MONASH University

School of Physics and Astronomy
ASP4200/ASP4020
Due Date: 5pm Thurs 18th March 2021

ASP4200/ASP4020 Computational Astrophysics

Assignment 1

Your mission, should you choose to accept it...

1. Write a one dimensional code to solve the equations of hydrodynamics using the Smoothed Particle Hydrodynamics method
2. Solve for the propagation of a one dimensional isothermal linear wave
3. Solve the Sod shock tube problem and compare your results to the exact solution

I suggest you proceed with the following steps, which will form the marking criterion (1 mark for successful completion of each step):

1. Write a code, using your favourite language, that prints hello world and defines arrays capable of storing position, velocity, mass, smoothing length, density, internal energy (u), pressure and sound speed for $n_{\rm max}$ particles, where $n_{\rm max}$ is a parameter.
2. Add a subroutine called `setup` that sets up a one dimensional array of n particles with positions evenly spaced between $x_{\rm min}$ and $x_{\rm max}$ and assigns a mass to each particle corresponding to a specified initial density ρ_0 and sets the velocity to a sine wave in the x direction with amplitude 10^{-4} times the sound speed. Also define and store a smoothing length h for each particle, set initially to 1.2 times the particle spacing. Set up 100 particles with $x_{\rm min} = 0.0$, $x_{\rm max} = 1.0$ and $\rho_0 = 1.0$. Set an integer n equal to the number of particles you have set up and pass this back out of the routine to your master code. *Hint: do NOT give the number of particles as an input to this routine, it should be an OUTPUT, ensuring that $n \leq n_{\rm max}$.*
3. Write a subroutine called `output` that writes the particle properties to a file. Call this routine after setting up the particles. Using the contents of the output file, plot the initial velocity as a function of position on your particles using a plotting program such as SPLASH. Your output files should be written with a format similar to the following:

```
write(lu,*) '# x,y,z,rest of column labels'
write(lu,*) time
do i=1,n
```

```
   write(lu,*) x(i),y(i),z(i), rest of particle properties
enddo
```

4. Implement a subroutine (preferably in a separate module) called `get_density` or similar that takes the particle positions and masses as input and returns the density computed using the SPH density estimate for each particle a:

$$\rho_a = \sum_b m_b W(|\mathbf{r}_a - \mathbf{r}_b|, h_a) \tag{1}$$

where h_a is the smoothing length for each particle and W is the cubic spline kernel (implement the kernel function itself in a separate routine, which also returns the scalar part of the kernel gradient). Store the density computed for each of your particles and write this to the output file alongside the positions and masses.

5. Implement periodic boundary conditions, so that neighbours can be found "across the box". *The simplest way is to add additional "ghost" particles that are copies of their corresponding particles on the other side of the box. To do this, create a routine called* `set_ghosts` *that sets the positions and particle properties of up* $n_{\rm ghost}$ *virtual particles. Place these particles at the end of the list of real particles, so that particles* $1 \rightarrow n$ *are the real particles, and particles* $n+1 \rightarrow n + n_{\rm ghost}$ *are the ghost particles.*

6. Add a routine called `equation_of_state` that returns the pressure and sound speed given the density. For an isothermal equation of state, set the sound speed to be constant.

7. Add a routine called `get_accel` that returns the acceleration on each particle based on the SPH summation:

$$\mathbf{a}_a \equiv \frac{{\rm d}\mathbf{v}_a}{{\rm d}t} = -\sum_b m_b \left[\frac{P_a + q^a_{ab}}{\rho_a^2} \nabla W_{ab}(h_a) + \frac{P_b + q^b_{ab}}{\rho_b^2} \nabla W_{ab}(h_b) \right], \tag{2}$$

where q^a_{ab} and q^b_{ab} are zero for the time being. Test this routine by ensuring that the acceleration computed on the initial setup is i) the same for all particles and ii) approximately zero. Organise the calls to density, equation of state and acceleration routines into a single `derivs` routine that calls these three in turn.

8. Add a timestepping routine that updates positions and velocities of particles using

the leapfrog integrator in the form:

$$\mathbf{x}^1 = \mathbf{x}^0 + \Delta t \mathbf{v}^0 + \frac{1}{2}(\Delta t)^2 \mathbf{a}^0, \tag{3}$$

$$\mathbf{v}^* = \mathbf{v}^0 + \Delta t \mathbf{a}^0, \tag{4}$$

$$\text{call derivs}(\mathbf{x}^1, \mathbf{v}^*), \tag{5}$$

$$\mathbf{v}^1 = \mathbf{v}^* + \frac{1}{2}\Delta t(\mathbf{a}^1 - \mathbf{a}^0), \tag{6}$$

Do *not* simply write one output file every timestep. Instead, add a code parameter called `dtout` that specifies the time interval between output files. Call your output routine so that a new output file is written once per `dtout` (where $\Delta t_{\rm out}$ is something like 0.05 in code units). I suggest setting `tprint = ifile*dtout` as the time for the next output file, where `ifile` is a counter for the number of files already written. Then simply write a file every time $t > t_{\rm print}$ and increment `tprint` and `ifile` accordingly.

