# Building a model for the solar system using ordinary differential equations

Tor-Andreas Bjone, Håvar Rinde Lund Jacobsen, Håkon Rinde Lund Jacobsen
(Dated: October 26, 2020)

We have built a model for the solar system using ordinary differential equations and object-orientation. Starting off with a simple two body system, we gradually build on by adding more features until we have a model for the entire solar system consisting of the Sun and all planets. Testing different parts of the model in special cases to find out what is the Earth's escape velocity and how we can improve Mercury's orbit precision by using the theory of general relativity. Testing parts of our model is also done to make sure that our algorithms are implemented correctly, for example by checking energy conservation. In total we implement four differential equation solvers, these are Euler's forward, Euler-Cromer's, Velocity Verlet and the Runge-Kutta fourth order methods. These methods are compared to see if efficiency and precision reflect what we would expect from a theoretical perspective.

## I. INTRODUCTION

In this report we will look at how we can create a model for the solar system, by solving ordinary differential equations for planetary motion, that is Newton's second law (A1). Starting off with a simple two dimensional system only consisting of the Earth and the Sun, we gradually build on to end up with a three dimensional model for the entire solar system. In the simplest two body case, we fix the position of the Sun to be the origin and look at the relative motion of the Earth. Assuming circular orbits for the Earth, we scale our equations such that positions are given in astronomical units (AU) and time is given in years. This will prove beneficial later on when we include more planets, because the acceleration for each planets are given by equation (B3), which is easy to calculate. Accelerations are calculated by using masses relative to the solar mass, and units in of $AU/(years)^2$.

Equations of motion are solved numerically. The goal is the make the numerical implementation as object-oriented as possible. We have implemented four numerical differential equation solvers in total. These algorithms are Euler's forward method and Euler-Cromer's method (C), the Velocity Verlet method(D) and the fourth order Runge-Kutta method (E). We have mainly focused on using the Velocity Verlet algorithm because it provides a good balance between precision and run-time, as well as being energy conserving (symplectic). For some specific cases we have compared the algorithms in terms of stability and run-time to see how they behave and to find out their strengths and weaknesses. In order to make sure that the algorithms have been implemented correctly, we have made unit tests for checking conservation of energy and angular momentum. These quantities should be conserved for stable elliptical planetary orbits in our solar system. Using real observational data from NASA we obtain correct initial conditions for positions and velocities. This allows us to predict future positions and velocities many years ahead, which is interesting and important when dealing with real physical systems. By including some features from the theory of general relativity, we are able to do corrections on Mercury's orbit such that it correspond better to observational data.

The report begins with an explanation of methods. Starting off with a two dimensional two body system (the Earth-Sun case). Assuming a fixed positioned Sun at first. Later we allow the Sun to move, and showing how to calculate correct initial conditions. Building on from here, we introduce Jupiter as our third planet and expand our system to three dimensions. Then generalising this to the $N$-body system, with our solar system as an example. Results and discussion of results are next, and finally the conclusions. We have included several appendixes to give additional information about the theoretical background, which we will also refer to throughout the report.

## II. METHOD

This section explains the numerical implementation of algorithms and the methods used for calculations in this project.

### A. The Earth-Sun binary system

The first and simplest case we have looked at is the Earth-Sun binary system in two dimensions. Assuming circular co-planar orbits in the $xy$-plane, we scale our equations of motions as described in Appendix (B) using equation (B2) and assume the Sun to be fixed in the origin. This means that we fix the center of mass to be the origin. This simplified case allows us to test out our algorithms and check that they obtain valid results. For this simple case we have tested the Forward Euler's algorithm (C) and the Velocity Verlet algorithm (D).

Implementing Velocity Verlet numerically: Defining a total number of integration points over a time period of one year. Using initial conditions $x = 1\ AU$ for position and $v_y = 2\pi\ AU/year$ for velocity. Also defining the scaling parameter $4\pi^2$. The source can be found at the GitHub-repository [2].

```
  int NumberofSteps = 1000;
  double FinalTime = 1.0;    // 1 Year simulation.
  double Step = FinalTime/((double) NumberofSteps);
  double time = 0.0;         // Initial time.
  // Initial values  x = 1.0 AU and vy = 2*pi AU/year
  double pi = acos(-1.0);
  double FourPi2 = 4*pi*pi;
  double x0 =  1.0; double y0 =  0.0;
  double vx0 = 0.0; double vy0 = 2.0*pi;

  vec x = zeros(NumberofSteps);
  x[0] = x0;
  vec y = zeros(NumberofSteps);
  y[0] = y0;
  vec vx = zeros(NumberofSteps);
  vx[0] = vx0;
  vec vy = zeros(NumberofSteps);
  vy[0] = vy0;
```

The algorithm is then run in a while loop:

```
int i = 0;
  while (time <= FinalTime)
  {
    double r0 = sqrt(x[i]*x[i]+y[i]*y[i]);
    double ax0 = acceleration(FourPi2, x[i], r0);
    double ay0 = acceleration(FourPi2, y[i], r0);

    x[i+1] = x[i] + Step*vx[i] + Step*Step*ax0/2;
    y[i+1] = y[i] + Step*vy[i] + Step*Step*ay0/2;

    double r1  = sqrt(x[i+1]*x[i+1]+y[i+1]*y[i+1]);
    double ax1 = acceleration(FourPi2, x[i+1], r1);
    double ay1 = acceleration(FourPi2, y[i+1], r1);

    vx[i+1] = vx[i] + Step*(ax1 + ax0)/2;
    vy[i+1] = vy[i] + Step*(ay1 + ay0)/2;

    time += Step;
    i += 1;
    }
```

The acceleration is calculated as intermediate steps:

```
double acceleration(double FourPi2, double k, double r)
{
  double a = -FourPi2*k/(r*r*r);
  return a;
}
```

The algorithm above produces the following results presented in figure (1). The results are as we expect, the Earth's orbit around the Sun is stable and circular. Using the same conditions on the same system, the Forward Euler's algorithm produces results with no visible difference, as we can see in figure (2). For more complex systems (for example the three body problem), which we will eventually arrive at, Forward Euler fails (low accuracy, not symplectic). Therefor we apply the Velocity Verlet method because it has better precision and conserves energy (symplectic).

Building on from here, we have mainly focused on using the Velocity Verlet algorithm. Even still we will include some comparisons of the results using the Euler methods and RK4 (E) in terms of stability (error) and floating point operations. From here on we will make use of object-orientation in the numerical implementation. Moving on from two dimensions to three dimensions.

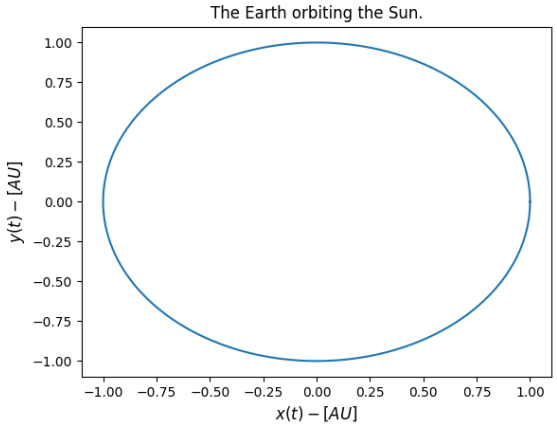


Figure 1. Binary Earth-Sun system in two dimensions. Solved using the Velocity Verlet algorithm.

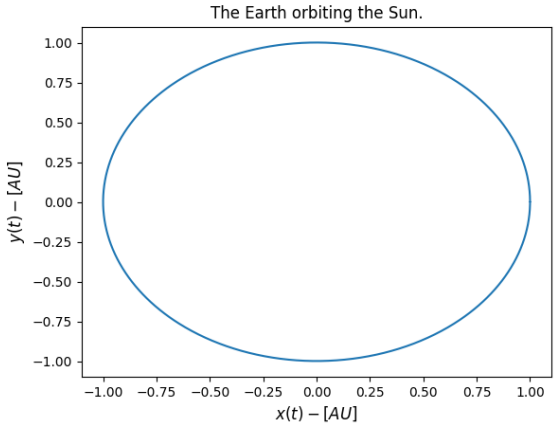


Figure 2. Binary Earth-Sun system in two dimensions. Solved using the Forward Euler's algorithm.

