FYS3150 - Project 3

Daniel Heinesen, Gunnar Lange

October 22, 2016

Abstract

We present numerical models of our solar system with varying levels of complexity.

Contents

1 Introduction

The n-body problem is a recurring theme in many physics publications (see for example **HERE**), due to its wide applicability in many different fields, such as **REFEREENCE**. Analytic solutions of these problems are notoriously hard to come by and are, in most situations, unattainable. Therefore, these problems are almost always solved by use of numerical techniques. We present two different numerical methods for solving the ODE's resulting from Newton's gravitational laws - Euler's method and the velocity-Verlet method. We investigate the stability of both methods for varying complexity of the system.

2 Theoretical model

In this section, we introduce the quantities and equations required to model the solar system. We begin by introducing the relevant equations, and subsequently discuss how to solve these numerically. We continue with a brief discussion of the units used, followed by a discussion of tests that we can implement to test our numerical results. We conclude with a brief diversion to relativistic effects in the solar system.

2.1 Newtonian Gravity

Newton's general law of gravitation, which describes the force of gravity between two objects, is given by:

$$\vec{F}_G = \frac{Gm_1m_2}{r^3}\vec{r} \tag{1}$$

Here F_G is the force of gravity on the first mass, G is the universal gravitational constant, m_i are the masses of the two objects, and $\vec{r}$ is the vector pointing from m_1 to m_2. This law can be combined with Newton's second law of motion, to give a set of coupled differential equations:

$$\begin{aligned} \frac{d^2x}{dt^2} &= \frac{F_{G,x}}{m_1} \\ \frac{d^2y}{dt^2} &= \frac{F_{G,y}}{m_1} \\ \frac{d^2z}{dt^2} &= \frac{F_{G,z}}{m_1} \end{aligned} \tag{2}$$

Where $F_{G,x}, F_{G,y}$ and $F_{G,z}$ are the components of the gravitational force in the x, y and z directions respectively. These equations can be reduced to a set of first-order equations by introducing the velocity components, v, such that:

$$\frac{dx}{dt} = v_x, \qquad \frac{dv_x}{dt} = \frac{F_{G,x}}{m_1} = a_x(x, y, z) \tag{3}$$

And equivalently for the y and z direction. This gives six coupled, first-order, linear, differential equations.

2.2 Implementing the initial conditions

To find a unique solution of the set of equations in section 2.1, we require some initial conditions, i.e. some $x(t = 0)$, $v_x(t = 0)$, and equivalently for the y and z direction. For our first model (described in section 3.1.1) we will, for simplicity, let the earth be 1 AU away from the sun, with an initial velocity that ensures a circular orbit. We show in appendix A that this results in the requirement that the initial velocity be given by:

$$v_0 = \sqrt{\frac{GM_\odot}{r}} \tag{4}$$

Where $M_\odot$ is the mass of the sun and r is the distance to the sun.

For the subsequent models, with multiple planets, we will use the initial positions and velocity provided on NASA's webpages[1], letting $t = 0$ correspond to **INSERT SOMETHNG HERE**.

We want to ensure that the center of mass of our solar system is stable at the origin. On NASA's webpage, we can choose the coordinate origin, however, it is not entirely clear if the velocities have been adjusted so that the center of mass is at rest. Therefore we adjusted the position and velocity of the sun to accommodate this. The position vector of the center of mass, $\vec{R}_{COM}$, is found from:

$$\vec{R}_{COM} = \sum_i m_i \vec{r}_i$$

The sum is over all bodies. $\vec{r}_i$ is the position vector of body i, and m_i is its mass. As the sun is by far the heaviest mass in the system, we can ensure that $\vec{R} = \vec{0}$ by letting the position vector of the sun, $\vec{r}_\odot$ be given by:

$$\vec{r}_\odot = -\frac{\sum_j m_j \vec{r}_j}{M_\odot}$$

Where the sum is over all bodies *except the sun*. We can do the same thing for the velocity of the center of mass, $\vec{V}_{COM}$, which is given by:

$$\vec{V}_{COM} = \sum_i m_i \vec{v}_i$$

Where $\vec{v}_i$ is the velocity of body i. We can then let the velocity of the sun, $\vec{v}_\odot$ be:

$$\vec{v}_\odot = -\frac{\sum_j m_j \vec{v}_j}{M_\odot}$$

Where the sum is again over all bodies *except* the sun. This ensure that the center of mass of our system is fixed at the origin.

