Week 8 — *Generic Particle Method Program*
Kepler orbits

The goal of the present exercise is to setup the first stone for our particle method program.
The documentation for the particle's code is provided on the Moodle. Please refer to the documentation to understand the structure of the code, class hierarchy and methods to implement.

Exercise 1: ***Input/Output object***

In order to read and write sets of particles we need to implement methods that read from files or write to files.

1. Implement the missing parts of the object

   ```
   class CsvReader
   ```

   which wants to read some particle characteristics from a file. Each line in the file will have the format

   ```
   x y z vx vy vz fx fy fz mass radius name
   ```

   To achieve this, use an ifstream object, read line by line using the function getline.

   ```
   while (is.good()) {
     getline(is, line);

     ...

   }
   ```

   For each line construct a stringstream and use the operator » towards a newly instantiated *Particle* object. For this to work as expected you have to implement the methods

   ```
   void Particle::initself(std::istream & is);
   void Planet::initself(std::istream & is);
   ```

   Finally insert the created particle to the system.

2. Implement the class

   ```
   class CsvWriter
   ```

   Each line in the produced file will have the format

   ```
   x y z vx vy vz fx fy fz mass radius name
   ```

   To achieve this, loop over all the particles stored in the *particles* member and use « operator towards an adequately opened files. Again you have to implement the methods

   ```
   void Planet::printself(std::ostream & stream) const;
   void Particle::printself(std::ostream & stream) const;
   ```

3. Construct an input file with your favorite text editor and test that your routines work as expected.

Exercise 2: ***Computing the gravitational forces***

For any particle i in the set, the force felt due to the gravitational field follows :

$$\boldsymbol{f}_i = \sum_{j \neq i} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \frac{\mathcal{G} m_i m_j}{r_{ij}^2}$$

where we have $\boldsymbol{r_{ij}} = \boldsymbol{p}_j - \boldsymbol{p}_i$ is the vector from particle i towards particle j with the distance being $r_{ij} = ||\boldsymbol{r}_{ij}||$, m_i is the mass of particle i and $\mathcal{G}$ is the gravitational constant.

1. Implement the computation of the force in the class

```
class ComputeGravity
```

2. What is the complexity of the computation of forces for all the particles ?

Exercise 3: ***Time integration***

In order to make the particles evolve with time an explicit scheme is needed:

1. $\boldsymbol{v}_i += \frac{dt}{2m_i} \boldsymbol{f}_i$
2. $\boldsymbol{p}_i += dt \boldsymbol{v}_i$
3. Compute the forces $\boldsymbol{f}_i$
4. $\boldsymbol{v}_i += \frac{dt}{2m_i} \boldsymbol{f}_i$

The purpose of the *ComputeVerletIntegration* is to perform this scheme.

1. Implement the class

```
class ComputeVerletIntegration;
```

Exercise 4: ***Kepler orbits***

Following *Kepler* theory, the two body gravitational problem yields a conic section trajectory with an eccentricity changing from ellipse to parabola and hyperbola. (see http://en.wikipedia.org/wiki/Kepler_orbit).

1. In the main you can implement the creation of the integration scheme and/or the force computation objects.
2. In the main you can implement the iteration loop, with CSV outputs made at a chosen frequency.
3. Set an input file with two particles of mass $m_1 = 1.$ and $m_2 = 1.$ with initial states being
   - $p_1 = (0, 0, 0)$ and $v_1 = (0, 0, 0)$
   - $p_2 = (1, 0, 0)$ and $v_2 = (0, 1.2, 0)$
4. Modify the $Verlet$ integrator so as to fix the first particle (which will play the role of the sun). Run the code with a gravitational constant of 1, a timestep dt of 5.10^{-3} and during 1000 steps.
5. Observe the trajectory using matplotlib or paraview. What is the trajectory like ?
6. Modify the initial condition to yield a different trajectory type.
7. Once everything is working, please decide what should the *evolve* method of the class *SystemEvolution* should contain. Adapt the code in consequence.