Project 3 - FYS3150 Computational Physics

Fredrik Østrem (fredost)

October 25, 2016

Abstract

In this project, we will study different simulations of the solar system. We will discuss the differences between Euler's forward algorithm and the Verlet algorithm in such simulations, and we will build up a program which can simulate the gravitational interaction between the Sun and all the planets. Finally, we will use this program to study the perihelion precession of Mercury due to general relativity.

Contents

1 Introduction

In this project, we will build a simple object-oriented program that simulates the gravitational interaction between the sun and planets in the solar system, by using the velocity Verlet algorithm. We will start writing a simple program for simulating the Earth-Sun system with the Sun fixed at the origin, and use this to compare the Verlet algorithm to an algorithm that we're already very familiar with: Euler's forward algorithm. We will expand this program to simulate three bodies (Earth, Sun and Jupiter) and then the entire solar system (Sun, Moon and all the eight planets). We will also use our code to explore two specific problems: the escape velocity of Earth in the gravity of the Sun, and the perihelion precession of Mercury due to general relativity.

1.1 Units and constants

We will throughout this assignment assume that the base units are expressed in *astronomical units* (AU) for distance, *years* (yr) for time, and *solar masses* ($\mathrm{M}_\odot$) for mass. All other units will be derived from these, such as energy ($\mathrm{M}_\odot\mathrm{AU}^2/\mathrm{yr}^2$) and angular momentum ($\mathrm{M}_\odot\mathrm{AU}^2/\mathrm{yr}$).

When using these units, we get nice expressions for relevant constants:

- The distance between the Earth and the Sun is $r = 1\,\mathrm{AU}$.
- The velocity of the Earth is $v = 2\pi\,\mathrm{AU}\,\mathrm{yr}^{-1}$.
- The universal gravitational constant is $G = 4\pi^2\,\mathrm{AU}^3\,\mathrm{M}_\odot^{-1}\,\mathrm{yr}^{-2}$.

2 Methods

2.1 Discretization of a two-body system

a) We start by only considering a simple two-body gravitational system, such as the Earth-Sun system. We can express the gravitational interaction of the Sun on the Earth by the differential equation

$$\mathbf{a} = \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = -\frac{F_G}{M} \cdot \frac{\mathbf{r}}{r} = -\frac{GM_\odot \mathbf{r}}{r^3}$$

where $\mathbf{x}$ is the position of the Earth, $\mathbf{r}$ is the position of the Earth relative to the Sun, and M and $M_\odot$ are the masses of the Earth and the Sun.

To be able to solve this differential equation numerically, we want to discretize it. If we use Euler's forward algorithm to discretize this with time steps Δt, we get the equations

$$\begin{aligned}
\mathbf{x}_{n+1} &= \mathbf{x}_n + \mathbf{v}_n \cdot \Delta t \\
\mathbf{v}_{n+1} &= \mathbf{v}_n + \mathbf{a}_n \cdot \Delta t = \mathbf{v}_n - \frac{GM_\odot \mathbf{r}_n}{r_n^3} \cdot \Delta t
\end{aligned}$$

where

$$\mathbf{r}_n = \mathbf{x}_n - \mathbf{x}_{\odot,n}$$

We can also use the Verlet algorithm, which is based on Netwon's laws of motion. If we use that, we get the equations

$$\begin{aligned}
\mathbf{x}_{n+1} &= \mathbf{x}_n + \mathbf{v}_n \cdot \Delta t + \frac{1}{2}\mathbf{a}_n \cdot \Delta t^2 \\
\mathbf{v}_{n+1} &= \mathbf{v}_n + \frac{\mathbf{a}_n + \mathbf{a}_{n+1}}{2} \cdot \Delta t \\
\mathbf{a}_n &= -\frac{GM_\odot \mathbf{r}_n}{r^3}
\end{aligned}$$

Since the mass of the Sun is about 300 000 times larger than that of the Earth, we can assume that the Sun is position in the center of mass, and we will assume that it lies in the origin.

As we will see later, Euler's forward algorithm is not numerically stable with respect to total energy, while Verlet keeps the total energy of the system constant.

