FYS4150 Project 3: Celestial Mechanics with Numerical Methods

Olav Fønstelien

October 26, 2020

Abstract

Analytic solutions to n-body gravitational problems are only possible in rare exceptions or with simplifications for $n \geq 3$. To study these problems we therefore need efficient and accurate numerical methods. In this report we investigate the Velocity Verlet method, which has a truncation error $O(h^3)$, and which, with 7 FLOPs per axis and iteration, is numerically efficient. Crucially, this method is also symplectic (energy conserving). We develop an object-oriented solver which we verify by reproducing the orbits of the Solar System, and use it to conduct numerical experiments.

For circular orbits, the Velocity Verlet method gives reliable results for 10^2-10^3 integration points per rotation, while for elliptic orbits, 10^3-10^4 should be used. Mechanical energy and angular momentum are conserved with high accuracy, with relative error often down to 10^{-6} if the orbits are stable. However, accuracy is dependent on angular and radial position, and is different for position and velocity. Conservation of energy and angular momentum serves as a reliable indicator of numerical stability.

The numerical experiments in this report show Earth's fluctuating orbit when simulated in a Sun-Earth-Jupiter system with various masses of Jupiter. With Jupiter at 100 times it's actual mass, Earth's orbit is stable, but fluctuates in a band approximately 0.15-0.20 AU wide. At 1000 times its actual mass, Earth leaves its orbit around the Sun and starts to swirl around Jupiter. We also show that the model correctly predicts orbit eccentricity, and that closed elliptic orbits are a feature of the inverse-square radius to force relationship in Newton's gravitational law. Conservation of mechanical energy and angular momentum is still preserved in this alternate domain, however. Our model also correctly reproduces the perihelion precession of Mercury at 43 arc seconds per century when we introduce a general relativistic correction to Newton's gravitational law. This requires a very high accuracy, at 10^7 integration points per rotation. The error of the reference calculation without the general relativistic error is 0.22 arc seconds per century.

Visit my GitHub repository at `https://github.com/fonstelien/FYS4150/tree/master/project3` for source code and raw data from the numerical simulations.

1 Introduction

For all its beauty, the limits of differential calculus quickly becomes apparent when it comes to the study of planetary movements. The classic n-body problem for describing the movement of bodies in interacting gravitational fields can be perfectly accounted for at $n = 2$, but problems arise already at $n = 3$. Hence the need for numerical methods.

In this report we will develop an n-body *solver* for gravitational problems based on the *Velocity Verlet* algorithm [5]. We will use the solver to perform numerical experiments to answer questions about how a planet's orbit around the Sun is affected by the presence of other planets, how the orbit would be affected by a *change* in Newton's gravitational law, and what this tells us about the *accuracy* of that law. We will also touch on general relativity when we reproduce the perihelion precession of Mercury with high accuracy, a famous test of general relativity [4]. The validity of the results must always be validated against the preservation of energy and angular momentum, which we keep a keen eye on all along.

Section 3 starts by looking at the most basic numerical method of all for solving differential equations involving the movement; Euler's forward method. Then we introduce the Velocity Verlet method and see how it outperforms Euler's even if computationally more demanding. We also give a framework for an object-oriented implementation of the Velocity Verlet algorithm. Then, in Section 4, we investigate our model's performance and present the results of the experiments, before discussing their interpretation and wrapping up with some suggestions for further studies in Section 5. But first, in the next section, let us revisit some of the basics of celestial mechanics, which will serve as the mathematical background for the testing of the algorithms and the interpretation of the results.

2 Celestial Mechanics: Kepler and Newton's Laws

It was *Johannes Kepler* (1571-1630) who first described the planets' orbits correctly. By observation, he found that all planets move in elliptic orbits with the sun at one of the two foci; and that the square of their periods T are directly proportional to the cube of their semi-major axis a [2] (Kepler's first and third laws, respectively);

$$T^2 \propto a^3. \tag{1}$$

Perfect harmony prevailed, it seemed. Later, *Isaac Newton* (1642-1727) would describe with his gravitational law the mutual attractive force F_G between two bodies of mass M, m at distance r apart as

$$F_G = \frac{GMm}{r^2}, \tag{2}$$

where G is the gravitational constant. Further, as a result of Newton's second law, the centripetal force acting on a body with mass m moving along a circular

path with radius r at angular velocity ω is given by

$$F_C = mr\omega^2. \tag{3}$$

Kepler, having died 12 years before Newton's birth, would have been astonished when he discovered that when combining these laws, his own would have been confirmed. For the simple case of a circular orbit, with the Sun's mass M, the planet mass m, and the angular velocity $\omega = 2\pi/T$, we get by equating F_G and F_C that

$$mr\left(\frac{2\pi}{T}\right)^2 = \frac{GMm}{r^2} \Rightarrow \frac{r^3}{T^2} = \frac{GM}{4\pi^2}. \tag{4}$$

An important result of this combination of Newton's gravitational and second laws is that they postulate the conditions for closed elliptic orbits. Following [2], a planetary orbit's *eccentricity* is given by

$$e = \frac{r_0 v_0^2}{GM} - 1, \tag{5}$$

where r_0 is the planet's distance from the Sun at any of the orbit's vertices, and v_0 is the velocity at that point. Calculating e we get the trajectories:

$$\begin{aligned} e < 1 \quad &: \quad \text{closed elliptic} \\ e \geq 1 \quad &: \quad \text{parabolic/hyperbolic trajectory.} \end{aligned} \tag{6}$$

The special case $e = 0$ gives a circular orbit, and for the cases $e \geq 1$, the planet escapes into space, never to return. This is easily seen by reminding ourselves that a planet's mechanical energy, that is; the sum its kinetic energy, $E_K = \frac{1}{2}mv^2$, and potential energy, $E_P = -GMm/r$, is preserved during flight, such that

$$E_M = E_K + E_P = \frac{1}{2}mv^2 - GMm/r = const. \tag{7}$$

Now, if the sum at the vertex is zero or positive we have that

$$\frac{1}{2}mv_0^2 \geq \frac{GMm}{r_0} \Rightarrow \frac{r_0 v_0^2}{GM} \geq 2 \Rightarrow e \geq 1, \tag{8}$$

and since $E_K \geq E_P$, the planet will escape. For the circular orbit, we recall from Equation (4) that $F_G = F_C$, such that

$$\frac{mv_0^2}{r_0} = \frac{GMm}{r_0^2} \Rightarrow \frac{r_0 v_0^2}{GM} = 1 \Rightarrow e = 0. \tag{9}$$

We note that in a circular orbit, $E_K = \frac{1}{2}E_P$.

