Problem Set 12

Exercises for the lecture Fundamentals of Simulation Methods, WS 2020

Prof. Dr. Frauke Gräter, Prof. Dr. Friedrich Röpke
Tutors: Benedikt Rennekamp (benedikt.rennekamp@h-its.org), Fabian Kutzki (fabian.kutzki@h-its.org), Giovanni Leidi (giovanni.leidi@h-its.org), Siddhant Deshmukh (sdeshmukh@lsw.uni-heidelberg.de)
Offices: Heidelberg Institute for Theoretical Studies, Schloss-Wolfsbrunnenweg 35

Hand in until Wednesday, 17.02, 23:59
Tutorials on Friday, 19.02
(Group 1, Giovanni Leidi 09:15)
(Group 2, Benedikt Rennekamp 09:15)
(Group 3, Siddhant Deshmukh 11:15)
(Group 4, Fabian Kutzki 14:15)

---

1. SPH simulation of a planet-planet collision system [20 pt.]

For this last problem set you are going to run a SPH code to simulate a collision between two planets. The source code (`sph_source.py`) is available on the course web page.

In order to run this Python code you have to install the `numba` package, either through `conda` or `pip`. You can find further instructions for the installation on:
`https://numba.pydata.org/numba-doc/latest/user/installing.html`
In case you cannot manage to install `numba` on your machine, you can work with the `sph_source_numpy.py` already available on the course web page. For that you will just need the `numpy`, `matplotlib` and the `math` packages (it's just 2-3 x slower).

This code is a basic implementation of the SPH algorithm with the following methods:

- polytropic equation of state ($\gamma = 2$)
- Gaussian kernel to interpolate physical quantities among particles
- constant kernel width h
- direct summation algorithm to get the gravitational forces among particles
- Leapfrog time marching scheme to evolve the space and velocity coordinates in 3D.
- adaptive time stepping according to the sonic CFL condition:

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

In order to simulate a collision between two planets, first you need to get an approximate model of a planet in *hydrostatic equilibrium* (HSE). HSE describes the local balance of pressure and gravitational forces ($\vec{\nabla} P = \rho \vec{g}$), so no motions are present in the system aside from a bulk velocity. However, the random placement of few SPH particles does not guarantee HSE and there is no obvious way how to evenly distribute the particles in a spherical object. This is why the initial conditions are relaxed to damp out spurious velocities that arise from the setup. In order to do so, a damping term ($\sim -\nu \cdot \mathbf{v}$) is added to the acceleration.
Use the following set of parameters:

- N = 800
- h = 0.1
- $\nu = 1$
- K = 0.1
- $\epsilon = \frac{h}{100}$
- CFL = 0.8
- G = 0.1
- $m_p = 0.0025$
- $\gamma = 2$

You will find a description of each parameter in the Python script.

1. Generate the spatial coordinates x_i ($i = x, y, z$) of each particle according to a normal distribution $\sim N(0, 1)$. Set each velocity component $v_i = 0$. For $t > 100$ set $\nu = 0$[1] and for $t > 170$ reset $\nu = 1$. Run the simulation until $t_{max} = 200$ and store the final positions and velocities[2] **[3 pt.]**

2. Why doesn't $|\mathbf{v}|_{max}$ appear in Eq. 1 contrary to what seen in the Eulerian finite-volume algorithm? **[1 pt.]**

3. What would be the effect of increasing or decreasing the value of the *softening length* ϵ on the evolution of the system? **[2 pt.]**

4. Load the data from the output file[3] and create a scatter plot of the Mach number ($M_i = \frac{v_i}{c_s}$, $i = x, y, z$) as a function of radius[4]. Explain what you see and why this plot may be a good way to check whether the system is approximately in HSE or not. Hint: compare the dynamical time scale $\frac{R}{< c_s >}$ to the time scale required by velocity fluctuations to span the radius of the star R: $\frac{R}{< |\mathbf{v}| >}$, where $< . >$ is some average value across the system. **[3 pt.]**

[1] The proper way to do this is `pars[i_nu]=0`.
[2] To store the output: `numpy.savez('output.npz', x=x , v=v )`
[3] To load the data frame: `data=numpy.load('output.npz')`. To get the variables stored in `data`: `x=data['x']` and `v=data['v']`
[4] $c_s = (\gamma \frac{P}{\rho})^{\frac{1}{2}}$. You will need to recompute the density and the pressure of each particle in post-processing using the formula you saw in class. You can also have a look at how these quantities are computed in the Python script

You can now simulate a collision between 2 identical planets. For that you need to double the number of particles, so N=1600, and set $\nu = 0$.

1. Load the positions and velocities for the relaxed planet that you just stored, and assign these values to the first 800 elements of the new position and velocity arrays. Do the same for the last 800 elements of the arrays, but this time add an offset of the type: `x=x+3`, `y=y+1.5`, v_x `=` v_x`-0.3`. Let the system evolve until $t_{max} = 30$. **[2 pt.]**

   If you want to get the animation for the evolution of the system, try to plot the particle positions in the x-y plane at each time step and save the plot in `.jpg` or `.png` format (e.g. `sph_000.jpg`, `sph_001.jpg` etc.). You can then make a movie using `ffmpeg`:
   ```
   ffmpeg -r 24 -i sph_%03d.jpg x_y.mp4
   ```

2. Plot the time evolution of the total energy of the system. Remember that:

$$E_{\rm TOT} = E_{\rm GRAV} + E_{\rm KIN} + E_{\rm INT}. \tag{2}$$

   The internal energy for particle i is:

$$e_i = m_i \frac{P_i}{\rho_i(\gamma - 1)} \tag{3}$$

   How large is the relative error? What happens at later times? **[2 pt.]**

3. How would you improve this simple SPH code based on what you have learned in class? (Hint: Look at some of the bullet points on the first page describing the methods that are used in the code). Write a *pseudo-code* in which you describe the main steps of the new algorithm as in a cooking recipe, starting from the setup of the initial conditions until the end of the time loop. **[7 pt.]**