Using ordinary differential equations to build a three dimensional model for the solar system

Are Frode Kvanum*
Department of Geoscience, University of Oslo
(Dated: October 26, 2020)

This project studies how to use a system of coupled ordinary differential equations written as an initial value problem to build a model for the solar system. This is done by studying different discretized methods for solving initial value problems, as both Euler's forward algorithm and the velocity Verlet algorithm is considered. By using the developed solvers, it is shown that the asymmetry in the equation for computing time in the Euler solver, results in the energy of the system not being conserved. It is however shown that the symplectic Verlet solver is conserving energy. With using the Verlet solver, different scenarios of the two body Earth-Sun case is studied. The algorithm is able to conserve angular momentum, thus proving a result obtained from Kepler's second law. Different form of the gravitational force exerted by the Sun is also tested, where it is seen that $F(r) \propto -1/r^{\beta}$ only destabilizes the result for $\beta = 3$. By expanding the simulation with Jupiter, the Verlet solver is able to determine how the force exerted by Jupiter perturbs the kinetic and potential energy of the Earth. Though Jupiter has to be included with a mass equal to that of the Sun for it to destabilize the Verlet solver.

INTRODUCTION

Ordinary differential equations are functions containing one independent variable which relates the derivative of said function. Though a function on the form $\frac{dy}{dt} = f(t, y)$ might appear simple at first glance, their extensibility into numerous fields of science such as biology, engineering, physics and more have given these equations a status of high importance. To illustrate their influence of how we describe the world today, Newton's laws would not have been possible without the theory of ordinary differential equations. (Hjort-Jensen [2]) As can be seen by describing Newton's second law, *the change in motion is proportional to the force acting on the body*, as a second order ordinary differential equation $m\frac{d^2r}{dt^2} = \underline{F}$.

For this project, the basic theory for ordinary differential equations will serve as the starting point when building a model for the solar system. A prominent feature of ordinary differential equations that becomes apparent when discretizing them, is that they can be used to simulate systems which evolve over time. The only constraint is that they need to be supplied with initial conditions with information of the system's values at the starting position. This is called an initial value problem, and as the system is discretized, it can easily be solved using different finite difference methods. A downside of solving a differential equation with a finite difference method, is that it involves numerous calculations of the same equations for all included timesteps. Calculations gets even more hairy if the system we are considering is a set of coupled differential equations, as this would introduce even more equations to solve for each timestep, thus resulting in a solution being tedious to find using only pen and paper. Luckily, as have been seen in the previous projects, computers are an ideal tool for solving a lot of equations quickly. What would be deemed a daunting task analytically is a simple task for a computer, requiring only the algorithm for solving the system of difference equations to be translated into computer code. As such, the solar system model will be considered a system that evolves over time, with the calculations for how it develops over time being left as an exercise for the computer's central processing unit.

It was previously alluded to the existence of several finite difference methods, through the usage of plurals when referring to these algorithms. It will be shown in the following sections that, by manipulating Taylor Expansions while discretizing the starting equations that relate the objects in the solar system, we can obtain different schemes that solve the same initial value problem. This project will develop two algorithms for solving the resulting coupled ordinary differential equations, Euler's forward algorithm and the velocity Verlet method. A point of interest that emerges when considering several algorithms that solve the same initial value problem is in what way the solution of the algorithms deviates. This will be inspected when the deviation becomes apparent, as though the developed algorithms can be considered numerical methods, their differences can be explained by relating the numerical mathematics to the laws of the physical system.

Though a thorough discussion of the source code developed for this project will not be provided, it is noteworthy to mention that the solar system model will be developed in C++. Mainly for its fast runtime when doing heavy calculations, but also due to the language supporting Object Oriented Programming, a programming paradigm which will be used due to its ability to easily relate the planets in the solar system as C++ Objects. This way of relating the solar system to C++ code also simplifies the potential inclusion of additional moons and planets to the solar system model, resulting in a clean and ex-

* afkvanum@mail.uio.no

tensible code. Even though the majority of the code will not be discussed, some of the algorithmic aspects will be highlighted whenever relevant. For the source code in its entirety, refer to the GitHub repository which contains all developed source code found in the Appendix.

The model will be built and discussed incrementally, that is, we will start off by only considering the case of an Earth - Sun system, with additional planets being introduced as it is confirmed that the model works for systems with fewer planets. Our starting equation for the Earth - Sun system will therefore be

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

as it is assumed that the Earth's orbit around the sun is close to circular. In the following section, I will show how to derive the relation $v^2 r = GM_\odot = \frac{4\pi^2 \text{AU}^3}{\text{yr}^2}$ from the prior equation by using the theory of circular dynamics. It will also be shown how the latter equation can be discretized such that the finite difference scheme algorithms can be set up for solving the discretized equations.

THEORY

Scaling with respect to the Sun

Note that the derivation of the equations will be discussed in two dimensions as to reduce bloating and increase readability. Adding a third dimension to this problem is analogous to the following derivation. The Earth - Sun system is a hypothetical solar system where the Earth is orbiting the Sun. For this simplification of the solar system, the only force in this problem is gravity, which for this specific system is given by Newton's law of gravity

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

where $M_\odot$ is the mass of the Sun and M_{Earth} being the mass of the Earth. $G = 6.67 \times 10^{-11}[\frac{Nm^2}{kg}]$ is the gravitational constant and $r = 1AU$ is the distance between the Sun and the Earth. As we assume that mass of the Sun greatly exceeds that of the Earth, the motion of the sun can be safely neglected. By assuming that the Earth is orbiting around the sun in an xy-plane. Throughout this project, I will stick to the convention that a is the acceleration vector and v is the velocity vector, denoted with subscripts x, y, z to signify dimension. Newton's second law of motion gives us the following equations

$$a_x = \frac{dv_x}{dt} = \frac{F_{Gx}}{M_{\text{Earth}}}$$

and

$$a_y = \frac{dv_y}{dt} = \frac{F_{Gy}}{M_{\text{Earth}}}$$

By assuming that our Earth - Sun system is in Cartesian coordinates, we can write $r = \sqrt{x^2 + y^2}$, which gives us that $x = r\cos\theta$ and $y = r\sin\theta$. From this, we can write $\frac{F_{Gx}}{r} = r\cos\theta$. Resulting in $F_{Gx} = -\frac{GM_\odot}{r^3}x$, similar derivation for the y-direction which ends up being. $F_{Gy} = -\frac{GM_\odot}{r^3}y$.

The next step in this scaling is to get rid of $GM_\odot$. Considering that the Earth is orbiting the Sun in a spherical orbit, we know that the acceleration can be expressed as an centrifugal acceleration. That is

$$\underline{a} = \frac{\underline{v}^2}{r} = \frac{\underline{F_G}}{M_{\text{Earth}}}$$

Which can be rewritten as $\underline{v}^2 r = GM_\odot$. Moreover, as the motion is circular, we can express v as $v = \frac{2\pi r}{1Yr}$. By then using the fact that $r = 1(AU)$ and $GM_\odot = v^2 r$, we obtain the following scaling for $GM_\odot$

$$GM_\odot = \frac{4\pi^2 (AU)^3}{(Yr)^2}$$

Thus, our scaled coupled ordinary differential equations ends up being

$$\frac{dv_x}{dt} = a_x = -\frac{4\pi^2 x}{\sqrt{x^2 + y^2}}, \quad v_x = \frac{dx}{dt}$$

and

$$\frac{dv_y}{dt} = a_y = -\frac{4\pi^2 y}{\sqrt{x^2 + y^2}}, \quad v_y = \frac{dy}{dt}$$

Quick overview of Kepler's laws

Johannes Kepler deduced three laws of planetary motion in the early parts of the seventeenth century. Though the laws he inferred were only derived from observations, they still stand today as important laws of planetary motion. The three laws defined by Kepler, usually referred to as Kepler's laws, state the following:

(1) All planets move along elliptical paths with the Sun at one focus.

(2) A line segment connecting any given planet and the Sun sweeps out area at a constant rate.

(3) The square of a planet's orbital period about the Sun, P_{orb}, is proportional to the cube of its semi-major axis, an i.e., $P_{\text{orb}}^2 \propto a^3$.

These laws are stated as in de Pater and Lissauer [1]. Kepler's second and third law can be described in further detail by translating them into precise mathematical

terms. For Kepler's second law, assume that the line connecting the Sun with the given planet sweeps out the area A at a constant rate. Then we have that

$$\frac{dA}{dt} = \text{constant}$$

with the constant being dependent on the given planet. The case for Kepler's third law, some assumptions have to be made. Firstly, the planet's orbital period about the Sun has to be in years. Secondly, the distance has to be given in AU. Then, the proportionality $P^2_{\text{orb}} \propto a^3$ can be written as an equality

$$P^2_{\text{yr}} = a^3_{\text{AU}}$$

These mathematical definitions have been derived following the approach of de Pater and Lissauer [1]. With Kepler's laws defined and described mathematically, we have obtained a tool which can be used to test the physical properties of our model for the solar system.