The last of Kepler's observations was that the planets move in their orbits such that the line joining it to the sun sweeps over equal areas in equal intervals of time, whatever the line's length. Again, see [2]. This observation of constant *areal velocity*, called Kepler's second law, can be written

$$\frac{dA}{dt} = \frac{1}{2}r^2\frac{d\theta}{dt} = const. \tag{10}$$

This would also be explained as a consequence of Newton's discoveries. The *angular momentum* of a body with mass m moving at velocity $\mathbf{v}$ at distance $\mathbf{r}$ from a fixed point O is defined as $\mathbf{L}_O = \mathbf{r} \times m\mathbf{v}$. Now, with $\mathbf{r}$ and $\mathbf{v}$ at an angle ϕ to each other, $|\mathbf{r} \times \mathbf{v}| = rv \sin\phi$; and the component of $\mathbf{v}$ perpendicular to $\mathbf{r}$ is the body's velocity on a circular path about O, so $v \sin\phi = r\frac{d\theta}{dt}$. Bringing this all together, we get that Kepler's second law foresaw the conservation of angular momentum:

$$\mathbf{L}_O = L_O\mathbf{e} = (mrv\sin\phi)\mathbf{e} = mr^2\frac{d\theta}{dt}\mathbf{e} = 2m\frac{dA}{dt}\mathbf{e} = \textit{const. vector} \quad , \tag{11}$$

where $\mathbf{e}$ is a unit vector. We should note that the conservation of angular momentum of a body holds for any trajectory, but that it's relation to Kepler's second law only holds for circular or closed elliptic orbits.

3 Numerical Methods for Celestial Mechanics

We will start with developing a solver for the simplified 2-body problem with a fixed Sun at the center and a planet moving around it. We will use this for proving the correctness our algorithm, as well as studying its efficiency with regards to CPU time and accuracy. Then, we will proceed to solving the general n-body problem in an object-oriented manner.

3.1 Sun-Earth System with Fixed Sun

The position at time $t + h$ of a body moving along a straight line with velocity $v = v(t)$, is given by the Taylor expansion

$$x(t + h) = x(t) + hv(t) + O(h^2), \tag{12}$$

where $O(h^2)$ is the truncation error. The velocity $v(t + h)$ at the new position $x(t + h)$, is again given by a new Taylor expansion

$$v(t + h) = v(t) + h\frac{dv}{dt}(t) + O(h^2), \text{ where } \quad \frac{dv}{dt}(t) = a(t). \tag{13}$$

We discretize time in the time interval $[t, t + \Delta t]$ by letting $t_i = t + ih$, where $i \in [0, n]$ and $h = \Delta t/n$. The discrete equations for the body's motion are then given as

$$\begin{aligned} x_{i+1} &= x_i + hv_i \\ v_{i+1} &= v_i + ha_i \end{aligned} \quad . \tag{14}$$

With these equations, called *Euler's forward method*, we can can iteratively calculate the object's movement along the x axis. These are expanded to three dimensions with corresponding equations for movement along the y and z axes.

To apply Euler's forward method on Newton's gravitational law (2), we plug in Equation (2) for the acceleration. On vector form and in spherical coordinates the acceleration is given as

$$\mathbf{a}_i = \frac{GM}{r_i^3}\mathbf{r}_i. \tag{15}$$

where M is the mass of the attracting body. In the Solar System it is convenient to refer time, mass and space to the Earth-Sun system. Hence, we will scale away GM from Equation (16) using the relationship we found in Equation (4), namely $GM = 4\pi^2 r^3/T^2$, where M is the mass of the Sun, $r = 1$ AU, and $T = 1$ year, such that

$$GM = 4\pi^2 \quad [\mathrm{AU}^3/\mathrm{yr}.^2]. \tag{16}$$

Now, the with cartesian representation of $\mathbf{a}_i$, we get the full set of forward Euler equations in three dimensions for solving the problem of planetary movement about the Sun, where the Sun is kept fixed at the center;

$$\begin{aligned} x_{i+1} &= x_i + hv_{x,i}, \quad v_{x,i+1} = v_{x,i} - 4\pi^2 h\frac{x_i}{r_i^3}, \\ y_{i+1} &= y_i + hv_{y,i}, \quad v_{y,i+1} = v_{y,i} - 4\pi^2 h\frac{y_i}{r_i^3}, \\ z_{i+1} &= z_i + hv_{z,i}, \quad v_{z,i+1} = v_{z,i} - 4\pi^2 h\frac{z_i}{r_i^3}, \end{aligned} \tag{17}$$

where $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. A framework of the algorithm is given in Listing 1 below.

Listing 1: Euler's Forward algorithm for Sun-Earth system with fixed Sun.

```
// Declarations
n = "number of steps per year"
y = "years to calculate"
N = y*n // total number of steps
h = 1.0/n // time step length
array(3) position = "initial x,y,z"
array(3) velocity = "initial vx, vy, vz"
array(3) pos0 // for storing postion after update

// Iterations
FOR i = 1...N DO
  pos0 = position
  position = position + h*velocity
  r = norm(pos0)
  velocity = velocity - h*4*pi^2/r^3*pos0
END DO
```

