THE SOLAR SYSTEM: AN EXERCISE IN NUMERICAL INTEGRATION

FYS3150: COMPUTATIONAL PHYSICS

SEBASTIAN G. WINTHER-LARSEN
TRYGVE L. SVALHEIM
GITHUB.COM/GREGWINTHER

Abstract. Two, three, multi. With Euler-Cromer and then Velocity Verlet

Contents

Date: October 24, 2016.

1. Introduction

2. Theory

2.1. **Newtonian gravtiation.** When one studies the solar system and the way it moves, one inevitably needs to consider the gravity of the situation. The classical law of gravitation is given by Newton's law

$$F_G = \frac{GM_1M_2}{r^2}, \tag{1}$$

where F_G is the gravitational force between to bodies, G[1] is the gravitational constant, M_1 and M_2 are the masses of the two bodies and r is the distance between them.

In three dimensions, by employing Newton's second law of motion, one will get the following componental equations for the acceleration due to gravitational pull on a particular body i.

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_i}, \quad \frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_i}, \quad \frac{d^2z}{dt^2} = \frac{F_{G,z}}{M_i}, \tag{2}$$

each of which can be integrated in order to find the position of the bodies at any given time and for a particular inital velocitiy and position. Moreover, a neat vectorial expression for the Newtonian gravity (equation 1) in three dimensions is

$$\mathbf{F}_G = \frac{GM_1M_2}{r^3}\mathbf{r} \tag{3}$$

where $\mathbf{r}$ gives the posistion of the body in a cartesian coordinate system.

2.2. **Relativity.** The theory of general relativity (GR), put forth by Albert Einstein in 1915, is the current description of gravity in modern physics. On curiosity of the solar system that cannot be explained is the perihelion precession of Mercury's orbit. For every Mercury year the perihelion of the planets orbit moves sligthly. The observed value of this effect is 43 arc-seconds per century. Einstein showed himself that GR could explain this anomaly.

Closed elliptical orbits are a special feature of the factor $(1/r^2)$ in Newton's law of gravitation. Any change to this factor will result in a change in the orbit. For a small such correction, each orbit will be almost the same, but as time progresses the orientation of the elliptical orbit will itself rotate. In this study we will not compute a space-time manifold in order to make this correction, but rather add a general relativistic error term to the Newtonian gravitational force (equation 1.

$$F_G = \frac{GM_\odot M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right], \tag{4}$$

where $M_{MERCURY}$ is the mass of Mercury, r is the distance between Mercury and the Sun, $l = |\mathbf{r} \times \mathbf{v}|$ is the orbital angular momentum per unit mass, and c is the speed of light in vacuum.

3. Units

In order to make the system easy to work with and the calculations easier as well, a change of units is warranted. In this study we will therefore express the mass of any body as a fraction of solar masses, that is $M_\odot = 2 \times 10^{30} kg$. This means that the Earth[2], for instance will have a mass of $M_{Earth} = 3 \times 10^{-6}$.

Additionally, the units for distance will be astronmical units AU. This is the mean distance between the earth and the sun. For a simple system and a particular

[1] $G = 6.67408 \times 10^{-11} m^3 kg^{-1} s{-}2$

[2] In SI-units, $M_{Earth} \approx 6 \times 10^{24} kg$

coordinate system with the sun at origin, the initial position for the Earth could simply be $\mathbf{r}_{Earth} = (1, 0, 0)$.

The unit for time will be Earth years, yr. This means that velocity will be in units AU/yr.

Changing the variables will have consequences for the gravitationan constant G and velocities v as well. This can be deduced easily enough by picking a sample system consisting of the earth and the sun. Inserting in equation 1 gives

$$F_G = \frac{GM_\odot M_{Earth}}{r^2} \tag{5}$$

Furthermore, assuming the orbit of the earth is perfectly circular, will give the following relation

$$F_G = \frac{M_{Earth} v^2}{r} \tag{6}$$

Combining equations 5 and 6 yields the following

$$v^2 r = gM_\odot = 4\pi^2 AU^3 yr^{-2} \tag{7}$$

Because the mass of the sun as a fraction of solar masses is $M_\odot = 1$ and the distance between the earth and the sun is $r = 1AU$ we land at

$$G = 4\pi^2 AU^3 yr^{-2} \quad v_{Earth} = 2\pi AU yr^{-1} \tag{8}$$

4. Algorithms and implementation

4.1. **Object orientation.**

Object orientation allows for more general code to be written and for easy reuse. Moreover, object orientation makes sense for humans, who tends to classify objects we see and interact with in everyday life. In this study we make use of all the advantages of object oriented programming by implementing several classes. A class diagram illustration of the program is included in appendix A. Bear in mind that this diagram does not represent the functionality of the program to a full extent, as many methods and a few classes are left out. Rather, it helps one to understand how the class hierarchy is built up and how the program as whole is supposed to function.