Implementing planets in the object-orientated code is shown below. We specify a given number of planets $n$, which is two in the Earth-Sun system. Then we use the planet-class to specify the initial conditions as

$$(mass, x, y, z, v_x, v_y, v_z),$$

where *mass* is given in units of solar masses, positions in $AU$ and velocities in $AU/year$. The example below uses data from NASA [4]. The source for this code is our GitHub project repository [2].

```
int n = 2; //number of planets
planet *planets[n];
```

```
\\Sun
planets[0] = new planet(1,
    -6.005339779329579E-03,6.491104601150000E-03,
    8.587420750830977E-05,
    au_pr_yr(-7.374879897223016E-06),
    au_pr_yr(-4.969570018104684E-06),
    au_pr_yr(2.194527060357223E-07), "Sun");

//Earth
    planets[1] = new planet(solar_mass(5.972*1E24),
    9.826038297983306E-01, 1.626617426930922E-01,
    7.845130873662143E-05,
    au_pr_yr(-2.970939298989496E-03),
    au_pr_yr(1.691727251702593E-02),
    au_pr_yr(-7.366459664051695E-07), "Earth");
```

When we include the motion of the Sun, we have to use initial conditions such that energy, momentum and angular momentum are conserved. We can use real data for initial conditions from NASA [4] to ensure this, like shown above, but for the Earth-Sun system we might as well calculate these conditions roughly by first assuming circular orbits. Using NASA's data we will obtain elliptical orbits, which we can compare. Assuming that the center of mass is in the origin. Starting off with momentum, we know that the Earth's initial velocity vector in three dimensions is

$$\vec{v}_{Earth} = (0, 2\pi, 0),$$

in units of $AU/year$. The Earth's momentum $\vec{p}_{Earth} = M_{Earth}\vec{v}_{Earth}$ is then

$$\vec{p}_{Earth} = M_{Earth}(0, 2\pi, 0).$$

The Sun's momentum has to be the same, with opposite sign. This means that

$$\vec{p}_{Sun} = M_{Sun}(0, -\frac{2\pi M_{Earth}}{M_{Sun}}, 0).$$

From this we see that the Sun's initial velocity has to be

$$\vec{v}_{Sun} = (0, -2\pi\frac{M_{Earth}}{M_{Sun}}, 0),$$

in order for momentum to be conserved. The initial angular momentum (B9) of the Earth is

$$L_E = M_E x_E v_E,$$

where $M_E = M_{Earth}$, $x_E = 1\ AU$ and $v_E = 2\pi\ AU/year$ in the $y-$direction. This has to equal the angular momentum of the Sun for the angular momentum to be conserved. The angular momentum of the Sun initially is

$$L_S = M_S x_S v_S,$$

where $M_S = M_{Sun}$, $x_S$ is unknown and $v_S = -2\pi\frac{M_E}{M_S}$ in the $y-$direction. Requiring that $L_E = L_S$ and solving for $x_S$ gives us the initial position of the Sun along the $x-$axis, assuming $y = z = 0$, such that angular momentum is conserved. The initial position for the Sun is then

$$x_S = -\left(\frac{M_E}{M_S}\right)^2\ AU.$$

This shows that our assumption by fixing the center of mass at the origin is a good approximation for finding valid initial conditions. Since the mass of the Sun is far greater than the mass of the Earth

$$\frac{M_{Earth}}{M_{Sun}} \approx 3 \cdot 10^{-6},$$

using values from table (I). We see that $x_S = -9 \cdot 10^{-12}\ AU$, which is a quite small distance on an astronomical scale. This also explains why we can assume a fixed positioned Sun as the center of mass when modelling the solar system.

We have made unit tests in order to check that the total energy and total angular momentum is conserved for the Earth-Sun system. Using equation (B4) for kinetic energy and equation (B6) for potential energy. The angular momentum can be calculated using equation (B9) for each of the planets, and checking that the magnitude is constant. Running the tests [2] from terminal window is shown below:

```
havarjacobsen@eduroam-193-157-196-107 FYS3150_project_3 % make all
c++ -O3 -o ./main.out unit_test.cpp
planet.cpp diff_solver.cpp
-larmadillo -llapack -lblas
./main.out
==========================
======================
==============================
All tests passed (3 assertions in 1 test case)
```

We are using an epsilon error tolerance of $10^{-4}$ for these tests.

For a circular orbit, both kinetic energy and potential energy should be conserved. This is evident when looking at the equations for kinetic energy (B5) and potential energy (B7). Circular orbits have constant-magnitude velocity and constant distance to the center of mass (radius), therefor these quantities are conserved.

Knowing that the Earth's orbit around the Sun is stable using our circular orbit approximation, we have calculated by trial and error how different initial velocities effect the orbit. We expect the Earth to escape the Sun's gravitational field when the velocity approaches the theoretical escape velocity, given by equation (B11).

With our scaling, $v_{escape} = 2\pi\ AU/year$. As long as $v < v_{escape}$, we expect stable elliptical orbits because the total energy of the Earth will remain negative, as explained in appendix (B).

We have tested the alternative force model (B12) in order to investigate how much nature deviates from a perfect inverse square law, like Newton's gravitational law (B1). We have investigated this $\beta \in [2, 3]$ with the Velocity Verlet algorithm on the Earth-Sun system, assuming circular orbits with the initial conditions derived above for conservation of momentum and angular momentum.

### B. Three body system

By adding Jupiter to our solar system, we have a three body system. Using data from NASA [4], we obtain values for positions and velocities such that energy, momentum and angular momentum are conserved. Using the Velocity Verlet algorithm (D), we calculate orbits as before with equation (B3) for the acceleration.

We have investigated how much Jupiter alters the motion of the Earth around the Sun. Then we increase Jupiter's mass by a factor of 10, and finally 1000, in order to check how to this impact the stable Earth-Sun system.

The three body system is implemented numerically with

```
int n = 3; //number of planets
planet *planets[n];
\\Sun
planets[0] = new planet(1,
    -6.005339779329579E-03,6.491104601150000E-03,
    8.587420750830977E-05,
    au_pr_yr(-7.374879897223016E-06),
    au_pr_yr(-4.969570018104684E-06),
    au_pr_yr(2.194527060357223E-07), "Sun");

//Earth
    planets[1] = new planet(solar_mass(5.972*1E24),
    9.826038297983306E-01, 1.626617426930922E-01,
    7.845130873662143E-05,
    au_pr_yr(-2.970939298989496E-03),
    au_pr_yr(1.691727251702593E-02),
    au_pr_yr(-7.366459664051695E-07), "Earth");
//Jupiter
    planets[2] = new planet(solar_mass(1.8982*1E27),
    2.465917496117004E+00,
    -4.485467614711858E+00, -3.656260354902041E-02,
    au_pr_yr(6.520238149039668E-03),
    au_pr_yr(3.993243263732226E-03),
    au_pr_yr(-1.624209573907791E-04), "Jupiter");
```

### C. N body system

Implementing the $N-$body system (the solar system) is just a generalization of the methods so far. When we include all the planets however, we need to run the algorithms over a longer time period to make sure that the outer planets complete at least one full orbit. Using data from NASA [4] to make sure that our initial conditions are valid, and masses from table (I).

The following is set up in the diff_solver.cpp class. We define the position and velocity information for planet $j$ at a time step $i$ to be $(x_j, y_j, z_j, v_{x,j}, v_{y,j}, v_{z,j})$. We use this to set up a position and velocity matrix for the $n$ planets.

$$X = \begin{pmatrix} x_1 & y_1 & z_1 & v_{x,1} & v_{y,1} & v_{z,1} \\ x_2 & y_2 & z_2 & v_{x,2} & v_{y,2} & v_{z,2} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n & y_n & z_n & v_{x,n} & v_{y,n} & v_{z,n} \end{pmatrix}$$

In the step function we set up this this matrix using armadillo[6] in the following way.

```
mat current_XV = zeros(n,6);
  //Updating current position
  for(int i=0;i<n;i++){
    for(int j=0;j<3;j++){
      //Getting the current position from the planet class
      current_XV(i,j) = planets[i]->position[j];
      //Getting the current velocity from the planet class
      current_XV(i,j+3) = planets[i]->velocity[j];
      }
    }
```

To calculate the next step we first set up a dXdt matrix for the planets similar to the position-velocity matrix.

$$dXdt = \begin{pmatrix} v_{x,1} & v_{y,1} & v_{z,1} & a_{x,1} & a_{y,1} & a_{z,1} \\ v_{x,2} & v_{y,2} & v_{z,2} & a_{x,2} & a_{y,2} & a_{z,2} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ v_{x,n} & v_{y,n} & v_{z,n} & a_{x,n} & a_{y,n} & a_{z,n} \end{pmatrix}$$