A problem with Euler's Forward method, though, is that it does not conserve energy, it is not symplectic [3]. This can be shown by evaluating a few iterations. A correction to this energy drift is possible with the small modification $x_{i+1} = x_i + hv_{x,i+1}$, known as the the *Euler-Cromer method*. However, this too has a truncation error $O(h^2)$, which means that it quickly becomes too coarse or inefficient if we want to calculate long time spans with high precision. An alternative method, which is both energy conserving and more precise is the *Velocity Verlet* method. Taking again the Taylor expansion for calculating the

movement of a body along a straight line, we have

$$\begin{aligned} x(t+h) &= x(t) + hv(t) + \frac{h^2}{2}a(t) + O(h^3) \\ v(t+h) &= v(t) + ha(t) + \frac{h^2}{2}\frac{d^2v}{dt^2}(t) + O(h^3) \end{aligned}, \tag{18}$$

where $O(h^3)$ is the truncation error. $\frac{d^2v}{dt^2}(t)$ can itself be found using a Taylor expansion, such that

$$\frac{d^2v}{dt^2}(t) = \frac{a(t+h) - a(t)}{h} + O(h), \tag{19}$$

which, since $h^2O(h)$ yields $O(h^3)$, gives us

$$\begin{aligned} x(t+h) &= x(t) + hv(t) + \frac{h^2}{2}a(t) + O(h^3) \\ v(t+h) &= v(t) + \frac{h}{2}(a(t+h) + a(t)) + O(h^3) \end{aligned}. \tag{20}$$

We see that we need $a(t+h)$ in order to calculate $v(t+h)$, which may not always be possible. For solving gravitational problems, however, the acceleration is dependent on position only, meaning that if we calculate $x(t+h)$ first, we can obtain $a(t+h)$ and then move on to calculating $v(t+h)$.

Discretizing as we did for Euler's Forward method, we get again the full set of equations for solving the problem of planetary movement about the Sun by the Velocity Verlet method, with the Sun fixed at the center:

$$\begin{aligned} x_{i+1} &= x_i + hv_{x,i} - 2\pi^2h^2\frac{x_i}{r_i^3}, \quad v_{x,i+1} = v_{x,i} - 2\pi^2h\left(\frac{x_{i+1}}{r_{i+1}^3} + \frac{x_i}{r_i^3}\right), \\ y_{i+1} &= y_i + hv_{y,i} - 2\pi^2h^2\frac{y_i}{r_i^3}, \quad v_{y,i+1} = v_{y,i} - 2\pi^2h\left(\frac{y_{i+1}}{r_{i+1}^3} + \frac{y_i}{r_i^3}\right), \\ z_{i+1} &= z_i + hv_{x,i} - 2\pi^2h^2\frac{z_i}{r_i^3}, \quad v_{z,i+1} = v_{z,i} - 2\pi^2h\left(\frac{z_{i+1}}{r_{i+1}^3} + \frac{z_i}{r_i^3}\right), \end{aligned} \tag{21}$$

where $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. See Listing 2 below for an algorithm framework.

Listing 2: Velocity Verlet algorithm for Sun-Earth system with fixed Sun.

```
// Declarations
n = "number of steps per year"
y = "years to calculate"
N = y*n // total number of steps
h = 1.0/n // time step length
array(3) position = "initial x,y,z"
array(3) velocity = "initial vx, vy, vz"
array(3) acceleration
array(3) acc0 // for storing acceleration after update
```

```
// Initializing
r = norm(position)
acceleration = -4*pi^2/r^3*position

// Iterations
FOR i = 1...N DO
  position = position + h*velocity - h^2/2*acceleration
  acc0 = acceleration
  r = norm(pos0)
  acceleration = -4*pi^2/r^3*position
  velocity = velocity - h/2*(acceleration + acc0)
END DO
```

Comparing the Verlocity Verlet method with Euler's Forward method, we see that Euler's is more efficient with regards to FLOPs (FLoting point OPersations) per iteration. Euler needs one multiplication and one addition for each position x_i, y_i, z_i; plus two multiplications and one subtraction for each velocity $v_{x,i}, v_{y,i}, v_{z,i}$. 15 FLOPs in total. However, r_i and the factor r_i^{-3} must be calculated once per iteration. This involves square root and floating point divide operations, which are much more expensive than regular FLOPs and are not pipelined properly in the CPU [1]. For Velocity Verlet, total number of FLOPs is $(3+4)\cdot 3 = 21$, and even if the method has both r_i, r_i^{-3} and r_{i+1}, r_{i+1}^{-3} terms, we store these between the iterations such that the most expensive operations are done only once, as we saw in Listing 2.

CPU time for both Euler Forward and Velocity Verlet has a linear relation to problem size, as we see in Table 1. As expected, Euler is the fastest, but more than what we can explain by the FLOPs ratio between the algorithms. A possible explanation is the square root operations, which are bound to make trouble with pipelining. Another possibility is the use of `C++ Armadillo` classes. In these completely CPU bound algorithms, these may cause some of the additional variables in Verlet to be stored outside of CPU register, meaning that they must be loaded from L1 cache.

This speed advantage of Euler relative to Verlet is easily offset when we compare the accuracy between the two. As we see in Figure 1, which shows Earth in a circular orbit around the Sun over 100 years, Velocity Verlet is remarkably stable. Even for $n = 100$ iteration steps per year, the accumulated error in the energy balance is on the order of 10^{-8}. For Euler's Forward, we see a complete breakdown even at $n = 1000$, partly stemming from its lower accuracy and partly from its non-symplecticity.

Figure 1 also shows that the 1/2 relationship between kinetic and potential energy is conserved for the Verlet results, in accordance with Equation (9) for eccentricity $e = 0$. We also see that after 100 years, Verlet has lagged about 1/8th of a year for $n = 100$, so to get good accuracy for long runs, we should use at least $n = 1000$. In the following, we will limit our discussion to Velocity Verlet.

