Project 3

Maziar Kosarifar (maziark)
Marie Reichelt (marierei)

October 2018

Abstract

In this project we are going to study and simulate the solar system. We are comparing two different approaches of solving first order differential equations. We will look at the masses of different celestial bodies and the effect that other masses will have on their orbits.

1 Introduction

An ordinary differential equation can be formulated as:

$$y'(t) = f(t, y(t))$$

Equations in this form can be used as a tool in many different scientific fields, from math and physics, to economics and engineering.

Previously in this course, we came across the Schrödinger equation, which is such an equation.

In this project, we are going to look at Euler's forward algorithm and velocity Verlet algorithm, both methods for solving ordinary differential equations. We are going to compare the algorithms and the results they produce.

Firstly, we go through two different methods for solving ordinary differential equations. These methods are then compared. Our implementation of the Solar System is then explained briefly, and the results produced are discussed.

2 Theory

For two orbiting celestial bodies, Newton's law of gravitation is given by:

$$F_G = \frac{GM_{\odot}M_{Earth}}{r^2} \tag{1}$$

where $M_{\odot}$ is the mass of the Sun, M_{Earth} is the mass of the Earth, and r is the distance between their centers. We have the gravitational constant given as G. Assuming the Sun has a much greater mass than that of the Earth, we can dismiss any possible effect that the mass of the Earth will have on the orbit of the Sun.

Newton's second law of motion goes as:

$$a = \frac{F}{m} \tag{2}$$

Following this, we get a set of equations to represent the position (x, y, and z components) of the gravitational force:

$$\left(\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{Earth}}, \frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{Earth}}, \frac{d^2z}{dt^2} = \frac{F_{G,z}}{M_{Earth}}\right) \tag{3}$$

From NASA's web page, we have the initial positions $\vec{x}$, velocities $\vec{v}$ and masses M of all objects in the Solar System. This information can be combined with (1) and (3) solve our system of differential equations.

2.1 Potential and Kinetic energy

For two masses, m_1 and m_2, at a distance r apart, the Potential energy can be defined as:

$$E_p(\vec{r}) = -G\frac{m_1 m_2}{r} \tag{4}$$

The Kinetic energy of an object is defined as:

$$E_k = \frac{1}{2}m_1 v^2 \tag{5}$$

Assuming that we have a perfectly circular orbit, we know that both Potential and Kinetic energy will be preserved. We have the equation:

$$v^2 r = Gm_2 \tag{6}$$

$$v = \sqrt{\frac{Gm_2}{r}} \tag{7}$$

This can be used to find the escape velocity of an object that is in an orbit around another object. Escaping this orbit requires a velocity so that the Kinetic energy is greater or equal to the Potential energy. This gives us:

$$v_{escape} = \sqrt{\frac{2Gm_2}{r}} \tag{8}$$

2.2 The general law of relativity

In the Newtonian model of gravitation, planets would have a closed orbit, meaning that after each orbit they will end up at their initial position. This cannot explain the perihelion precession of Mercury.

The general relativistic correction to the Newtonian gravitational force models the gravitational force is defined as:

$$F_G = \frac{Gm_1m_2}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right], l = |\vec{r} \times \vec{v}| \tag{9}$$

Here r is the distance between the two bodies, and c is the speed of light.

We intend to put this theory in test, and compare the result of the simulation, with the results obtained from observations.

2.3 Taylor expansion

We are using Taylor expansion for the position and velocity:

$$\vec{x}(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + O(h^3) = x(t) + hv(t) + \frac{h^2}{2}a(t) + O(h^3) \tag{10}$$

$$\vec{v}(t+h) = v(t) + hv'(t) + \frac{h^2}{2}v''(t) + O(h^3) = v(t) + ha(t) + \frac{h^2}{2}a'(t) + O(h^3) \tag{11}$$

2.4 Euler's forward algorithm

One of the very well known method for solving ordinary differential equations is the Euler's method. This method uses an initial value, and the derivative of the function as its variable.

