Simulating the Solar System using ordinary differential equations

Noah Oldfield and Johan Nereng

Department of Physics, University of Oslo, Norway

Sep 8, 2018

1 Introduction

ØK MARGINEN!

Legg till litt mer om motivasjon for prosjektet

The overall aim of this project is to develop a program that simulates the Solar System by numerically solving equations that describe the trajectory of the planets and the Sun. The motion of each of these objects is subject to the sum of forces acting upon that object, and it's trajectory may be described through a set of coupled ordinary differential equations (ODEs).

.

First, choice of numerical procedure (for solving the ODEs) is decided upon, empathizing run time and stability. In this project, both the velocity Verlet and Forward Euler method (here) were evaluated - upon which the velocity verlet was decided upon as the most suitable method due to **kort beskrivelse av hvorfor Verlet ble valgt**

After describing the procedures, algorithms using both methods were developed (here).

Having developed algorithms, both algorithms were tested in a prototype program using a simplified version of the Solar System, described in detail here.

All sims run over one year, except last stuff

I Intro: all data er henta fra NASA, med dato: posisjoner 20.oktober 2017 (for å kunne sammenlignde med årets pos)

1. Finn metode for simulering (Sun/Earth) 2. Implementer generell M-body algoritme (solsystemet) 3. Simuler hele solsystemet 4. Hva med relativitet?

assumes general understanding of forces, and the sum of forces on an object.

first prog: 2d, OO3d, plotting in 2d (xy axis) important test of the general theory of relativity was to compare its prediction for the perihelion precession of Mercury to the observed value.

2 Methods

2.1 Numerical procedures for solving ODEs

The procedures [1] described in this section are general numerical solvers for ordinary differential equations. This means that the procedures assumes that the problem at hand is properly discretized. Throughout this project, all discretization is carried out using the same template. The specific discretization paramters for each particular system is described in more detail later.

Discretization template

Time, t - running from an initial value t_0 to $t = T$, is discretized over n points with a step length $h = 1/(n-1)$, so that $t_i = t_0 + ih$, where $i = 0, 1, ..., n-1$. x_i, v_i denotes position and velocity (in the x direction) at time t_i, and $\frac{d^2 x_i}{dt_i^2} = f(x_i, t_i)$ denotes the discretized second derivative of position - or the acceleration. Same procedure for y and z direction.

2.1.1 Euler's Forward method

$$\begin{aligned} x_{i+1} &= x_i + hv_i \\ v_{i+1} &= v_i + hf(x_i, t_i) \end{aligned}$$

For the Euler's Forward method method, the local truncation error (LTE) is $O(h) = h^2$.

2.1.2 Velocity Verlet method

When using the Velocity Verlet method, x_{i+1} must be determined before v_{i+1} as $f(x_{i+1}, t_{i+1})$ depends on x_{i+1}.

$$\begin{aligned} x_{i+1} &= x_i + hv_i + \frac{h^2}{2} f(x_i, t_i) \\ v_{i+1} &= v_i + \frac{h}{2}\big(f(x_{i+1}, t_{i+1}) + f(x_i, t_i)\big) \end{aligned}$$

For the Velocity Verlet method method, the local truncation error (LTE) is $O(h) = h^3$.

2.2 Developing algorithms

Before applying the numerical procedures to a particular system, an algorithm for each method is first developed. The algorithms use non-specific parameters, which ensures flexibility in implementation (choice of system). The algorithms does however assume that the acceleration of an object at time t depends on the mass of the object and the state of the system at time t .
In order to evaluate the algorithm implementations, a specific system must be decided upon, which also fixes the number of floating point operations.

On facilitating implementation to code

In order to later develop effective and transparent code, the use vectorization is assumed. This means that $\vec{r} = (x, y, z)$, $\vec{v} = (v, u, w)$ contains the position and velocity components in the x, y, z direction. Similarly, derivatives and forces are assumed to be three dimensional.