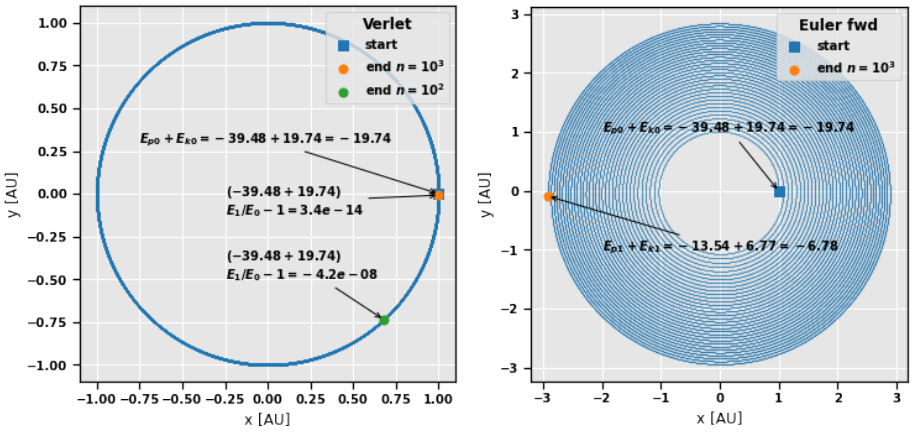

Figure 1: Trace of 100 years of the Sun-Earth system with fixed Sun and circular orbit for the Velocity Verlet method to the left and Euler's Forward method to the right. Euler quickly breaks down due to low precision and energy drift, while Verlet has far superior accuracy for a 10th of the iteration steps. Verlet conserves energy with high accuracy even for $n = 100$ steps per year, but will lag a bit unless n is selected high enough.

Table 1: Averaged CPU times for Euler's Forward and Velocity Verlet algorithms when solving the Sun-Earth system with fixed Sun. Each run is 1 year. As could be expected, as soon as the pipelining reaches its full potential, scaling is perfect. The speed difference between the methods is somewhat surprising, though, since that cannot all be explained by the $7/5 = 1.4$ FLOPs ratio between them. Both algorithms are CPU-bound. Results were obtained on an Intel i7-8550U CPU, with implementation in `C++` with `Armadillo 10.1.0`.

n	Euler's Forward [s]	Velocity Verlet [s]	ratio [%]
10^1	2.000e-07	1.000e-06	500
10^2	1.500e-06	3.500e-06	233
10^3	1.450e-05	2.980e-05	206
10^4	1.439e-04	2.817e-04	196
10^5	1.511e-03	2.879e-03	191
10^6	1.543e-02	2.699e-02	175
10^7	1.573e-01	2.776e-01	176
10^8	1.526e+00	2.730e+00	179
10^9	1.530e+01	2.735e+01	179

3.2 Solar System

We can extend the Velocity Verlet method in Equation (21) to the general n-body system quite easily. Newton's gravitational law for the acceleration of a body 1 at location $\mathbf{r}_1$ towards a body 2 with mass m_2 at location $\mathbf{r}_2$ is given by

$$\mathbf{a}_{12} = \frac{Gm_2}{r_{12}^3}\mathbf{r}_{12} \quad , \quad \text{where} \quad \mathbf{r}_{12} = \mathbf{r}_2 - \mathbf{r}_1. \tag{22}$$

The total acceleration $\mathbf{a}_1$ of body 1 in an n-body problem is then simply the sum of the distance vectors to the other bodies, $\mathbf{r}_{1i}$, weighted by their masses m_i over r_{1i}^3:

$$\mathbf{a}_1 = \sum_{i=2}^{n} \frac{Gm_i}{r_{1i}^3}\mathbf{r}_{1i}. \tag{23}$$

As in the Sun-Earth system we then scale away the planet masses and the gravitational constant using $GM = 4\pi^2$ AU^3/yr.^{2}, where M is the Sun's mass, such that

$$Gm_i = GM \cdot \frac{m_i}{M} = 4\pi^2 \cdot \frac{m_i}{M} = 4\pi^2 \cdot \mu_i. \tag{24}$$

For planet p we then get the following set of equations for solving the n-body problem using the Velocity Verlet method

$$\begin{aligned} x_{i+1}^{(p)} &= x_i^{(p)} + hv_{x,i}^{(p)} - 2\pi^2 h^2 \sum_{\substack{j=1 \\ j\neq p}}^{n} \frac{\mu_j x_i^{(pj)}}{(r_i^{(pj)})^3} \\ v_{x,i+1}^{(p)} &= v_{x,i}^{(p)} - 2\pi^2 h \sum_{\substack{j=1 \\ j\neq p}}^{n} \left(\frac{\mu_j x_{i+1}^{(pj)}}{(r_{i+1}^{(pj)})^3} + \frac{\mu_j x_i^{(pj)}}{(r_i^{(pj)})^3} \right) \end{aligned} \quad , \tag{25}$$

and correspondingly for the y and z axes.

As these over-crowded sub- and superscripts reveal, we need more than a few equations to completely describe an n-body system. There are 8 planets in the solar system, and counting the Sun and Pluto we have 60 equations to solve. To keep track of this all, we can pick one of two strategies which each have their merit. The first is an array-based scheme where we allocate an array of length n for each variable, and keep track of the planets that way. This would probably be the most computationally efficient with regards to pipelining in the CPU. The other is object orientation, which is probably the simplest to implement and which we will use here.

In the object oriented approach, each planet and the Sun will be an object. The `Planet` object should keep its static data, limited to its mass relative to the Sun in this case, and keep the state of its position and velocity. It should also provide services which enable us to update the position and velocity states and let us query key properties which relate to this object only, like kinetic energy and angular momentum about the reference center. An interface framework

for implementing a `Planet` object for the Velocity Verlet algorithm is given in Listing 3.

Having defined the `Planet` object, we next define an object for keeping track of all the `Planet`s and calling on their services for updating state. This `Solver` object should thus provide a data structure for keeping the `Planet` collection as well as services to add `Planet`s to the collection and initiating the Velocity Verlet algorithm. It should also provide services which let us query key properties which extend beyond the single `Planet` object, like potential energy, as well as total energies and angular momentum. See Listing 4 for an interface framework.

Apart from `total_potential_energy()`, where we must be careful to count the potential energies only once, the only interesting service to take a closer look at in the `Solver` object is `solve()`. The Velocity Verlet algorithm is executed in much the same way as for the Sun-Earth case in Listing 2, but extended to looping over all `Planet`s. See Listing 5 for a framework.

