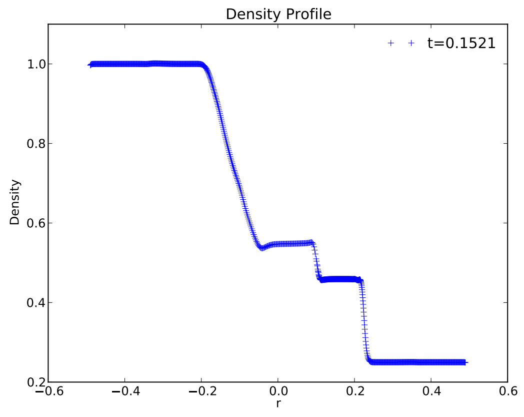


Iteration 80

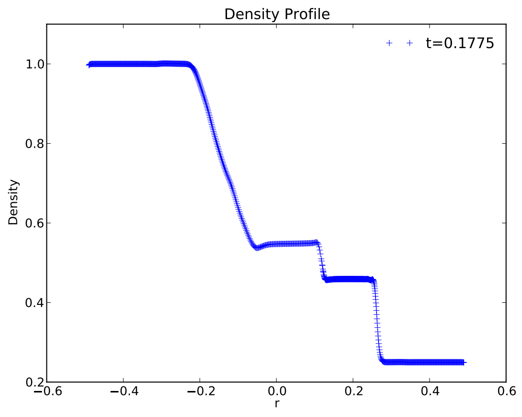


Iteration 100

Figure 3



Iteration 120



Iteration 140

Figure 4

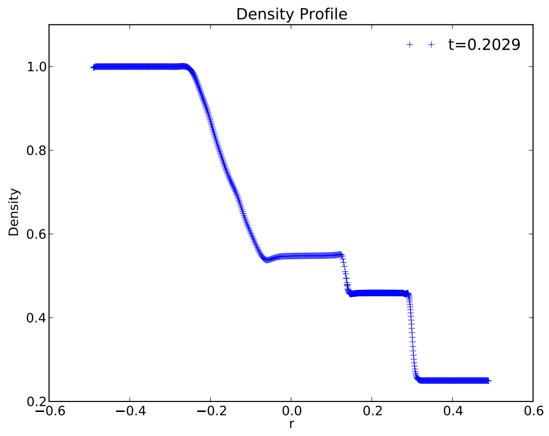


Figure 5: Iteration 160

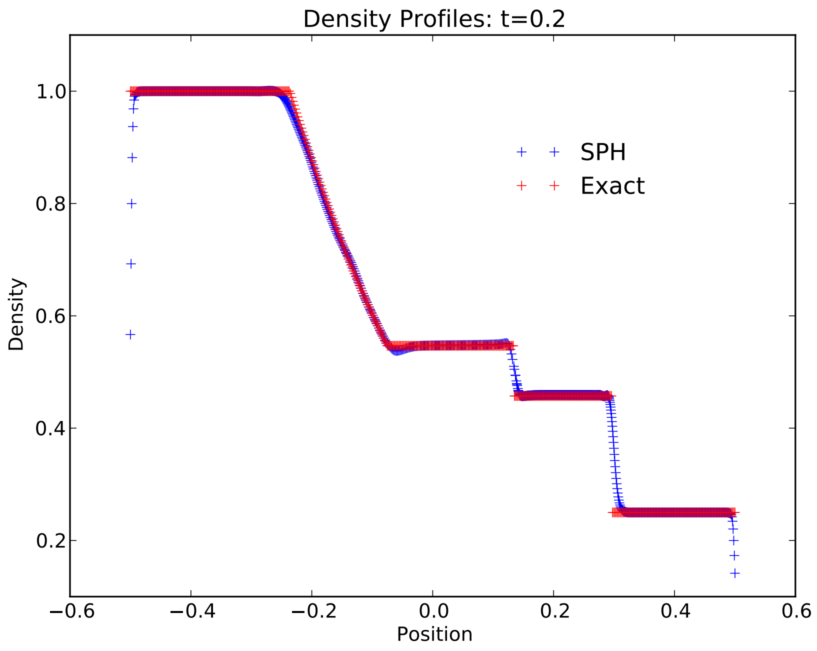


Figure 6: Density profiles of SPH and exact Riemann solver at $t = 0.2$.