The algorithms

The following algorithms approximates the position $r_j = (x_j(t), v_j(t), z_j(t))$ and velocity $v_j = (v_j(t), u_j(t), w_j(t))$ in three dimensions for $1, 2, .., j, ..., M$ objects with mass m_j at time $t_1, t_2, .., t_{n-1}$, provided that the the initial values are known. In addition, the algorithm assumes that the acceleration at a given time t of object j depends on the current state of the system, and the mass of the object itself - that is: $a_j = f(t, m_j)$. The algorithms overwrites r_j and v_j at every time step.

The Euler's forward algorithm below has floating point operations:
# flops$\sim$ $n \cdot (1 + j \cdot (4 + (\text{\#flops in } a_j))$

Algorithm for solving 3 dimensional M body system using Euler's forward method

```
1. Calculate $h = 1/(n-1)$
2. For $j = 1, 2, .., M$:
       Initialize vectors: $r_j$, $v_j$ and $a_j$ with dimension 3.
       Set $m_j$
       Set $r_j$ and $v_j$ to initial values.
3. Set $t = 0$
4. For $i = 1, 2, ..., (n-1)$:
       For $j = 1, 2, .., M$:
           Update: $a_j = f(r_j, m_j)$
       For $j = 1, 2, .., M$:
           Update:
           $r_j = r_j + hv_j$
           $v_j = v_j + ha_j$
       Update $t = t + h$
```

As mentioned under Numerical procedures for solving ODEs, the velocity Verlet method (algorithm below) makes use of the acceleration at t_{i+1}, $\tilde{a}_j$. Since $\tilde{a}_j = f(r_j(t_{i+1}), m_j)$, the position of all objects is updated before before $\tilde{a}_j$ is calculated. This is unproblematic as r_j is not part of the expression used to update v_j. In order to reduce the number of floating point operations (FLOPS), a_j is only calculated once (acceleration at t_0), and at the end of each time step, set equal to $\tilde{a}_j$. The implementation then yields that steps $1, 2, 3$, and 4 are done at $t = t_0$, while step 5 iterates over time.

The velocity Verlet algorithm below has floating point operations: # flops$\sim n \cdot (1 + j \cdot (10 + 2(\#\text{flops in } a_j))$, Which may be reduced to # flops$\sim n \cdot (1 + j \cdot (7 + 2(\#\text{flops in } a_j))$ by pre calculating $\frac{h^2}{2}$, and $\frac{h}{2}$.

Algorithm for solving 3 dimensional M body system using velocity Verlet method

```
1. Calculate $h = 1/(n-1)$
2. For $j = 1, 2, .., M$:
       Initialize vectors: $r_j$, $v_j$, $a_j$, $\tilde{a_j}$ with dimension 3.
       Set $m_j$
       Set $r_j$ and $v_j$ to initial values.
3. Set $t = 0$
4. For $j = 1, 2, .., M$:
         Update: $a_j = f(r_j, m_j)$ (initial acceleration)
5. For $i = 1, 2, ..., (n-1)$: (iterations over time start here)
       For $j = 1, 2, .., M$:
         Update $r_j = r_j + hv_j + \frac{h^2}{2}a_j$
       For $j = 1, 2, .., M$:
         Update: $\tilde{a}_j = f(r_j, m_j)$
       For $j = 1, 2, .., M$:
         Update $v_j = v_j + \frac{h}{2}(\tilde{a}_j + a_j))$
         Update $a_j = \tilde{a}_j$
       Update $t = t + h$
```

2.3 Sun Earth system

The first system used for algorithm implementation is the simplified model of the Solar System using only the Earth and the Sun, which constitutes a two body system. Newton's law of gravity (1) applied to the Earth with respect to the Sun is assumed to govern the motion of the Earth, in other words - the effects of relativity is neglected.

$$\vec{F} = \gamma \frac{M_1 M_2}{r^3} \vec{r} \tag{1}$$

Where γ is a constant (usuallly G), M_1, M_2 the masses of two objects and r the distance between their masses. In this project, all objects are assumed to be point particles