Listing 3: Planet object interface for Velocity Verlet algorithm solving the n-body problem.

```
BEGIN OBJECT Planet
  // Static data
  mass // mass relative to the Sun

  // Object state
  array(3) position
  array(3) velocity
  array(3) acceleration
  array(3) acc0 // for storing acceleration after position update

  // Services to update state
  update_position(dt) // dt is time step size
  update_velocity(dt)
  update_acceleration(other_planet) // call once for each other object

  // Services to query properties
  kinetic_energy()
  angular_momentum()
END OBJECT
```

Listing 4: Solver object interface for keeping track of the `Planet` objects in the n-body problem, and manage the update of their state.

```
BEGIN OBJECT Solver
  // Object state
  collection planets // collection of Planet objects

  // Services to update state
  add(new_planet) // adds new Planet object to collection
  solve(N, dt) // N is number of iterations; dt is time step
```

```
  // Services to query properties
  potential_energy(planet) // total potential energy of planet
  total_potential_energy() // total potential energy of all planets
  total_kinetic_energy() // total kinetic energy of all planets
  total_angular_momentum() // total angular momentum of all planets
END OBJECT
```

Listing 5: The `Solver` object's solve() service for executing the Velocity Verlet algorithm on the n-body problem.

```
BEGIN FUNCTION solve(N, dt)
  global collection planets // collection of Planet objects
  num_planets // number of planets in collection planets

  // Initializing acceleration
  FOR i = 1...num_planets DO
    planet1 = planets[i]
    FOR j = (i+1)...num_planets DO
      planet2 = planets[j]
      planet1.update_acceleration(planet2)
      planet2.update_acceleration(planet1)
    END DO
  END DO

  // Iterations
  FOR n = 1...N DO
    // Update position
    FOR i = 1...num_planets DO
      planets[i].update_position(dt)
    END DO

    // Update acceleration at new position
    FOR i = 1...num_planets DO
      planet1 = planets[i]
      FOR j = (i+1)...num_planets DO
        planet2 = planets[j]
        planet1.update_acceleration(planet2)
        planet2.update_acceleration(planet1)
      END DO
    END DO

    // Update velocity
    FOR i = 1...num_planets DO
      planets[i].update_velocity(dt)
    END DO
  END DO
END FUNCTION
```

We test our object-oriented model on the Solar System. To get a stable solution, we need to ensure that the initial relative positions and velocities of the planets correspond to closed elliptic orbits and that the total momentum of the Solar System is zero. *NASA Jet Propulsion Laboratory* [6] publishes highly accurate live data on the position and velocity of the objects in the Solar system, where these quantities naturally are preserved, which we use to build our model.

Figure 2 shows a simulation of the Sun and the planets in the Solar System over 250 years, which is a little over 1 rotation for Pluto (included even if strictly not defined as a planet). The model reproduces the elliptic orbits and preserves energy and angular momentum. We also recognize the partly overlapping orbits of Neptune and Pluto in the middle pane.

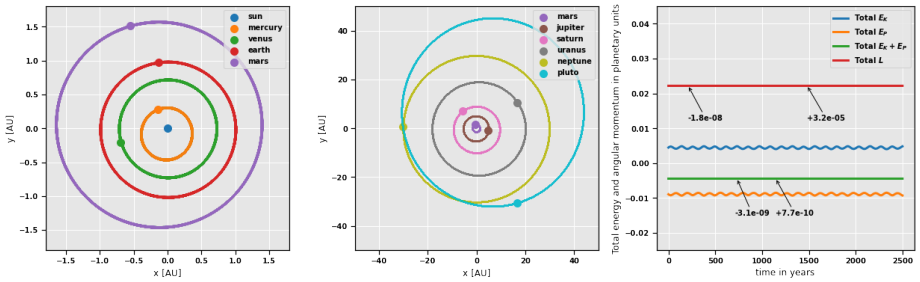

Figure 2: Solar system simulated using our object-oriented Velocity Verlet solver, with 10^4 iterations per year over 250 years (a little more than 1 rotation for Pluto). Simulation was run with non-fixed Sun. We see that the model reproduces distinct elliptic orbits of the planets in the middle and to the left. To the right, we see that the model preserves energy and total angular momentum with high accuracy. Annotations are relative error from initial values, and planetary units are on a Sun mass-AU-year base. Initial states collected from [6], with states as of 2020-Oct-15 at 00:00:00.0000.

4 Results

After having developed an object-oriented Velocity Verlet solver for the n-body problem in the former section, we may now move on and do numerical experiments with our model. But before we do that, we should try to develop a qualitative feeling about the model's accuracy and stability.

4.1 Accuracy and Stability of the Velocity Verlet Solver

In Figure 1 we saw that the accuracy of the energy balance and angular momentum is high even for relatively large time step h. The explanation for that is the $O(h^3)$ truncation error of the Velocity Verlet method. However, as we see in Figure 3, we may not assume that the position and velocity accuracies are

constant in three-dimensional space. In the example shown in the left and middle of Figure 3, Earth is rotated once about the Sun in a circular orbit starting at $(x, y, z) = (1, 0, 0)$, with various time step. We see that both the error in the position and in the velocity varies with rotational angle θ, but again that the sum of the error, when translated to energy, zeros out.

To the right in Figure 3, a planet is put at $(x, y, z) = (1, 0, 0)$ with zero velocity, and we see the error in the velocity as it accelerates against the Sun. As the planet moves towards the sun, the error in the velocity grows steeply. When the planet is close to the sun, at some point the $hv_{x,i}$ term in x_{i+1} may also cause a step-through of the gravitational center, causing a discontinuity. The result is that the planet shoots off into space.

