FYS3150 Computational Physics - Project 3

Nicholas Karlsen

This is an abstract

INTRODUCTION

Lastly, the source code for any code discussed in this report can be found on my Github at: https://github.com/nicholaskarlsen/FYS3150

THEORY, ALGORITHMS AND METHODS

Newton's law of universal gravitation

Between every body, there is a force of attraction inversly proportional to the square of the separation, or more precisely, the force acting on some body with mass m due to a mass m' is

$$\mathbf{F} = -G\frac{mm'}{|\mathbf{r}-\mathbf{r}'|^2}\hat{\mathbf{u}}_{\mathbf{r}-\mathbf{r}'}, \quad \hat{\mathbf{u}}_{\mathbf{r}-\mathbf{r}'} = \frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|} \tag{1}$$

Where G is the gravitational constant and $\mathbf{r}, \mathbf{r}'$ denote the position vectors of bodies with mass m, m' respectively.

Chosing the the 2D cartesian coordinate system, let $\mathbf{r}-\mathbf{r}' = (x_r, y_r)$, where the r suffix denote that these coordinates are to be understood as the relative coordinates between bodies m, m'. Further, $|\mathbf{r}-\mathbf{r}'| = \sqrt{x_r^2+y_r^2} = r$ is the distance between the two bodies.

By Newtons second law, the acceleration on body 1 due to the gravitational pull of body 2 can then be written as

$$\mathbf{a} = \frac{1}{m}\mathbf{F} = -G\frac{m'}{r^2}\frac{(\mathbf{r}-\mathbf{r}')}{r} = -G\frac{m'}{r^2}\frac{(x_r, y_r)}{r} \tag{2}$$

Written out component-wise in terms of the positions, we get the set of coupled differential equations

$$\frac{\partial^2 x}{\partial t^2} = -G\frac{m'}{r^2}\frac{x_r}{r}, \quad \frac{\partial^2 y}{\partial t^2} = -G\frac{m'}{r^2}\frac{y_r}{r} \tag{3}$$

Similar, for 3 dimensions where $\mathbf{r}-\mathbf{r}' = (x_r, y_r, z_r), r = \sqrt{x_r^2+y_r^2+z_r^2}$.

Relativistic Correction

The aforementioned model of gravity fails to predict the perihelion (see fig. 1) precession of mercury, which is observed to be 43" per century [3].

That is, the closed, uniform elliptical orbits predicted by the Newtonian model for gravity does not match with observations in Astronomy, where the perihelion of Mercury seems to shift over time. In fact, the perihelion precession of mercury was the first experimental confirmations of General relativity, which accurately predicts this phenomena.

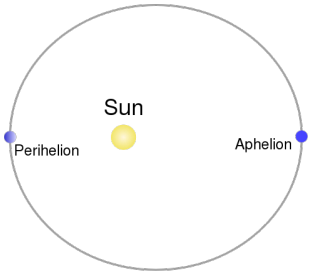


FIG. 1: In an eliptic orbit, the closest and farthest points in the orbit is defined as the Perhelion and Aphelion respectively [Image source]

And so, a correcting factor accounting for the relativistic effects is added to Newtons model, and the magnitude of the gravitational force becomes [3]

$$|\mathbf{F}| = G\frac{mm'}{r^2}\left[1+\frac{3l^2}{r^2c^2}\right] \tag{4}$$

Where l denotes the magnitude of the angular momentum of the orbiting body and c the speed of light.

Now, in order to find the perihelion precession we define a coordinate system such that

Solving ODEs numerically

Forward Euler

Consider a function $f(t)$, which derivative, $f'(t)$ is known and we want to find $f(t+\delta t)$. Take the taylor expansion of $f(t+\delta t)$

$$f(t+\delta t) = f(t)+f'(t)\delta t+f''(t)\delta t^2+\cdots+f^{(n)}(t)\delta t^n \tag{5}$$

If we then truncate this series after the second term we get

$$f(t+\delta t) = f(t)+f'(t)\delta t+\mathcal{O}(\delta t^2) \tag{6}$$

where the term $\mathcal{O}(\delta t^2)$ contains the numerical error associated by truncating the series early.