2.2 Implementation of the Earth-Sun system

b) When implementing the simulation of the Earth-Sun system, we may want to create and generalize classes to be able to reuse functionality. In particular, we may want to write:

- A `vec3` class, that stores a 3D vector and handles vector operations.
- A `CelestialBody` class, that stores the properties of some celestial body (in this case, the Earth and the Sun).
- A `SolarSystem` class, that stores all the celestial bodies of the simulation (in this case, the Earth and the Sun).
- A `Solver` class, that solves the differential equations for the movement of the celestial bodies, and that can be subclassed to provide different algorithms (such as Euler's forward algorithm or the Verlet algorithm).

Writing classes like this can have many advantages. In particular, we keep the information about an object and the functions that act on it bundled together, and we can easily replace or extend the functionality of a class by creating a subclass (like we do with the `Solver` class).

3 Results

3.1 Earth-Sun system

c) When we run the program `code/3b`, we will get two data files `3b_euler.dat` and `3b_verlet.dat`, that contain the data for the simulation over a five-year period with time steps at $1/1000$ of a year.

In figure 1, we see the x and y coordinates of the Earth's orbit around the sun (fixed at the origin). In figure 2, we can see the kinetic, potential and total energies over time, and in figure 3, we can see the total angular momentum of the system over time.

In all three plots, we can see that the Verlet algorithm (in blue) is stable; the Earth moves in a near perfect circle around the sun, total energy is constant and so is total angular momentum. This reflects what we already know about the conservation of energy and angular momentum in a closed system: that these quantities are preserved.

On the other hand, Euler's forward algorithm (in red) is numerically unstable: instead of moving in a circular orbit, the Earth's distance from the sun gradually increases, and its total energy and angular momentum increase significantly, despite being a closed system.

The Verlet algorithm is specifically designed for calculations of forces, being based on Newton's laws of motion. Because of this, it ensures that quantities such as total energy or total angular momentum is preserved when we have conservative forces, such as gravity. However, Euler's forward algorithm is a generic algorithms for all differential equations, and pretty much the simplest algorithm you could use.

The Verlet algorithm uses three times more FLOPs than Euler's forward algorithm: while Euler uses 4 FLOPs[1] per calculation per dimension, Verlet uses 12. (For a three-dimensional system such as ours, this means Euler will use 12 FLOPs while Verlet will use 36.) The naïvely implemented Verlet algorithm also has to recalculate the forces twice: once for the previous position, and once for the new position. However, this can be reduced by simply only recalculating when the position has changed; when it has not, you can just reuse the forces calculated for the previous Verlet step. It will often be the calculation of the forces that is the most expensive, especially in systems with more than one moving body, since the number of forces to be calculated is approximately N^2 for N bodies.

[1]**F**loating **p**oint **op**erations.

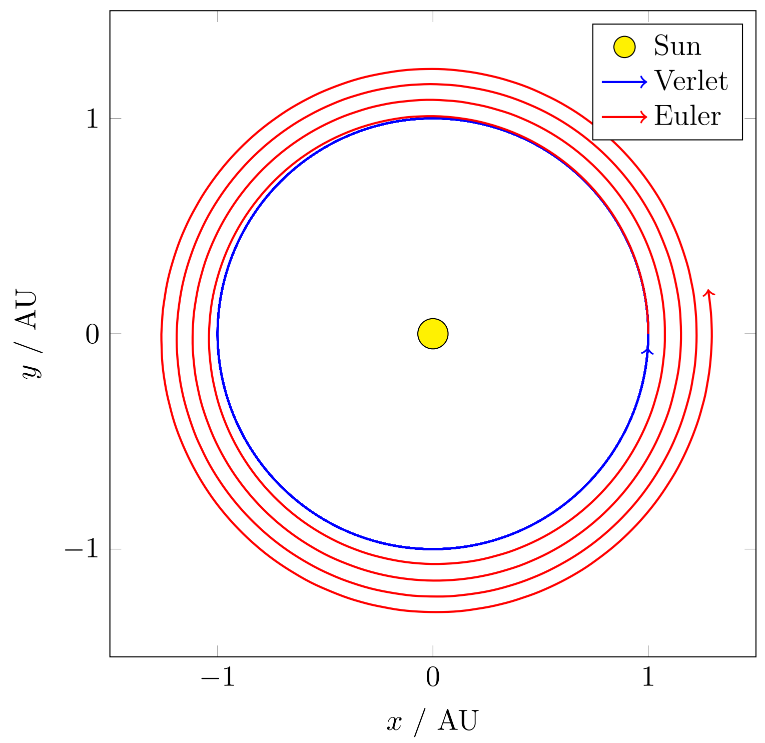

Figure 1: Five-year simulation of the Earth-Sun system, using the Verlet algorithm and Euler's forward algorithm. Sun is not to scale.