METHODS

Deriving the Euler's forward algorithm

When deriving a general formula for the Euler's forward algorithm, we will consider a discretized coupled ordinary differential equation over a timespan $t \in [t_0, t_n]$. Where $t_i = t_0 + ih$ given that $i = 0, 1, 2, \ldots, n$ and $h = \frac{t_n - t_0}{n}$. This implies that $x(t_i) = x_i$ and $v(t_i) = v_i$. Moreover, by Taylor Expanding around a point $(t_i + h)$, we get the following equations

$$x(t_i + h) = x(t_i) + hx'(t_i) + O(h^2)$$

and

$$v(t_i + h) = v(t_i) + hv'(t_i) + O(h^2)$$

Which result in the following equations, omitting the truncation error,

$$\begin{aligned} x_{i+1} &= x_i + hv_i \\ v_{i+1} &= v_i + ha_i \end{aligned} \tag{1}$$

Which can then be turned into the following algorithm

Algorithm 1 Forward Euler

Input: Initial state of the system, final time and n points
Output: The state of the system at the final time

Initialize $h \leftarrow \frac{\text{final time}}{n}$
Initialize time $\leftarrow 0$
while time $<$ final time **do**
    $x[\text{time} + \text{h}] = x[\text{time}] + \text{h}v[\text{time}]$
    $v[\text{time} + \text{h}] = v[\text{time}] + \text{h} * a[\text{time}]$
    time $\leftarrow$ time + h
end while

Deriving the velocity Verlet method

When deriving a general form for the velocity Verlet algorithm, the same considerations will be taken as the prior section where Euler's forward algorithm were developed. As such, formalities will be voided. Start off by Taylor expanding the position around the point $(t_i + h)$

$$x(t_i + h) = x_(t_i) + hx'(t_i) + \frac{h^2}{2}x''(t_i) + O(h^3)$$

and the velocity

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

The second derivative of the velocity, written more compactly as v''_i, has no immediate physical quantity in which to relate to. Thus, a Taylor expansion for the first derivative of v is performed.

$$v'_{i+1} = v'_i + hv''_i + O(h^2)$$

which can be rewritten as $v''_i = \frac{v'_{i+1} - v'_i}{h} + O(h)$. Plugging this into second order derivative term for the velocity, results in the final coupled system, once again omitting the truncation error

$$\begin{aligned} x_{i+1} &= x_i + hv_i + \frac{h^2}{2}a_i \\ v_{i+1} &= v_i + \frac{h}{2}\left[a_{i+1} + a_i\right] \end{aligned} \tag{2}$$

This system of equations can be turned into the following algorithm

Algorithm 2 velocity Verlet

Input: Initial state of the system, final time and n points
Output: The state of the system at the final time

Initialize $h \leftarrow \frac{\text{final time}}{n}$
Initialize time $\leftarrow 0$
while time $<$ final time **do**
    $x[\text{time} + \text{h}] = x[\text{time}] + \text{h}v[\text{time}] + 0.5\text{h}^2 a[\text{time}]$
    $v[\text{time} + \text{h}] = v[\text{time}] + 0.5\text{h}\,(a[\text{time} + \text{h}] + a[\text{time}])$
    time $\leftarrow$ time + h
end while

Note for both the developed algorithms, that their implementation in the source code may differ from their conceptual idea. As such, the two prior algorithms give a general idea behind how the solar system is evolving over time, with respect to which quantities drive the change in position and velocity forward. As these algorithms require several additional computations when implemented, the resulting FLOPs will be based on their actual C++ implementation.

RESULTS

All results are generated using the results obtained from the souce code, on a laptop running Ubuntu 20.04.1. All computations are done in C++, with the compile g++, version 9.3.0, and optimized with the compiler flag -O3 which results in the code having the highest amount of compiler optimization enabled. The plots are generated using the Python package Matplotlib, version 3.3.2. All Python code regarding plotting and general overhead has been successfully ran using Python 3.8.5. Reproduction of the results is highly encouraged, and the reader is referred to the GitHub repository which contains instructions on how to compile and run the code.

Euler's forward algorithm

The implementation of Euler's forward algorithm 1 is done with the following C++ code

```
void SolarSystem::advance_fe(Planet &current, int idx, double step
     , double **a, double &Fx, double &Fy, double &Fz) {
    Fx = Fy = Fz = 0.0;

    gravitational_force(current, idx, Fx, Fy, Fz);

    a[idx][0] = Fx / current.get_M();
    a[idx][1] = Fy / current.get_M();
    a[idx][2] = Fz / current.get_M();

    for (int j = 0; j < m_dimension; j++) {
        double new_pos = current.get_pos(j) + step*current.get_v(j
     );
        current.set_pos(j, new_pos);
        double new_vel = current.get_v(j) + step*a[idx][j];
        current.set_v(j, new_vel);
    }
}
```

Listing 1. C++ implementation of the time advancement step for Euler's forward algorithm.

From inspecting the above code, it can be counted that for each planet, the algorithm performs three divisions. Furthermore, for each dimension of said planet, the algorithm performs two multiplications and two additions. Moreover, the call to *gravitational_force* costs 24 FLOPs for each Planet excluding the current Planet. Also in *gravitational_force* is a call to the Planet method *distance*. *distance* performs 8 FLOPs, as well as a call to the CMath function *sqrt*. As I am not able to easily determine in what way the function *sqrt* calculates the square root, getting the precise number of FLOPs performed by the algorithm unmanageable. Therefore it will be assumed that *sqrt* performs one FLOP, though this is an underestimation. The total FLOPs from *distance* is thus 14. In total, assuming three dimensions, for each Planet a total of $15 + 38(\text{Total planets} - 1)$ FLOPs are performed. Note that for the analysis, all FLOPs are assumed to have an equal execution time.

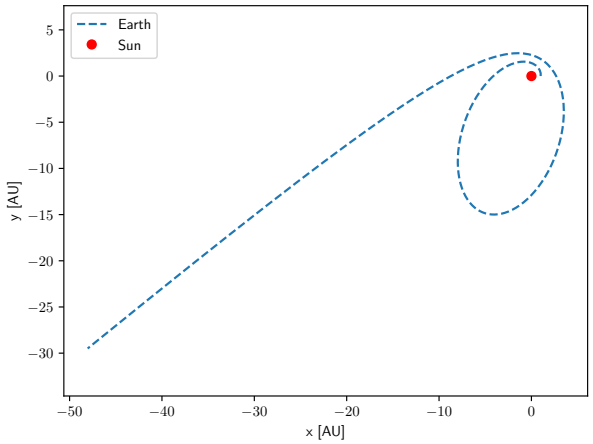

FIG. 1. Position of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.05 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

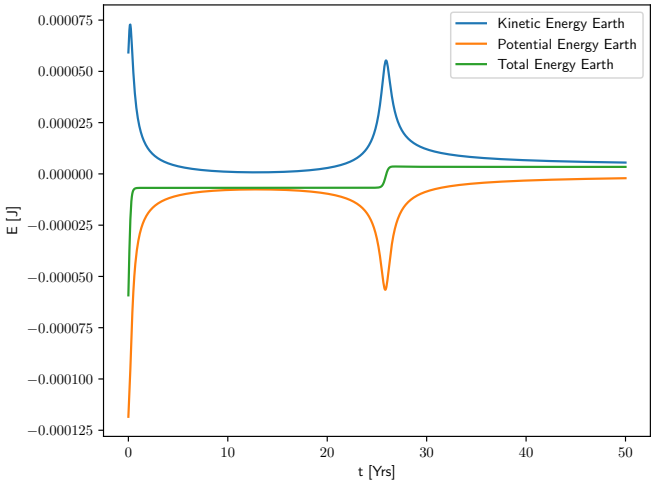

FIG. 2. Potential, kinetic and total energy of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.05 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

The velocity Verlet algorithm

The implementation of the velocity Verlet algorithm is done with the following C++ code

A similar inspection of the implementation, which is the C++ implementation of the velocity Verlet algorithm, as that of Listing 17 will be performed. For each planet, six divisions and two calls to *gravitational_force* is performed. In addition to the non-advancing calls, position and velocity advancements are performed in separate loops, the reason for which will be explained in the ensuing discussion. Again for simplicity, it is assumed that the calculations are performed in three dimensions. The posi-

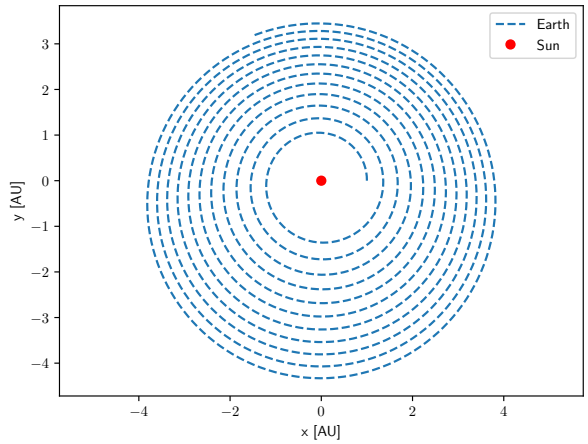

FIG. 3. Position of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

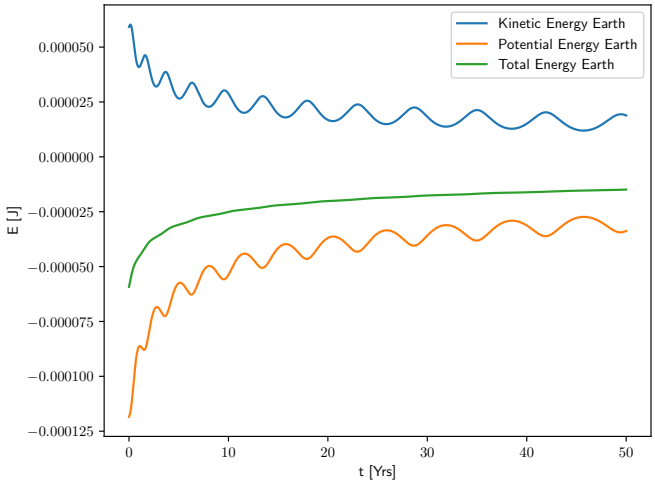

FIG. 4. Potential, kinetic and total energy of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

tional loop 18 FLOPs, whereas the velocity loop performs 12 FLOPs. In total, for each Planet, the velocity Verlet algorithm performs $36 + 76(\text{Total planets} - 1)$FLOPs.