We use (10) and (11) at set $h = \Delta t$:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t \tag{12}$$

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t \tag{13}$$

2.5 Velocity Verlet algorithm

The Verlet and Leapfrog algorithms are both nummerically stable and easy to implement[1]. Here we consider a second-order differential equation like the second law of Newton:

$$m\frac{d^2x}{dt^2} = F(x,t)$$

We rewrite as:

$$\frac{dx}{dt} = v(x,t) \tag{14}$$

$$\frac{dv}{dt} = \frac{F(x,t)}{m} = a(x,t) \tag{15}$$

To find the position we perform Taylor expansion:

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3) \tag{16}$$

$$v(t+h) = v(t) + hv^{(1)}(t) + \frac{h^2}{2}v^{(2)}(t) + O(h^3)$$

$$v(t+h) = v(t) + ha(t) + \frac{h^2}{2}a'(t) + O(h^3) \tag{17}$$

We have :

$$a'(t) = \frac{a(t+h) - a(t)}{h} \tag{18}$$

We substitute (18) for (17), so for velocity Verlet we have:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \tag{19}$$

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+\Delta t)}{2}\Delta t \tag{20}$$

2.6 Discretisation

When discretising the differential equations, we use astronomical units. The unit of length is known as 1 AU. This is the average distance between the Sun and the Earth. We have 1 AU = 1.5×10^{11}m. To make the time scale to be more manageable, we use years instead of seconds.

For Euler's forward algorithm, (10) and (11), we have:

$$x_{i+1} = x_i + v_i \Delta t \tag{21}$$

$$v_{i+1} = v_i + a_i \Delta t \tag{22}$$

And for the velocity Verlet algorithm, (19) and (20), we have:

$$x_{i+1} = x_i + v_i \Delta t + a_i \frac{\Delta t^2}{2} \tag{23}$$

$$v_{i+1} = v_i + \frac{a_i + a_{i+1}}{2} \Delta t \tag{24}$$

2.7 Eventual differences between Euler and Verlet

We see that Euler's algorithm has an error that is proportional to the step size, thus it can give an answer that makes little sense no matter the step size. With velocity Verlet, the odd terms cancel each other out when we add up the Taylor expansions. This makes the total error of the velocity Verlet algorithm smaller than the total error for the forward Euler algorithm.

From the graph we observe that the velocity Verlet algorithm is a slower algorithm than the forward Euler algorithm when we have δ_t to be the same. This is shown in figure 1. But for large values for δ_t, Verlet is much more precise.

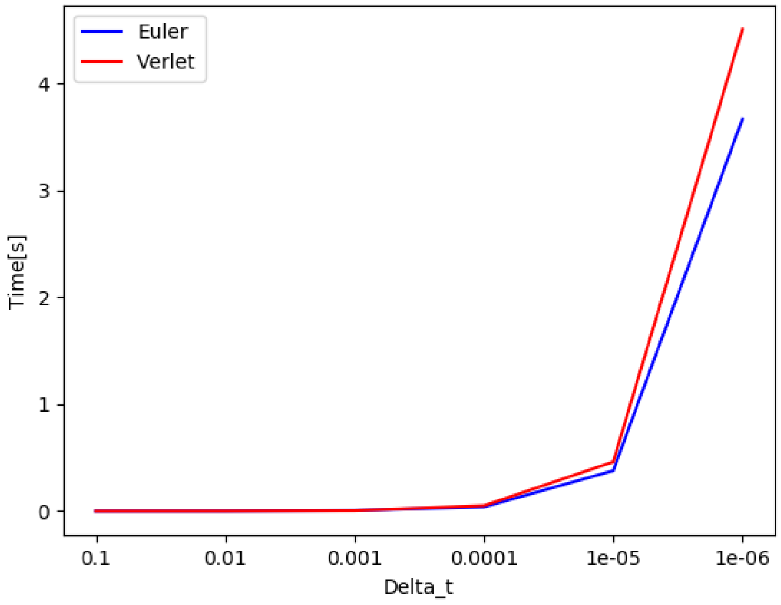

Figure 1: Comparing the errors of Forward Euler and Velocity Verlet with the same time steps

2.8 Implementation

Since all celestial objects share certain fundamental features that can be represented with variables and vectors, we used Object Oriented programming.

There are four main classes, CeletialObj, SolarSystem, VerletAlgorithm, and EulerAlgorithm.

CelestialObj helps us store the main variables of every planets' position, velocity vector, and mass, while the SolarSystem class governs the all bodies involved in the simulation.

This approach simplifies the programming difficulties, and make testing simpler.

In each iteration, the most time consuming part is the calculation of gravitational force. Since it should be calculated for every two planets, therefore it requires N^2 calculation, which can be divided in half, using the third law of Newton. ($O(N^2/2)$)

Following is the pseudo code for our implementation of the forward Euler algorithm and the velocity Verlet algorithm are presented to highlight the differences.