Initially, the Sun is defined as the origin, which means that r is the distance to the Sun. Later, the Solar System's barycenter will be used as the origin. More on this under The multi body system. Assuming vector notation; $\vec{r} = (x, y, z)$ and $\vec{v} = (v, u, w)$, Newtons law of motion: $\vec{F} = m\vec{a}$ yields:

$$\frac{d^2\vec{r}}{dt^2} = \frac{\vec{F}}{M_E} \tag{2}$$

Using the Earth's mass, M_E as M_1, and the mass of the sun $M_\odot$ as M_2, and combining (1) and (2) then leads to a system of coupled differential equations:

$$\frac{d\vec{v}}{dt} = -\frac{GM_\odot}{r^3}\vec{r} \tag{3}$$

$$\frac{d\vec{r}}{dt} = v_x \tag{4}$$

$$\tag{5}$$

Which may be discretized and solved using the methods previously discussed under Numerical procedures for solving ODEs. However, prior to this, the Sun-Earth system is first scaled.

2.3.1 Scaling

The following scaling will also be applied to the other systems used in this project, therefore it's written assuming that the Earth is only one of several bodies in the Solar System.

Using the assumption that the trajectory of an astronomical object is (approximately) circular and in one plane, the Centripetal force may be used to scale the system. Applying one standard astronomical unit, 1 AU, as the length unit, and 1 $yr = 1$ *year* as the time unit, the velocity of the object may then be expressed as $v = 2\pi AU/yr$. So:

$$GM_\odot = v^2 r = 4\pi^2 \frac{AU^3}{yr^2} \tag{6}$$

Furthermore, the Solar mass $M_\odot = 1$ is set as the unit mass, which means that the gravitational constant is scaled as $G = 4\pi^2$. This also means that the mass of an object is evaluated as a ratio of the Sun's mass. M_E for example is defined as $M_E = \frac{\text{The Earth's mass in kgs}}{\text{The Sun's mass in kgs}} = 3.0024584 \cdot 10^{-6} M_\odot = 3.0024584 \cdot 10^{-6}$.

2.4 Algorithm implementation

Having scaled and applied the discretization template discussed under Numerical procedures for solving ODEs to the Sun-Earth system, the algorithms for both the Euler's Forward method and the velocity Verlet method is now implemented into code using the Sun-Earth, with the goal of developing a functioning program. This is done using $C++$ with the Vec3 class and Solar System Shell provided through the Github page of the course Computational Physics at the University of Oslo [3]. The program written for the purpose of this project may be found on the authors common Github repository.

In order to ensure correct algorithm implementation, the initial code is written without the use of object orientation. Pending verification of successful implementation and evaluation of the numerical methods, the code will be rewritten with object orientation. More on this under Object orienting the code.

The code is compiled into a program using the $C++$ optimization flag -03 to ensure limitation of syntax discrepancies.

2.5 Evaluation of initial implementasjon

In order to evaluate the implementation described above, the program is first tested with $n = 1000$ steps, and the results plotted for each of the two numerical methods. This is expected to produce a circular Earth orbit, bringing the Earth back to it's starting position, with some deviations due to discretization and the LTE. The velocity Verlet method is expected to produce more accurate results on account of having the smallest LTE among the two methods.

A more thorough testing of the algorithm implementation is then carried out, focusing on two parameters; stability and runtime. Stability is measured through absolute error in relation to a completely circular orbit, and run time is measured by using the Chrono library from C++ - where the Euler implementation is expected to run faster than the velocity Verlet implementation, since it requires fewer FLOPS.

As the overall aim of the project is to simulate the Solar System, the code will eventually be further modified to encompass multi body systems. As multi body systems tend not to follow circular orbits, this means that comparing simulation results with a known analytical solution will eventually be infeasible. Therefore, while the absolute error is still a good measure of the stability - that is, while the program is limited to a two body system with a known solution - two alternative parameters for program stability are introduced; conservation of energy, E, and conservation of angular momentum, ω. By cross checking the absolute error, a well established parameter for program stability, with conservation of E and $\vec{L}$, this projects seeks to establish the validity of these alternative parameters for program stability when simulating multi body systems. More on conservation of these quantities below.