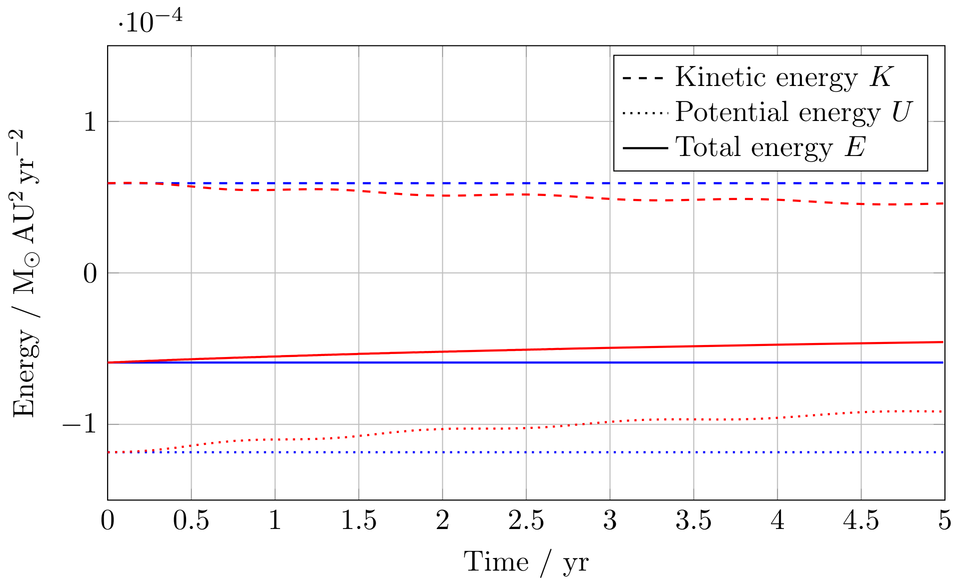

Figure 2: Kinetic, potential and total energy of the five-year simulation from figure 1. Blue is with Verlet algorithm, red is with Euler's forward algorithm.

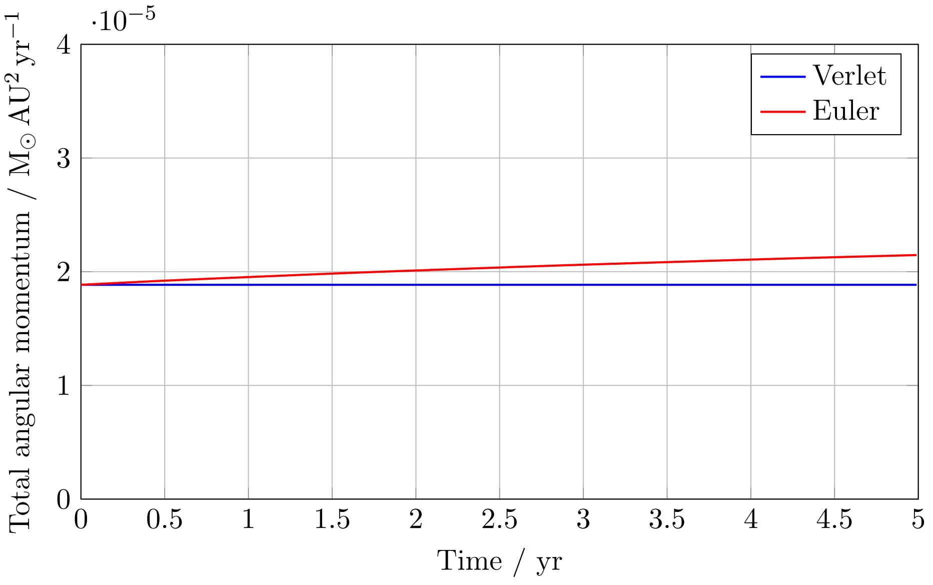

Figure 3: Total angular momentum L of the five-year simulation from figure 1. Blue is with Verlet algorithm, red is with Euler's forward algorithm.