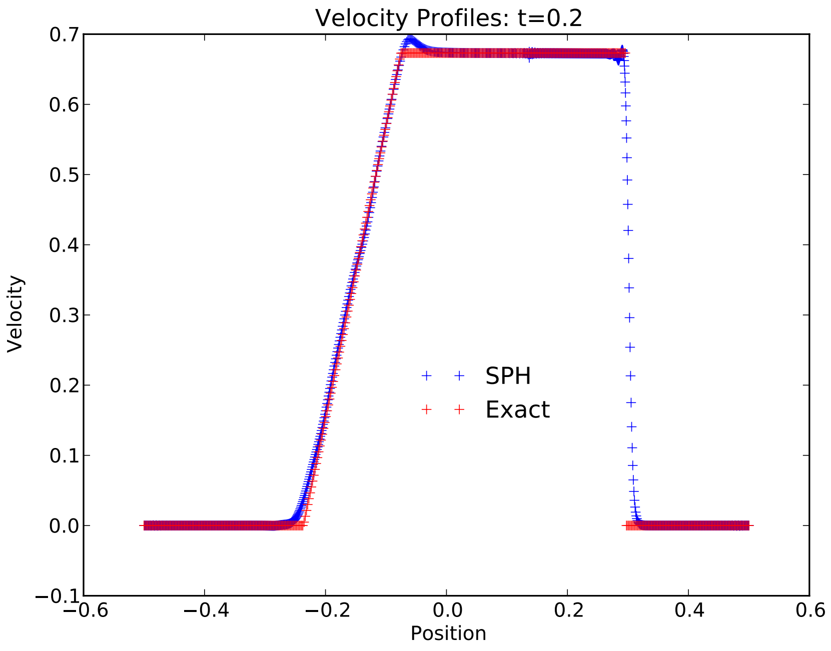


Figure 7: Velocity profiles of SPH and exact Riemann solver at $t = 0.2$.

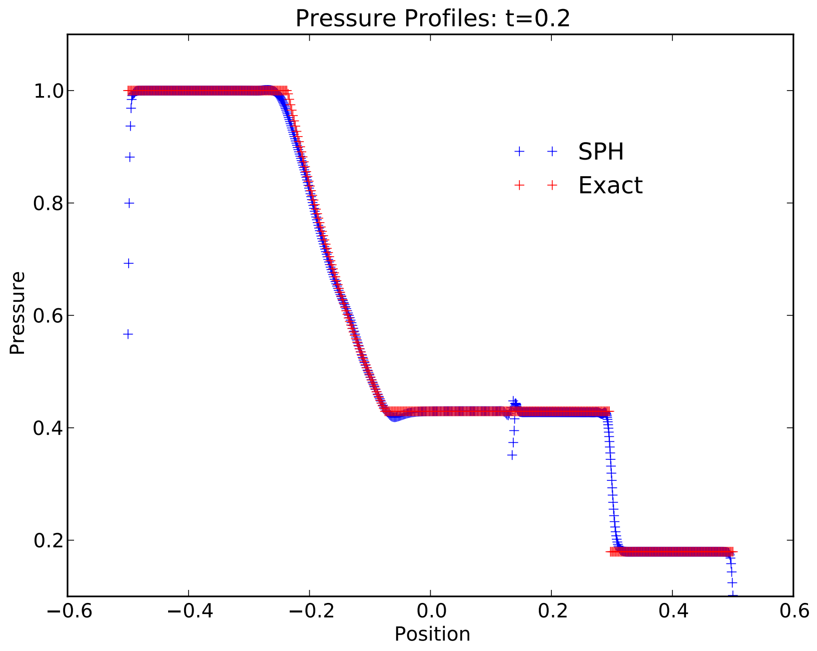


Figure 8: Pressure profiles of SPH and exact Riemann solver at $t = 0.2$.