In order to measure the parameters above, the following measurements are introduced and applied to each solution produced for each test value of n; ϵ_r - absolute error in relation to the analytical solution - a circle with radius $1AU$, ϵ_E - absolute deviation from sum of kinetic and potential energy at t_0 (E_0), and

ϵ_L - absolute deviation from (total) angular momentum at t_0 (L_0).

$$\epsilon_r = max\Big(\frac{|r(t) - r_{analytical}|}{r_{analytical}}\Big) \tag{7}$$

$$\epsilon_E = max\Big(\frac{|E(t) - E_0|}{E_0}\Big) \tag{8}$$

$$\epsilon_L = max\Big(\frac{|L(t) - L_0|}{L_0}\Big) \tag{9}$$

On conservation of Angular momentum and conservation of potential and kinetic energy

In general, when considerer the gravitational forces acting upon two object, each of those opposite forces acting upon one of the objects, acts in the direction of the other object. This means that the torque τ for object 1 and 2 may be expressed as:

$$\vec{\tau}_1 = \vec{r} \times \vec{F}_1 = 0$$
$$\vec{\tau}_2 = \vec{r} \times \vec{F}_2 = 0$$

This, combined with the relation $\frac{d\vec{L}}{dt} = \sum_j \vec{\tau}$, implies that $\vec{L} = constant$, or that angular momentum is conserved when no external torque is acting upon the system. For a multi body system, the net force acting upon each object is the sum of forces acting upon that object. These (gravitational) forces each act in the direction of another object in the system - which again means that angular momentum is also conserved for multi body systems under the same requirements as for a two body system.

The law of conservation of energy states that the total energy of an isolated system is preserved over time. Both the two body and the multi body systems are isolated systems, and as this project does not include any energy transformations except between kinetic and potential (gravitational) energy, the sum of these quantities is constant.

2.6 On future use of the Velocity Verlet method

Under discussion of results, Initial implementation, the velocity Verlet method is established as better suited for the purpose of this project than the Euler's Forward method. In will be used in all further simulations. In addition, a number of steps $n = 10^5$ appears to be an ideal trade off **eller hva? konkluder!** stability

2.7 Escape velocity and effects of the gravitational force

Using $n = 10^5$, the velocity Verlet is now further tested. This is done by evaluating the escape velocity of a planet in a two body system with the Sun.

The escape velocity, v_{esc}, is defined as the minimum velocity required to to escape the gravitational field of star system. This requires the planet's gravitational potential and kinetic energy to equal:

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

The planet's starting position is set to 1 AU, and it's initial velocity set to the analytical expression for v_{esc} above. If the potential energy at any point is great that the kinetic energy however, the planet will not escape the Sun's gravitational field. As the program uses several simplifications and is a numerical solver, the planet is not expected to escape. Therefore, by trial and error, different initial velocities are tested in order to find a more representative v_{esc} for the simulated system.

In addition, in order to evaluate the effects of the gravitational force, the force is modified (11) and the code rerun, using the Sun-Earth system.

$$F_G = \frac{GM_{cdot}M_E}{r^\beta} \tag{11}$$

Where $\beta \in [2, 3]$ is tested, empathizing the effects on the Earth-Sun system when $lim_\beta \to 3$.

2.8 Object orienting the code

Using object oriented code with C++, the algorithm implementation of the velocity Verlet on the two body system is modified in order to obtain modularity, re-usability and polymorphism.

As the specific C++ syntax applied may be found under on the authors common Github repository, this section only includes a brief description of the overall train of thought in the object orientation process.

First, a (solar) system class is defined, to which planets may be added, as well as a trajectory governing force (gravity). Other objects than planets may also be added, as the system only serves to store the objects that are a part of the system, not their types. In addition, other governing forces than gravity may be defined. This is utilized later, when the effects of relativity on the simulation are explored. The system class also includes a method to reset the forces at each time step

As this project only makes use objects of the same type (planets), a celestial bodies class is defined in order to facilitate the simulation of an $M > 2$ body system. By defining the celestial bodies to store $\vec{r}$ and $\vec{v}$ at the current time, m, and the name of each celestial body, any number of objects may be added to the system. The force (class) is set up in a way that enables it to calculate all forces between objects at their current position - $\tilde{a}$ in the velocity Verlet algorithm.