2.3 Discretizing the equations

We will solve the equations in 2 numerically. Therefore we require discretized versions of the these equations. We want to simulate from $t = t_0$ to $t = t_f$, and choose N evenly spaced points in this interval. This gives us a timestep, h, of:

$$h = \frac{t_f - t_0}{N}$$

Let now $t_i = a + ih$, where i goes from 0 to N. Furthermore, let $x_i = x(t_i)$ and $v_{x,i} = v_x(t_i)$, and equivalently for the y and z direction. Finally, discretize the acceleration as $a_{x,i} = a_x(x_i, y_i, z_i)$. This allows us to discretize the derivatives in equation 3. We will do this in two separate ways; with the forward Euler algorithm and with the Velocity-Verlet algorithm.

2.3.1 Euler's forward method

Euler's forward algorithm amounts to discretizing the derivatives from section 2.1 as:

$$\frac{dg}{dt} \approx \frac{g_{i+1} - g_i}{h} \tag{5}$$

This approximation is based on Taylor-expanding the function $g(t)$ to the second order, as explained **HERE**. Inserting this approximation for the derivatives in equation 5, and rearranging some of the terms, gives:

$$\begin{aligned} v_{x,i+1} &= v_{x,i} + h a_{x,i} \\ x_{i+1} &= x_i + h v_{x,i} \end{aligned} \tag{6}$$

[1] http://ssd.jpl.nasa.gov/horizons.cgi#top

And again equivalently for the x and y direction. This is the Euler forward algorithm. Combining this with the initial conditions, $\vec{v}(t_0) = \vec{v}_0$ and $\vec{x}(t_0) = \vec{x}_0$, makes it possible to solve the equations of Newtonian gravity. Closer inspection (through Taylor expansion, such as found **HERE** here), reveals that the local error is proportional to h^2, whereas the global error is proportional to h. Thus, whilst this is a straightforward algorithm, the errors quickly accumulate. It is therefore of interest to investigate a slightly improved algorithm:

2.3.2 Velocity-Verlet Algorithm

The Velocity-Verlet algorithm is based on the idea of Taylor-expanding the acceleration, in addition to the velocity and position. The details can be found in appendix B, but the method produces the following equations:

$$\begin{aligned} x_{i+1} &= x_i + hv_{x,i} + \frac{h^2}{2}a_{x,i} \\ v_{x,i+1} &= v_{x,i} + \frac{h}{2}\left(a_{x,i+1} + a_{x,i}\right) \end{aligned} \tag{7}$$

And similarly for the x and the y equations. Note that a_{i+1} depends on x_i, y_i and z_i, and therefore all three position coordinates must be computed before we can compute the velocity. As shown in our analysis in the **APPENDIX**, the local error is proportional to h^3, and in turn the global error is proportional to h^2, as shown **HERE**.

Notice that a_{xi+1} can be reused in the next step, so this method does not incur any additional calls to the acceleration function in the main loop, as compared to the Forward Euler algorithm. The method only costs a few more FLOPs;

- We must perform two additional multiplications for each dimension, specifically $(h^2/2) \cdot a_{x,i}$ and $(h/2) \cdot a_{x,i+1}$
- We must perform two further additions per dimension, namely adding $(h^2/2)a_{x,i}$ and $(h/2)a_{x,i+1}$ to respectively x_{i+1} and $v_{x,i+1}$.

This gives four additional FLOPs per timestep and dimension, i.e. 12 FLOPs per timestep. Thus if we have N points in the main-loop, this gives an additional $12N$ FLOPs. Note that $h^2/2$ and $h/2$ can be precomputed, and therefore do not contribute to the number of FLOPs in the main loop.