Algorithm 1 Forward Euler Algorithm

```
1: procedure ForwardEuler(CelBodies, $\Delta t$)
2:     $calculate_Forces()$
3:     for body ← bodies do
4:         $body_p \leftarrow body_p + body_v * \Delta t$
5:         $body_v \leftarrow body_v + body_a * \Delta t$
```

Forward Euler Algorithm, is of $O(N + \frac{N^2}{2})$.

Algorithm 2 Verlet velocity

```
   procedure Verlet(CelBodies, $\Delta t$)
2:     $calculate_Forces()$
       for body ← bodies do
4:         $body_p \leftarrow body_p + body_v * \Delta t + \frac{body_a}{2}\Delta t^2$
           $body_v \leftarrow body_v + \frac{1}{2} body_a * \Delta t$
6:     $calculate_Forces()$
       for body ← bodies do
8:         $body_v \leftarrow body_v + \frac{1}{2} body_a * \Delta t$
```

Verlet velocity Algorithm, is of $O(2N + N^2)$, which runs slower than Euler algorithm, but the results are more accurate, so much that for same level of precision, it can easily compete with Forward Euler algorithm.

Since this project involves plotting the solar system with different number of elements, we ended up programming a rather generic python code, that can read a text file, and plot the result.

```python
import numpy as np
import math
import sys
import matplotlib.pyplot as plt

n = 3
d = 3
f = "body2.txt"
if (len(sys.argv) > 2):
  n = int(sys.argv[1])
  d = int(sys.argv[2])

if (len(sys.argv) > 3):
  f = sys.argv[3]

fig = plt.figure()
planets = []
# number of planets;

file = open(f)

lines = [line.rstrip('\n') for line in file]
file.close()
```

```python
planets = []
for r in range(n) :
  planets.append([lines[i] for i in range(len(lines)) if i%n==r])

for r in range (d):
  name  = [a.split()[0] for a in planets[r]]
  pos_x = [a.split()[2] for a in planets[r]]
  pos_y = [a.split()[3] for a in planets[r]]
  plt.plot (pos_x, pos_y, label = name[0])

plt.xlabel('x(AU)')
plt.ylabel('y(AU)')
plt.legend()

plt.show()
```

The implementation runs in 3D, but the plots are in 2D.

3 Results and discussion

3.1 Conservation of energy and angular momentum

We are looking at a two body system with only the Sun and the Earth. Analytically we are finding the velocity for Earth that will give it a perfectly circular orbit around the Sun. We expected both Potential and Kinetic Energy to stay the same for the orbit to stay perfectly circular.

From the graphs produced, we see that the velocity Verlet algorithm has a better preservation of energy than the forward Euler algorithm has. Velocity Verlet already has a good precision when the step size is 0.1, whilst forward Euler is not very accurate before the step size is 0.001.

This says that forward Euler requires more steps than Verlet to get the same accuracy. We have earlier seen that velocity Verlet is a more expensive algorithm than forward Euler, but since it doesn't require as much precision, this method can be superior in some cases. This was discussed in more detail in section (2.7).

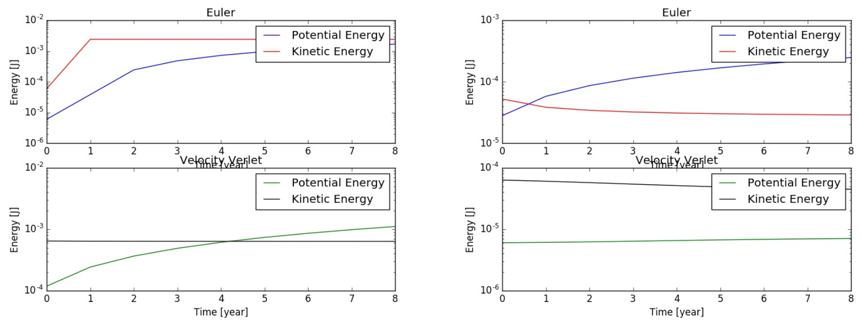

Figure 2: Conservation of energy with a precision of 1 (left) and 0.1 (right). Upper graphs shows forward Euler and lower graphs shows velocity Verlet.