Lastly, a solver class is defined - which upon construction is supplied with a specific system. The solver serves to advance that system from t_0 to T, by iterating over n time steps. This is done by calling a numerical procedure specific function, taking n as an argument. In this project, the velocty Verlet method is implemented, but other methods may be defined for the solver. The velocity Verlet algorithm is (almost) directly implemented, as it was devloped in order to encompass multi body systems.

In order to verify the integrity of the algorithm, a simple simulation of the Sun-Earth system is carried out by using $n = 1000$. This is expected to produce a plot similar to the one produced under initial testing (of the velocity Verlet algorithm).

2.9 NASA data and system barycenter

As the focus of this project now shifts from the simplified Solar System model using only the Earth and the Sun, to a model containing three, then all planets, initial positions and velocities of the objects are expected to approximate actual values. To accomlish this, ephemerides for the relevant Solar System bodies are extracted from the limited interface to JPL's HORIZONS system provided by NASA [4], in AU. These ephemerides use the Solar System's barycenter as the origin, which means that the Sun is no longer fixed at the origin - this is described further in the discussion of results section under the three body system.

Masses and velocities are also extracted from the HORIZONS system interface and scaled to fit unit's of measurements used in this project.

2.10 Three body system

The final step before simulating the Solar System with all planets, is to test the object oriented code using a three body system. This is carried out by adding the Sun, the Earth, and Jupiter to the system class, using HORIZONS system data as described above. The system is solved using the velocity Verlet function from the Solver class on a selection of n **WHAT VALUES?**, and the stability of the solution is evaluated by conservation of angular momentum, which is established as a suitable stability measurement in the discussion of results section under Inital implementation.

After initial stability testing of the Sun-Earth-Jupiter system using the object oriented code, the mass of Jupiter, M_J is increased. First by a factor of 10^1, then by 10^3. This is expected to have an effect on the trajectory of all three bodies; on the Sun - a pronounced trajectory, on Jupiter - a pronounced trajectory, and on the Earth - **SOMETHING??**. Especially in the $M_J = 10^3 M_J$ case. First, stability tests are executed for each case, then plots are produced, visualizing the effect on the system.

2.11 Simulating the Solar System

After having established the integrity of the applied simulation methods for multi body systems under discussion of results (the three body system), the Solar System is now simulated using all 8 planets and the Sun. All 9 objects are initialized using HORIZONS system data. Then, using $n =$ **HVA DA?** - established as suitably stable for the multi body system implementation under discussion of results, the program is executed and the results plotted. **NOE MER?**

check for stability?

2.12 Effects of relativity

As previously mentioned, the Solar System simulator developed in this project does not include the effects of relativity. In order to address this discrepancy, a simplified model of the Solar System is implemented in the object oriented program, this time using the Sun and the planet Mercury, and no other planets. Historically, predicting the perihelion precession of Mercury, in comparison with observations, was used to test the general theory of relativity - which is why this planet is chosen for this specific test in this project.

A general relativistic correction to the Newtonian gravitational force is now introduced [2]: (12). Where, $M_{mercury}$ is the mass of Mercury, r the distance between Mercury and the Sin, $l = |\vec{r} \times \vec{v}|$ the magnitude of the orbital angular momentum per unit mass of Mercury, and c the speed of light in vacuum.

$$F_G = \frac{GM_\odot M_{mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right] \tag{12}$$

The modified force is introduced into a modified version of the already well tested initial two body system program. Mercury is set to start at perihelion on the x axis with $r = 0.3075$ AU, and u_0 set to 12.44 AU/yr [2]. Then, the system is solved for $T = 100\ yr$, using $N = 10^6$. **10 i 6?**