2.4 Choice of units

As we will be modelling our solar system, a natural unit of length is the average distance between the earth and the sun, the astronomical unit, AU. In SI-units, $1\text{AU} \approx 1.496 \cdot 10^{11}\text{m}$. As the most massive object in the solar system is the sun, we choose one solar mass, $M_\odot$ as our mass unit. In SI units, $M_\odot \approx 1.99 \cdot 10^{30}$. We will use the mean tropical year, which is about 365.24 days of 864000 seconds**REFERENCE HERE**. This choice of units has one considerable advantage - it simplifies the expression for the gravitational constant, G. As shown in appendix A, with this choice of units the gravitational constant becomes:

$$G = 4\pi^2 \text{ AU}^3\text{yr}^{-2}\text{M}_\odot^{-1}$$

As discussed in section 2.7, we will also use the speed of light. With our units, this is given by:

$$c = 63\ 198 \text{ AU/yr}$$

2.5 Conserved quantities in the system

To check the consistency of our results, we will investigate whether certain quantities, which we expect to be conserved, are indeed conserved.

2.5.1 Conservation of mechanical energy

As there are no external forces in the system, we expect the mechanical energy to be conserved. There are two kinds of energy to consider in our system: kinetic and potential energy. The kinetic energy of a single body is simply given by:

$$E_k = \frac{1}{2}mv^2$$

Where m is the mass of our celestial body and v is the absolute value of its velocity. This is the standard formula for the kinetic energy. The potential energy can be found by integrating the force, choosing infinity as a reference point, i.e. $U(\infty) = 0$:

$$U = -\int_r^\infty \frac{GMm}{r'^2}dr' = -\frac{GMm}{r}$$

This is the potential energy between the bodies. Thus the total energy (which should be conserved) is given by:

$$E_{tot} = \sum_i \frac{1}{2}m_i v_i^2 - \sum_{i<j}\sum_j \frac{Gm_i m_j}{r_{ij}} \tag{8}$$

Where i runs over all the bodies and r_{ij} is the norm of the distance between body i and j. Note that the second sum of the potential energy only goes to $i < j$, to avoid double-counting the energy.

2.5.2 Conservation of angular momentum per unit mass

As there is no external force, there is also no external torque. Therefore, the net angular momentum per unit mass should be conserved in our system. The angular momentum per unit mass of a body is generally given by:

$$\vec{l} = \vec{r} \times \vec{v}$$

Where $\vec{v}$ is the velocity of the body, and $\vec{r}$ is the distance to the center of mass. The total angular momentum per mass is then is then:

$$\vec{l}_{tot} = \sum_i \vec{r}_i \times \vec{v}_i \tag{9}$$

Where the sum is over all bodies.

Equations 8 and 9 both describe quantities which should be conserved. Thus, an important consistency check for our code, is to test if these quantities are indeed (approximately) constant throughout our simulation.

2.6 Escape velocity

Another way we can check the consistency of our results is to investigate the escape velocities of the planets. This can be done by equating the potential energy to the kinetic energy. This is a difficult problem for a many-body problem. Therefore, we will only implement this in the two-body case, with the sun and the earth. Equating potential and kinetic energy of the earth gives:

$$\frac{1}{2}v_{esc}^2 = \frac{GM_\odot}{r}$$

Solving:

$$v_{esc} = \sqrt{\frac{2GM_\odot}{r}} \tag{10}$$

Which is the escape velocity of the earth, if we ignore the other planets. Note that this velocity is to be taken *relative to the sun*. We can experiment with the initial velocity of our planets, and check if the velocity at which a planet manages to escape equals this quantity. This gives us another way to check the consistency of our results.

2.7 The perihelion precession of Mercury

One of the tests for Einstein's theory of relativity is the perihelion precession of Mercury. Over time, the perihelion (the point where Mercury is closest to the sun) changes. This was initially thought to be due to the gravitational attraction of other planets, but the effect was too large.It turns out that this effect is due to general relativity, which predicts that gravity warps spacetime. This boils down to modifying the gravitational force to:

$$\vec{F}_G = \frac{GM_\odot M_{\text{Mercury}}}{r^3}\left[1 + \frac{3l^2}{r^2c^2}\right]\vec{r} \tag{11}$$

Where $l = |\vec{l}|$ is the absolute value of the angular momentum per mass of the body the force acts on, and c is the speed of light. This assumes that all other effects, such as gravitational attraction from other planets, have been ignored. In this case, experimental data shows that the perihelion angle precesses $43'' = 0.2085$ mrad per 100 earth years. We will investigate if this relativistic correction can correctly predict the precession, by comparing the motion of Mercury (removing all other planets) with Newtonian gravity and with the relativistic expression for gravity.