The System class contains all the information about the initial conditions of the system, which integration method should be used and also some helpful methods, like one that writes data to file.

The InitialCondition superclass contains a method for setting up particles. Depending on what system one wants to model, we have implemented three subclasses of IntialCondition: TwoBody, ThreeBody and MultiBody. The name of these classes speek mostly for themselves.

The intersting objects, contained within the InitialCondition subclasses are instances of the Particle class. The class could have another name, like "planet" or "body" because instances of it will represent the largest bodies in the solar system; the sun and (eventually) all nine planets[3]. Every particle instance takes a position vector, a velcity vector and a mass as inputs and has only these attributes[4]. The extra abstraction to "particle" if therefore fitting, and the class could be employed elsewhere, for instance in a molecular dynamics simulation.

The Integrator superclass har two subclasses EulerCromer and VelocityVerlet each of which must owerwrite the integrateOneStep method. These are the most important part of the program, because it is where the essential part of the two algorithms are implemented. The integrateOneStep method is called iteratively for a specified number of steps from a class instance of the system class, hence the name

[3]Including Pluto for historical reasons.

[4]The position and velocity vectors are defined by its own class Vec3.

of the method. Within the integrateOneStep method, the integration is performed for every particle in the system. A further description of the integrator algorithms follows.

4.2. **Euler-Cromer.** Analytically the Euler-Cromer method, or the semi-implicit Euler method, can be expressed in one dimension by the following recursive relations

$$v_{n+1} = v_n + a_n dt \tag{9}$$

$$x_{n+1} = x_n + v_{n+1} dt, \tag{10}$$

where a_n is the acceleration, v_n is the velocity and x_n is the position, after a certain number of steps n. dt is the time step and $t_n = t_0 + ndt$ is the time after n steps. One sees that the algorithm is relatively cheap, with $4n$ FLOPS. In C++ code the integrateOneStep method of the EulerCromer class looks like this

```
void EulerCromer::integrateOneStep(std::vector<Particle*>
    particles) {

  m_system->computeForces();

  for (int i=0; i<particles.size(); i++) {
    Particle *p = particles.at(i);

    // Acceleration vector
    vec3 a = (p->getForce()) / (p->getMass());

    // Velocity update
    p->getVelocity() += m_dt*a;

    // Position update
    p->getPosition() += m_dt*p->getVelocity();
  }
}
```

Notice that the forces used to compute the acceleration are computed within the System class and that the method uses three-dimensional vectors for more realistic results.

The Euler Cromer method is a first-order integrator which means that it commits a global error of the order of dt. This means that by decreasing the time step one will get a more accurate result.

4.3. **Velocity Verlet.** The Verlet method is an integration method designed to integrate Newton's equation of motion. The method provides good numerical stability and more precise results compared with a regular method like the Euler-Cromer method. The reason for this is that it is symplectic, which means that it conserves state-space volume[5].

The velocity Verlet method can also be represented by a (one-dimensional) recursive relation

$$x_{n+1} = x_n + v_n dt + \frac{1}{2} a_n dt^2 \tag{11}$$

$$v_{n+1} = v_n + \frac{a_n + a_{n+1}}{2} dt. \tag{12}$$

[5]One would probably read an entire book to understand this, so we will leave it at that.

These equations tell us that the algorithm is slightly more expensive compared to the Euler-Cromer scheme at $11n$. Here follows an implementation of the algorithm in C++.

```
void VelocityVerlet::integrateOneStep(std::vector<Particle
   *> particles) {

  m_system->computeForces();

  for (int i=0; i<particles.size(); i++) {
    Particle *p = particles.at(i);

    // Acceleration vector
    vec3 a = (p->getForce()) / (p->getMass());

    // Position update
    p->getPosition() += p->getVelocity()*m_dt + 0.5*a*m_dt*
        m_dt;

    m_system->computeForces();

    // New accelration vector
    vec3 anew = (p->getForce()) / (p->getMass());

    // Velocity update
    p->getVelocity() += (a + anew)*m_dt/2;
  }
}
```

One sees that it is necessary to compute forces and update acceleration once more in order to update the velocity, which means more operations must be conducted. Time will show if the extra cost of this algorithm is worth the increase in precision it promises.