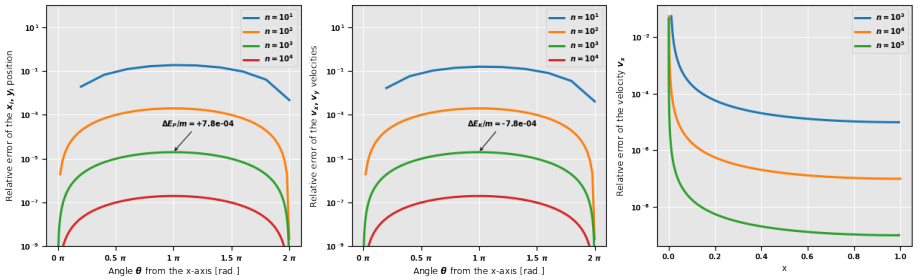

Figure 3: Relative error of the Velocity Verlet algorithm depending on location for various time steps. To the left we see the error in the position for Earth following a circular orbit of radius 1.0. $\varepsilon = |1 - \sqrt{x_i^2 + y_i^2}|$. Start conditions were `pos = {1.0, 0.0, 0.0}` and `vel = {0.0, 2*PI, 0.0}`, and the simulation is run for 1.0 rotation. In the middle we see the same, but for the velocity. We see that the kinetic and potential energy error cancel each outher out. $\varepsilon = |2\pi - \sqrt{v_{x,i}^2 + v_{y,i}^2}|/2\pi$. To the right we see the velocity error of a planet accelerating from standstill towards the sun. Close to the sun, the Velocity Verlet method becomes unstable. Initial conditions were `pos = {1.0, 0.0, 0.0}` and `vel = {0.0, 0.0, 0.0}` with fixed sun. $\varepsilon = |v(x_i) - v_{x,i}|/v(x_i)$.

4.2 Preservation of Mechanical Energy and Angular Momentum: Earth's Orbit with a Super Massive Jupiter

We saw in Section 2 that a planet's mechanical energy and angular momentum is preserved during flight. Let us put our model to the test by introducing a super massive Jupiter in the Sun-Earth System, and see how it preserves these key quantities. Jupiter's mass in the Solar System is a little less than one thousandth of the Sun's, so it would be interesting to see how much larger it may be before Earth's orbit is influenced significantly, and how this measures up to Kepler's first law, which states that all planets' orbits are elliptic.

In Figure 4 we have simulated the Sun-Earth-Jupiter system, with varying

multiples of Jupiter's mass. Sun was kept fixed at the center. We see that for 1x and 10x Jupiter masses, Earth's orbit is largely undisturbed, only a little delayed in the latter due to a gentle radial fluctuation in the trajectory. This tendency is more pronounced for 100x Jupiter masses. We see that the orbit still is regular, but that it fluctuates within a band of about 0.15-0.20 AU. As a consequence, Kepler's first law does not hold in this experiment.

Up until 100x Jupiter, mechanical energy and angular momentum are preserved with high accuracy, meaning that the solver performs well.

Another 10-fold increase to 1000x Jupiter causes trouble, though. As we see to the far right in Figure 4, Earth is quickly thrown out of its regular orbit, getting dangerously close to the sun, where at around 5 years, the model breaks down. Earth then swirls around Jupiter for some 20 years, before again being thrown against the sun, and then shoots off into space. The simulation was run with 10^3 integration points per year in all cases. Increasing to 10^4 or even 10^5 did not change the stability significantly.

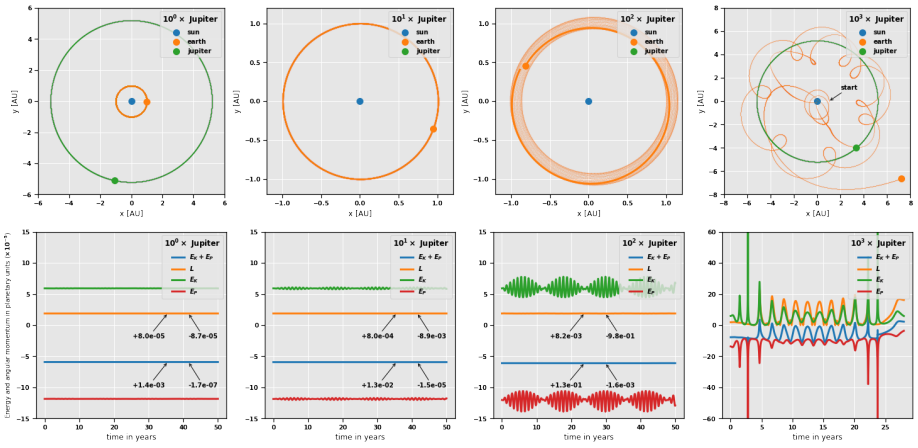

Figure 4: Sun-Earth-Jupiter system with varying sizes of Jupiter with Velocity Verlet. In the top row we see the Earth's worsening orbit for increasing mass of Jupiter, before Earth's orbit completely breaks down. In the 100x Jupiter pane in the top row, the shaded area is the band within which Earth's orbit fluctuates, while the thick line shows the trajectory in the last year. In the bottom row we see that up until 100x Jupiter mass, Earth's angular momentum and energy balance are both stable. For 1000x Jupiter mass, everything breaks down when Earth comes close to first the Sun, and then starts swirling around Jupiter, before it takes off into space. Annotations are relative error from initial values. Note the 10^{-5} Sun-masses-AU-year scaling. Initial conditions for Earth were `pos = {1.0, 0.0, 0.0}` and `vel = {0.0, 2*PI, 0.0}`, and for Jupiter `pos = {-5.2, 0.0, 0.0}` and `vel = {0.0, -2*PI/SQRT(5.2), 0.0}`, giving both a counter-clockwise circular rotation. Sun was fixed. 1000 integration points per year in all cases.

4.3 Newton's Gravitational Law: Eccentricity and Forms of the Force

Kepler's third law about the $T^2 \propto a^3$ relationship between period and elliptical axis got its confirmation in the combination of Newton's gravitational law and second law, Equation (4). This again gave us the eccentricity factor e, which easily tells us about the shape of an object's trajectory based on initial conditions r_0, v_0;

$$e = \frac{r_0 v_0^2}{GM} - 1. \tag{26}$$

We note again that $e < 1$ gives a closed elliptical or circular orbit, and $e \geq 1$ lets the object escape into space. Thus, with $r_0 = 1$ and using $GM = 4\pi^2$, we have initial velocities giving

$$\begin{aligned} \text{circular orbit:} \quad & v_0 = 2\pi \\ \text{elliptic orbit:} \quad & v_0 < 2\pi\sqrt{2} \quad . \\ \text{escape :} \quad & v_0 \geq 2\pi\sqrt{2} \end{aligned} \tag{27}$$

Figure 5 shows the Sun-Earth system with various initial velocities giving closed and open trajectories, which are in accordance with Equation (30). We also see that the mechanical energy is negative for the closed trajectories, and zero or positive for the open, again in accordance with Equation (30), as we showed in Equations (7), (8) and (9).