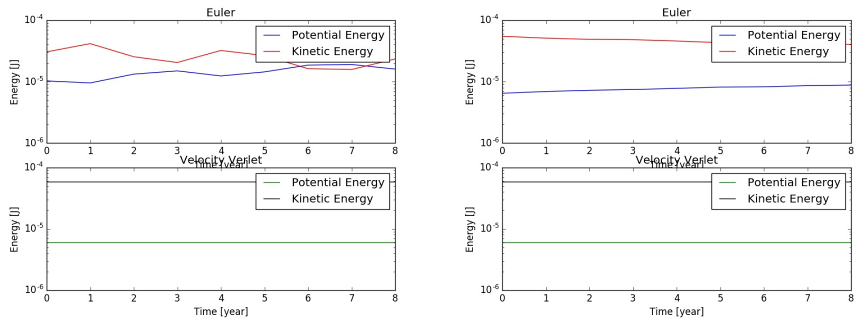

Figure 3: Conservation of energy with a precision of 0.01 (left) and 0.001 (right). Upper graphs shows forward Euler and lower graphs shows velocity Verlet.

3.2 The three-body problem

When adding another body to the system, things get complicated. The three-body problem is a special case of the n-body problem. Whilst the motion will be stable over time when we only have two bodies who's mass interact with each other, this is not the case if we have three bodies. Therefore we want to use our program to observe what effect adding Jupiter has to the motion of the Earth.

Even though Jupiter is a massive planet, it is not as heavy as the Sun. The Sun weighs 2×10^{30} kg, whilst Jupiter only weighs 1.9×10^{27} kg. Jupiter is also much further away from the Sun that what Earth is, more than 5 times as far. Therefore Jupiter should not have a great effect on Earth's orbit.

From the graphs we observe exactly this. This is shown in figure 2. Earth's orbit is slightly affected, but not enough to make a big change. When the mass of Jupiter is set to be ten times its original mass is slightly affected, but when the mass is increased to be 100 times the mass we observe a similar, but bigger change. For the mass of Jupiter 1000 times the original, we observe that the mass of Jupiter starts affecting the orbit of the Sun. The Sun moves around Jupiter, making a half circle pattern. Earth is still moving in the system.