Since $f(t), f'(t)$ are known, this allows us to find $f(t+\delta t)$, which is the basis of the Forward Euler algorithm. Following, i have written out in pseudocode

how this algorithm can be used to solve a system governed by some known force $\mathbf{F}$ and initial conditions $\mathbf{v}(0) \simeq v_0, \mathbf{r}(0) \simeq r_0$.

Forward Euler Algorithm

```
for i = 0, ..., N − 1
    v_{i+1} = v_i + a_i Δt
    r_{i+1} = r_i + v_i Δt
```

Notice that the value of $\mathbf{v}_{i+1}$ is known when computing $\mathbf{r}_{i+1}$. By using this newly calculated velocity in the computation instead, we get the Euler-Cromer method, which wont be discussed further in this report.

Euler-Cromer Algorithm

```
for i = 0, ..., N − 1
    v_{i+1} = v_i + a_i Δt
    r_{i+1} = r_i + v_{i+1} Δt
```

Velocity-Verlet

The Velocity-Verlet method is another method for solving ODE's numerically, which just like Forward Euler is derived from the manipulation of Taylor expansions. However, this time i refer you to [2] for the full derivation and simply state the result

$$
\begin{aligned}
f(t+\delta t) &= f(t) + f'(t)\delta t + \frac{f''(t)\delta t}{2} + \mathcal{O}(\delta t^3) \\
f'(t+\delta t) &= f'(t) + \frac{\left(f''(t+\delta t) + f''(t)\right)\delta t}{2} + \mathcal{O}(\delta t^3)
\end{aligned}
\tag{7}
$$

Which again, can be applied to solve a Physical system where $\mathbf{F}$ is known. The algorithm for doing this is written out below

Velocity-Verlet Algorithm

```
for i = 0, ..., N − 1
    r_{i+1} = r_i + v_i Δt + (1/2) a_i (Δt)^2
    v_{i+1} = v_i + (1/2)(a_{i+1} + a_i) Δt
```

RESULTS AND DISCUSSIONS

Object orientation

When designing the class `n_solver`, i wanted to strike a balance between utilizing the benefits, and expandability you get from object orientation whilst also not abstracting the data too much. As such, the class simply manipulates arrays of a particular format in a particular way. Not abstracting the process by creating objects for each planet (or something else of that nature), which seems to be a pitfall of object orientation as i perceive it.

As such, the class can easily be expanded by adding new solvers and differential equations, to solve similar multi-body problems governed by a different set of coupled differential equations.

A particular benefit in the way i wrote my code, is that there is no difference if the input data is in 2 Dimensions or 3. The code will work just the same either way by making full use of the way Numpy arrays work.

I also made an attempt to utilize just-in-time (JIT) compilation by using jitclass from the Numba package, however, jitclass is poorly documented in its current state and imposes many restrictions as to what is permitted compared to regular python, or even the standard jit flag that is used for regular python functions. Eventually, i reached the point where i would have to sacrifice the modularity of my program in order to run it using jitclass, which obviously was not an option. So i gave up on the endeavor.

Earth-Sun System

Working in units $M_\odot, AU, Yr$ for mass, length and time respectively, i initialized a 2D system using the `n_solver` class where the sun is fixed at the origin and the earth has initial velocity $\mathbf{v}_0 = (0, 2\pi)$ at position $\mathbf{r}_0 = (1, 0)$, corresponding to an orbit of eccentricity, $e = 0$, a perfect circle.

The system was then simulated for a time period of 10 years using a different number of integration points for both the Velocity-Verlet and Forward Euler algorithms. The resulting orbits shown in Fig. 3, shows that the Velocity-Verlet solution yields a reasonable orbit for $N = 10^3$ integration points, whilst the Forward Euler algorithm doest produce a comparable orbit until $N = 10^6$ integration points.

CONCLUSIONS

[1] M. Hjorth-Jensen, Computational Physics - Lecture Notes 2015, (2015).