The perihelion angle, θ_p, is calculated using $\theta_p = \frac{y_p}{x_p}$,where x_p, y_p are the perihelion values of x and y. This calculated value is then compared to the observed value. newline

Subtracting all classical effects (such as the perturbation of the orbit due to gravitational attraction from the other planets), the observed value of the perihelion precession is $43''$ per century [2]. The program's ability to reproduce this will determine

må forklares litt mer her precession of Mercury to the observed value. the noe of mercus is now examined. 4. Kjør Sol og merkur - Aphelion/Perihelion beskrive (og vinkel perihelion) - Forklar generell relativitet ledd i kraft

3 Results

3.1 Testing of initial implementation

Figure 1 shows a plot of the output from the initial algorithm implementation. The Sun-Earth system, with the Sun fixed at the origin, is solved over $T = 1yr$, using $n = 1000$ steps, and both the Euler's forward method and the The velocity Verlet method are utilized. It is clear that both methods have produced near circular orbits. However, it is also evident that the Euler's Forward method has produced an increasingly erroneous solution, while the velocity Verlet method appears more stable.

In order to quantify the differences observed in figure 1, further testing of the implementation is carried out. Table 1 shows a comparison in ms of run time, ϵ_r, ϵ_E, and ϵ_L, between the Euler's forward method and the The velocity Verlet method implementations as a function of number of steps n in $T = 1yr$. The table shows that both methods have approximately the same order of magnitude in regards to run time, and that the Euler's method is approximately 1.5 times faster for all test values of $n < 10^2$.

In regards to stability, both methods have decreasing ϵ_r for higher values of n, however by using the Velocity Verlet method, $epsilon_r$ decreases faster than when using the Euler's method. In addition, the Verlet method appears to reach a minimum ϵ_r at around $n = 10^5$ of approximately $7 \cdot 10^{-6}$, with $n = 10^6$ producing an ϵ_r that is $8 \cdot 10^{-9}$ larger.

Lastly, it is evident that Verlet conserves both energy and angular momentum for all tested values of $n > 2$, while the Euler's method has errors of the same magnitude as for ϵ_r.

3.2 Escape velocity

1.3 (V_esc) Presenter plot med analytisk v_esc. Deretter v_esc ved trial and error. Kommenter forskjell (høyere eller lavere enn analytisk?)