This matrix is calculated in the diffEq function by first setting up the matrix with armadillo and filling in the previous velocity values.

```
mat dydt = zeros(n,6);
  //Filling in previous velocity values for updating the position
  for(int i=0;i<n;i++){
    for(int j=0;j<3;j++){
      dydt(i,j) = current_XV(i,j+3);
    }
}
```

The distance vectors between the planets are set up using the current time step.

```
for(int k=0;k<3;k++){
        r_i[k] = current_XV(i,k);

for(int j=0;j<n;j++){
    for(int k=0;k<3;k++){
        r_j[k] = current_XV(j,k);
```

There is a for-loop over $r_j$, because this has to be calculated for each planet before the acceleration is calculated. We calculate the acceleration either with or without the relativistic correction. The acceleration for each planet $i$ is calculated as shown in equation B3.

```
if(relativistic_correction==true){
          a_i += (-Gconst * planets[j]->mass *
          1/powf(norm(r_i-r_j),beta) * r_ij) *(1+3*pow(l,2)*
          1/(powf(norm(r_i-r_j),beta)*pow(speed_of_light,2)));
        }
        else{
          a_i += -Gconst * planets[j]->mass *
          1/powf(norm(r_i-r_j),beta) * r_ij;
        }
```

This is then returned to the step function. If you use Euler you can just return the next time step as as $X_{i+1} = X_i + dXdt\Delta t$. For Velocity-Verlet and Euler-Cromer we need to calculate a temporary $X_{i+1}$, and then use this to calculate $dXdt_{i+1}$. In the program we have implemented this in the following way:

```
mat dydt = diffEq(current_XV);
    mat next_XV = dydt*deltaT + current_XV;
    mat dydt_next = diffEq(next_XV);
```

We implement the Velocity Verlet method as described in appendix D and the RK4 method as in E.

```
if(method=="Velocity-Verlet"){
    //Velocity-Verlet
    mat new_XV = zeros(n,6);

    mat dydt = diffEq(current_XV);
    mat next_XV = dydt*deltaT + current_XV;

    mat dydt_next = diffEq(next_XV);

    for(int i=0;i<n;i++){
      for(int j=0;j<3;j++){
        new_XV(i,j) =current_XV(i,j)+deltaT*dydt(i,j)
        +deltaT*deltaT/2*dydt(i,j+3);

        new_XV(i,j+3) = current_XV(i,j+3)
        +deltaT/2*(dydt_next(i,j+3)+dydt(i,j+3));
      }
    }
    return new_XV;
  }


    if(method=="RK4"){
    //RK4
    mat k1 = diffEq(current_XV);
    mat k2 = diffEq(current_XV + k1*deltaT*1./2);
    mat k3 = diffEq(current_XV+k2*deltaT*1./2);
    mat k4 = diffEq(current_XV+k3*deltaT);
    mat dydt = (k1 + 2*k2 + 2*k3 + k4)/6;
    return current_XV + dydt*deltaT;
```

In the solve function we set up a time vector with armadillo[6] with length $N \cdot \Delta t$, where $N$ is the number of integration points.

```
vec t = linspace(0,N*deltaT,N);
```

We then write this to file. To check our runtime we set up a clock using the time.h library. The clock is started right before the integration and ended right after. The runtime will be slower than optimal because we are writing to file.

```
runtime = 0;
    //Starting clock for timing the runtime
    start = clock();
```

We then integrate $N$ steps using the step function and update each of the planets new positions with their pointers. Then we write this to an outfile and end the clock and save the runtime.

```
    //Starting integration
    for(int steps=0;steps<N;steps++){
      mat new_XV = step(method);
      for(int i=0;i<n;i++){
        for(int j=0;j<3;j++){
          planets[i]->position[j] = new_XV(i,j);
          planets[i]->velocity[j] = new_XV(i,j+3);
        }
      }
      //To save space in our textfile we dont save every timestep
      k+=1;
        if(k>=interval){
          outfile << new_XV << endl;
          k=0;
        }
    }
    //Ending the clock and saving the value
    finish = clock();
    runtime = ( (finish - start)*1./CLOCKS_PER_SEC );
    outfile.close();
```

The solve function is given a step_saved integer. This is used to calculate an interval in which no steps are saved. We then let a $k$ run for each time step. When $k$ is equal to this interval length we save the time-step to file. In this way the program runs faster because it doesn't have to save every step and our file size is smaller, but we still get the precision of a high number of integration points.

```
//making an interval that
//says how many points between every saved step
  int timesteps_pr_year = 1./deltaT;
  int interval, points_saved;
  if(step_saved>timesteps_pr_year){
    interval = 1;
    points_saved = N;
  }
  else{
    interval = timesteps_pr_year*1./step_saved;
    points_saved = step_saved*1./timesteps_pr_year * N;
  }
```

### D. Mercury

The classical prediction on Mercury's orbit does not correspond exactly to the observations, where as the predictions made by the theory of general relativity do. We have implemented the relativistic correction using equation (B13) to calculate and compare the perihelion angle with the value obtained from Newton's gravitational law (B1), as explained in appendix (B) under the general relativity section. The speed of Mercury at perihelion is approximately 12.44 $AU/year$ and the distance to the Sun from this point is 0.3075 $AU$. The perihelion angle $\theta$ is calculated by using

$$tan\theta = \frac{y_p}{x_p},$$

where $(x_p, y_p)$ is the perihelion position on Mercury's orbit around the Sun.

### E. Algorithms

The algorithms used in this project are Euler's methods (C), Velocity Verlet (D) and Runge-Kutta of

fourth order (E). These methods differ in terms of the number of floating point operations used for calculations (which affects run-time) and accuracy (precision, error). For the methods involved in this project, the slowest methods are also the most accurate. The reason we have chosen to mainly focus on Velocity Verlet it strikes a balance between run-time and accuracy, as well as being symplectic (energy-conserving).

The Velocity Verlet algorithm requires roughly twice the amount of floating point operations than that of the Euler's forward method. The RK4 algorithm requires roughly fourth times as many floating point operations as the Euler's forward method. This will be reflected in the run-time for each algorithm, which we have tested for the Earth-Sun system with elliptical orbits. We have also checked how well these methods do for different time steps by checking energy conservation (stability).

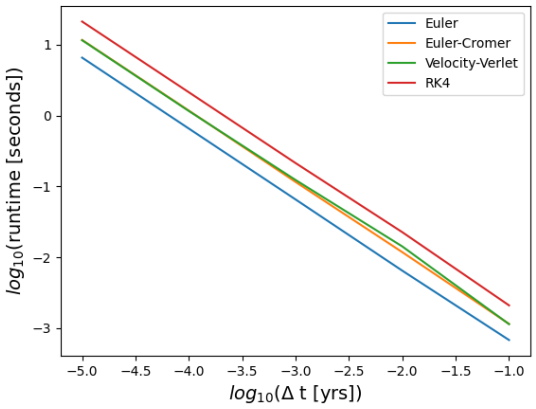


Figure 3. Run-time (logarithmic scale, $y-$axis) for different algorithms when simulating the Earth-Sun system over a period of 100/ : *years*. Calculating orbits with time steps of $10^{-2}, 10^{-2.5}, ..., 10^{-5}$ *years* shown on a logarithmic scale on the $x-$axis.

## III. RESULTS

This section describes the results obtained. We discuss these results and compare our findings to what we would expect.

Figure (3) shows the run-time for the four different algorithms: Euler, Euler-Cromer, Velocity Verlet and RK4 when solving the Earth-Sun system with circular orbits. As expected when evaluating the number of floating point operations, the Euler's Forward method is the fastest. Euler-Cromer and Velocity Verlet are approximately equal, and RK4 is the slowest. When considering the amount of floating point operations involved for the different algorithms, this is roughly as expected.

Figure (4) shows the relative total energy of the Earth, $E/E_0$, where $E$ is the current total energy and $E_0$ is the initial total energy, in the Earth-Sun system assuming elliptical orbits. As we can see, energy is conserved. We checked this already when developing the methods with unit tests. This is a graphical illustration of the result. We clearly see that Euler is the least accurate method in terms of energy conservation, next is Euler-Cromer and Velocity Verlet, and finally the RK4 is the most accurate method. When taking into account the number of floating point operations (run-time), the Velocity Verlet method is the most efficient method overall.