5. Data

As we wish to determine the orbits of our solar system as accurately as possible, we use date from HORIZON Web-Interface, provided by the Jet Propulsion Laboratory (NASA) at the California institute of technology: `http://ssd.jpl.nasa.gov/horizons.cgi`. Here of data from several celestial bodies can be downloaded. It is simple to find both position and velocity for all particles in our bodies at high precision. Units of length can be set to *AU* and velocities to *AU* per day. The velocities must be converted for our simulation. We have faith that these data are accurate.

6. Results

The object-oriented model for the solar system that has been built is very flexible, as the reader will soon come to know. We start by constructing some simple models with two and three bodies, then the entire solar system, and lastly, simulate the perihelion precession of Mercury, as discussed in the theory section.

6.1. **Two-Body system: Earth and Sun.**

6.2. **Three-Body system: Earth, Sun and Jupiter.**

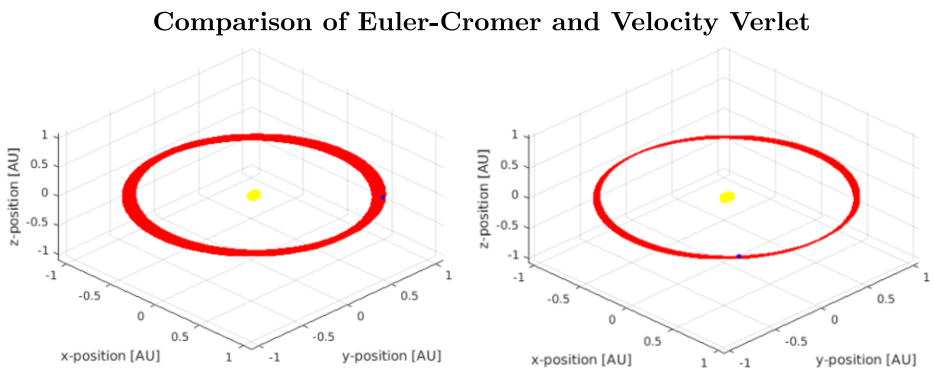


FIGURE 1. Figure showing the difference in precision between the Euler-Cromer method (left) and the velocity Verlet method (right). The time step is $dt = 10^{-2}$ and the number of points is $N = 10^6$