3 Methods

In this section we give an overview of the methods that we employed to solve the problems outlined in the previous sections. We discuss the models we have implemented, the numerical implementation of these models and the equations from the previous section and finally the implementation of initial conditions.

3.1 Models

We will implement four different models, to test our simulations under multiple circumstances.

3.1.1 First model - The earth-sun system

We begin with a simplified model of our solar system, containing only the earth and the sun. We simulate this using both the Euler forward algorithm and the Velocity-Verlet algorithm, which enables us to compare these methods.

3.1.2 Second model - The three body problem

We slightly expand our system by also including the most massive planet - Jupiter. We can then investigate how large the effect of Jupiter is on the orbit of the earth. We will also make this slightly more extreme, by increasing the mass of Jupiter by a factor 10 and a factor 1000. In the latter case, we will be approaching a binary star system, as the mass of Jupiter is about 1000 times less than the mass of the sun. Thus we expect a fairly chaotic behavior, which will allow us to study the stability of our Verlet solver.

3.1.3 Third model - Entire solar system

We then include all eight planets in the solar system, as well as Pluto for historical reasons. We also include all interactions between the planets. We use the Velocity-Verlet method to solve this numerically, which gives us an opportunity to study the stability of the Verlet-method on larger scales.

3.1.4 Final model - The mercury-sun system

Finally, we model only mercury and the sun, to investigate the perihelion precessions of mercury described in section 2.7. Here we use the Verlet-solver, and implement both the Newtonian and the relativistic expressions for the gravitational force.

3.2 Some interesting features of our numeric scripts

3.2.1 Data flow

To ease the flow of data, we implement a class structure to our code. We have three interacting classes:

1. Particle class - a class which stores all variables related to a single planet.
2. System class - A class which stores all and computes all species related to the entire system.
3. ODESolver class - A class which computes acceleration based on the algorithms detailed in section 2.3

Finally we also implement a tailored vector class, which implements some key features of vector algebra needed to solve the equations in section 2.1, such as addition, multiplication, norms, inner products and the cross product.

3.2.2 Retrieving the initial conditions

As described in section 2.2, the initial positions and velocities of all planets can be retrieved from NASA's webpages. It is possible to establish a telnet connection to these websites, and retrieve the data automatically. We took inspiration from**INSERT STACKEXCHANGE PAGE HERE** when implementing this.

With the class structure and the initial conditions in place, we can then simply initialize the solar system by a code akin to the one shown below:

```
infile=open('Planet_initial_data.txt')
solar_system=System()
for line in infile: //Read in planet data
        particle_name, initial_pos, initial_vel, mass=line.split()
        if particle_name is not "Sun": //So that COM is at x0=0, v0=0
                solar_system.create_particle(initial_pos, initial_vel, mass)

total_position=sum(particle_mass*particle_position)
total_momentum=sum(particle_mass*particle_velocity)
sun_pos=-total_position
sun_vel=-total_momentum/mass_of_the_Sun

solar_system.create_particle(sun_pos, sun_vel, mass_of_the_Sun)
```

And then we can simply solve the differential equations by calling:

```
steps_per_year=1e7
years=10
timestep=1/steps_per_year
solver=ODESolver(timesteps)
N=steps_per_year*years

for i in range(0, years, N):
        solver.Velocity_verlet_one_step //update parameters in solar system
        solar_syststem.dump_positions_to_file
```

3.3 Checking the consistency of our results

3.3.1 Checking conservation of energy and angular momentum per unit mass

We check conservation of energy for every 100^{th} timestep in all of the models described in section 3.1, except for the case with relativistic gravity. As our background in general relativity is relatively limited, we are unsure whether or not energy and angular momentum per mass are conserved in this case.

Because magnitudes of deviations are notoriously difficult to interpret, we will instead record and analyze the relative error at a time, t, defined as:

$$\epsilon_{rel} = \frac{E_{\text{initial}} - E(t)}{E_{\text{initial}}} \tag{12}$$

Where E is the energy. We employ the same expression to check conservation of angular momentum per unit mass.