Figure (5) shows how the Earth's orbit changes due to an increase in initial velocity. When the velocity is

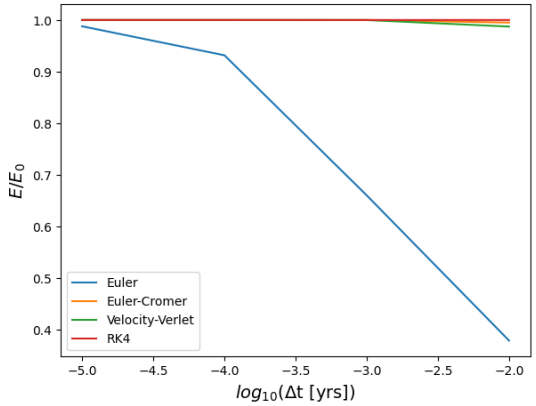


Figure 4. Stability check for different algorithms, by checking conservation of energy on the $y-$axis. Earth-Sun system, elliptical orbits. Calculating orbits with time steps of $10^{-1}, 10^{-2}, ..., 10^{-5}$ *years* shown on a logarithmic scale on the $x-$axis. Plotting 20 time steps per. Total period is 100 *years*.

increased from $2\pi$ $AU/year$, the orbit changes from circular with the Sun in center to an elliptical with the Sun in focus. When the initial velocity equals the theoretical escape velocity, the Earth's total energy equals zero and the Earth is able to escape the gravitational field. The theoretical value for the escape velocity therefor seems valid.

Figure (6), (7) and (8) show the results from calculating orbits in the Earth-Sun system using the alternative force model (B12). Assuming co-planar orbits in the three dimensional plane, we have plotted the projection to the two dimensional $xy-$plane. There is not much

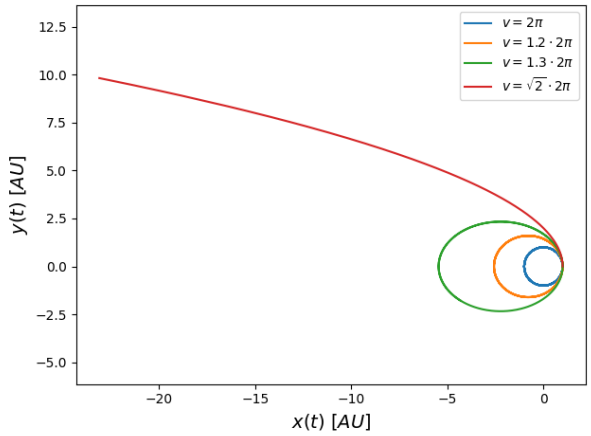


Figure 5. Earth's orbit around the Sun for different initial velocities up to and including the theoretical escape velocity.

of a visual difference between $\beta = 2.95$ and $\beta = 2$. The orbits still remain stable. When $\beta = 3$, the orbits deviates more and more for each year.

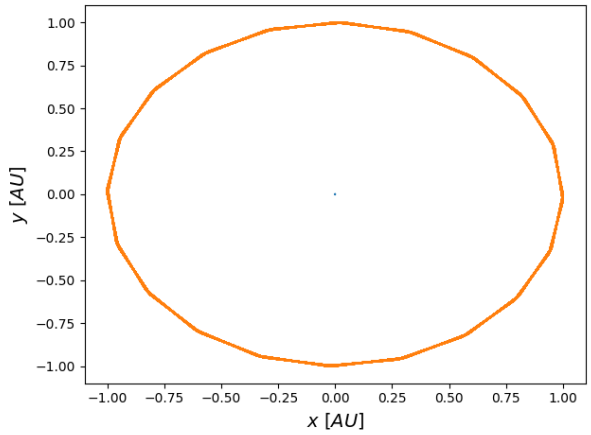


Figure 6. Earth-Sun system with alternative force model (B12), using $\beta = 2.9$ and circular orbits. Velocity Verlet algorithm. Time steps of $10^{-4}$ *years* and total time period of 100 *years*.

The three body system consisting of the Earth, the Sun and Jupiter with real data for initial positions and velocities from NASA [4] is shown in figure (9). We can see from this that the Earth's orbit is barely impacted by including Jupiter. The reason for this is that the mass of the Sun is roughly 1000 times larger than Jupiter, and the distance between the Earth and Jupiter is roughly 5 times as large as the distance between the Earth and the Sun (I). Jupiter's contribution to the Earth's acceleration is therefore negligible. It is also worth noting that the z-axis is scaled here so it looks like the inclination of Jupiter is higher than it really is.

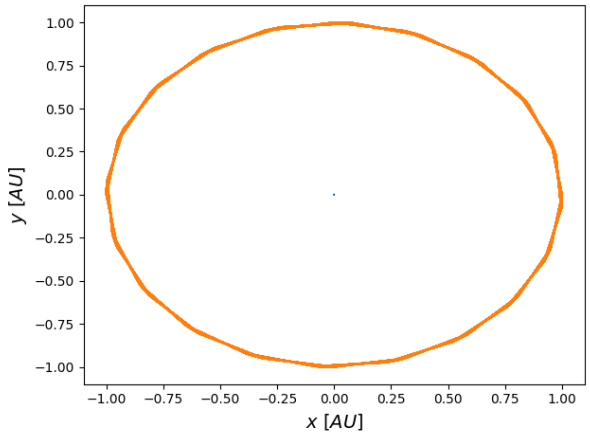


Figure 7. Earth-Sun system with alternative force model (B12) and circular orbits, using $\beta = 2.95$. Velocity Verlet algorithm. Time steps of $10^{-4}$ *years* and total time period of 100 *years*.

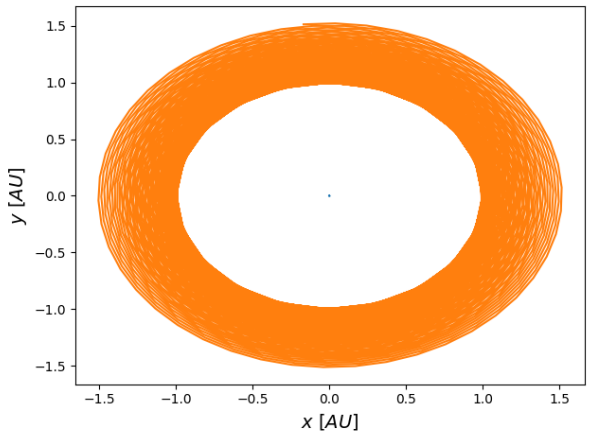


Figure 8. Earth-Sun system with alternative force model (B12) and circular orbits, using $\beta = 3$. Velocity Verlet algorithm. Time steps of $10^{-4}$ *years* and total time period of 100 *years*.

When we increase the mass of Jupiter by a factor of 10, as shown in figure (10), we see that Jupiter clearly has an impact on the Earth's orbit. The orbit of the Sun is also visually impacted by this. Increasing Jupiter's mass by a factor of 1000, as shown in figure (11), clearly disrupts the whole three body system. The mass of the Sun and Jupiter are then approximately equal, resulting in chaos.

Figure (12) shows the plot of the solar system including all planets and the Sun, excluding moons. Using the Velocity Verlet algorithm. The system is simulated over a period of 250 *years* in order for Pluto to complete one full orbit. Figure (13) shows the inner solar system,

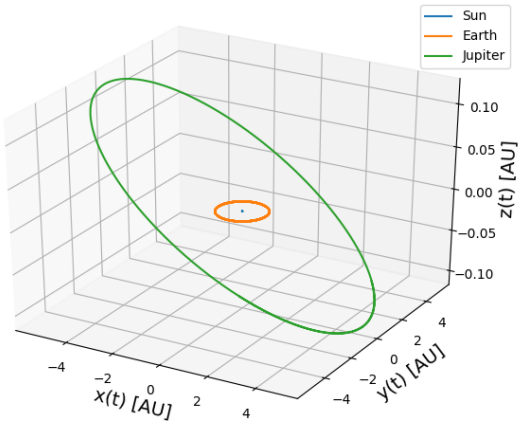


Figure 9. Three body system consisting of the Earth, the Sun and Jupiter. Using real data from NASA [4]. Solved using Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and time period of 15 *years*.

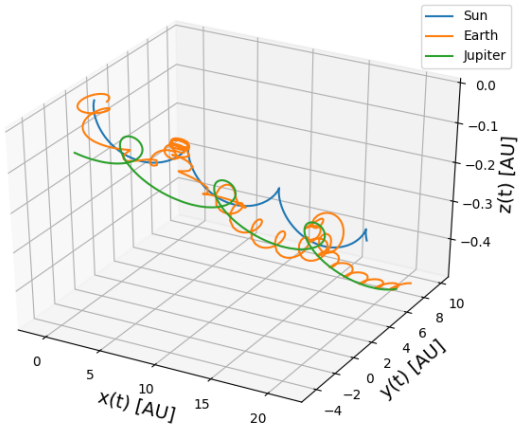


Figure 11. Three body system consisting of the Earth, the Sun and Jupiter with an increased mass by factor of 1000. Using real data from NASA [4]. Solved using Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and time period of 15 *years*.