3.2 Escape velocity

d) Through trial and error (as shown in figure 4), I found the best approximation of the escape velocity be

$$\begin{aligned} v_{\text{escape}} &\approx 2\pi \cdot 1.42\,\text{AU}\,\text{yr}^{-1} \\ &\approx 8.92\,\text{AU}\,\text{yr}^{-1} \end{aligned}$$

where $1\,\text{AU}\,\text{yr}^{-1} \cdot 2\pi$ is the speed at which a circular orbit would be made.

We know that the gravitational potential energy of Earth, $U(r)$, is

$$U(r) = -\frac{GMM_\odot}{r}$$

If we let $U_\infty = \lim_{r\to\infty} U(r)$ be the potential energy at "infinity", we easily see that $U_\infty = 0$. This means that to escape the gravitational well of the sun, we need to have a total energy $E_{\text{tot}} \geq U_\infty$; the escape velocity occurs exactly where $E_{\text{tot}} = U_\infty = 0$.

This means that the escape velocity v_{escape} is given by the equation:

$$\frac{1}{2}Mv_{\text{escape}}^2 - \frac{GMM_\odot}{r} = 0$$

which when rearranged gives

$$v_{\text{escape}} = \sqrt{\frac{2GM_\odot}{r}}$$

When we insert the quantities G, $M_\odot$ and r for the Earth-Sun system, we get

$$v_{\text{escape}} = 2\sqrt{2}\pi\,\text{AU}\,\text{yr}^{-1} \approx 8.89\,\text{AU}\,\text{yr}^{-1}$$

which is not far off our earlier approximation.

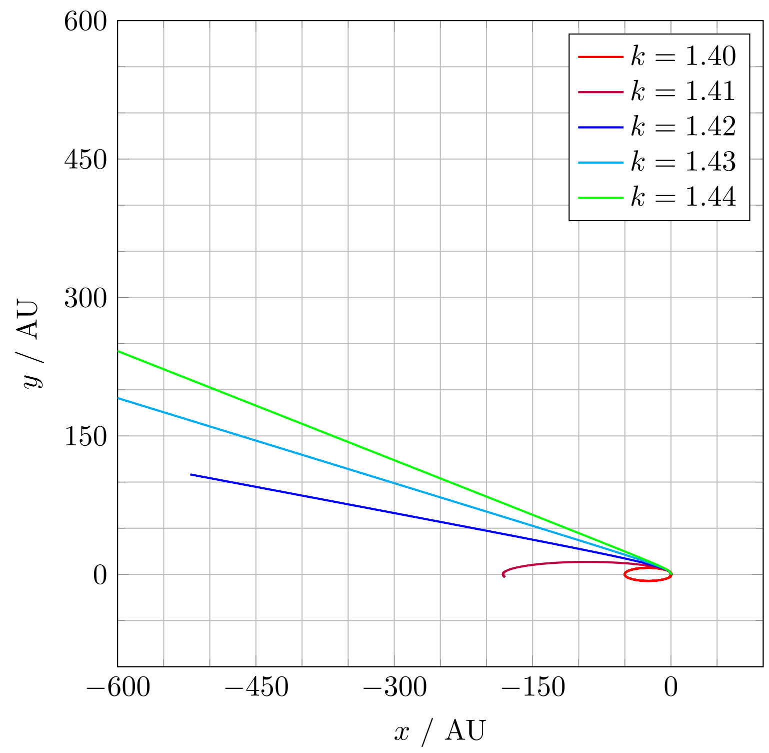

Figure 4: Plot of a planet's orbit at different initial velocities $v_0 = 2\pi k \cdot 1\,\mathrm{AU}\,\mathrm{yr}^{-1}$.

3.3 Earth-Jupiter-Sun system

e) We now add a third body, Jupiter, to our simulation. Because of the way we designed the program, this change only needs one or two more lines of code. We try three different "Jupiters": one with the realistic mass ($1 \times 10^{-3}\,\mathrm{M}_\odot$), one with ten times this ($1 \times 10^{-2}\,\mathrm{M}_\odot$) and one with 1000 times this ($1\,\mathrm{M}_\odot$).

As we see in figure 5, adding the realistic Jupiter does not significantly affect the orbit of the Earth. If we change the mass of Jupiter to 10 times its real mass, we see that it will disturb Earth's orbit slightly, making it slowly lose speed and move towards the Sun. This will likely cause the Earth to crash into the Sun after a number of years. When we use a mass 1000 times that of Jupiter, Earth's orbit becomes completely unstable, and the Earth will crash into the sun after only a few years.