[2] M. Hjorth-Jensen, Ordinary differential equations - Computational Physics Lectures (2017)

[3] M. Hjorth-Jensen, Building a model for the solar system using ordinary differ- ential equations - Project 3 (2018)

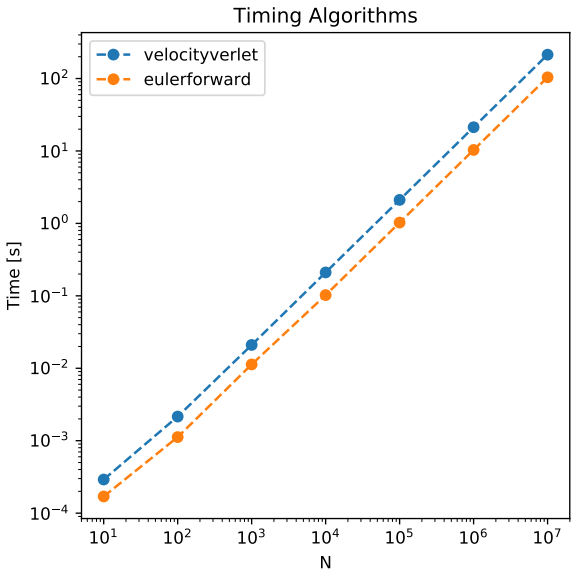
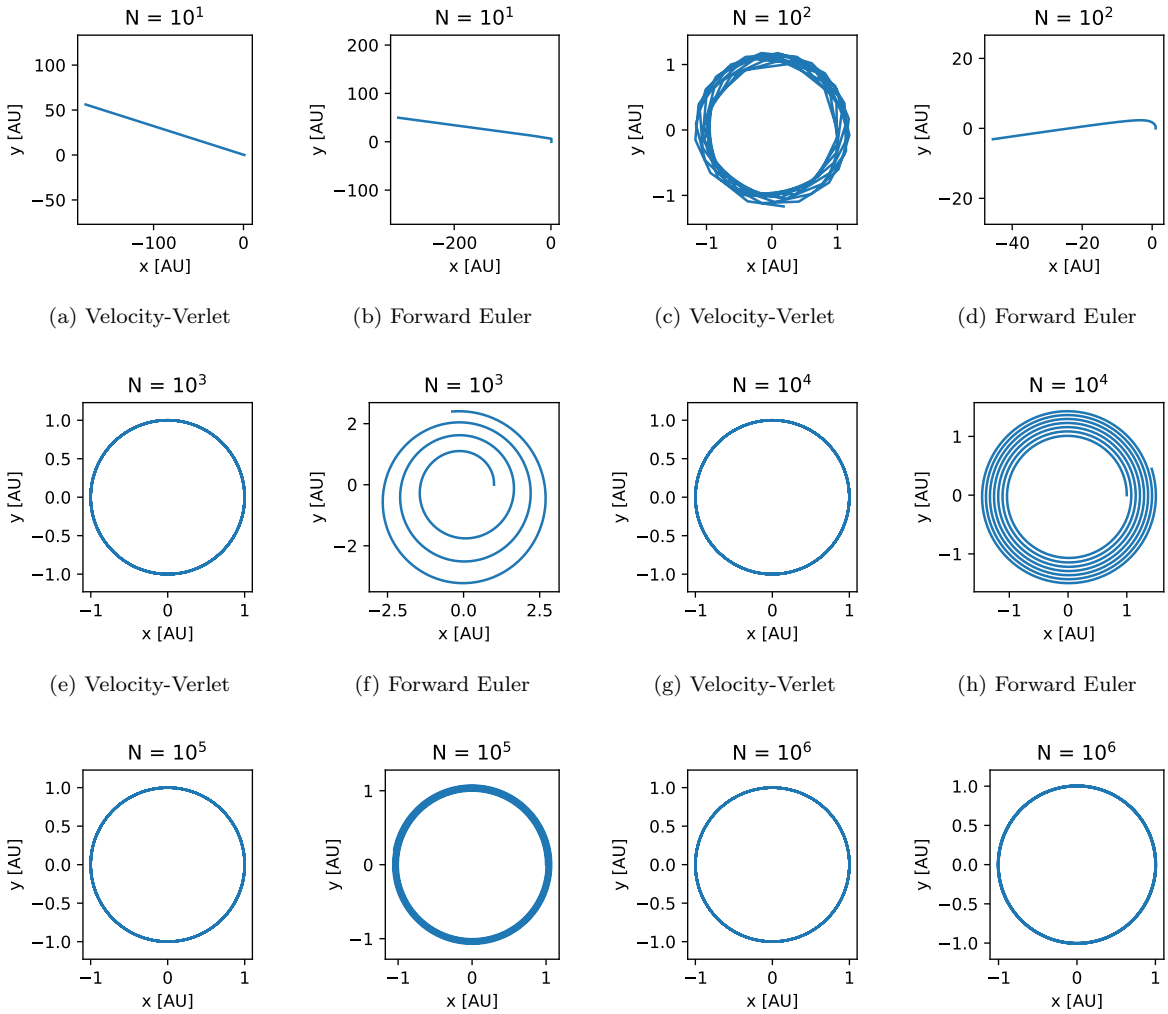