Conservation of angular momentum

Both Figure 13 and 14 show how the angular momentum of the Earth change over time. First for the case of an circular orbit where initial velocity $v_{y0} = 2\pi$, then for the case of an elliptical orbit, i.e. $v_{y0} = 5.0$. Both simulations are run over the same time, with identical timestep $h = 0.0005$.

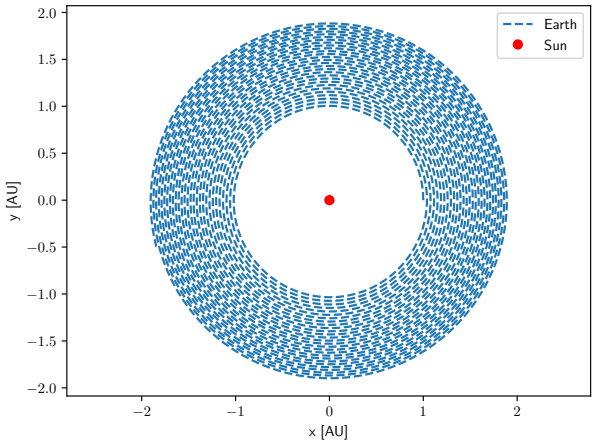

FIG. 5. Position of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

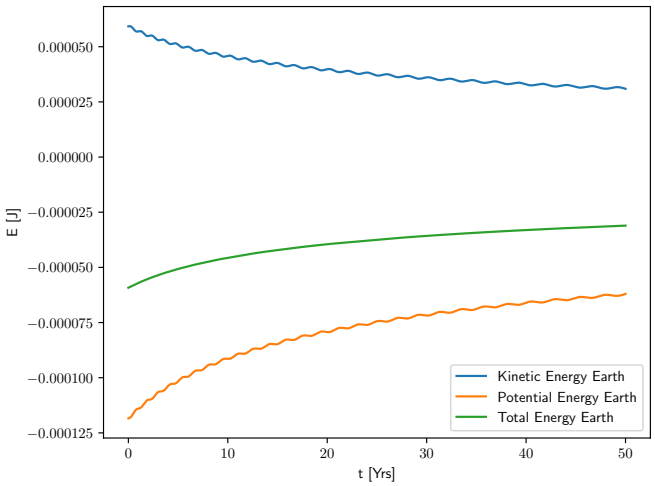

FIG. 6. Potential, kinetic and total energy of the Earth around a static Sun simulated using Euler's forward algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

TABLE I. The total runtime for both Euler's forward algorithm and the velocity Verlet algorithm, for the case of a final time set to 50 yrs. The decrease in timestep is a result of the amount of mesh points being increased

Timestep	Forward Euler [ms]	velocity Verlet [ms]
0.05	1.166	1.347
0.005	11.079	13.491
0.0005	106.921	132.946
0.000 05	1123.608	1380.719

```cpp
void SolarSystem::advance_vv(Planet &current, int idx, double step
     , double **a, double **a_new, double &Fx, double &Fy, double
      &Fz, double &Fx_new, double &Fy_new, double &Fz_new) {
    Fx = Fy = Fz = Fx_new = Fy_new = Fz_new = 0.0;

    gravitational_force(current, idx, Fx, Fy, Fz);

    a[idx][0] = Fx / current.get_M();
    a[idx][1] = Fy / current.get_M();
    a[idx][2] = Fz / current.get_M();

    for (int j = 0; j < m_dimension; j++) {
        double new_pos = current.get_pos(j) + current.get_v(j)*
     step + 0.5*step*step*a[idx][j];
        current.set_pos(j, new_pos);
    }

    gravitational_force(current, idx, Fx_new, Fy_new, Fz_new);
    a_new[idx][0] = Fx_new / current.get_M();
    a_new[idx][1] = Fy_new / current.get_M();
    a_new[idx][2] = Fz_new / current.get_M();

    for (int j = 0; j < m_dimension; j++) {
    double new_vel = current.get_v(j) + 0.5*step*(a_new[idx][j] +
     a[idx][j]);
        current.set_v(j, new_vel);
    }
}
```

Listing 2. C++ implementation for the time advancement step of the velocity Verlet algorithm.

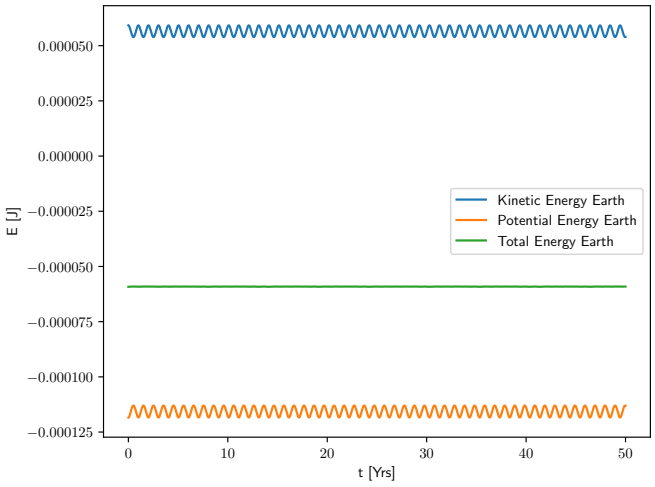

FIG. 8. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.05 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

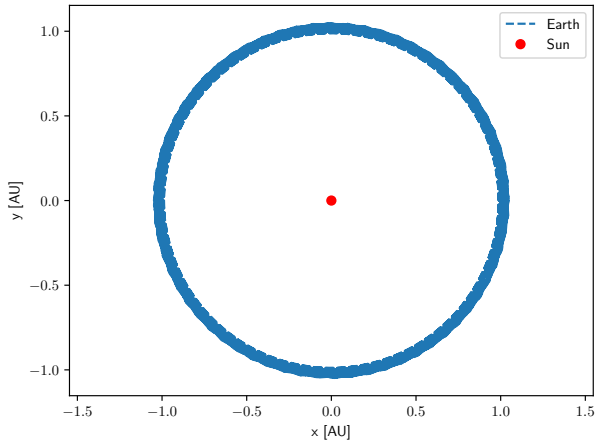

FIG. 7. Position of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.05 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

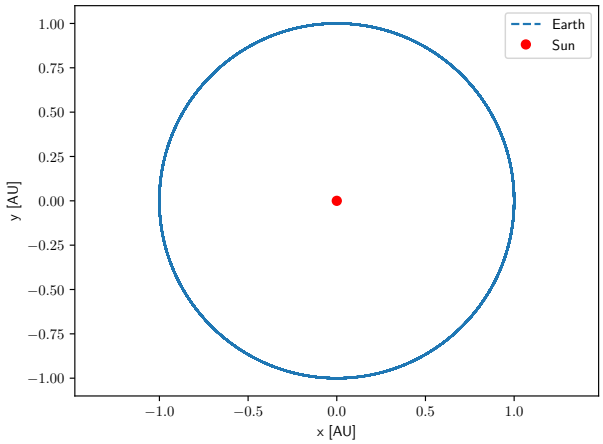

FIG. 9. Position of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

Modifying Newton's law of gravity

The following Figures 15 - 17 have been created by replacing the exponent of the radii in the denominator with $\beta \in [2, 3]$ in the gravitational force.

Considering an elliptical orbit

When considering an elliptical orbit, the initial values have to slightly modified to ensure that the orbit doesn't end up as a circular one. The initial values in position is not changed, resulting in $x_0 = 1(\text{AU})$ as in the case for the circular orbit. For the initial velocity value, which for the circular orbit only was $v_{y0} = 2\pi$, is reduced to $v_{y0} = 5.0[\frac{\text{AU}}{\text{yr}}]$. These initial values ensures that the resulting orbit is elliptical.

