Solar System: Unit 1: Basic Functions

Computer Modelling

4pm Monday 15th January 2024

Aims

Your task for this first unit of the project is to write two basic functions that will form part of the rest of your simulation code.

The two functions are:

1. `compute_separations(particles)`
2. `compute_forces_potential(particles, separations)`

Each is described in a section below.

Read the background information before trying this.

You can test your functions with code like this:

```
particles = [
    Particle3D("A", mass_a, position_a, velocity_a),
    Particle3D("B", mass_b, position_b, velocity_b),
    Particle3D("C", mass_c, position_c, velocity_c),
]
separations = compute_separations(particles)
forces, potential = compute_forces_potential(particles, separations)
```

You will need to choose values for the particle masses, positions, and velocities that will help you see if your functions are working correctly. Choose something simple, and print out the separations, forces, and potential so you can check if they are right - think about what values they should have.

1 Computing separations

Your first function, `compute_separations(particles)`, will compute the vectors between particles, for later use in calculating forces.

1.1 Function inputs and outputs

Kind	Name	Type	Meaning
Input	`particles`	list	Particles whose separations we want.
Output	`separations`	3d array	Vector separation of each pair of particles.

The particle list input will be a list of `Particle3D` objects. You can create one as shown in the previous section.

The output, `separations`, should be a 3D numpy array of shape $(n, n, 3)$, where n is the number of particles. The element (i, j, k) of the array should contain the k'th component (0=x, 1=y, 2=z) of the vector separating particle i and particle j, where i and j are the indexes of particles in the list.

You should make this function faster by noticing that the vector from $\boldsymbol{a}$ to $\boldsymbol{b}$ is the negative of the vector from $\boldsymbol{b}$ to $\boldsymbol{a}$.

2 Computing forces and potential

Your second function, `compute_forces_potential(particles, separations)`, will compute the gravitational forces on the particles and the total gravitational potential energy of the system.

2.1 Function inputs and outputs

Kind	Name	Type	Meaning
Input	`particles`	list	Particles in the system.
Input	`separations`	3d array	Vector separation of each pair of particles.
Output	`forces`	2d array	Force on each particle.
Output	`potential`	float	Total system potential energy.

The `particles` input should be the same as the input to the `compute_separations` function. The `separations` input should be the output from that same function.

The first output, `forces`, should be an array of shape $(n, 3)$ containing the forces where the (i, j) element should be the j'th component (0=x, 1=y, 2=z) of the total force vector on the i'th particle.

The second output, `potential`, should be a single float value, which should be the total potential energy of the system.

You should make this function faster using Newton's Third Law, and must use the units specified in the background information.

3 Submission

3.1 Submitting

Submit a single file through Learn by the date at the top of this page. Successful submissions are emailed a receipt.

The marker should be able to import your file and use the functions in it without it doing anything else, like prompting for user input. If your code does anything like that then remove it before submitting.

Follow the instructions in the background information document to submit.

3.2 Marking Scheme

This assignment counts for 10% of your total course mark.

1. `compute_separations` returns the appropriate shape array [**2**]
2. `compute_separations` all array values are correct [**4**]
3. `compute_forces_potential` returns the appropriate type of both outputs, array and float [**1**]
4. `compute_forces_potential` returns the correct force array [**3**]
5. `compute_forces_potential` returns the correct potential [**2**]
6. The code is good quality, well-organized and with sensible naming, layout, and comments/docstrings [**6**]
7. The code is efficient and uses the speed-ups mentioned above. [**2**]

Total: 20 marks.