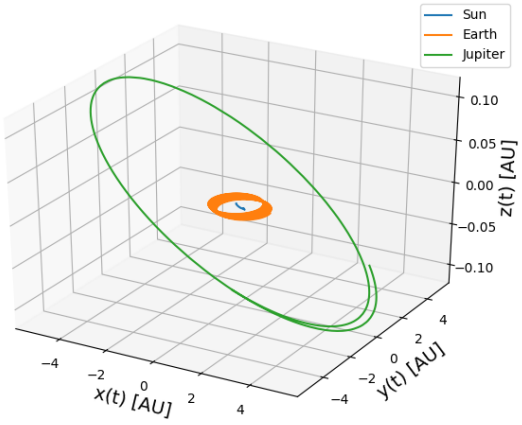


Figure 10. Three body system consisting of the Earth, the Sun and Jupiter with an increased mass by factor of 10. Using real data from NASA [4]. Solved using Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and time period of 15 *years*.

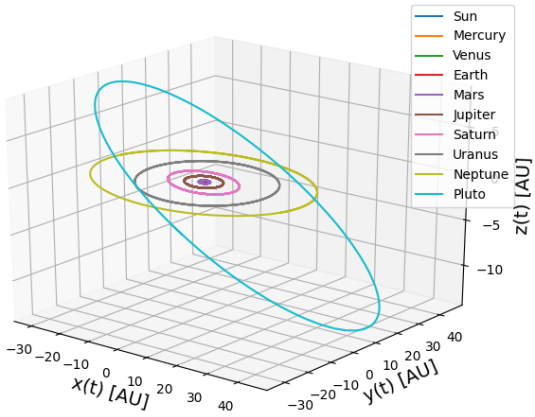


Figure 12. The solar system consisting of all planets and the Sun. Using real data from NASA [4] for initial positions and velocities. Solved using the Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and total period 250 *years*.

orbits are calculated over a period of 15 *years*.

Using the correction on Mercury's orbit from general relativity (B13), we have calculated the perihelion angle to be 0.7064289 radians. The classical prediction is 0.70644458 radians. Figure (14) shows the orbit of Mercury around the Sun over a century, when not including any other planets.

## IV. CONCLUSION

We have implemented and tested four ordinary differential equation solvers and compared their precision and stability upon what we expect from the theoretical background. These algorithms behave accordingly to our expectations when investigating properties like run-time (which reflects floating point operations) and stability (which reflects general error and energy conservation). The least accurate, but fastest method is the Euler's forward algorithm. Velocity Verlet and Euler-Cromer are approximately equal when comparing run-time, but the Verlet algorithm is overall more stable. The RK4 algorithm provides the best solution of them all, but at the expense of twice as long run-time as Verlet and

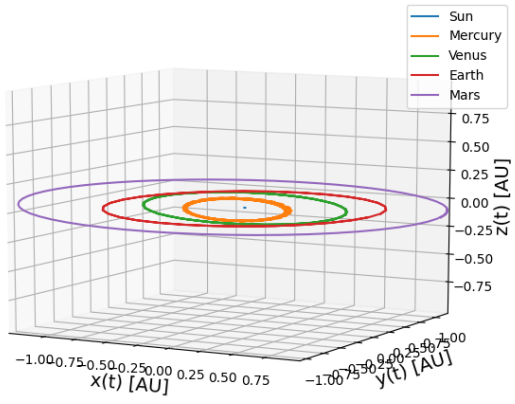


Figure 13. The inner solar system consisting of planets and the Sun. Using real data from NASA [4] for initial positions and velocities. Solved using the Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and total period 15 *years*.

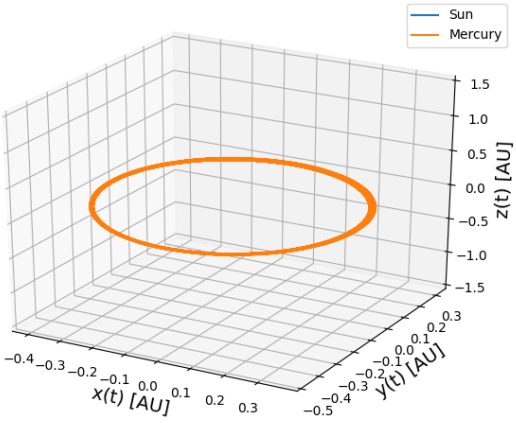


Figure 14. Mercury's orbit around the Sun, assuming no other planets. Using real data from NASA [4]. Solved using the Velocity Verlet algorithm with time steps of $10^{-4}$ *years* and total period 100 *years*.

Euler-Cromer. While RK4 is not a symplectic method, it still beats the energy conserving methods Verlet and Euler-Cromer on stability, even though the differences are small.

When considering the simple two body system consisting of the Earth and the Sun, we have seen that the theoretical expectation value (B11) of the escape velocity correspond well to the experimental result, as shown in figure (5). For the same system, we have seen that the alternative force model (B12) works surprisingly well compared to the Newtonian inverse-square law (B1). For $\beta \in [2, 2.95]$, there are almost no visual difference in the orbits. This is evident when comparing figures (1) and (7). For $\beta = 3$ however, the orbits become unstable due to weaker gravitational attraction. Over a period of 100 *years*, the approximately circular orbits have deviated from a radius of 1 *AU* to 1.5 *AU*, an increase of 50%.

For the three body system consisting of the Earth, Jupiter and the Sun, we have seen a justification for why we can assume a fixed position Sun as the center of mass. This is due to the fact that the Sun's mass are roughly 1000 times larger than that of Jupiter, therefore Jupiter, or any other planet in our solar system, have an insignificant impact on the center of mass (9). This would not be the case if Jupiter's mass where increased by a factor of 10, as figure (10) shows. If Jupiter and the Sun were to have roughly equal mass, the solar system would look very different. This is evident from figure (11).

The Velocity Verlet algorithm are able to produce stable orbits for all the planets in our solar system for a period of 250 *years*, using time steps of $10^{-4}$ *years*. This is just enough time for Pluto to complete one full orbit, as seen in figure (12). This was one of our main accomplishments - to produce a object-oriented numerical implementation that can calculate the orbits of every planet in our solar system. Using the theory of general relativity, we are able to use our model to calculate a more accurate value for the perihelion angle.

Further improvements could include the motion of various moons and asteroids, like the asteroid belt between Jupiter and the Earth. Since we were focusing on using object-orientation programming, we could have improved our model to automatically fetch data from NASA. Longer simulations in order to test our algorithms even further would have been beneficial, but proved to be impractical because of the lack of resources available. It would be interesting to have included an animation of the planetary orbits in time for the entire solar system.

## REFERENCES

[1] http://compphysics.github.io/ComputationalPhysics/doc/Projects/2020/Project3/html/Project3-bs.html - FYS3150 - Project 3, 2020. Morthen Hjorth-Jensen. October 21, 2020.

[2] https://github.com/Trrn13P/FYS3150_project_3 - FYS3150 - Project 3, 2020. GitHub project repository. Tor Andreas Bjone, Håvar Jacobsen and Håkon Jacobsen. October 21, 2020.

[3] http://compphysics.github.io/ComputationalPhysics/doc/pub/ode/html/ode.html - Computational

Physics Lectures: Ordinary differential equations. Morten Hjorth-Jensen. October 21, 2020.

[4] https://ssd.jpl.nasa.gov/horizons.cgi#top - NASA. Jet Propulsion Laboratory. Initial conditions. October 21, 2020.

[5] http://www.scholarpedia.org/article/N-body_simulations_(gravitational) - N-body simulations (gravitational). Timescales, Equilibrium and Collisionality. Michele Trenti and Piet Hut (2008). October 21 2020.

[6] http://arma.sourceforge.net - Armadillo C++ library. Version 10.1.0.