Escape velocity

The following Figures 27 - 28 where created with a simulation setting the total energy equal to zero, that is $E_{\text{tot}} = \frac{mv^2}{2} - \frac{GM_{\odot}M_{\text{Earth}}}{r} = 0$ which results in the

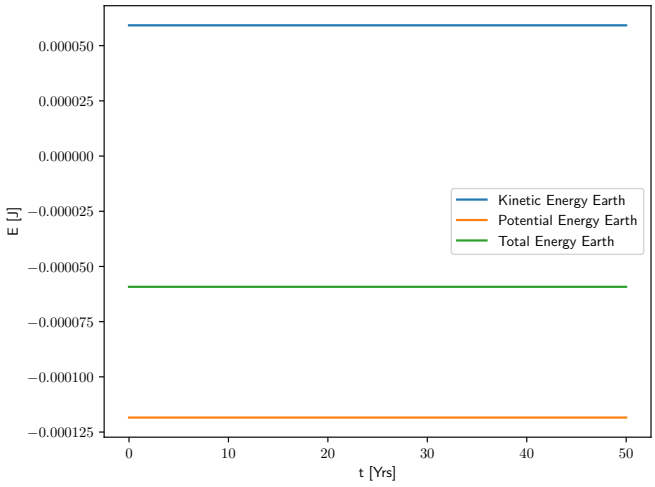

FIG. 10. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

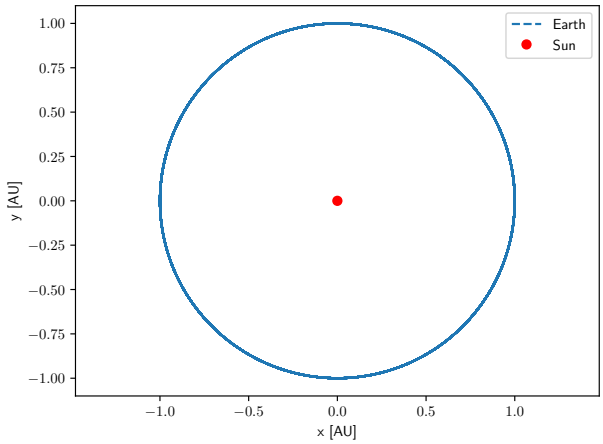

FIG. 11. Position of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

following escape velocity $v_{\text{esc}} = \sqrt{\frac{2GM_\odot}{r}}$. This velocity has then been used as the initial velocity for $v_{y0} = v_{\text{esc}}$.

Systems with more than two bodies

The following Figure 29 - 34 considers a three-body system where the Sun is static, for increasing magnitudes of mass for Jupiter. Furthermore, Figure 35 and 36 show respectively a plot for a Sun-Earth-Jupiter system where all Planets are moving, and a full solar system simulation where the data for the initial positions for the

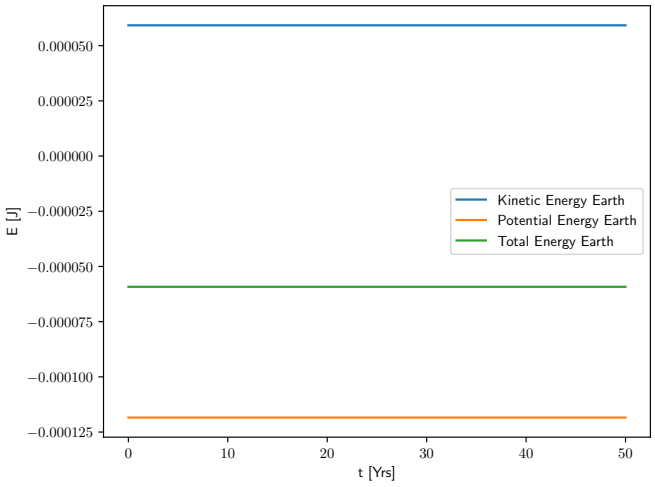

FIG. 12. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$

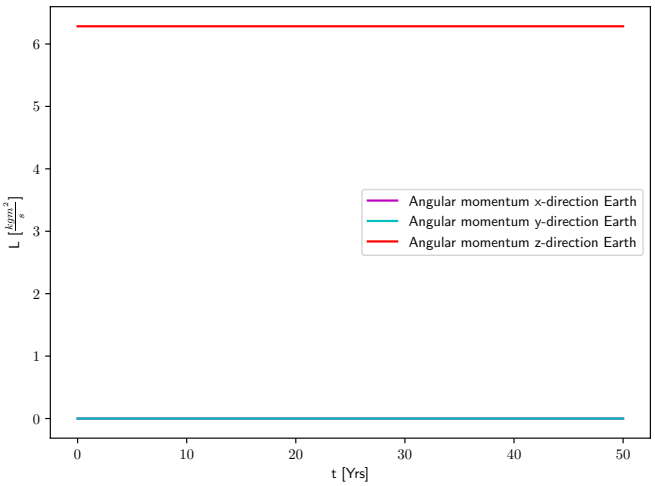

FIG. 13. Angular momentum in the x,y and z direction for the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$ resulting in a circular orbit

final simulation are their positions and velocities as of 12.10.2020 downloaded from NASA [3].

CONCLUSIONS AND DISCUSSION

The binary Earth-Sun system

When discussing the binary Earth-Sun system, it is of note that the calculations that advance the system in time has been performed for the Earth. I.e. the sun is only considered a static point in space. The initial conditions

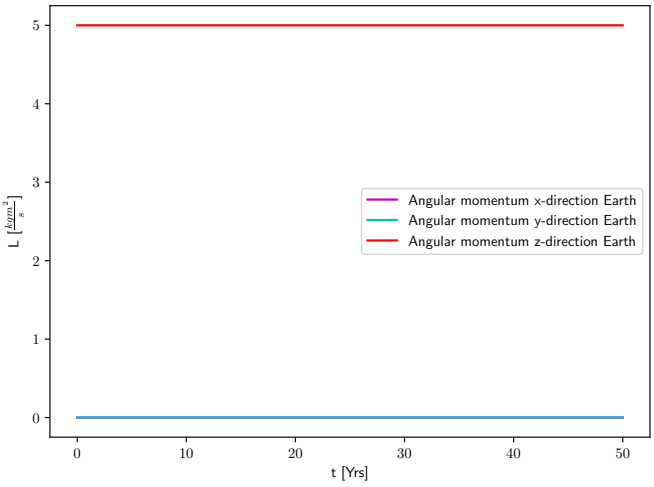

FIG. 14. Angular momentum in the x,y and z direction for the Earth around a static Sun simulated using the velocity Verlet algorithm over 50 years with a timestep of 0.0005 yrs with an initial velocity $v_{x0} = 0.0, v_{y0} = 2\pi, v_{z0} = 0.0$ resulting in a a elliptical orbit

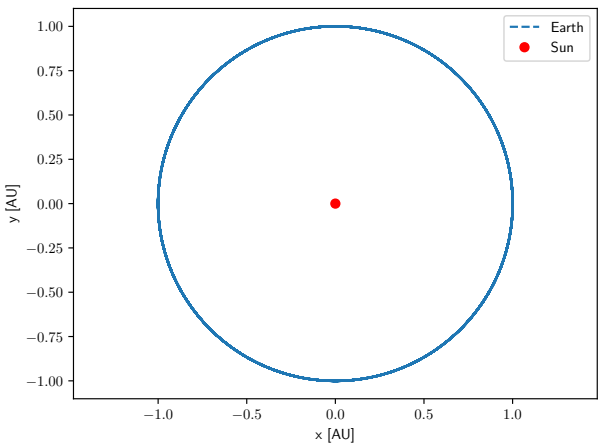

FIG. 15. Position of the Earth around a static Sun simulated using the velocity Verlet solver, with $\beta = 2.5$ over 50 years and a timestep equal to 0.005. The initial velocity is $v_{y0} = 2\pi$ to ensure a circular orbit

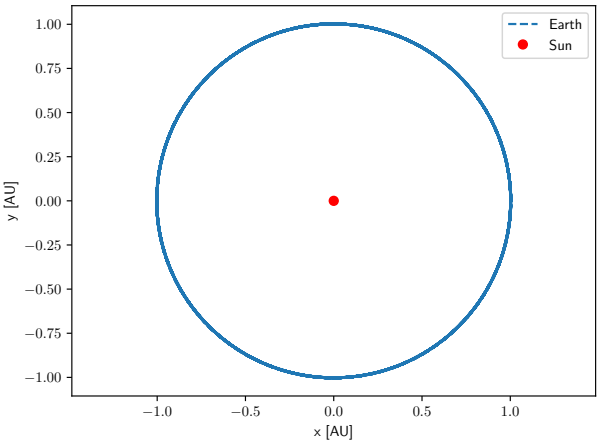

FIG. 16. Position of the Earth around a static Sun simulated using the velocity Verlet solver, with $\beta = 2.9$ over 50 years and a timestep equal to 0.005. The initial velocity is $v_{y0} = 2\pi$ to ensure a circular orbit

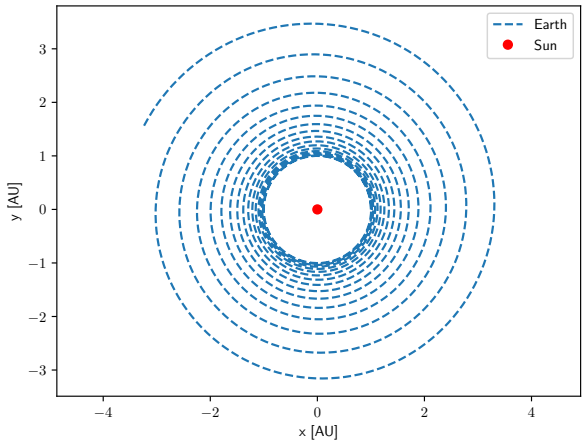

FIG. 17. Position of the Earth around a static Sun simulated using the velocity Verlet solver, with $\beta = 3.0$ over 50 years and a timestep equal to 0.005. The initial velocity is $v_{y0} = 2\pi$ to ensure a circular orbit

chosen are $x_0 = 1\text{AU}, y_0 = 0, z_0 = 0, v_{x0} = 0, v_{y0} = 2\pi, v_{z0} = 0$. This is the case for Figure 1 - 12. Figure 1 - 6 represent the just explained system simulated using Euler's forward algorithm. By inspecting Figure 1, where the system is advanced with a timestep of 0.05, Earth does not follow a circular orbit around the Sun. The orbit is more closely resembling that of an elliptical one, though after one revolution around the Sun, Earth achieves its escape velocity and leaves the Sun's orbit. This behavior is not reproduced in Figure 3 and 5. For the latter Figures, where the step size has been increased to 0.005 and 0.0005 respectively, the Earth's orbit is expanding per revolution though no escape is apparent. This ever increasing orbit can be understood by inspecting Figure 4 and 6, which both show that the total energy of the system is continuously increasing over time. This in contrast to the energy budget for the grainiest time step in Figure 2. For this case, there appearers to be a sudden spike in kinetic energy after the Earth has performed one revolution around the Sun, which in turn enables the total energy to suddenly increase as if it spiked. Though the total energy retains the maximum value of this "spike" for the rest of the simulation. Furthermore, as the value of the total energy is close to zero after the sudden shift