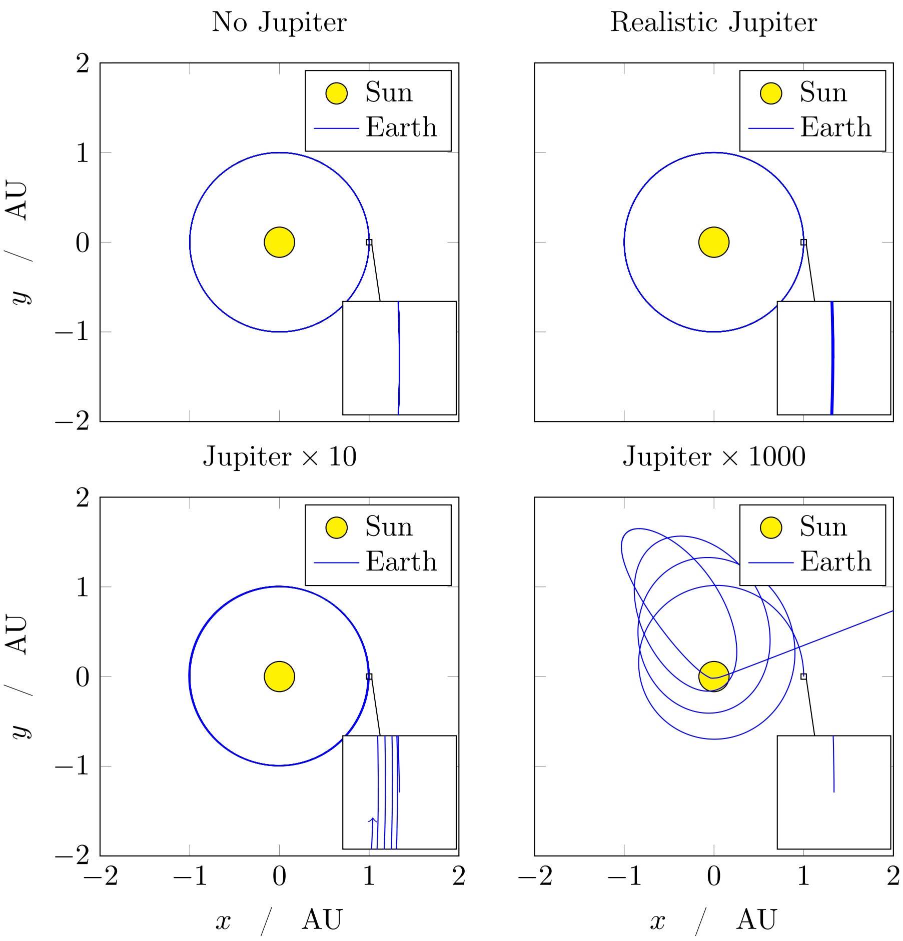

Figure 5: Five-year simulation of the Earth-Jupiter-Sun system, with different Jupiter masses. Sun is not to scale.

3.4 Complete solar system

f) We add all the remaining planets of the solar system, in addition to our own Moon, to the program, and run a simulation for 25 years. The paths of all of the planets are visualized in figures 6 and 7.