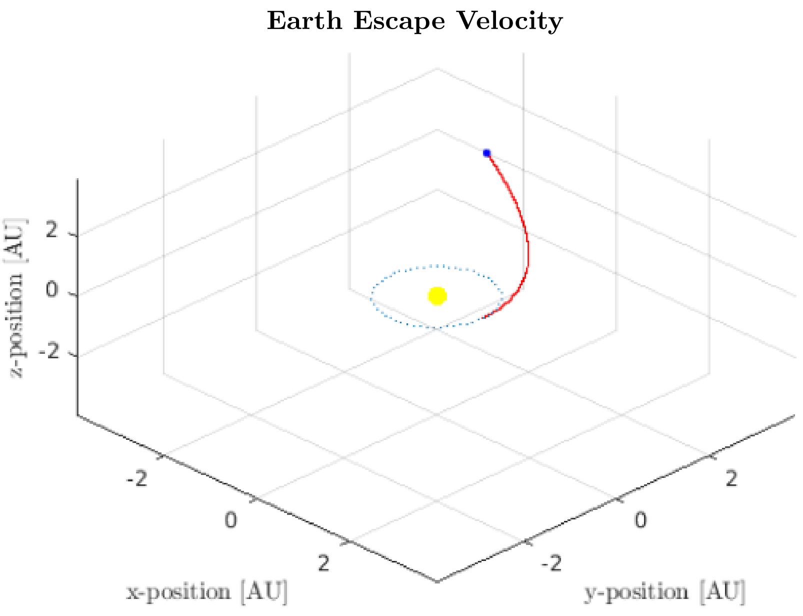


FIGURE 2. Illustration of Earth escaping orbit at $\sqrt{8}\pi AU/yr$

6.3. **Multi-Body system: all planets.**

6.4. **The perihelion precession of Mercury.**

7. Discussion

8. Conclusion

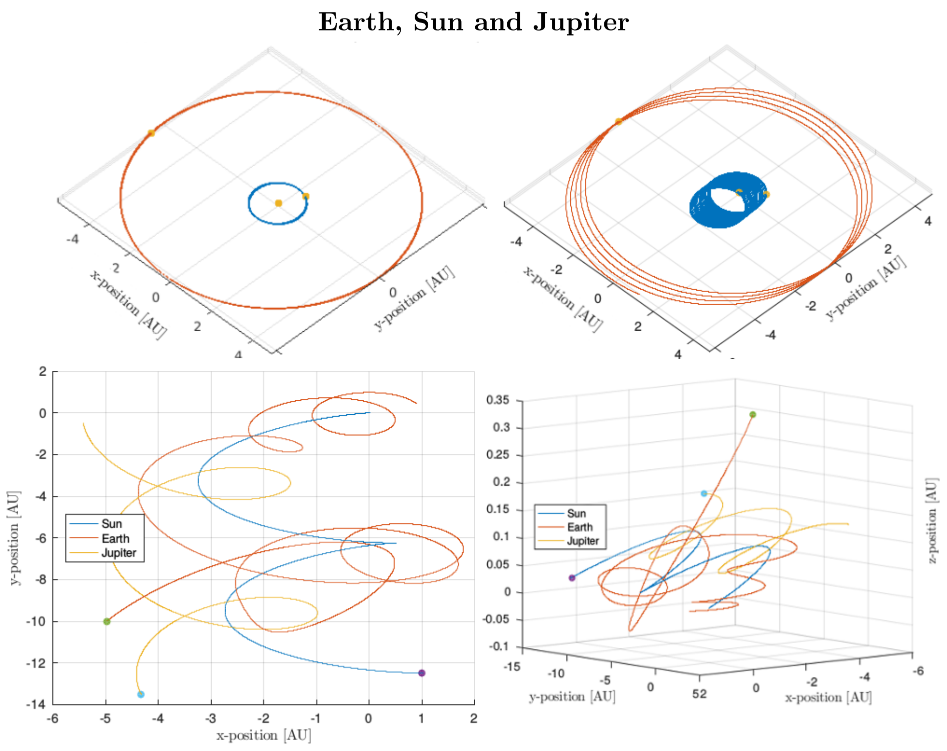
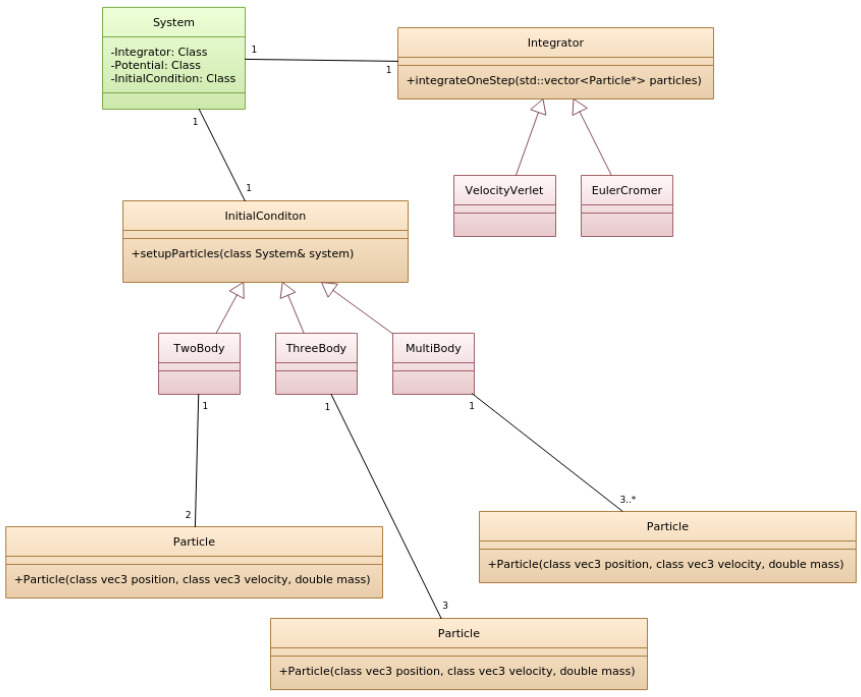


FIGURE 3. Model outcome for a Jupiter with different masses. Top left: Normal mass. Top right: Jupiter mass 10× normal. Bottom: Jupiter mass 1000× normal, in 2D (left) and 3D (right)

APPENDIX A. CLASS DIAGRAM


System
-Integrator: Class
-Potential: Class
-InitialCondition: Class
1
1
Integrator
+integrateOneStep(std::vector<Particle*> particles)
VelocityVerlet
EulerCromer
1
1
InitialConditon
+setupParticles(class System& system)
TwoBody
ThreeBody
MultiBody
1
1
1
2
Particle
+Particle(class vec3 position, class vec3 velocity, double mass)
3
Particle
+Particle(class vec3 position, class vec3 velocity, double mass)
3..*
Particle
+Particle(class vec3 position, class vec3 velocity, double mass)