Now, it would be interesting to put Kepler's third law to the test by modifying Newton's gravitational law. We re-phrase it by replacing the inverse square r^2 factor by r^β, and adjust β to see what happens to the planet orbit;

$$F_G = \frac{GMm}{r^\beta}, \tag{28}$$

and the corresponding potential energy E_P is then on the form

$$E_P = \frac{GMm}{(-\beta + 1)r^{\beta - 1}}. \tag{29}$$

To reflect this in our Velocity Verlet solver, we update `update_acceleration()` service in `Planet`, and `potential_energy()` service in `Solver`. When we change β away from 2, we should still expect that energy and angular momentum are conserved in this alternate physical domain. Kepler's third law should not hold, however, since that is a consequence of Newton's gravitational law in combination with Newton's second law.

Figure 6 shows just that. As we see to the left, even a slight change of 1/1000th clearly makes the planet drift out of its regular orbit. Increasing β to 2.5 gives rise to a beautiful doughnut shape, which however does not seem to be repeating itself. The only closed orbit that seems possible in this domain is the circular, which occurs at Further increasing β to 3, we see a collapse of the orbit for $v_0 < 2\pi$. The orbit is circular for $v_0 = 2\pi$, and open for $v_0 > 2\pi$. In all cases of β, we see that both energy and angular moment are conserved.

If we concentrate on $\beta = 3$, in a way it still makes sense to talk about *eccentricity* in this domain, too. With scaling, we get the same initial velocities as in Equation (30), except for the elliptic orbit, which will not occur:

$$\begin{aligned} \text{collapse:} \quad & v_0 < 2\pi \\ \text{circular orbit:} \quad & v_0 = 2\pi \quad , \\ \text{escape :} \quad & v_0 > 2\pi \end{aligned} \tag{30}$$

where we have assumed $r_0 = 1$. The circular is easily explained, since $r_0 = 1$. The collapse we have demonstrated to the right in Figure 6. Escape for $v_0 > 2\pi$ is a consequence of the mechanical energy, which is positive. Using the modified gravitational law $v^2/r = 4\pi^2/r^3$, we get for the circular orbit that

$$E_M = E_K + E_P = \frac{1}{2}mv_0^2 - \frac{4\pi^2 m}{2r_0^2} = \frac{1}{2}m\left(\frac{4\pi^2}{r_0^2}\right) - \frac{4\pi^2 m}{2r_0^2} = 0. \tag{31}$$

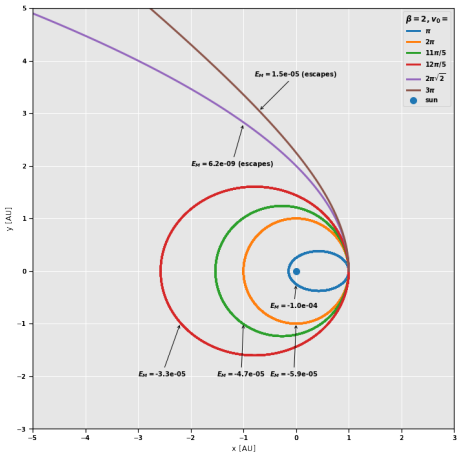

Figure 5: Various degrees of eccentricity versus initial velocity v_0 (Equation 30) with Velocity Verlet. We see that mechanical energy $E_M < 0$ gives closed elliptic or circular orbits, while $E_M \geq 0$ give open trajectories (body escapes). Simulations were run with 10^4 integration points per year, over 20 years. Initial conditions were `pos = {1.0, 0.0, 0.0}` and `vel = {0.0, v0, 0.0}` with fixed sun.

4.4 General Relativity: The Perihelion Precession of Mercury

A modification to Newton's gravitational law that is real, comes from the general relativistic correction given by [4] as

$$F_G = \frac{GMm}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right], \tag{32}$$