FIG. 2: Time taken to solve the Earth-Sun system for different number of integration points for $t_n = 10\,Yr$

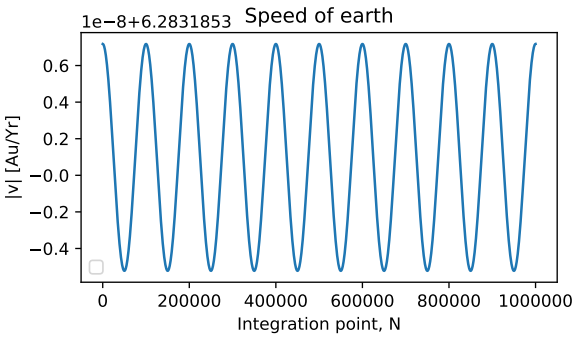


(a) Velocity-Verlet (b) Forward Euler (c) Velocity-Verlet (d) Forward Euler

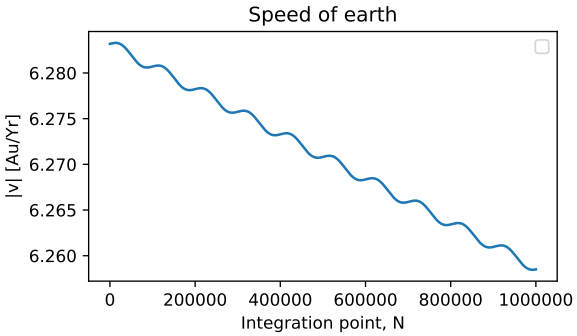
(e) Velocity-Verlet (f) Forward Euler (g) Velocity-Verlet (h) Forward Euler

(i) Velocity-Verlet (j) Forward Euler (k) Velocity-Verlet (l) Forward Euler

FIG. 3: The orbit of earth around a stationary sun for a timeperiod of 10 Years with simulated with a varying number of integration points N (see fig titles) using the Velocity-Verlet and Forward Euler algorithms