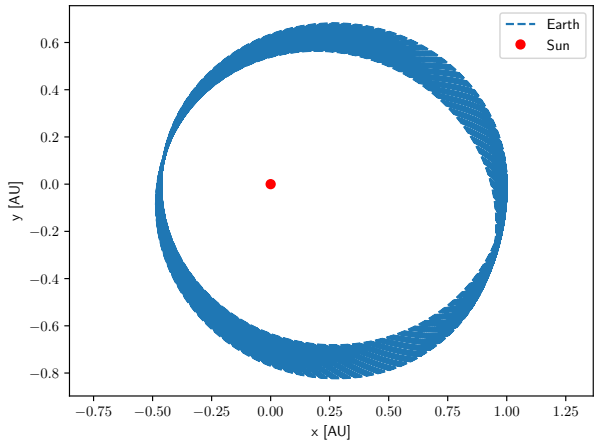

FIG. 18. Position of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

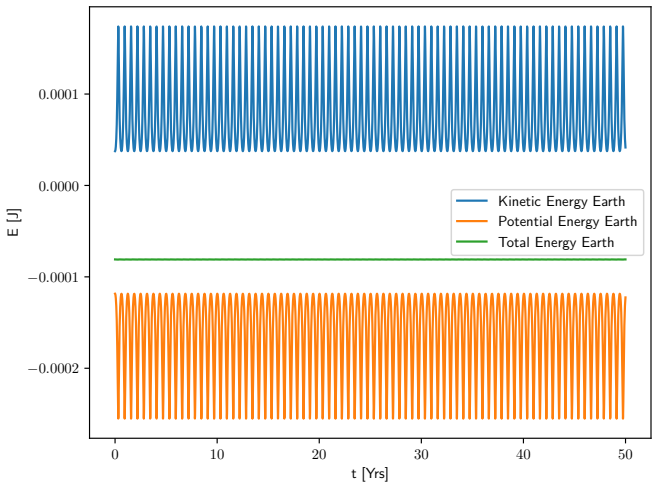

FIG. 19. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

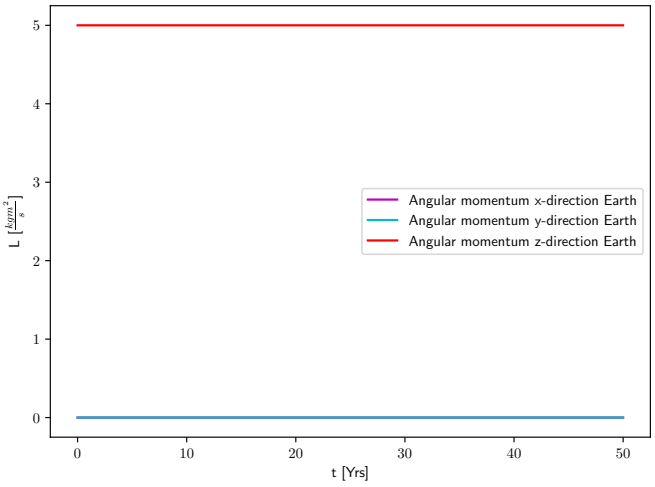

FIG. 20. Angular momentum in the x,y and z direction for the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

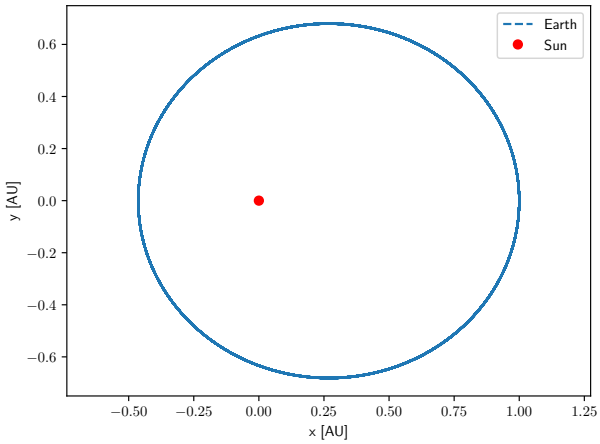

FIG. 21. Position of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. The initial velocity is $v_{y0} = 5.0$.

in value, this might indicate that this simulation actually triggered the Earth to escape the Sun's orbit.

Figure 7 to 12 have been produced by simulating using the velocity Verlet algorithm. The positional figures show that as the time step is decreased, the orbit of Earth oscillates less. That is, oscillation only appears in Figure 8, which is the case where the timestep is 0.05, though the apparent oscillation does not alter the total energy. This can be explained by inspecting the complementary energy-budget figure. As time moves, the potential and kinetic energy oscillates interrelatedly, such that the Earth's total energy at all times is constant. This behavior is expected for elliptical orbits, which the Earth closely achieves when the system is simulated using the velocity Verlet algorithm for longer timesteps. When the timestep is 0.005 or 0.0005 as in Figures 9 and 11 respectively, there are no changes in either kinetic or potential energy, resulting in perfectly circular orbits. Due to the only force acting upon the Earth in the simulation is the gravitational pull exerted by the Sun, which is a conservative force, it should cause the total mechanical energy of the system to be equal to 0.

By comparing the results from both algorithm, it was expected that the result of Euler's forward algorithm would not conserve energy. Inspecting Equation 1 tells us that Euler's forward algorithm is asymmetric in time, due to the position at the next time step is computed using the velocity from the previous time step. What this

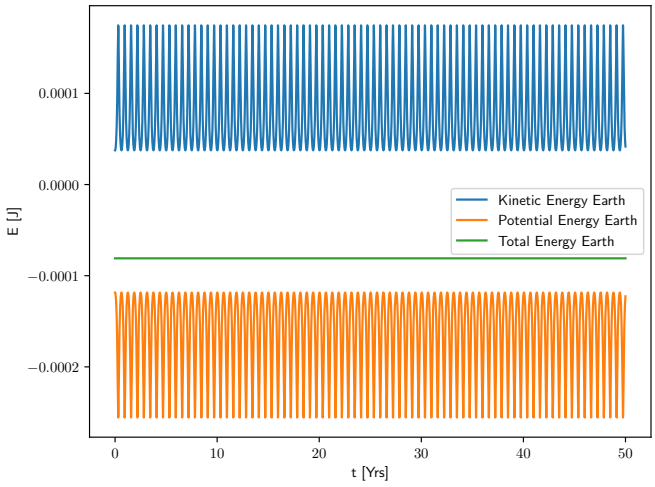

FIG. 22. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. The initial velocity is $v_{y0} = 5.0$.

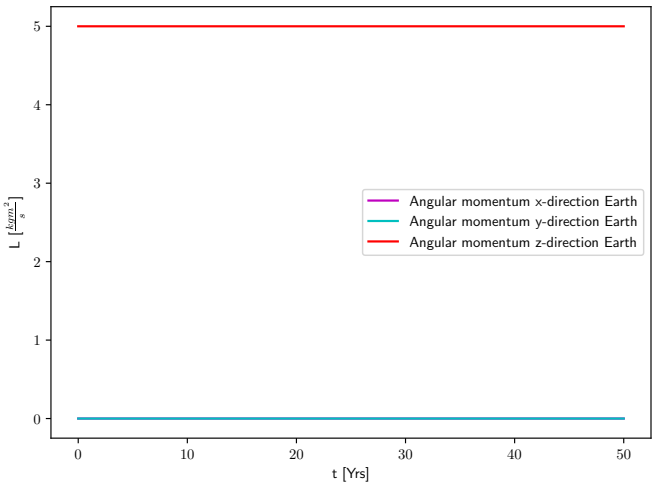

FIG. 23. Angular momentum in the x,y and z direction for the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. The initial velocity is $v_{y0} = 5.0$.

implies is that the algorithm does not conserve energy when simulated over prolonged time, potentially resulting that a planet can escape its orbit as exemplified in Figure 1. The velocity Verlet algorithm in Equation 2, is an example of a symplectic, which without diving into the mathematical meaning, leads to the algorithm conserving energy. I.e. the forward Euler scheme would have been symplectic if it used the velocity at the next time step to compute the position in the next time step (Euler-Cromer algorithm). Thus, the correct behavior of the solar system emerging from the velocity Verlet simulation can be traced back to the algorithm itself conserving energy due to it being symplectic.