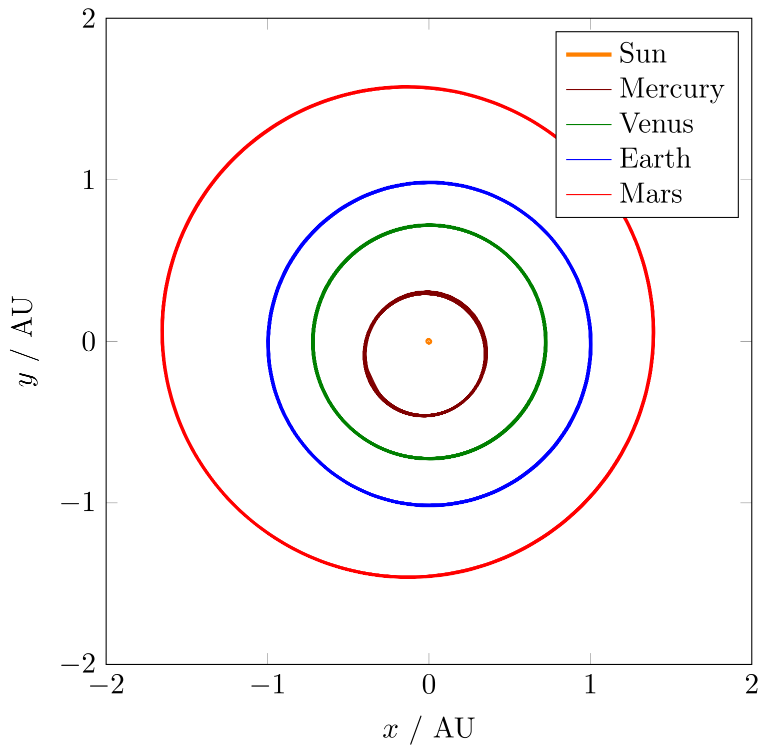

Figure 6: 25-year simulation of the inner solar system (Sun, Mercury, Venus, Earth, Mars).

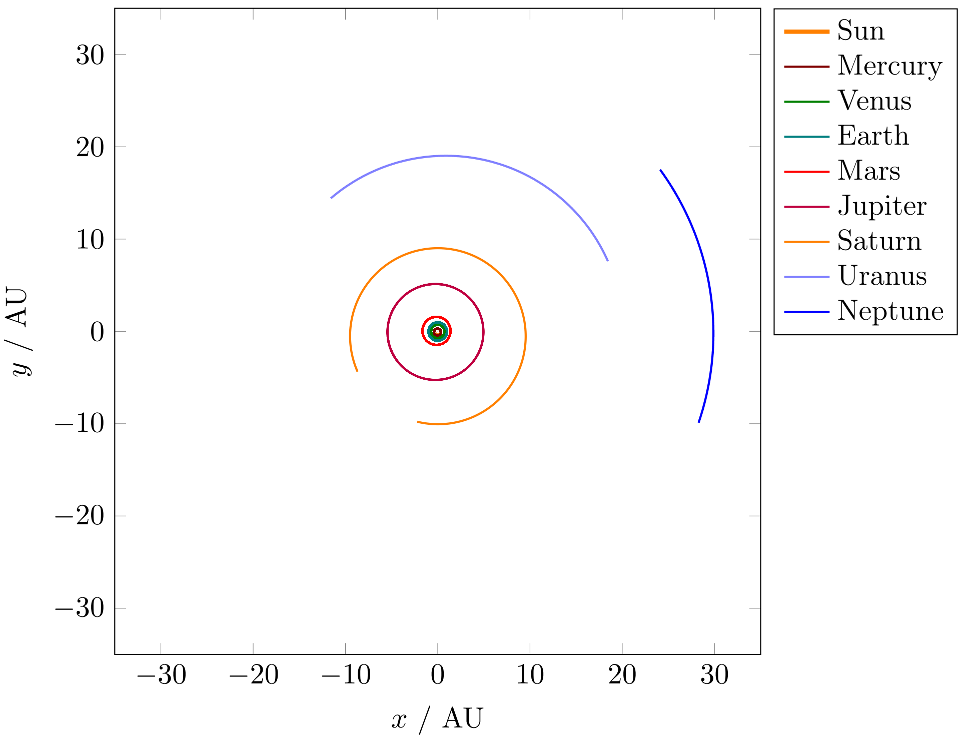

Figure 7: 25-year simulation of the outer solar system (Jupiter, Saturn, Uranus, Neptune).

3.5 Perihelion precession of Mercury

g) We rewrite our simulation to only handle the Mercury-Sun two-body system, with the Sun fixed at the origin. However, we modify the gravitational force wih a corrective factor for relativistic effects, so that it becomes

$$F_G = \frac{GM_{\text{mercury}}M_{\odot}}{r^2} \cdot \left(1 + \frac{3 \cdot |\vec{r} \times \vec{v}|^2}{r^2 c^2}\right)$$

We then run the simulation and note where Mercury is when it is at perihelion (at speed $12.44\,\mathrm{AU\,yr^{-1}}$). When we do this for normal Newtonian gravity and gravity with relativistic corrections, we see that Mercury is at perihelion at the angles shown in figure 8.

We see that perihelion with the normal Netwonian gravity barely moves at all, keeping within $\approx$ 1/1000 of a degree from where it started. Even this difference is most likely a result of errors in the numeric calculations we do, and so we can assume that perihelion occurs at the same position every time.