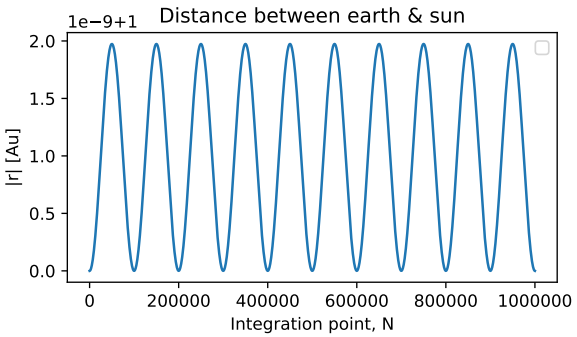


(a) Velocity-Verlet

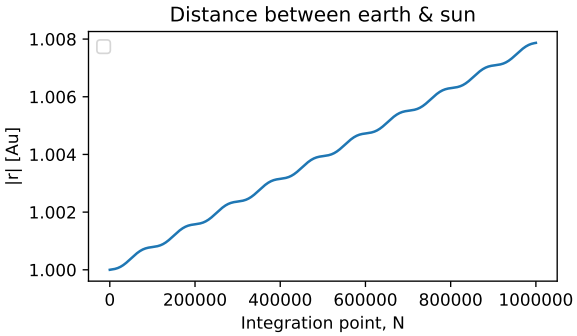


(b) Forward Euler

FIG. 4: The fluctuation of the total energy of the Earth in the Earth-Sun system for solutions using the Velocity-Verlet algorithm (a) and the Forward Euler algorithm (b) in a 10 Year simulation