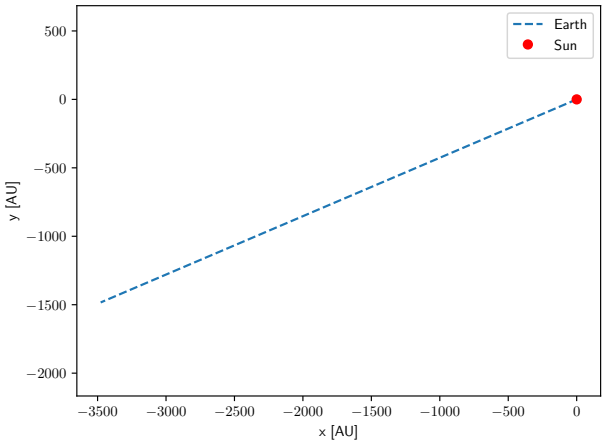

FIG. 24. Position of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years, with $\beta = 3.0$ and a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

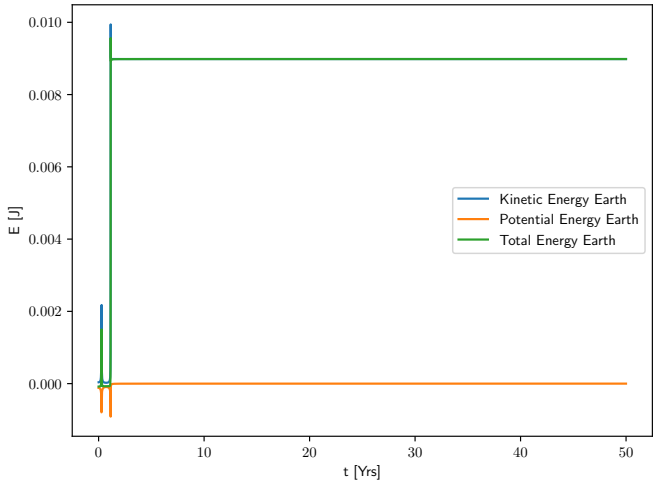

FIG. 25. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years, with $\beta = 3.0$ and a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

Table I compares the runtime of each algorithm. It has been computed that Euler's forward algorithm performs a total of $15 + 24(\text{Total planets} - 1)$ FLOPs for each Planet, whereas the velocity Verlet algorithm performs a total of $36 + 48(\text{Total planets} - 1)$FLOPs per Planet. As both algorithm have the same computational complexity, their efficiency is bounded by the number of FLOPs performed. As Table I shows, Euler's forward algorithm have a shorter runtime than the velocity Verlet algorithm, as expected by forward Euler performing less FLOPs per Planet. Even though the Euler's forward have a shorter runtime than the velocity Verlet, velocity Verlet is still preferred as it conserves energy. As is shown in the implementation of the velocity Verlet algorithm, the additional FLOPs

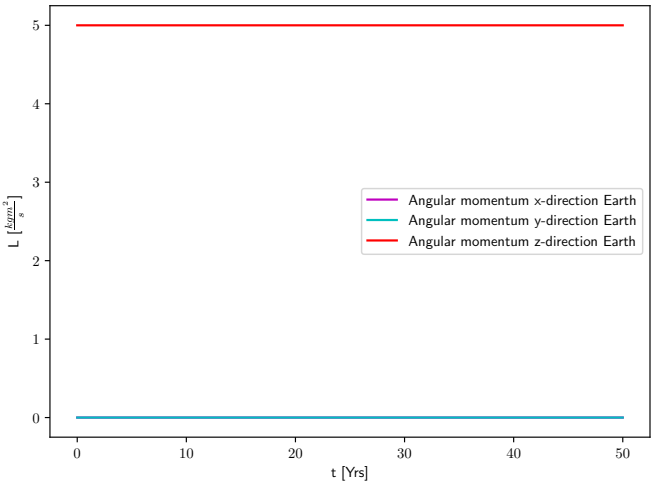

FIG. 26. Angular momentum in the x,y and z direction for the Earth around a static Sun simulated using the velocity Verlet solver over 50 years, with $\beta = 3.0$ and a timestep equal to 0.005. The initial velocity is $v_{y0} = 5.0$.

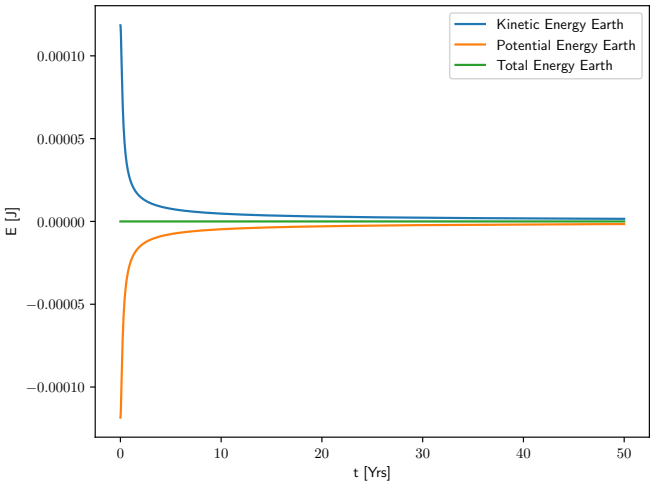

FIG. 28. Potential, kinetic and total energy of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. With an initial velocity $v_{y0} = v_{\text{esc}} = 8\pi^2$

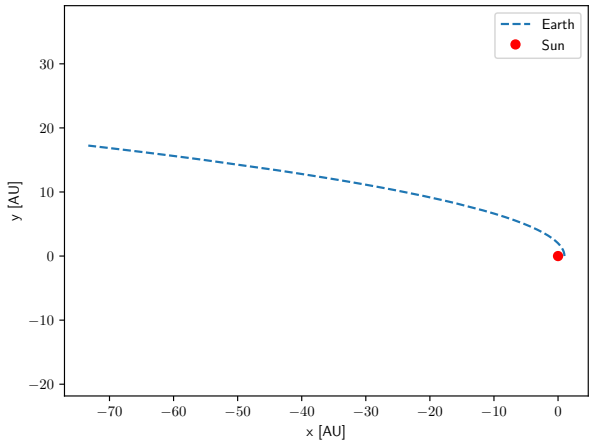

FIG. 27. Position of the Earth around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. With an initial velocity $v_{y0} = v_{\text{esc}} = 8\pi^2$

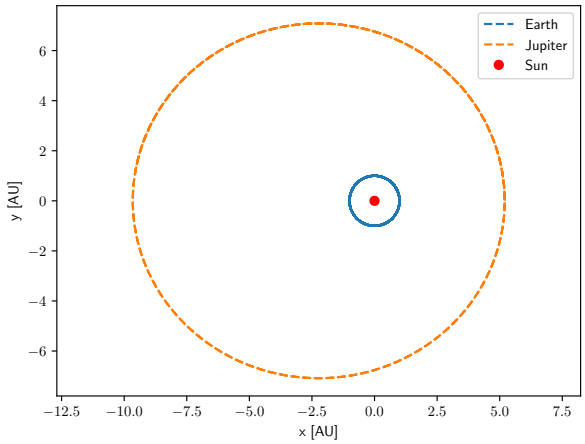

FIG. 29. Position of the Earth and Jupiter around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. Initial velocity for the Earth is $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

stems from the algorithm having to perform an additional *gravitational_force* call each time step. Which in turn results in the algorithm being symplectic. The trade off between marginally increased runtime but more correct simulations results in this project only considering the velocity Verlet algorithm from this point on.

Conservation of angular momentum

From Kepler's second law, we had that *A line segment connecting any given planet and the Sun sweeps out area at a constant rate.* Since the model is assuming that the celestial bodies have a constant mass, Kepler's second law is implying that the angular momentum has to be conserved as well. One approach to describe how this law explain the planetary movement, is by considering a case where the orbit is not circular, but elliptic. Due to total energy being conserved, potential and kinetic energy have to counter each other to stay in equilibrium. As kinetic energy is dependent on the planets velocity, and potential energy is dependent on the distance between the two celestial bodies, a decrease in distance would infer an increase in velocity. This results in the moving body having higher velocity when close to the static body, and a slower velocity when far away from the static body. Kepler's second law, which mathematically stated that

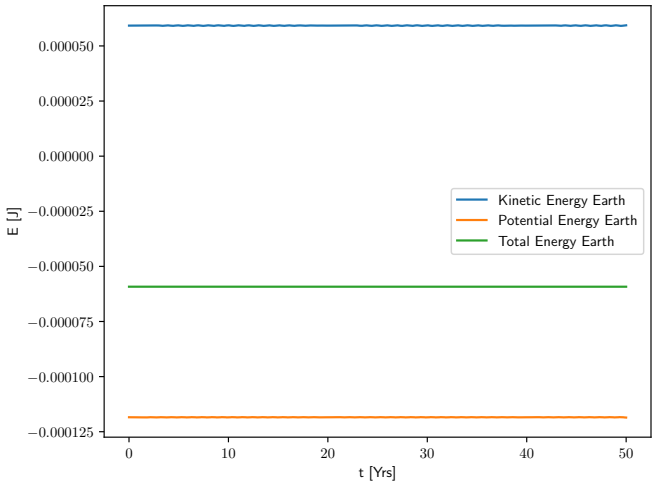

FIG. 30. Potential, kinetic and total energy of the Earth, with Jupiter added to the system, around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. With an initial velocity for the Earth $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

