Solutions to Problem Set 12

Fundamentals of Simulation Methods 8 ECTS
Heidelberg University WiSe 20/21

Elias Olofsson
ub253@stud.uni-heidelberg.de

February 16, 2021

1 SPH simulation of a planet-planet collision system (20 pts)

In the first exercise, we use the supplied Smoothed Particle Hydrodynamics code in `sph_source.py` with the parameters given in the specification, to simulate an approximate model of a single planet in hydrostatic equilibrium. We first generate spatial coordinates $\boldsymbol{x}_0 = (x_0, y_0, z_0)$ according to the normal distribution $\sim \mathcal{N}(0, 1)$ and with stationary initial velocities $\boldsymbol{v} = 0$ for all $N = 800$ particles. Then we simulate the system with an added artifical damping term $\nu = 1$ for time $t < 100$, after which we set it to $\nu = 0$ for $t > 100$ and $t < 170$, and finally re-activate the dampening to $\nu = 1$ for $t > 170$ and the remainder of the simulation until $t_{\max} = 200$. We store the final positions and velocities in `output_task1.npy`.

The reason why the term $|\boldsymbol{v}|_{\texttt{max}}$ does not appear in the expression

$$\Delta t = \mathrm{CFL} \cdot \frac{h}{\max\left(c_s\right)}, \tag{1}$$

which adaptively determines the time step size Δt, is because SPH inherently deals with a Lagrangian formulation of fluid dynamics, where quantities are defined in relation to the bulk motion of the fluid medium instead of a inertial frame of reference. Thus, the derivatives we deal with are typically the material derivative (also known as total, convective or Lagrangian derivative etc.) which specifies rate of change for some quantity in relation to the velocity field of the bulk fluid. I presume this is the reason why $|\boldsymbol{v}|_{\texttt{max}}$ is missing from Eq.(1), since it is uninteresting in our chosen frame of reference.

The effect of reducing the softening length ϵ I believe would reduce the radius of the planet when it reaches hydrostatic equilibrium. Since this parameter ϵ reduces the effect of the singularity in the gravitational force for two particles in close approach to each other, it effectively creates a hard limit on how strong the attractive force between to particles are allowed to grow, which will in turn implicitly define the average particle-particle separation distance within the planet, when the system reaches static equilibrium.

By loading the data in `output_task1.npy` from the final state of the planet at $t = 200$ into `post-processing.ipynb`, we can analyze the the distribution of Mach numbers for each of the particles, and plot it as a function of radial distance from the planet center of mass, as seen in Fig.(1).

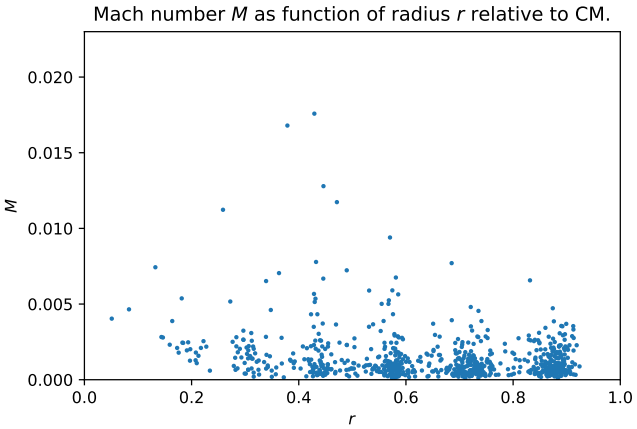

Figure 1 – Distribution of Mach numbers for all 800 particles, as a function of radial distance from the planet center of mass, for a single planet relaxed to approximate hydrostatic equilibrium.

Here in this figure, we can see that the system is very close to a hydrostatic equilibrium. Since the Mach number specifies the ratio of local bulk velocity magnitude to the rate at which local perturbations are transported within the fluid, having small average Mach numbers signifies that particles are slow compared to the bulk fluid motion. Thus, since we have Mach numbers on the order of ~ 0.005, fluid particles are moving very slowly in the final state of our simulation, which is a close approximation to the ideally completely static case we would have in true hydrostatic equilibrium.

Similarly, we can compare the dynamic time scale $t_{\text{dyn}} = R/\langle c_s \rangle$ with the time scale the average particle would need to span the radius of the star $t_{\text{span}} = R/\langle |\boldsymbol{v}| \rangle$, where R is the radius of the star. Calculating these quantities for the final state at $t = 200$ for the simulation, we arrive at $t_{\text{dyn}}/t_{\text{span}} \approx 0.0014$, i.e. perturbations within the fluid propagate much faster than the velocity of the individual fluid elements, which agrees with the conclusion drawn from Fig.(1).

Then, we simulate a planet-planet collision by importing two copies of the saved data from the final state from the previous simulation, and give them slightly different initial conditions such that the two planets are sent on a collision-course with each other. A short video of the collision can be seen at YouTube (`https://youtu.be/6hgo1GgrV1U`).

Making some slight modifications to the provided code enables us to export the total relative energy and plot its time evolution, as seen in Fig.(2).