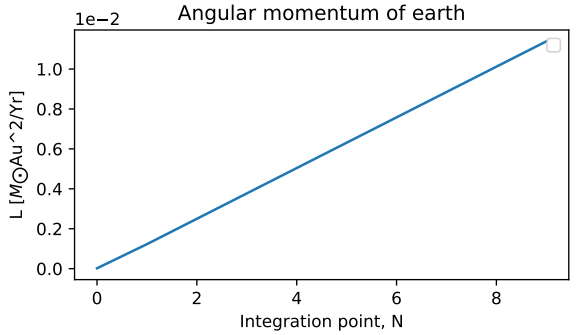


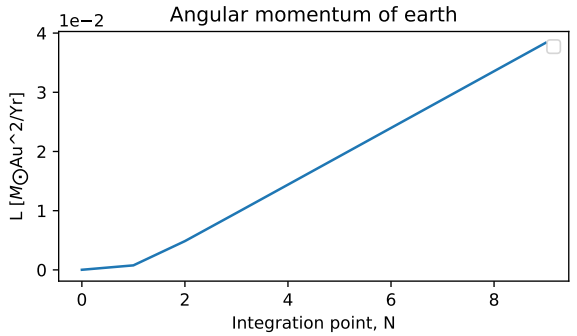
(a) Velocity-Verlet

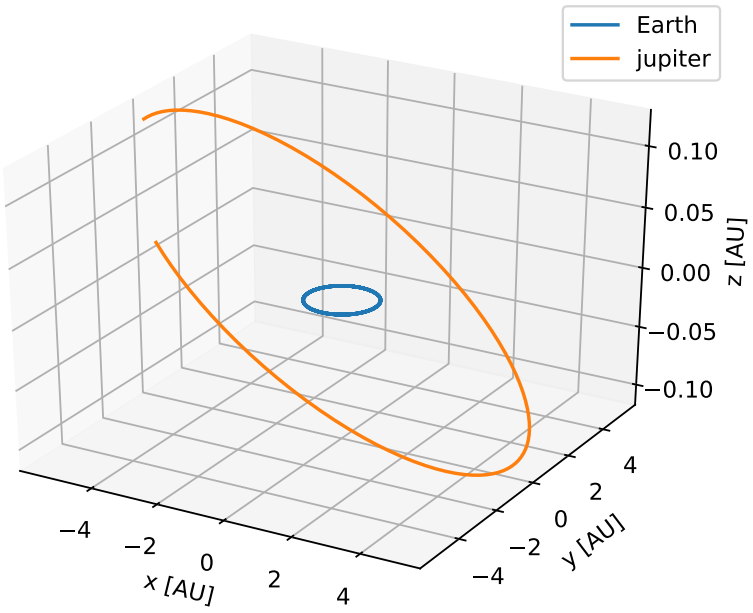


(b) Forward Euler

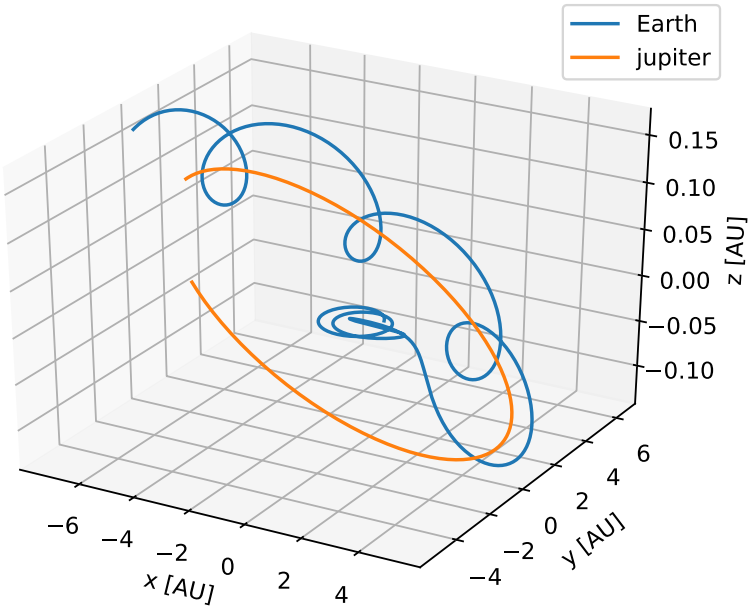
FIG. 5: The fluctuation of the total energy of the Earth in the Earth-Sun system for solutions using the Velocity-Verlet algorithm (a) and the Forward Euler algorithm (b) in a 10 Year simulation



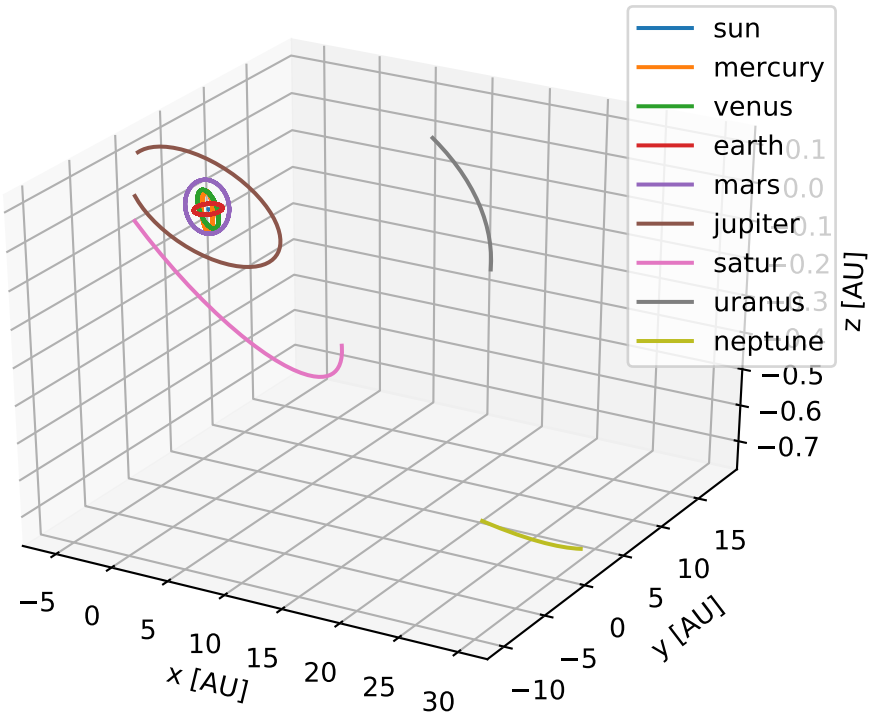



Earth
jupiter
0.10
0.05
0.00
−0.05
−0.10
z [AU]
−4
−2
0
2
4
x [AU]
−4
−2
0
2
4
y [AU]

(a)


Earth
jupiter
0.15
0.10
0.05
0.00
−0.05
−0.10
z [AU]
−6
−4
−2
0
2
4
x [AU]
−4
−2
0
2
4
6
y [AU]

(b)


sun
mercury
venus
earth
mars
jupiter
satur
uranus
neptune
x [AU]
y [AU]
z [AU]
−5
0
5
10
15
20
25
30
−10
−5
0
5
10
15
−0.5
−0.6
−0.7