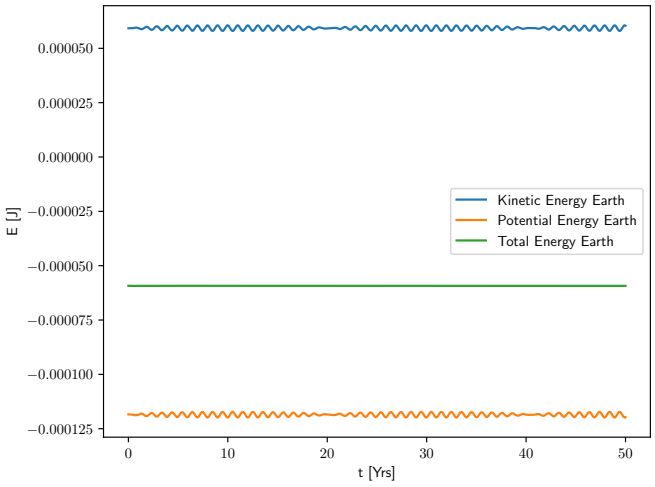

FIG. 32. Potential, kinetic and total energy of the Earth, and a Jupiter with ten times increased mass added to the system, around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. With an initial velocity for the Earth $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

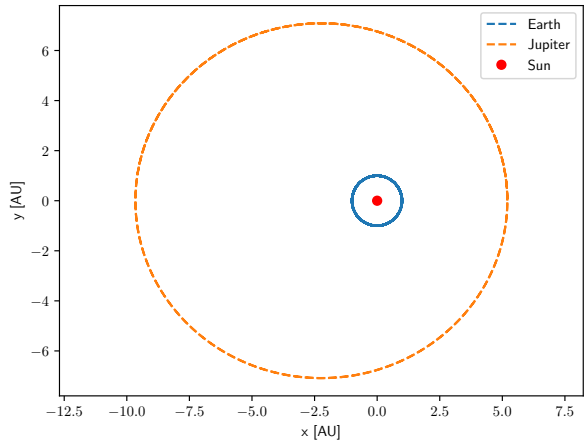

FIG. 31. Position of the Earth and a Jupiter with ten times increased mass around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. Initial velocity for the Earth is $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

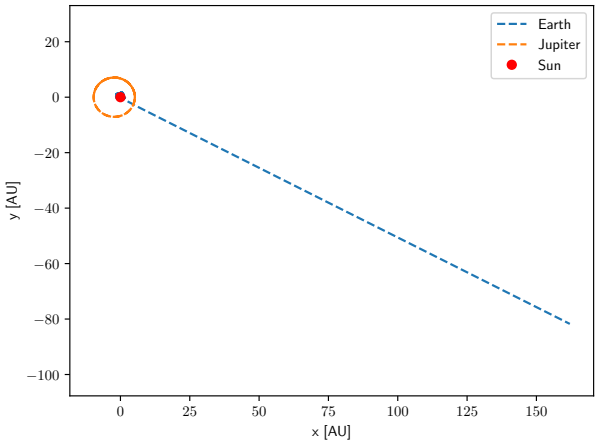

FIG. 33. Position of the Earth and a Jupiter with 1000 times increased mass around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. Initial velocity for the Earth is $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

$\frac{dA}{dt}$ = constant, would then hold true from this description of the energy relation. Due to the variable magnitude of the velocity as a result of a changed distance between the celestial bodies, the total area which is swept over stays constant. As a circular orbit is just as special case of an elliptical orbit, where the eccentricity of the ellipse is zero, this would also stay true since the distance hence velocity would stay unchanged.

This can be related to the conservation of angular momentum, which is defined as $\underline{L} = \underline{r} \times \underline{p}$. For angular momentum to stay constant, the relationship between the distance and the velocity has to be in such a way that they counter each other for all timesteps. From what was just explained by considering the relationship between potential and kinetic energy, and how this affects velocity as the distance in changed, the relationship between velocity and distance holds true for planetary motion to ensure the angular momentum is conserved. Thus, Kepler's second law gives a relationship between how velocity changes with distance, and ensures that angular momentum is conserved.

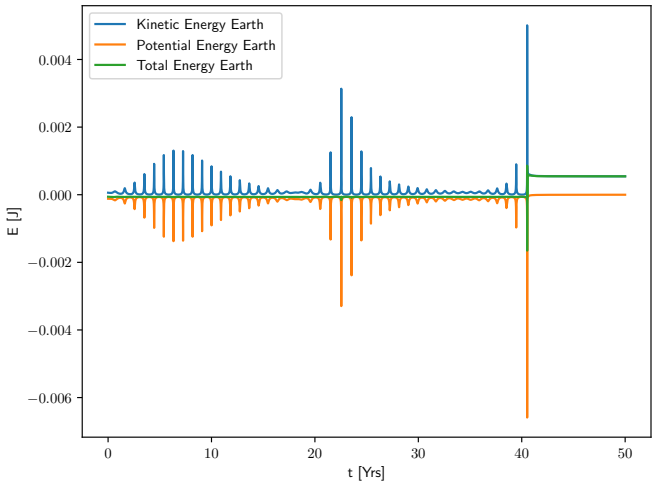

FIG. 34. Potential, kinetic and total energy of the Earth, and a Jupiter with 1000 times increased mass added to the system, around a static Sun simulated using the velocity Verlet solver over 50 years with a timestep equal to 0.0005. With an initial velocity for the Earth $v_{y0} = 2\pi$, and for Jupiter is $v_{y0} = \pi$

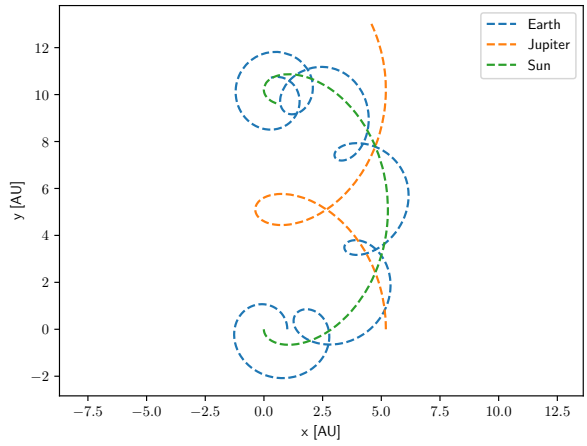

FIG. 35. Position of the Sun, Earth and Jupiter simulated using the velocity Verlet solver over 10 years with a timestep equal to 0.0001. Position for the Sun is chosen to ensure a COM system

Figure 13 and 14 shows the angular momentum plotted for respectively a circular and elliptical orbit. As both plots show, angular momentum is conserved for all three directions. This is as expected from Kepler's second law.

Considering an elliptical orbit for the Earth-Sun system

Figures 18 - 23 show the case of an elliptical orbit for the Earth around a static Sun, using the Verlet Solver. The initial values are as explained in the Results section,

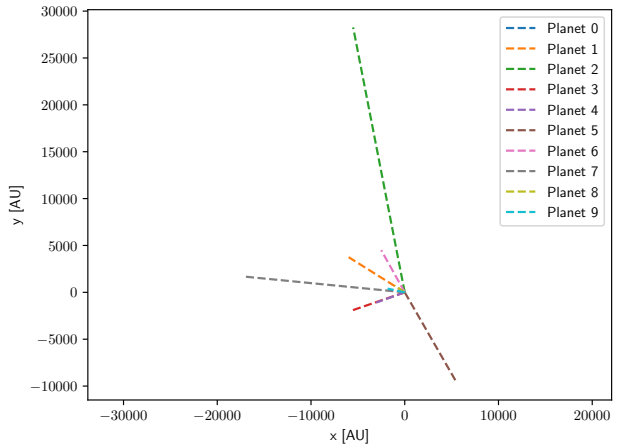

FIG. 36. Positions of all planets in the solar system, including the Sun and Pluto. Note that the plot is not labeled correctly, as it is not a correct simulation due to a bug in the velocity Verlet solver. The simulation is ran for 50 years, with a timestep of 0.005

only a slight reduction of the initial velocity v_{y0} to 5.0 results in the orbit changing shape to an ellipse. Figures 18 - 20 shows the simulation with a timestep of 0.005, which results in an orbit that is not completely elliptical, though when the timestep is increased to 0.0005 as in Figure 21 - 23, the orbit is completely elliptical. As expected, the kinetic and potential energy do vary, but the total energy is conserved throughout the simulation. Also for both simulations, the angular momentum is conserved.

Testing a modified gravitational force

Figures 15 - 17 shows how the circular orbit binary Earth-Sun system reacts when the inverse square force is replaced by

$$F_G = \frac{GM_\odot M_{\text{Earth}}}{r^\beta}$$

where $\beta \in [2, 3]$. As can be seen in the positional plots, the orbit is closed for all $\beta < 3.0$, though for the case of $\beta = 3.0$ the total energy is not conserved, instead it is increasing. Doing an analogous experiment for the elliptical orbit, though only testing for $\beta = 3.0$, results in an sudden spike in kinetic energy. This spike in kinetic energy makes it so that the Earth escapes the Sun's orbit with an extremely high velocity which makes this simulation seem suspicious. From the results made by simulating the solar system with a modified inverse square force, it can be concluded that setting $\beta = 3.0$ would destabilize the system. Moreover, when $\beta = 2.9$ for the circular orbit, the orbit is experiencing some oscillation, which is reduced further when $\beta = 2.5$.