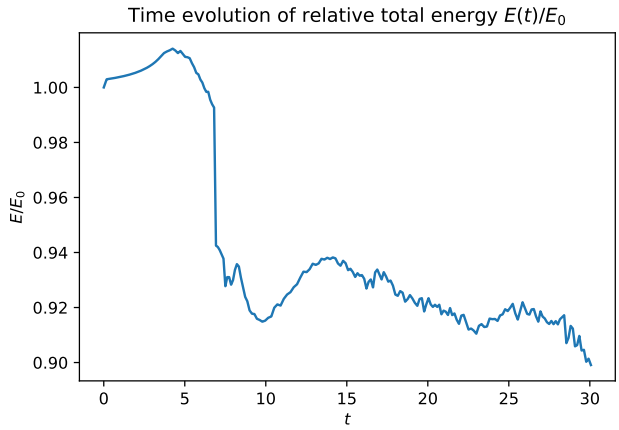

Figure 2 – Total energy evolution for the planet-planet collision relative to the total initial energy of the two planet system.

Here we can see how the relative total energy error evolves in time. First we get a slight increase in total energy as the two bodies approach each other, after which it dramatically drops when the impact occurs. In the aftermath of the collision, we have some slight large amplitude oscillations which gradually fades into higher frequency noise with smaller amplitudes.

I'm not entirely sure on the reason for this deviation of total energy of the system, since SPH is supposed have very good conservative properties. I would speculate that it may be due to the violent collision which quite dramatically alters the initially rather static system, and quite possibly creates shocks and contact discontinuities, which SPH has problems dealing with. Perhaps this simulation would fare better if artificial viscosity was introduced, as discussed in the lecture, to improve the treatment of shocks and discontinuities. Also, I could imagine that the softening parameter ϵ also introduces some error in the energy as well, since it weakly alters the forces, and thus the actual physics, for particles in close approach. So when the two planets collide, many of the fluid particles get pushed very close to each other in the compression during impact, which implies that the effect of the softening length is large at this time. I would argue that this must have a large cumulative effect on the total energy error during the most violent part of the collision.

To improve this given SPH-code, one could swap the Gaussian kernel for a kernel with a finite support, i.e. a kernel for which not all of its domain maps to non-zero values. An example would be the cubic spline shown in the lectures, which gives a zero value for all particle pairs which has a separation distance larger than $2h$, where h is the kernel width. This change has the benefits of reducing the double sum over all particles to a sum over all particles and its immediate neighbours. In other words, the complexity

reduces from $\mathcal{O}(N^2)$ to $\mathcal{O}(NN_{\text{neigh}})$, where the number of neighbours N_{neigh} should be much smaller than the number of particles N for an appropriately chosen kernel width h.

Furthermore, to ensure that this benefit is attained for changing and very different particle densities within the simulation, the kernel width h should optimally be allowed to vary in proportion to the local density. A method to do this is to require that the mass in the kernel volume remains constant, i.e. that $\rho_i h_i^3 = \text{const.}$ for all particles i.

Another technical detail this particular code could improve on is to use vectorization to a larger extent, instead of explicit for-loops. This could yield large performance improvements by exploiting the underlying array programming implemented in python and numpy, where hardware capabilities to perform vector instructions are used to speed up computations with large sets of data elements.

Moreover, to improve the treatment of shocks and discontinuities in the Eulerian fluid, one has to introduce an term with artificial viscosity to the calculation of the acceleration, which is supposed to allow for the necessary dissipation of energy which occurs at shock fronts. Additionally, the introduction of artificial viscosity requires the time integration of one more thermodynamic quantity, which can either be chosen as internal energy or entropy.

Lastly, one could improve the accuracy in the time integration by using a higher order scheme. The Verlet scheme has good and stable characteristics through its symplectic properties, but it is however only second order accurate. Other, higher order symplectic schemes are available, for example R. D. Ruth's third and fourth order schemes from 1983, which would improve on the rate of convergence but also increase the complexity [1]. Below we use the third order scheme, with the other improvements listed above, in a pseudo-code which outlines the general approach one would take in creating such a simulation code. The only thing not captured here is the inclusion of artificial viscosity, since I'm not totally sure on how the time integration of entropy or internal energy should be done correctly, and especially together with the Ruth 3rd order scheme.

```
# Define constants
...
# Define system parameters
...
# Initialize particles; positions, velocities and masses.
x = init_position()
v = init_velocity()
m = init_mass()

# Main loop
t = 0
while t < t_max do
    # Update kernel widths.
    h(t) = get_kernel_width(x(t))

    # Determine size of next time step.
    dt = get_timestep(x(t),h(t))

    # Evolve state in time (ruth3)
    a(t) = get_acceleration(x(t),h(t),m)
    v(t+dt/3) = v(t) - 7/24*a(t)*dt
    x(t+dt/3) = x(t) + 2/3*v(t+dt/3)*dt

    a(t+dt/3) = get_acceleration(x(t+dt/3),h(t),m)
    v(t+dt*2/3) = v(t+dt/3) - 3/4*a(t+dt/3)*dt
    x(t+dt*2/3) = x(t+dt/3) - 2/3*v(t+dt/3)*dt

    a(t+dt*2/3) = get_acceleration(x(t+dt*2/3),h(t),m)
    v(t+dt) = v(t+dt*2/3) + 1/24*a(t+dt*2/3)*dt
    x(t+dt) = x(t+dt*2/3) + v(t+dt)*dt

    # Update time.
    t = t + dt

    # Calculate energies etc.
    E = get_energies(x,v)

    # Output/plot data.
    ...
```

References

[1] Etienne Forest and Ronald D Ruth. Fourth-order symplectic integration. *Physica D: Nonlinear Phenomena*, 43(1):105–117, 1990. URL `https://www.slac.stanford.edu/pubs/slacpubs/5000/slac-pub-5071.pdf`.