3.3 Effects of gravitational force

1.4 (Tyngekrafta) Multiplot med bane for sol jord med ulike beta

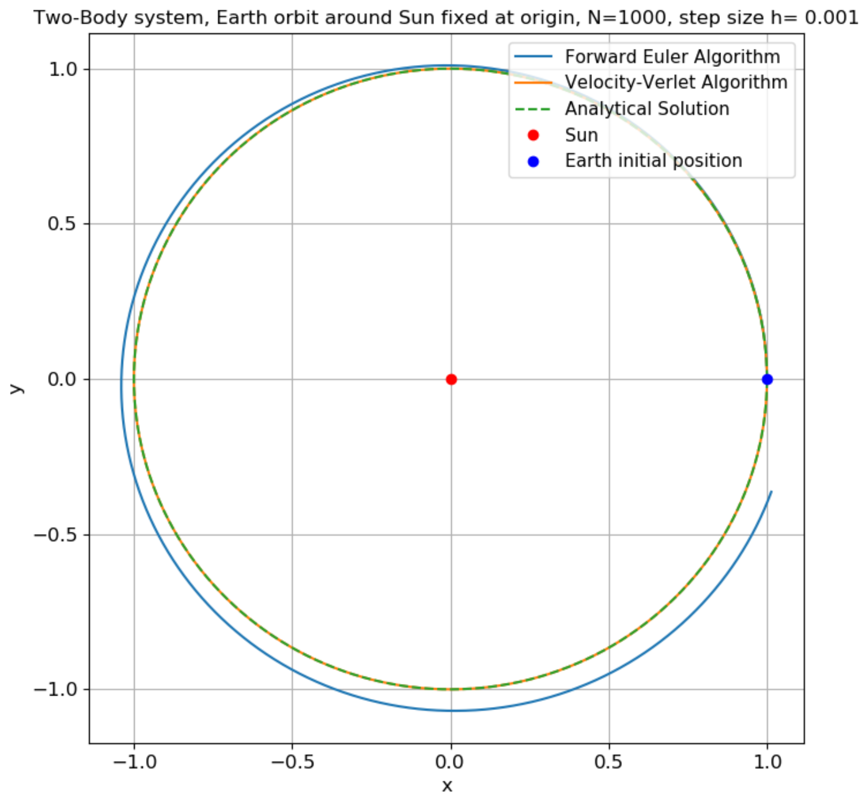

Figure 1: Plot of numerical solutions of the two body system: Sun-Earth, generated by the Euler's Forward and the velocity Verlet method

3.4 Object Orientation: Verifying integrity of methods

Figure 2 visualises the Sun-Earth system solved for $T = 1$ using the object oriented code with $n = 10^5$. The solution appears to satisfy the expected results; circular orbit with $r \approx 1$ AU. **Verification of conservartion L needed!**

3.5 Multi Body: Verifying integrity of implementation

2.1 - Plot ESJ oppå ES - For en N som tidligere har vært stabil - presenter momentumbevaring. Samme toleranse? Må vi øke N for samme stabilitet?

$n = 10^j$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Forward Euler						
Runtime [ms]	0.02	0.03	0.16	1.48±0.01	15.03±0.04	133.15±3.28
ϵ_r	3.67604	0.712392	0.076873	0.00787107	0.000794016	8.39334e-05
ϵ_E	1.47043	0.347594	0.0683006	0.00777134	0.000788617	7.76796e-05
ϵ_L	3.39498	0.229527	0.0359955	0.00390459	0.000392581	3.71361e-05
Velocity Verlet						
Runtime [ms]	0.01	0.03	0.24	2.40±0.02	24.28±0.08	198.00±3.62
ϵ_r	0.186713	0.00197411	2.02018e-05	8.62426e-07	6.99042e-07	6.99806e-07
ϵ_E	0.0248946	3.37737e-06	0	0	0	0
ϵ_L	0.00585689	0	0	0	0	0

Table 1: Comparison in ms of run time, ϵ_r, ϵ_E, and ϵ_L, between the Euler's forward method and the The velocity Verlet method implementations as a function of number of steps n in $T = 1yr$

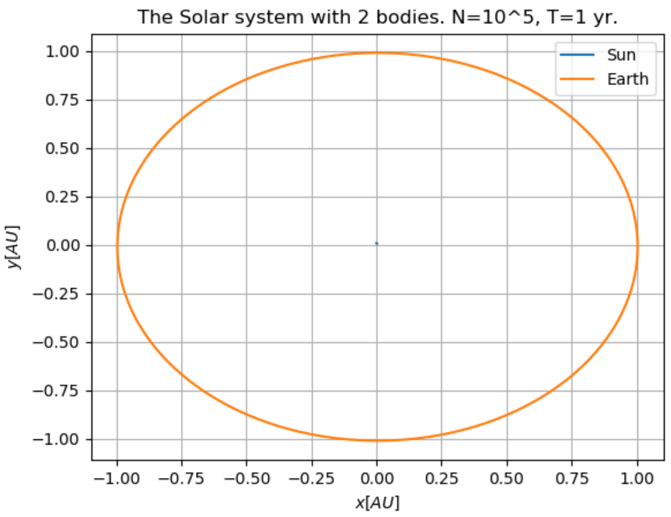

Figure 2: Plot of the Sun-Earth system using object oriented code with $n = 10^5$

3.6 Multi Body: The effects of another object on the Earth's trajectory

- plot for jupmassse x10 og x1000

3.7 Solar System:verifying integrity of implementation

- Plot over solsystemet for ett år <- simuleringa fungerer som forventa - Stabilitetssjekk: Angmom bevart?