Escaping the Sun

what keeps a planetary object in orbit around another planetary object is that the smaller of the two objects stays in the gravitational field of the larger of the two objects. Furthermore by looking at the energy budget of a celestial body, as long as the total energy of the moving body is less than 0, the body is in orbit. Thus the escape velocity can be found by assuming $E_{\text{tot}} = 0$. From this we get that, for a planet that begins at a distance 1 AU from the Sun, the initial velocity has to be $v_0 = \sqrt{2GM_\odot}$. For this model's case, this results in an escape velocity equal to $v_{\text{esc}} = \sqrt{8\pi^2}$.

Figure 27 and 28 show a simulation of the binary Earth-Sun system with the velocity Verlet solver over 50 years with a timestep equal to 0.0005, where the initial velocity $v_{y0} = \sqrt{8\pi^2}$. As can be seen in the energy budget for this simulation, the total energy is as expected 0 from the beginning, which results in the expected behavior of the Earth instantaneously escaping the Sun's orbit.

Adding Jupiter to the Earth-Sun system

Figure 29 and 30 show a three-body system where Jupiter has been introduced, with a still static Sun. As can be seen, the introduction of Jupiter barely affects the Earth's orbit around the sun. This can be seen by inspecting the energy budget in Figure 30, which shows slight oscillations in the Earth's kinetic and potential energy, though not enough to perturb the total energy. Thus the Earth's orbit around the Sun remains closed and circular.

By increasing the mass of Jupiter by a magnitude of 10 we get Figure 31 and 32. Figure 32 in comparison to 30 show a higher degree of oscillation in the Earth's kinetic and potential energy at some timesteps, which might relate to the two planets relative distance to each other. This since the oscillations appear to have a local maximum. Still the total energy of the Earth is not perturbed, thus the Earth still retain its circular orbit about the Sun.

When increasing Jupiter's mass by a magnitude of a 1000, that is, close to the mass of the Sun, we get Figure 33 and 34. As can be seen clearly from the positional plot, the total energy is not conserved as the Earth ends up leaving the Sun's orbit. This behavior can be seen by inspecting the energy plot, where it can be seen that the total energy is approximate to zero before spiking and ending up greater than zero. This system is thus not stable when solved using the Verlet solver.

Figure 35 and 36 show two simulations with the Verlet solver where all celestial bodies are in motion. The first Figure 35 considers a real three-body calculation, where all celestial objects are in motion. Though it was intended for the system to have the center of mass at its origin, as can be seen there seems to be some momentum in the system regardless which is making the system drift in the positive y-direction. The same can not be said for Figure 36. Even though it was supposed to be a simulation for the entire solar system, the resulting simulation shows that almost all celestial bodies escapes the Sun's orbit. Though the system is initialized with values obtained from [3], as well as adjusting the initial values with the systems center of mass position and velocity, the plot still show that when adding more planetary bodies to the system, the simulation breaks down. Some thoughts on why this is will be given in the following section

Considering the Verlet solver for multi body systems

As the previous section have alluded to, there seems to be a model breaking bug in the Verlet solver which appears when more celestial bodies are included in the simulation. Even though the solver have been implemented as Object-Oriented code which is general and supports the addition of a theoretically infinite number of planets, there are errors in the calculations which only appears when there are more than one moving body. There are a couple of reasons why the bug only appears when considering multi-body systems, though to be honest I have tried to debug all possible errors I could come up with.

Why the solver should work, and has been seen throughout this Project, works for a two-body system with a static Sun, is that there is only one moving body where parameters are changed for each iteration. As such, when only updating the moving Earth to a static Sun, the fact that the Sun is not moving results in all computations involving distance being constant no matter where the Earth is (when considering circular orbits). Since the Earth never has to take into consideration that the Sun has moved when recomputing distance, the results will always be correct. In a similar manner, as the Sun-object is never computed directly in the Verlet-loop, the positional computations for the Sun never have to take into consideration that the Earth has just recomputed it's position. Which as has been naively implemented in this solver, is only one variable which is rewritten instead of e.g. a positional array containing all computed positions. This was originally motivated by saving memory, as computing positions in three directions for all celestial bodies in the solar system for a large simulation would have not been possible memory wise.

Though there is currently in the implementation file only one position variable, one way in which I tried to debug and rewrite the code, was to introduce two positional variables. One for the current position, and one for the position at the next step. With this, I was able to rewrite the entire *advance_vv* method by dividing it into two parts. The first part calculated the position at the next timestep for all planets, then a second loop throughout all planets computed the velocity, using the fact that all planets have already gotten their next position calculated.

This was then taken into account in the *gravtational_force* method, which then could specifically use the fact that all planets have been moved in space to get their acceleration with respect to all planets new position. As it stands now, if one planet is moved to a new position, that position is used for all other distance calculations for all subsequent planets. The above potential solution which was just described was implemented to fix this problem, which is obviously incorrect, but it did not fix the bug. The all planets simulation simply just scaled differently, but it retained its erroneous form.

As such, what causes the solver to break when adding multiple moving planets is not something I was able to find out and fix. As it seems, the solver is not able to handle to many moving objects, which leads me to believe that errors might arise when computations involving distances are involved. Though what was then implemented to fix which position parameter was used, which solved the problem where two position of different timestep for two objects where used in the same computation, did not solve the problem. Due to this solution not working, my final theory for what might cause this bug is rounding errors, again probably arising from the distance calculation, but this is just a guess. The reason why I would suggest distance calculations as the source of errors, is that the solver worked throughout the entirety of project, until several objects where added, which in turn added more distances. Its regrettable in the end that the Verlet solver would not expand further than for two planets about a static Sun. Thankfully the algorithm is already written relatively general, and if what causes the bug is spotted, ironing it out and expanding the Verlet solver to simulate the entire solar system is left open to anyone who want to pick up this thread.

Final thoughts

Throughout this Project, the goal has been to develop a general solver which would be able to simulate the entire solar system. Though that was regrettably not reached, a solver which simulates a binary or even three-body system with a static Sun has been developed. All in three dimensions, though all examples which has been shown have only included the x,y-plane.

An important discovery made in this project is the importance of selecting the right algorithm for solving systems of coupled first order differential equations. Our first simulations being a solver which implemented the forward Euler algorithm, which tended to not conserve energy, resulting in corrupted simulations where the Earth was constantly gaining energy. By choosing an algorithm that was symplectic, such as the velocity Verlet algorithm, the results followed how we expected the Earth to behave when orbiting a static Sun. Both angular momentum and total energy was conserved, which without the conservation of these quantities would cause the Earth to eventually leave the Earth's orbit. Moreover, by utilizing the Verlet solver, we were able to simulate elliptical orbits, how the system reacted when changing the form of the gravitational force and simulate the Earth escaping the Sun's orbit when initialized with a velocity $v_{\text{esc}} = \sqrt{8\pi^2}$. The final being the minimum amount of velocity required to set the total energy equal to zero.

Though the Verlet algorithm required $36+76(\text{Total planets}-1)$FLOPs compared to the forward Euler algorithm's $15+38(\text{Total planets}-1)$FLOPs. The timing difference which was shown in Table I only differed in millisecond, with the Verlet solver obviously being the slowest. Even though the Verlet solver where doing additional computations, both algorithms scaled similarly to decreased timestep. When also considering that the Verlet solver gave answers that conserved physical quantities compared to the forward Euler solver, it was an apparent choice when further developing and testing the algorithm.

With the Verlet solver, Jupiter was also able to be included, and it was seen that by including Jupiter it would have an affect on the system. Jupiter in itself might have only been able to slightly perturb the kinetic and potential energy of the Earth, though when expanding Jupiter's mass by a factor of 10, the slight perturbations became more prominent. Though neither simulations where able to destabilize the solver, as the Earth's orbit remained closed about the Sun. The closed orbit was able to be broken when expanding Jupiter's mass to that of the Sun itself, which eventually caused the energy of the Earth to spike and enable it to escape the Sun's orbit. However, when setting the Sun in motion for this three body problem, some peculiarities arose as the entire system was drifting in the positive y-direction.

As has already been noted, the solver was not able to simulate the entire solar system due to a model breaking bug which I was not able to determine. Though some theories on what might cause it have been discussed, and some unsuccessful attempts of patching have been explained, the final model did not extend to simulate the case in which it set out to do. This result is quite regrettable indeed, especially considering that it seems to be something which would be easy to iron out, but this is merely speculation.

In summary, this project has proven why differential equations are as powerful a modeling tool as they are claimed to be, when implemented and solved correctly. With simple mathematical models, complex systems such as the entire solar system can be simulated, even without straining the memory or the CPU of an old laptop used for the purpose of studying. An important takeaway from this project is to always account for bugs, as their impact on a programs execution can have devastating effects. Even though the CPU is good at crunching numbers, its results have to make sense in the context of which they exist if they aspire to not be discarded as erroneous data. The latter was sadly the case this time.

Source code

The code developed for this project can be accessed at `https://github.com/AreFrode/FYS3150/tree/master/assignment3`

The following code has been written for this project

- main.cpp
- planet.cpp
- planet.hpp
- solarsystem.cpp
- solarsystem.hpp
- test.cpp
- main.py
- plotter.py
- plot_energy.py
- plot_momentum.py
- twobody_plotter.py
- threebody_plotter.py
- manybody_plotter.py

[1] I. de Pater and J.J. Lissauer. *Planetary Sciences*. Cambridge University Press, 2015. ISBN 9781316195697.

[2] M. Hjort-Jensen. Computational physics, August 2015 (accessed September 24, 2020). URL `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.

[3] NASA. Horizons web-interface. `https://ssd.jpl.nasa.gov/horizons.cgi#top`. (accessed October 25, 2020).