3.3.2 Checking escape velocity

We experiment with the initial velocity of earth in the earth-sun system until we see that the orbit is no longer elliptic but becomes parabolic. We then compare this to the value calculated in section 2.6.

3.4 Investigating the perihelion processions of mercury

To find the perihelion processions of mercury, we must develop an algorithm which checks for a minimum of the distance between the sun and mercury. We do this by storing the absolute value of the position vector for three different timesteps, and subsequently check if the value in the middle of these three is lower than the two others. If this is the case, we save it as a perihelion passing, and compute the angle. This is illustrated in the pseudo-code below:

```
previous_r =0
previous_previous_r=0

for k in range(0, time, N):
    r_vec = mercury_position - sun_position
    r = r_vec.length();
    if previous_r < r and previous_r < previous_previous_r:
        theta = arctan2(r_vec.y,r_vec.x);   //Save as perihelion passing
```

4 Results

4.1 Results from the first model - earth-sun

4.1.1 Position

The Euler-Forward algorithm

5 Discussion

5.1 Discussing the perihelion precession of mercury

One interesting aspect of our investigation of the precession of mercury is that it broke down when we computed the angular momentum at every time step. When we used the initial angular momentum of mercury, however, (given by $\vec{l} = \vec{r} \times \vec{v}$) we got the correct result. **WHY**

6 Conclusion and outlook

6.1 Conclusion

6.2 Outlook

Appendices

A Initial velocity for circular orbits and the value of G

If we assume a two-body system, consisting of the earth and the sun, transform to a system where the sun's velocity is zero, and assume circular orbit, we can equate the force of gravity to the centripetal force. For the earth this gives:

$$\frac{GM_\odot m_e}{r^2} = m_e \frac{v_{circ}^2}{r}$$

Where m_e is the mass of earth. Rearranging gives:

$$v_{circ} = \sqrt{\frac{GM_\odot}{r}} \tag{13}$$

Inserting $r = 1$AU gives for the earth $v_{circ} = 29\ 780$ m/s $= 6.279$ AU/yr. This is the velocity we must give the earth (in a frame where the initial velocity of the sun is zero), to ensure that it orbits in a circular orbit of radius 1AU.

From this equation, we can also find a value for G in the units astronomical units per year. We simply say that the earth takes one year to complete one revolution, which gives $v_{circ} = 2\pi r/T = 2\pi$ AU/yr. Solving equation 13 for G gives:

$$G = \frac{v_{circ}^2 r}{M_\odot}$$

Seeing as $r = 1$ AU, this gives:

$$G = 4\pi^2 \ \ \mathrm{AU}^3\mathrm{yr}^{-2}\mathrm{M}_\odot^{-1}$$

B Deriving the Velocity-Verlet algorithm

Here will derive the Velocity-Verlet method. We will derive this for a single dimension (x), however the three-dimensional generalization is straightforward, as it simply amounts to repeating the exact same equation with x exchanged with y and z respectively.

The Velocity-Verlet method is based on Taylor-expanding the acceleration, a, around $x = x_0$. This gives:

$$a(x_0 + h) = a(x_0) + ha'(x_0) + O(h^2)$$

Or, discretizing as described in section 2.3, gives approximately:

$$a_{i+1} \approx a_i + ha_i'$$

Or, rearranging slightly:

$$ha_i' \approx a_{i+1} - a_i \tag{14}$$

Now we expand the velocity in the same way, but include another term in the Taylor expansion. This gives:

$$v_{i+1} = v_i + ha_i + \frac{h^2}{2}a_i' + O(h^3)$$

Inserting from equation 14 gives:

$$v_{i+1} = v_i + a_i h + \frac{h}{2}\left(a_{i+1} + a_i\right) + O(h^3)$$

Finally, we Taylor expand the position in the same way to get:

$$x_{i+1} = x_i + h v_i + \frac{h^2}{2} a_i + O(h^3)$$

These are the equations of the Velocity-Verlet algorithm described in equation 7. Notice how the error is proportional to h^3, as opposed to the Forward-Euler algorithm, where the error is proportional to h^2.