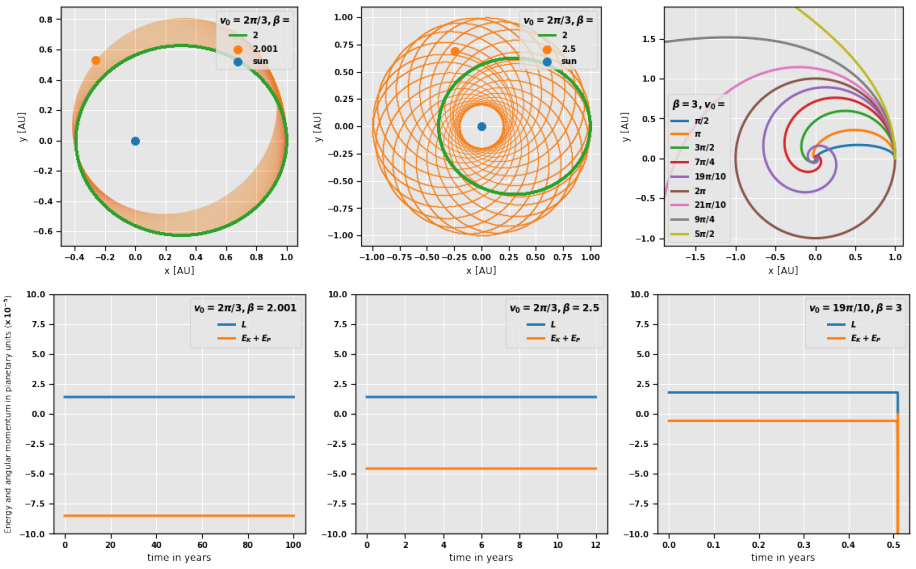

Figure 6: Alternate forms of the gravitational force simulated with Velocity Verlet. The inverse square radius, r^2, has been replaced by r^β as in Equation (29). To the upper left we see that even a slight deviation from $\beta = 2$ makes closed elliptic orbits impossible. Over 100 years, the orbit has shifted upwards in this case. The green line is the orbit in the orbit over 100 years with $\beta = 2$. Upper middle, with $\beta = 2.5$, shows a beautiful doughnut shaped orbit, which appears not to be repeating itself. Again, the green line shows the orbit with $\beta = 2$ in the same time frame. Upper right shows collapse, circular and escape trajectories for $\beta = 3$. The lower row shows that energy and angular momentum are conserved in this alternate physical domain. The exception is the numerical breakdown of the model to the far right in the lower right picture, which corresponds to crashing into the Sun. Initial conditions were `pos = {1.0, 0.0, 0.0}` and `vel = {0.0, v0, 0.0}` with fixed sun, and with simulation times as shown on the x-axes of the lower row. 10^4 integration points per year.

where $l = |\mathbf{r} \times \mathbf{v}|$ is the per-unit-mass angular momentum, and c is the speed of light. As we saw in the previous section, modifying Newton's gravitational law led to non-closed planetary orbit, and the correction given here will do the same. The observed perturbation of Mercury's perihelion, when other effects like the pull from other planets is taken away, is 43 arc seconds per century [4], and adding this general relativistic factor to the calculation of acceleration in our Velocity Verlet solver should reproduce the same result.

To reproduce this, however, requires 100 years of simulation with extremely high accuracy. To get good performance while we iterate towards the number of integration points per year which gives an acceptable accuracy, we write a special function which performs this special task. This model considers only

two dimensions, since any two-body problem is planar.

Table 2 shows the results for 10^3 to 10^8 integration points. We see that the error in the uncorrected calculation drops to about 0.2 arc secs at 10^7, and the corrected has a perturbation of 43.1 arc seconds, which is the result we wanted. It is also worthwhile to increase the number of integration points another 10-fold to see that the perturbation does not proceed beyond 43 arc seconds. We see that at $n = 10^8$, the perturbation is 43.0 arc seconds, indicating that the angle is converging and that the calculation is correct.

Table 2: Perturbation per century of Mercury's perihelion. We get a useful result of 43.1 arc seconds when the uncorrected error drops to below 1 for $n = 10^7$. The calculation with $n = 10^8$ indicates that the angle converges and that the calculation is correct. The simulation was done on a Sun-Mercury system with fixed Sun and initial conditions `pos = {0.3075, 0.0}` and `vel = {0.0, 12.44}`.

n	arc secs w/o correction	arc secs w/ correction
10^3	-1.034e+05	-1.032e+05
10^4	-1.429e+03	-1.264e+03
10^5	1.549e+01	1.335e+01
10^6	-1.780e+00	4.615e+01
10^7	2.165e-01	4.314e+01
10^8	-1.995e-02	4.296e+01

5 Discussion and Conclusion

Euler's Forward method showed us that energy conservation is a requirement when solving differential equations involving motion, and that even if Euler-Cromer deals with this, the higher accuracy of the Velocity Verlet method makes it more suitable, even if it is more computationally demanding. For circular trajectories, Velocity Verlet is reliable even for as little as 10^2-10^3 integration points per rotation, while for elliptic orbits, we need to increase the number to 10^3-10^4.

Its accuracy is not independent of angular or radial position. We saw that it varies both in circular and axial trajectories, and when the distance between two bodies becomes too small, the method breaks down numerically. An interesting observation relating to this which remains to be explained is that when simulating a circular orbit with Velocity Verlet using 100 integration points per year over 100 years, the total distance traveled in Figure 1 was short by 1/8th of a year, while both kinetic and potential energy were conserved with high accuracy. The planet seems to have *moved* slower than its velocity. The explanation may lie in that the position and velocity equations have different accuracies.

The object-oriented Velocity Verlet solver we developed could be used to

conduct numerical experiments on n-body problems. It correctly reproduced the orbits of the planets in the Solar System, and predicted correctly the trajectories in our experiment with eccentricity. We saw that by controlling the preservation of mechanical energy and angular momentum, the numerical stability could be monitored even when we changed Newton's gravitational law. Our experiments with the gravitational force's relation to radius suggested that closed orbits are a phenomenon of the inverse-square radius.

By introducing a general relativistic correction to Newton's gravitational law, our model reproduced the famous perihelion precession of Mercury with high accuracy. This confirms both that purely elliptic orbits are a mathematical phenomenon, and that the observed precession of Mercury can be explained by the general theory of relativity.

Some interesting experiments which could be conducted with the Velocity Verlet solver are to simulate the formation of new, stable orbits by initiating the planets in other states than they are in the physical world; how the orbits would be affected by the slow arrival of a Jupiter-sized planet from outer space; the and to investigate the planetary orbits around twin stars.

For future work, it would also be interesting to investigate the expansion of the Velocity Verlet solver to include an adaptive time step algorithm. This would both increase computational efficiency, while improve numerical stability. Another interesting possibility is to expand it to electric and magnetic forces, which would render it a powerful tool for solving a wider set of physics problems.

References

[1] Georg Hager and Gerhard Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, Boca Raton, 2010.

[2] Russell Charles Hibbeler. *Engineering mechanics: statics & dynamics, 9th edition*. Prentice Hall, New Jersey, 2001.

[3] Morten Hjort-Jensen. Computational physics, handwritten lecture notes, 2-october-2020. https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/HandWrittenNotes/NotesOctober2.pdf, [Online; accessed 16-October-2020].

[4] Morten Hjort-Jensen. Project 3, computational physics fys4150. http://compphysics.github.io/ComputationalPhysics/doc/Projects/2020/Project3/html/Project3.html, [Online; accessed 26-October-2020].

[5] Morten Hjort-Jensen. Computational physics, lecture notes fall 2015. https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf, August 2015.

[6] NASA Jet Propulsion Laboratory. Solar System Dynamics. https://ssd.jpl.nasa.gov/horizons.cgi, 2020. [Online; accessed 16-October-2020].