9. Evolve your particles from time $t = 0$ until $t = 5$ wave periods, with a timestep determined by an appropriate stability condition, and plot the resulting velocity on the particles at the end time. Compute and plot the total kinetic energy as a function of time, and use this plot to evaluate the period of your wave and the error with respect to the expected period, to at least two decimal places.

10. Allow the smoothing length to vary by setting the smoothing length for each particle after the density calculation to:

$$h_a = h_{\rm fac} \left(\frac{m_a}{\rho_a} \right)^{1/n_{\rm dim}}. \tag{7}$$

    Implement 'iterations' of the smoothing length by repeatedly calling the density routine followed by evaluation of (7) at least 3 times before calling `get_accel`.

11. Evolve the linear wave using a variable smoothing length. As previously, compute the wave period to at least two decimal places and evaluate the error with respect to the expected answer.

12. Implement an artificial viscosity term in (2) using

$$q^a_{ab} = \begin{cases} -\frac{1}{2}\rho_a v_{{\rm sig},a} \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab}, & \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

    and similarly for q^b_{ab}, where

$$v_{{\rm sig},a} \equiv \alpha c_{{\rm s},a} - \beta(\mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab}) \tag{9}$$

where α and β are input parameters to the acceleration routine with default values of $\alpha = 1$ and $\beta = 2$. Ensure that the sound speed is computed in a way consistent with the equation of state being employed (preferably create a routine in your equation of state module that returns c_s for the chosen equation of state). Rerun your wave test with artificial viscosity and plot the kinetic energy as a function of time, demonstrating that the amplitude of the wave now decays with time.

13. Write a `setup` routine for an isothermal shock tube problem, with particle spacing adjusted so that the density set to 1.0 for $x < 0$ and density set to 0.1 for $x \geq 0$. Use a particle spacing of 0.001 to the 'left' of the shock, and 0.01 to the 'right', with particles placed between $x_{\rm min} = -0.5$ and $x_{\rm max} = 0.5$. *Hint: recall that the number of particles is an OUTPUT from this routine, so let the particle spacing determine the number of particles.*

14. Implement fixed boundary conditions by tagging the first and last $n_{\rm bound}$ particles as 'boundary' particles (choosing e.g. $n_{\rm bound} = 6$) and setting $v = 0$ for these particles.

15. Run the isothermal shock tube to $t = 0.1$ and compare your numerical solution with the exact solution to the Riemann problem (e.g. using the exact solution supplied in SPLASH). *Hint: to get the exact solution plotting working you need the time printed to the header of your files and the columns labelled appropriately as described above.*

16. Implement the evolution of the internal energy in your timestepping routine. In your `get_accel` routine, implement the PdV work term and the heating term from the artificial viscosity:

$$\frac{{\rm d}u_a}{{\rm d}t} = \sum_b m_b \frac{P_a + q^a_{ab}}{\rho_a^2} \left(\mathbf{v}_a - \mathbf{v}_b\right) \cdot \nabla_a W_{ab}(h_a) \qquad (10)$$

Hint: try to only perform ONE loop over the neighbours, where both sums required in the acceleration and thermal energy derivative are computed at the same time, enabling you to re-use quantities already computed between particle pairs.

17. Implement the adiabatic equation of state $P = (\gamma - 1)\rho u$, where γ is an input parameter.

18. Run the Sod (1978) shock tube problem, with initial conditions for $x < 0$ given by $\rho = P = 1.0$ and initial conditions for $x \geq 0$ given by $\rho = 0.125$ and $P = 0.1$ implemented using equal mass particles with a particle spacing of 0.001 for $x < 0$ and 0.008 for $x \geq 0$. Evolve this to $t = 0.2$ using $\gamma = 1.4$.

19. Plot a comparison of your numerical solution compared to the exact solution.

20. Examine the effect of the artificial viscosity parameters α and β on the shock solution. Plot solutions obtained with $\alpha = \beta = 0$, $\alpha = 0$ and $\beta = 2$, and $\alpha = 1$ and $\beta = 2$.

Collate all your plots (e.g. into a single LaTeX document) and submit them electronically along with a tarball of your source code to daniel.price@monash.edu