## Appendix A: Newton's second law

Introducing Newton's second law in classical mechanics. The second law states that the net force $F$ that acts on an object with mass $m$ is equal to the object's acceleration $a$ times the mass.

$$F = ma. \tag{A1}$$

The force and acceleration may vary in time $t$, $F = F(t)$ and $a = a(t)$. In this case, the acceleration is given as the double derivative of position $r$ with respect to $t$. Likewise the velocity $v(t)$ is the first derivative of position $r$ with respect to $t$. This reads

$$a(t) = \frac{d^2}{dt^2}r(t) = \frac{d}{dt}v(t) = r^{(2)}(t),$$

and

$$v(t) = \frac{d}{dt}r(t) = r^{(1)}(t).$$

The second law can therefor be written as

$$F(t) = mr^{(2)}(t).$$

Newton's second law is a second order (ordinary) differential equation. We have decoupled this into two first order differential equations above.

Applying this to an example. Considering an object with mass $m$ connected to a spring with stiffness $k$ (units $Newton/meter$). Motion along a straight line in three dimensions. Hook's law tells us that the spring force $F$ is proportional to the displacement $r$ from equilibrium. Position (displacement) $r$ is a function of time.

$$F(t) = -kr(t)$$

Assuming equilibrium at $r = 0$. Since $F$ is the net total force, this has equal Newton's second law (A1), giving

$$mr^{(2)}(t) = -kr(t),$$

or

$$r^{(2)}(t) = -\frac{k}{m}r(t).$$

We may define a oscillatory frequency (angular frequency), $\omega^2 = k/m$. This second order differential equation can then be solved through using numerical or analytical integration techniques. We know the analytical solution to be the equation of an harmonic oscillator, $r(t) = Acos(\omega t) + Bsin(\omega t)$ for some constants $A$ and $B$. The equation can be solved numerically using Euler's forward method (C) or Velocity Verlet method (D).

## Appendix B: The Solar System

Introducing Newton's gravitational law, scaling and applications to the $N-$ body problem for calculating planetary orbits. Introducing Kepler's second law, escape velocity, alternative force models and corrections using the theory of general relativity. Starting off with a simple two-body problem and gradually building on, we end up with a model for the solar system [1].

The gravitational law states that the force $F_G$ acting between two objects with mass $m_1$ and $m_2$ with a distance $r$ between them is given by

$$F_G = G\frac{m_1 m_2}{r^2}, \tag{B1}$$

where $G = 6.67 \cdot 10^{-11} Nm^2/kg$ is Newton's gravitational constant. Both masses will experience this attractive force. We may treat the masses as point particles. This force is conservative (the force only depends on position), meaning it conserves energy.

**The Earth-Sun-system**. In a hypothetical solar system with only the Earth orbiting around the Sun, we can apply equation (B1) to solve for the motion of the earth. This is a two body problem. The gravitational force then reads

$$F_G = G\frac{M_{Sun}M_{Earth}}{r^2}.$$

We assume the Sun to be fixed in the origin (neglecting its motion), since its mass is by far much larger than the mass of the Earth. In three dimensions, the position vector is given by $\vec{r} = (x, y, z)$ in Cartesian coordinates. Assuming the Earth's orbit to be co-planar. The net

force on the Earth is the gravitational pull from the Sun and applying Newton's second law (A1), the get three expressions for the acceleration $a$ of the Earth:

$$a_x = -\frac{d^2x}{dt^2} = -\frac{F_{G,x}}{M_{Earth}} = -\frac{GM_{Sun}}{r^3}x,$$

$$a_y = -\frac{d^2y}{dt^2} = -\frac{F_{G,y}}{M_{Earth}} = -\frac{GM_{Sun}}{r^3}y,$$

$$a_z = -\frac{d^2z}{dt^2} = -\frac{F_{G,z}}{M_{Earth}} = -\frac{GM_{Sun}}{r^3}z,$$

in the $x-$, $y-$ and $z-$direction respectively. The minus sign indicates that the gravitational pull on the Earth from the Sun acts towards the origin. The distance $r$ is given by

$$r = \sqrt{x^2 + y^2 + z^2}.$$

The components of the velocity vector $\vec{v} = (v_x, v_y, v_z)$ can be obtained by integrating the acceleration with respect to time, since $a = dv/dt$. Positions can be found similarly by integrating the velocities, since $v = dr/dt$. This results in six coupled first order differential equations.

When dealing with the solar system, its convenient to use astronomical units (AU) for distances and years as the unit for time. Therefor we want to scale our equations, getting rid off the constants $GM_{Sun}$. Assuming the Earth moves in a circular motion around the Sun (approximation). The acceleration will then be perpendicular to the velocity, pointing towards the center of mass (the Sun) at all times with constant magnitude. This is called centripetal acceleration:

$$a = \frac{v^2}{r} = \frac{F_G}{M_{Earth}} = \frac{GM_{Sun}}{r^2},$$

giving $v^2 r = GM_{sun}$. The velocity along a circumference (co-planar orbit) is given by

$$v = 2\pi r/T,$$

where $T$ is the orbiting-time period. In astronomical units, the distance $r$ from the Earth to the Sun is $r = 1\,AU$ and $T = 1\,year$, meaning that the Earth's velocity can be expressed as

$$v = 2\pi\,\frac{AU}{year}.$$

Our scaling parameter then reads

$$GM_{Sun} = 4\pi^2\,\frac{(AU)^3}{(year)^2}. \tag{B2}$$

We can express this simply as

$$GM_{Sun} = 4\pi^2,$$

with units of $(AU)^3/(year)^2$.

Applying this to the equations above for the acceleration components, we get

$$a_x = -\frac{4\pi^2 x}{(\sqrt{x^2 + y^2 + z^2})^3},$$

$$a_y = -\frac{4\pi^2 y}{(\sqrt{x^2 + y^2 + z^2})^3},$$

$$a_z = -\frac{4\pi^2 z}{(\sqrt{x^2 + y^2 + z^2})^3},$$

in units of $AU/(year)^2$. On vector form, the Earth's acceleration can be written as

$$\vec{a} = (a_x, a_y, a_z) = -\frac{4\pi^2}{r^2}\hat{r},$$

using $\hat{r} = \vec{r}/r$ is the unit position vector pointing from the origin (Sun) to the Earth.

**By including Jupiter**, we have a three body problem. Using the superposition principle, the Earth's acceleration (A1) can be calculated by adding up the gravitational forces from the Sun and from Jupiter, using equation (B1) respectively. Figure (15) illustrates the three body system consisting of the Sun, Earth and Jupiter in a reference system relative to the center of the Sun.

Denoting the gravitational force between the Earth and Jupiter $F^{EJ}$, this reads

$$F_{EJ} = G\frac{M_J M_E}{r_{EJ}^2},$$

where $M_J = M_{Jupiter}$ and $M_E = M_{Earth}$. The position vector between the Earth and Jupiter is $r_{EJ} = r_E - r_J$, where $r_E$ and $r_J$ are the positions of the Earth and Jupiter relative to the Sun, respectively.

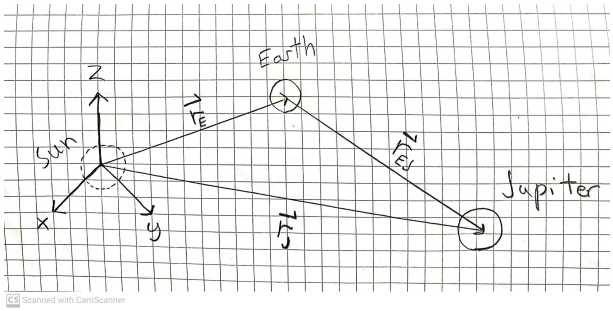


Figure 15. A system consisting of three planets; the Earth and Jupiter orbiting the Sun. Coordinates relative to the Sun.