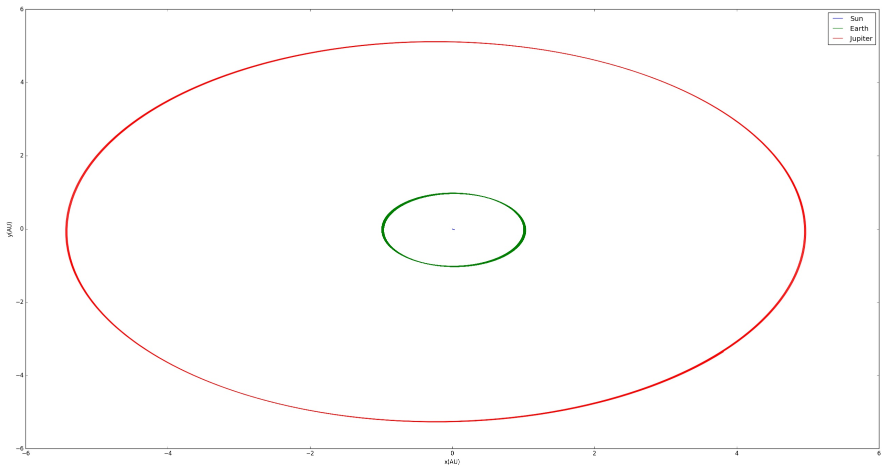

Figure 4: Jupiter with its ordinary mass

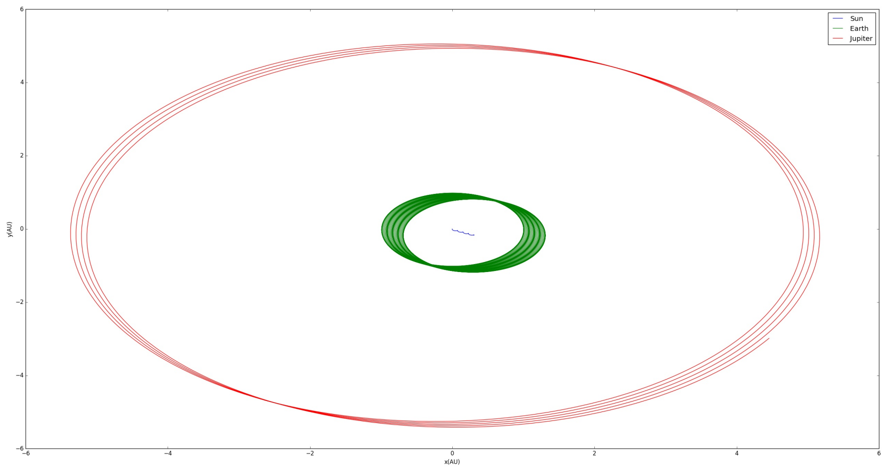

Figure 5: Jupiter with ten times its mass

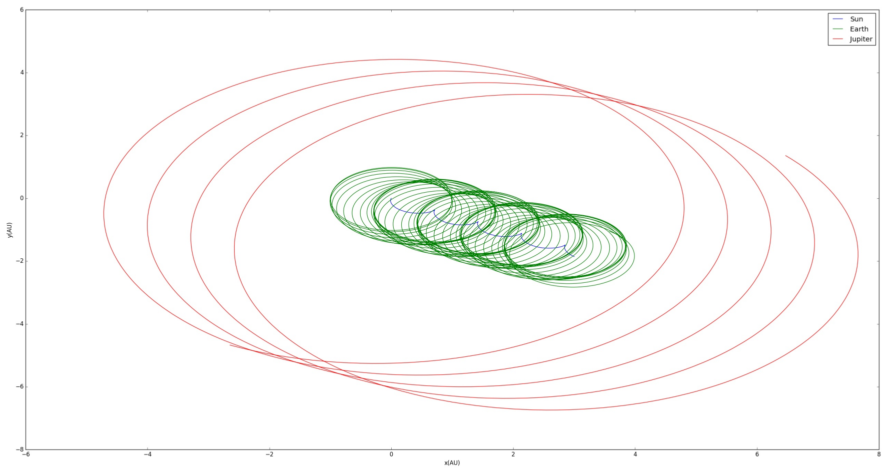

Figure 6: Jupiter with 100 times its mass

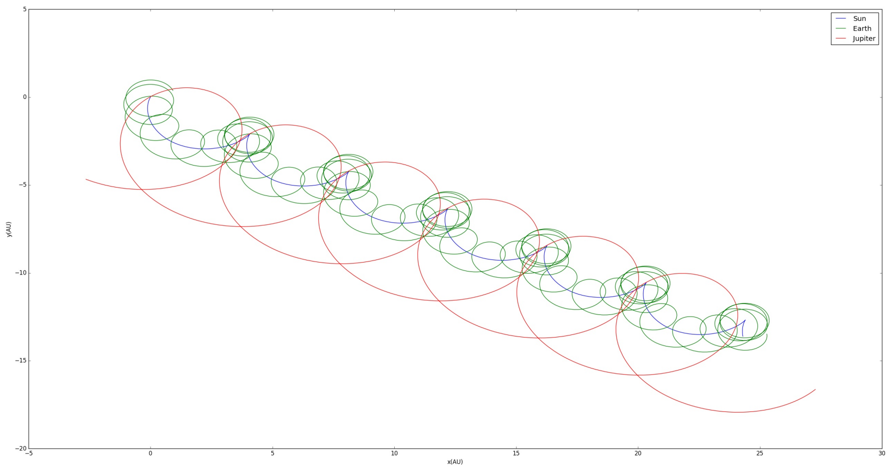

Figure 7: Jupiter with 1000 times its mass

3.3 Escape velocity

The escape velocity of an object at a distance of r from a different object of mass m_2, is defined as:

$$v_{esc} = \sqrt{\frac{2Gm_2}{r}} = \sqrt{\frac{2 \cdot 4\pi^2 \cdot 1}{1}} = \sqrt{8} \cdot \pi \approx 2.82842712475\pi AU/year \tag{25}$$

when using astronomical units. When plotting the escape velocity we get it to be approximately $2.87\pi AU/year$. The orbit of the Earth when escaping the gravity of the Sun is shown in figure 7.

3.4 The perihelion precession of Mercury

To test the general relativistic correction to the Newtonian gravitational force, we implemented the general law of relativity for gravitation and run the algorithm for the duration of 100 years. For each year there are $N = 10^6$ points, so in total we run it for $N = 10^8$ steps.

When running the algorithm with the initial position of $[0.3075, 0, 0]$ and a stationary Sun, the position of the perihelion of the last orbit was $[0.3075, -5.18083 \cdot 10^{-5}, 0]$. The angle between the first and the last perihelion is $34.752°$.