On the other hand, with the relativistically corrected gravity, the angle changes at a rate equal to approximately 0.011 degrees per century; this is very close to the 43 arc seconds (0.01194 degrees) per century we already know that Mercury's precession due to general relativity is. This precession can be explain by the fact that general relativity predicts a slightly stronger gravitational pull than Newtonian gravity when two bodies are close to each other, which in this case will alter the shape of the orbit slightly, causing a precession of the perihelion.

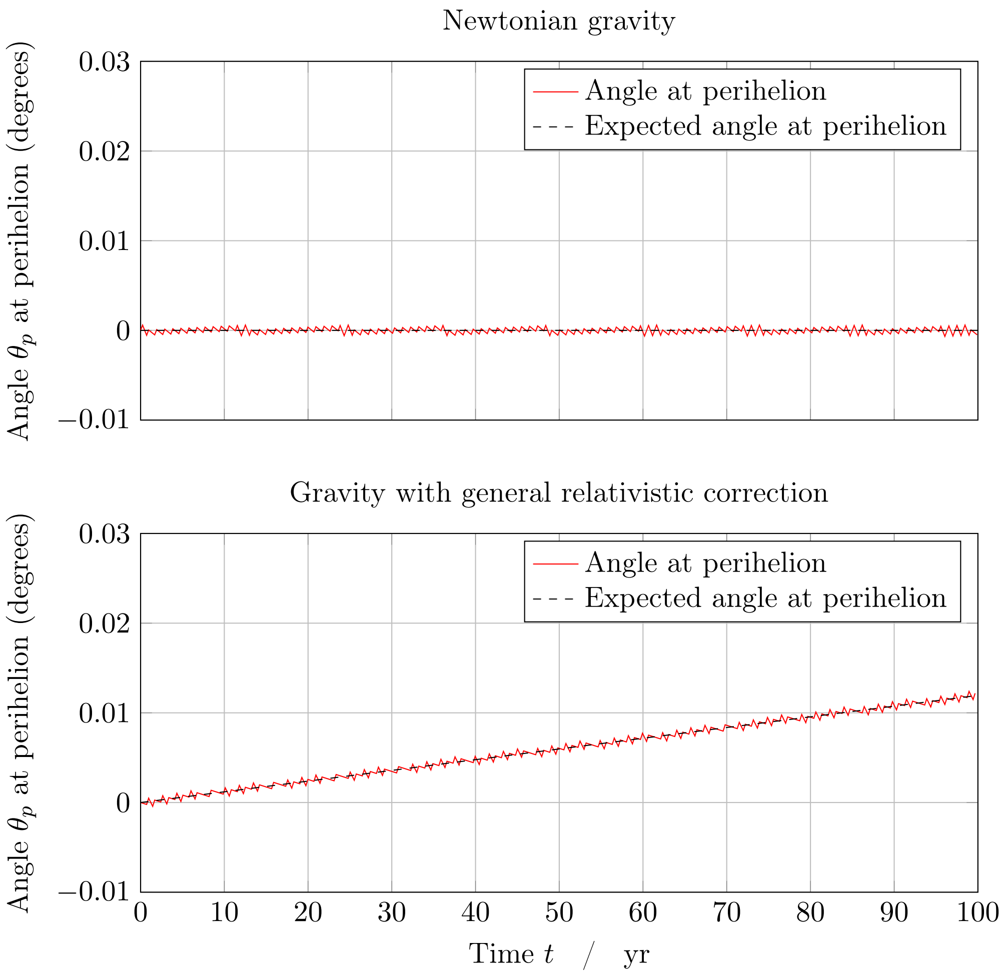

Figure 8: Angle of Mercury at perihelion as it changes over time, in the case of normal Newtonian gravity (top) and Newtonian gravity with relativistic corrections (bottom).

A Appendix

All files used in this project can be found at `https://github.com/frxstrem/fys3150/tree/master/project3`. The following code files are used:

b)
- `code/3b.cc`

d)
- `code/3d.cc`

e)
- `code/3e.cc`

f)
- `code/3f.cc`

g)
- `code/3g.cc`

In addition, the following files contains classes used in all of (or most of) the above programs:

- `code/CelestialBody.cc`
  - `CelestialBody`
- `code/SolarSystem.cc`
  - `SolarSystem`
  - `RelativisticSolarSystem`
- `code/Solver.cc`
  - `EulerSolver`
  - `VerletSolver`