The gravitational force between the Earth and the Sun is as before. We denote this by

$$F_E = G\frac{M_S M_E}{r_E^2},$$

where $M_S = M_{Sun}$. Similarly for Jupiter, we have

$$F_J = G\frac{M_S M_J}{r_J^2}.$$

Using Newton's second law (A1) and the superposition principle, we find the acceleration of the Earth:

$$\vec{a}_E = \frac{\vec{F}_E + \vec{F}_{EJ}}{M_E} = -G\frac{M_S}{r_E^2}\hat{r}_E - G\frac{M_J}{r_{EJ}^2}\hat{r}_{EJ}.$$

Using that $GM_S = 4\pi^2$, we can rewrite $GM_J$ as

$$GM_J = GM_S\frac{M_J}{M_S} = 4\pi^2\frac{M_J}{M_S}.$$

The acceleration of the Earth can then be written as

$$\vec{a}_E = -\frac{4\pi^2}{r_E^2}\hat{r}_E - \frac{4\pi^2(M_J/M_S)}{r_{EJ}^2}\hat{r}_{EJ}.$$

Where

$$\vec{r}_E = (x_E, y_E, z_E),$$

and

$$\vec{r}_{EJ} = (x_E - x_J, y_E - y_J, z_E - z_J).$$

The components of the acceleration vector in the $x-$, $y-$ and $z-$ direction are obtained generally by multiplying $\vec{a}$ with the unit vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$. For the Earth, its acceleration in $x-$ direction reads

$$a_{E,x} = -\frac{4\pi^2}{r_E^3}x_E - \frac{4\pi^2(M_J/M_S)}{r_{EJ}^3}x_{EJ}.$$

This can be generalized for a $k \in \{x, y, z\}$:

$$a_{E,k} = -\frac{4\pi^2}{r_E^3}k_E - \frac{4\pi^2(M_J/M_S)}{r_{EJ}^3}k_{EJ}.$$

For Jupiter, the acceleration vector becomes

$$\vec{a}_J = -\frac{4\pi^2}{r_J^2}\hat{r}_J - \frac{4\pi^2(M_E/M_S)}{r_{JE}^2}\hat{r}_{JE},$$

with $\vec{r}_J = (x_J, y_J, z_J)$ and $\vec{r}_{JE} = -\vec{r}_{EJ}$. Decomposing the vector is similar to that of the Earth, for a $k \in \{x, y, z\}$:

$$a_{J,k} = -\frac{4\pi^2}{r_J^3}k_J - \frac{4\pi^2(M_E/M_S)}{r_{JE}^3}k_{JE}.$$

**For a $N$-body problem**, we might as well include the motion of the Sun. Given a collection of $N$ planets including the Sun, we can calculate the acceleration of a planet $i$, for $i = 1, 2, ..., N$ by adding up the contributions from all the other planets, this reads

$$\vec{a}_i = \sum_{j=1,\, j\neq i}^{N} \frac{4\pi^2}{r_{ij}^2}\left(\frac{M_j}{M_S}\right)\hat{r}_{ji}, \tag{B3}$$

with components for a $k \in \{x, y, z\}$:

$$a_{i,k} = \sum_{j=1,\, j\neq i}^{N} \frac{4\pi^2}{r_{ij}^3}\left(\frac{M_j}{M_S}\right)k_{ji},$$

The acceleration has units of $AU/(year)^2$ because of the scaling we have chosen (B2).

Using our own solar system, we can obtain masses and relative distances for each of the planets from table (I). Using equation (B3), and any of the methods discussed in Appendix (C), (D) or (E), we can calculate velocities and positions for each planet over a given time period. Using the NASA site for initial conditions [4], we obtain values for positions and velocities such that energy, momentum and angular momentum are conserved in the system.

**The total mechanical energy** $E = K + V$ is conserved for closed planetary systems like our solar system. $K$ is the kinetic energy and $V$ is the potential energy. The total kinetic energy for a $N$-body system is given by

the sum of all the kinetic energies for each planet, this reads

$$K = \sum_{i}^{N} m_i v_i^2, \tag{B4}$$

where $m_i$ is the mass and $v_i$ is the velocity. Each planet has a kinetic energy

$$K_i = \frac{1}{2} m v_i^2. \tag{B5}$$

For a system in dynamical equilibrium (constant inertia), the Viral Theorem states [5] that the total energy potential energy is $V = 2E = -2K$, and $K = E$. The potential energy is given by

$$V = -\frac{G}{2} \sum_{i}^{N} \sum_{j=1,\, j\neq i}^{N} \frac{m_j m_i}{|\vec{r}_j - \vec{r}_i|}. \tag{B6}$$

The potential energy for one planet in the $N-$body system is given by

$$V_i = -G m_i \sum_{j=1,\, j\neq i}^{N} \frac{m_j}{|\vec{r}_j - \vec{r}_i|}. \tag{B7}$$

Table I. Masses and relative distances for planets in our solar system [1]. $1\ AU = 1.5 \cdot 10^{11} m$. The mass of the Sun is approximately $M_{Sun} = 2 \cdot 10^{30} kg$.

| Planet | Mass - $[kg]$ | Distance - $[AU]$ |
|---|---|---|
| Earth | $6 \cdot 10^{24}$ | 1 |
| Jupiter | $1.9 \cdot 10^{27}$ | 5.20 |
| Mars | $6.6 \cdot 10^{23}$ | 1.52 |
| Venus | $4.9 \cdot 10^{24}$ | 0.72 |
| Saturn | $5.5 \cdot 10^{26}$ | 9.54 |
| Mercury | $3.3 \cdot 10^{23}$ | 0.39 |
| Uranus | $8.8 \cdot 10^{25}$ | 19.19 |
| Neptun | $1.03 \cdot 10^{26}$ | 30.06 |
| Pluto | $1.31 \cdot 10^{22}$ | 39.53 |

**Kepler's second law** tells us that planets orbiting the Sun sweeps out equal areas in equal time intervals. This is illustrated in figure (16) for an elliptical orbit. If $\Delta T$ is the same, then $A_1 = A_2$. This means that the planet moves slower when its distance to the Sun is greater. Kepler's second law actually tells us that angular momentum is conserved for planets moving in elliptical orbits around the Sun (B10). Since

$$A_1 = A_2,$$

for equal times $\Delta T$, we have that

$$\frac{dA}{dt} = constant. \tag{B8}$$

Generally angular momentum $L$ is given by

$$\vec{L} = m(\vec{r} \times \vec{v}), \tag{B9}$$

where $m$ is the mass of the object, $r$ is the distance and $v$ is the velocity. Assuming that $v$ can be decomposed in orthogonal components as

$$\vec{v} = \vec{v}_r + \vec{v}_\theta = v_r \hat{r} + v_\theta \hat{\theta},$$

Where $v_r$ is the velocity parallel to the straight line radial distance $r$ and $v_\theta$ is the velocity in the direction of the angle $\theta$ (angular velocity direction), perpendicular to $r$. We see that

$$\vec{L} = m(\vec{r} \times (\vec{v}_r + \vec{v}_\theta)),$$

where $\vec{r} \times \vec{v}_r = \vec{0}$. Skipping the vector notation, the angular momentum can then be written as

$$L = m r v_\theta.$$

Looking at figure (16), we can approximate the area swept for a small change in angle $d\theta$ as a triangle with an area $dA$, given as

$$dA = \frac{1}{2} r \cdot r d\theta = \frac{1}{2} r^2 d\theta.$$

Kepler's second law (B8) tells us that $dA/dt = constant$. We divide the above equation with $dt$ on both sides to obtain

$$\frac{dA}{dt} = \frac{1}{2} r^2 \frac{d\theta}{dt},$$

where

$$\frac{d\theta}{dt} = \frac{v_\theta}{r},$$

therefor

$$\frac{dA}{dt} = \frac{1}{2} r v_\theta = \frac{1}{2} \frac{L}{m}. \tag{B10}$$

This result tells us that angular momentum (B9) is conserved for elliptical planetary orbits around the Sun in our solar system.

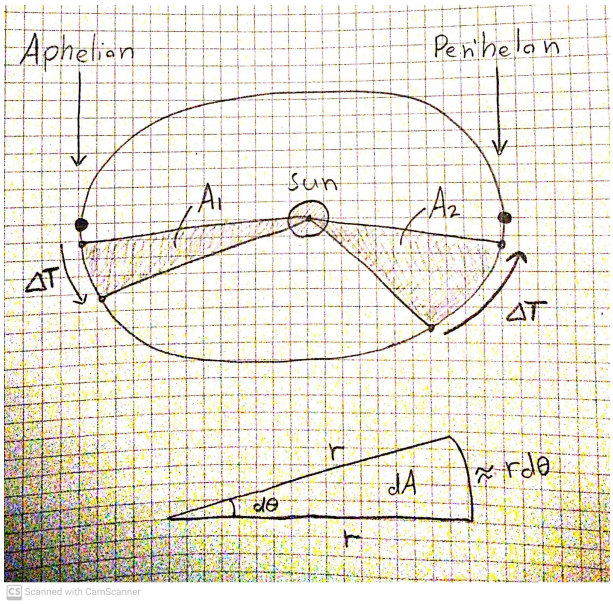


Figure 16. Kepler's second law: Equal areas in equal time intervals, also known as conservation of angular momentum.

**Escape velocity** is the velocity needed for an object to escape the gravitational field. We can calculate the escape velocity for a planet in a $N-$body system by setting its kinetic energy (B5) equal to the potential energy (B7), and solving for $v$.