The observed value of the perihelion precession of Mercury over a time span of a century is $43°$. This is when taking all effects on the orbit. In our system, we only consider a two-body system that consists of the Sun and Mercury.

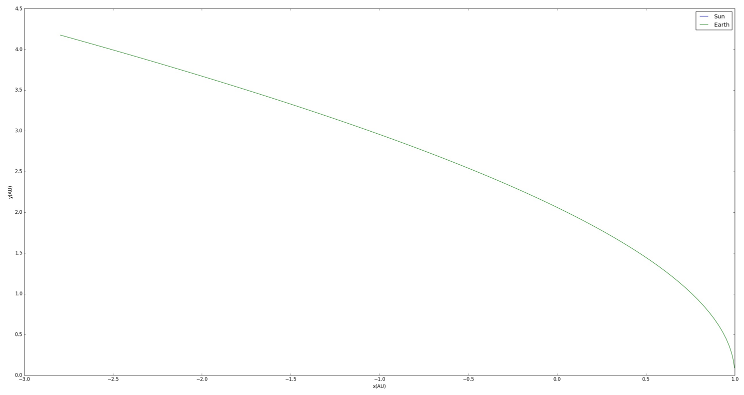

Figure 8: Jupiter with 1000 times its mass

```cpp
vec3 CelestialObj::calcForce(CelestialObj &other, bool Ens){

    vec3 f = calcForce(other);

    // 1 + \frac{3l^2}{r^2c^2}
    double c2 = 3993960777.9406304;
    vec3 r = this->position - other.position;
    vec3 v = this->velocity;
    double rel = 3.0 * (r.cross(v).lengthSquared())/(r.lengthSquared()*
    c2);
    return f * (1 + rel);
}
```

4 Conclusion

In this project we have looked at the differences between the forward Euler algorithm and the velocity Verlet algorithm. We have shown that the Verlet algorithm has a much better precision that the Euler algorithm when it comes to forces that are dependent on positions. The velocity Verlet algorithm is more expensive per iteration, whilst the forward Euler algorithm needs a smaller step size to be equally as precise. This in total, makes the velocity Verlet a cheaper method in our case.

We have also observed how stable our Solar System is, but also seen what happens when we change different factors. We looked at a Solar System with two massive objects, namely the Sun and Jupiter. This made the whole Solar System become unstable.

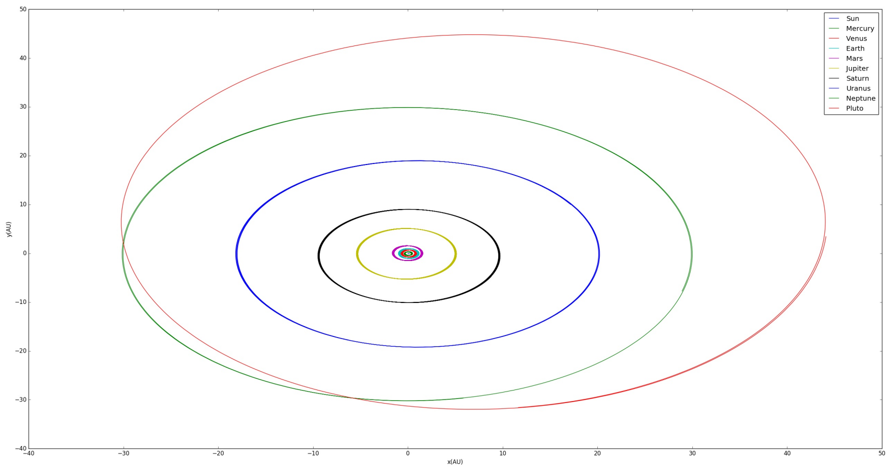

Figure 9: The whole Solar System, including Pluto

5 Source files

Source files are located in:
https://github.com/maziark/FYS3150-2018/tree/master/Project3

Since in this project we used QT Creator, the make file can be used to compile the code, to run different tests, one should uncomment the function call in the main function. Results are stored in text files, that can be plotted with Python.

References

[1] M. Hjort-Jensen, Computitional Physics Lecture Notes Fall 2015, Department of Physics, University of Oslo, Oslo, Norway, page 241-281 (2015).

[2] T.Sauer, Numerical Analysis, Pearson, Boston, United States, page 284-295 and 313-317 (2006)