3.8 Effects of relativity

4. Kjør Sol (flytende sol) og merkur - finn vinkel etter 100 år med newtonsk - finn vinkel etter 100 år med generell relativitet

. You may use that the speed of Mercury at perihelion is 12.44 AU/yr, and that the distance to the Sun at perihelion is 0.3075 AU. You need to make sure that the time resolution used in your simulation is sufficient, for example by checking that the perihelion precession you get with a pure Newtonian force is at least a few orders of magnitude smaller than the observed perihelion precession of Mercury. Can the observed perihelion precession of Mercury be explained by the general theory of relativity

4 Discussion of results

4.1 Initial implementation

On the basis of figure 1 appearing as expected - with both methods yielding near circular oribts, and the Verlet method being more accurate than the Euler method, it is assumed that the program is working as intended. The increasing error in *Euler* over time can be explained through table 1, which shows that Euler's forward method does not conserve angular momentum and energy.

In terms of stability, the Velocity Verlet method algorithm implementation is decidedly more stable for all values of n tested. In addition, $n = 10^5$ appears to produce the most accurate results. However, the difference between ϵ_r at $n = 10^5$ and $n = 10^6$ is small, and it is not clear whether this difference is the result of machine representation error or some other discrepancy.

In regards to runtime, Euler is faster than Verlet, just as expected, since the Verlet algorithm has more FLOPS. The difference appears to be given by an approximate factor of 1.5. All in all, $n = 10^5$ is decided upon as a suitable number of steps for the purpose of this project, both in regards to run time, and in regards to accuracy.

Both energy and angular momentum is conserved for $n = 10^5$ when using the Verlet based algorithm. This is a requirement when simulating such systems as this project focuses on (as explained under Evaluation of initial implementasjon), further empathizing that the velocity Verlet method is better suited for this project.

Because ϵ_r is an unsuitable measurement when a multi body system is examined (due to complexity of analytical solution), conservation of angular momentum is instead defined as the stability measurement to be used for all other purposes in this project. It is however important to note that angular momentum is conserved when ϵ_r is ~ 0.002 - which may indicate that the instability threshold increases when measured through conservation of angular momentum.

4.2 Escape velocity

1.3 (V_esc) Er det en sammenheng med hva vi så på energibevaring elns?

4.3 Effects og gravitational force

1.4 (Tyngekrafta) Kommenter noe på beta-> med høyere beta, avtar tyngekrafta fortere med avstand -> planetene stikekr av

4.4 Object Orientation: Verifying integrity of methods

As shown in the results section, the object orientation of the code appears to have been successfully. However, the implementation was not tested in depth for run time ϵ_r, ϵ_E, ϵ_L - so there may have been some (smaller) deviations from results produced by the non object oriented code. On the other hand, angular momentum was preserved, which is serves as a sufficient parameter for the purpose of this project.

4.5 The three body system

As the data from NASA uses a barycenter that includes objects that are not in use in this project, the barycenter may be barycenter - include bodies not used in our simulating - may have effects on solution.

4.5.1 Integrity of methods

-Verlet funker med 3 bodies - se at jordas bane (og solas) blir i økende grav påvirka.

4.5.2 Effects on Earth's trajectory

4.6 Solar system

4.6.1 Integrity of methods

- Ser plottet ut som forventa? Noen planeter ikke fullført bane -> de bruker lengre tid enn 1 yr! - Stabilitetssjekk: Angmom bevart ved en viss N med lik nøyaktighet som før?

- sucessfully simulated a simplified version of the solar system using all planets and newtonian gravity - not accounted for relativty and other astronomical objects

4.7 Effects of relativty

beskriv avviket på vinkel. Er det viktig å ta med generell relativitet?

5 Conclusions

References

[1] Morten Hjorth-Jensen. Computational physics lectures: Ordinary differential equations, Oct 5 2017.

[2] Morten Hjorth-Jensen. Computational physics assignment paper: Project 3, Oct 2018.

[3] Morten Hjorth-Jensen. Repository containing material pertinent to fys3150 at the university of oslo, 2018.

[4] NASA. Jpl's horizons system web-interface, 2018.