In the binary Earth-Sun system, we can calculate the Earth's escape velocity by

$$\frac{1}{2}M_E v^2 = -G\frac{M_S M_E}{r},$$

$$v_{escape} = \sqrt{\frac{2GM_S}{r}}. \tag{B11}$$

Using that $GM_S = 4\pi^2$ with the scaling (B2) and $r = 1\ AU$, the escape velocity is

$$v_{escape} = \sqrt{2}\,2\pi,$$

in units of $AU/year$. For our solar system, this could be a good approximation for the actual escape velocity when including all the planets because the Sun contributes the most to the gravitational potential field.

**Alternative force models**. We can replace the inverse-square force (B1) for the Earth-Sun system by

$$F_G = G\frac{M_{Sun}M_{Earth}}{r^\beta}, \tag{B12}$$

for $\beta \in [2,3]$ and use this to determine by what amount Nature deviates from a perfect inverse-square law, such as Newton's gravitational law.

**General relativity**. Using the theory of general relativity we can correct Mercury's orbit. The observed value of Mercury's perihelion precession on does not correspond exactly to the classical Newtonian inverse-square law prediction. Closed elliptical orbits are a special feature of the inverse-square law. Generally any correction will lead to an orbit which is not closed. If the correction is small, each orbit around the Sun will be almost the as as the classical ellipse, and the orbit can be thought of as an ellipse whose orientation in space slowly rotates. The perihelion of the ellipse slowly precesses around the Sun. Adding a general relativistic correction to Newton's gravitational law (B1), the force becomes

$$F_G = G\frac{M_{Sun}M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right], \tag{B13}$$

where $l = |\vec{r}\times\vec{v}|$ is the magnitude of Mercury's orbital angular momentum per unit mass, $l = L/m$, and $c$ is the speed of light in vacuum. The perihelion angle $\theta_p$ can be calculated using

$$tan\theta_p = \frac{y_p}{x_p},$$

where $(x_p, y_p)$ is the position of Mercury at perihelion, the point which is closest to the Sun. The speed of Mercury at perihelion is 12.44 $AU/year$ and the distance to the perihelion from the Sun is 0.3075 $AU$.

## Appendix C: Euler's method

Introducing Euler's forward method for solving coupled first order differential equations. This is an example of a method which does not conserve energy. [3].

When we discretize a function $x(t)$, from $t = t_0$ to $t = t_n$ we can define a time step

$$h = \frac{t_n - t_0}{n},$$

such that $t_i = t_0 + ih$. The function $x(t)$ could be a position as function of time. In that case its derivative with respect to time is the velocity $v(t)$. If we were to Taylor expand $x(t_i + h)$, we use $t_i$ as a reference point close to $t_i + h$, since $h$ is a small step in time.

$$x(t_i + h) = x(t_i) + hx^{(1)}(t_i) + \frac{h^2}{2}x^{(2)}(t_i) + O(h^3), \tag{C1}$$

similarly for the velocity

$$v(t_i + h) = x(t_i) + hv^{(1)}(t_i) + \frac{h^2}{2}v^{(2)}(t_i) + O(h^3), \quad \text{(C2)}$$

where $O(h^3)$ is the error-term, or the missing part when approximating. In **Euler's forward method**, we use only the first order terms in the Taylor expansion, giving

$$x(t_i + h) \approx x(t_i) + hx^{(1)}(t_i) = x(t_i) + hv(t_i),$$

and

$$v(t_i + h) \approx v(t_i) + hv^{(1)}(t_i) = v(t_i) + ha,$$

where $a$ is the acceleration, the first order derivative of velocity with respect to time. The acceleration may depend on more or other variables than just $t$. For conservative forces like gravity (B), we have that $a = a(t, x)$. In other cases we might have $a = a(t, x, v)$, which is non-conservative because of the velocity-dependency. Euler's forward method does not conserve energy however, because the method is not symplectic. The **Euler-Cromer method** is a slightly improved version that does conserve energy:

$$v(t_i + h) \approx v(t_i) + ha,$$

and

$$x(t_i) \approx x(t_i) + hv(t_i + h).$$

Both of these methods have a local error that goes like $O(h^2)$. When the algorithm is repeated $n$ times, the global error is goes like $O(h)$ because $n \sim h^{-1}$.

Introducing a simpler and more compact form of notation for numerical purposes: $x(t_i + h) = x_{i+1}$. Summarised, **Euler's forward method** then reads:

$$x_{i+1} = x_i + hv_i, \quad \text{(C3)}$$

and

$$v_{i+1} = v_i + ha_i. \quad \text{(C4)}$$

## Appendix D: Velocity Verlet method

Introducing the Velocity Verlet method. This is an example of an energy conserving method (symplectic). [3].

The Velocity Verlet method provides a good balance between floating point operations and accuracy. It is also symplectic, meaning it conserves energy. This is useful when dealing with conservative forces like gravity in classical mechanics (B). Using the Taylor expansions (C1) and (C2), and the compact notation $x(t_i + h) = x_{i+1}$, we start off at

$$x_{i+1} = x_i + hx_i^{(1)} + \frac{h^2}{2}x_i^{(2)},$$

and

$$v_{i+1} = v_i + hv_i^{(1)} + \frac{h^2}{2}v_i^{(2)},$$

with $x_i^{(2)} = v_i^{(1)} = a_i = F(t_i, x_i)/m$ given by Newton's second law (A1). The term $v_i^{(2)}$ is unknown. We use the Forward Euler's method (C4) to determine this:

$$v_i^{(2)} = \frac{v_{i+1}^{(1)} - v_i^{(1)}}{h},$$

$$hv_i^{(2)} = v_{i+1}^{(1)} - v_i^{(1)} = a_{i+1} - a_i,$$

this yields

$$v_{i+1} = v_i + ha_i + \frac{h}{2}(a_{i+1} - a_i),$$

$$v_{i+1} = v_i + \frac{h}{2}(a_{i+1} + a_i).$$

The **Velocity Verlet algorithm** then goes like this: First step is to calculate the forward position:

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i,$$

with $a_i = F(x_i)/m$ for conservative forces. Non-conservative forces may have other dependencies, but the general algorithm is the same. Next intermediate step is to update the acceleration:

$$a_{i+1} = F(x_{i+1})/m,$$

then finally the forward velocity:

$$v_{i+1} = v_i + \frac{h}{2}(a_{i+1} + a_i).$$

Since both the position $x_{i+1}$ and velocity $v_{i+1}$ are calculated using second order terms from the Taylor expansion, the local error in both goes like $O(h^3)$. The global error goes like $O(h^2)$.

### Appendix E: Runge-Kutta 4 method

Introducing the fourth-order Runge-Kutta (RK4) method for solving coupled first order differential equations. This method does not conserve energy (not symplectic). It is still far more accurate than the Forward Euler's method discussed in Appendix (C) at the cost of more floating point operations. [3].

We assume that the acceleration $a$ is given. This is the function which we want to integrate in order to find velocities and positions. It could for example be the $N-$body planetary system where the acceleration $a(t,x)$ of each planet is given by (B3), and we wish to calculate the velocities and positions for all the planets. The algorithm then goes like this:

We calculate different slopes for straight line to approximate. These slopes are given as $k_j$ for $j = 1,2,3,4$ like showed below. Our step size is $h = (r_n - r_0)/n$ and we integrate over time $t$ in steps such that $t_i = t_0 + h$ and $i = 1,2,3,...,n$ for $n$ integration steps. The acceleration only depends on position $r(t)$.

$$k_1 = ha(t_i, r(t_i)) = ha(t_i, r_i) = ha(r_i),$$

$$k_2 = ha(t_i + h/2, r_i + k_1/2) = ha(r_i + hk_1/2),$$

$$k_3 = ha(t_i + h/2, r_i + k_2/2) = ha(r_i + hk_2/2),$$

$$k_4 = ha(t_i + h, r_i + k_3) = ha(r_i + hk_3),$$

then we calculate the velocity forward in time as

$$v_{i+1} = v_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

We calculate similar slopes in the velocity function:

$$k_1 = v_i,$$

$$k_2 = v_i + hk_1/2,$$

$$k_3 = v_i + hk_2/2,$$

$$k_4 = v_i + hk_3,$$

$$r_{i+1} = r_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

The global error for the RK4 method goes like $O(h^4)$, meaning it has higher accuracy than the Velocity Verlet method which has an error that goes like $O(h^2)$. Disadvantages however is that RK4 obviously requires a greater amount of floating point operations and that it does not conserve energy (which can become a problem for large time simulations). The RK4 method can prove better if the possible amount of integration points are limited.