Numerically Modeling Gravitationally Bound Systems

Noah Green
Michigan State University

April 1, 2016

Abstract

The RK4 and Verlet algorithms for calculating the solutions to ordinary differential equations are used here to calculate the orbits of various gravitationally bound systems resembling the solar system. It was found that the RK4 algorithm is more accurate, but the Verlet algorithm is easier to implement. Additionally, systems that have many bodies that move on different timescales become difficult to model, requiring many more time steps to accurately model objects moving on a short timescale (e.g. Mercury) than is necessary to accurately model objects moving on a long timescale (e.g. Pluto). This was remedied in a model solar system by treating the inner and outer solar systems separately, and ignoring the negligible effect of the gravitational force due to the first four planets on the outer four planets and Pluto.

1 Introduction

Modeling the gravitational force has many applications, from space travel to astronomy. I will discuss how to use the Runge-Kutta and Verlet algorithms to numerically calculate the motion of model solar systems due to classical gravitational interactions. In section 2, I cover the equations used to model gravitational interactions. In section 3, I discuss how these equations can be solved numerically. In section 4, I show these algorithms applied in different situations. Finally, in section 5, I summarize the conclusions of this study.

2 Differential Equations

The force due to gravity between two objects can be calculated using Newton's law of gravitation:

$$F_G = \frac{GMm}{r^2} \tag{1}$$

where G is the universal gravitational constant, M and m are the masses of the two object, and r is the distance between them. Newton's second law, $F = ma$ gives us that

$$\frac{d^2x}{dt^2} = \frac{F_{G,m,x}}{m} \tag{2}$$

with similar equations for y and z. In order to use the algorithms, these equations must be rewritten as a set of coupled first order ordinary differential equations. For a planet orbiting the sun, these equations are

$$\frac{dv_x}{dt} = -\frac{GM_\odot}{r^3}x, \tag{3}$$

$$\frac{dx}{dt} = v_x, \tag{4}$$

for the x coordinate, with similar equations for y and z coordinates.

3 Algorithms

Both the Verlet and Runge-Kutta methods are based on Taylor expanding the ODE for which we want to solve (e.g. equations (3) and (4)), discretizing the function in question, then using some trick to step through the discrete solutions based on initial conditions. Their derivation can be found in references [3] and [2]. I will cover how they can be used algorithmically.

3.1 RK4

In addition to a Taylor expansion, the Runge-Kutta method also utilizes midpoint approximations of the integral of a differential equation. Hence, different orders of this approximation, using any number of midpoints, can be used to get varying degrees of accuracy. The RK4 method uses two midpoints and the endpoints of each discrete interval. For a first order differential equation $\frac{dy}{dx} = f(t, y)$, we have

$$k_1 = hf(t_i, y_i) \quad k_2 = hf(t_i + h/2, y_i + k_1/2)$$

$$k_3 = hf(t_i + h/2, y_i + k_2/2) \quad k_4 = hf(t_i + h, y_i + k_3)$$

to get the next value in the discretized solution

$$y_{i+1} = y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{5}$$

Applying this to equations (3) and (4), first note that the right side of equation (3) can be represented for a planet with mass m as $F_x(r)/m$. First calculate k_1

$$x : k_1 = hv_{i-1} \quad v : k_1 = hF_x(r_i)/m. \tag{6}$$

where h is the discrete time step. For the positional k_2, k_3, and k_4, simply add $k_1/2$, $k_2/2$, or k_3 respectively to the i^{th} position. The velocity k_2, k_3, and k_4 then come automatically by recalculating the force at the new position and multiplying it by h/m. The complete code that I used can be found in appendix D, but the abridged version follows:

```
for( int i = 1; i <= _Ntime; i++ ){
 vector< vector< double > > k1, k2, k3;
 for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){

 ... Calculate k1, k2, k3, k4...

        _vPlanets.at(iPlanet).SetX(_vPlanets.at(iPlanet).X(i-1)
                                  +(k1.at(iPlanet).at(0)
                                  +2.*k2.at(iPlanet).at(0)
                                  +2.*k3.at(iPlanet).at(0)
                                    +k4i.at(0))/6.,i);
    _vPlanets.at(iPlanet).SetY(_vPlanets.at(iPlanet).Y(i-1)
                                  +(k1.at(iPlanet).at(1)
                                  +2.*k2.at(iPlanet).at(1)
                                  +2.*k3.at(iPlanet).at(1)
                                    +k4i.at(1))/6.,i);
    _vPlanets.at(iPlanet).SetZ(_vPlanets.at(iPlanet).Z(i-1)
                                  +(k1.at(iPlanet).at(2)
```

```
                                    +2.*k2.at(iPlanet).at(2)
                                    +2.*k3.at(iPlanet).at(2)
                                      +k4i.at(2))/6.,i);
        _vPlanets.at(iPlanet).SetVx(_vPlanets.at(iPlanet).Vx(i-1)
                                     +(k1.at(iPlanet).at(3)
                                     +2.*k2.at(iPlanet).at(3)
                                     +2.*k3.at(iPlanet).at(3)
                                       +k4i.at(3))/6.,i);
        _vPlanets.at(iPlanet).SetVy(_vPlanets.at(iPlanet).Vy(i-1)
                                     +(k1.at(iPlanet).at(4)
                                     +2.*k2.at(iPlanet).at(4)
                                     +2.*k3.at(iPlanet).at(4)
                                       +k4i.at(4))/6.,i);
        _vPlanets.at(iPlanet).SetVz(_vPlanets.at(iPlanet).Vz(i-1)
                                     +(k1.at(iPlanet).at(5)
                                     +2.*k2.at(iPlanet).at(5)
                                     +2.*k3.at(iPlanet).at(5)
                                       +k4i.at(5))/6.,i);
      }
    }
  }
```

This algorithm is the more accurate of the two covered in this paper, with a global error of $O(h^4)$.

3.2 Verlet

The velocity Verlet method is more straightforward, but goes like $O(h^3)$. The equations to find the next discrete solution value are as follows.

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i, \tag{7}$$

and

$$v_{i+1} = v_i + \frac{h}{2}\left(a_{i+1} + a_i\right). \tag{8}$$

where a denotes the acceleration. The trickiest part of this algorithm is that the position needs to be calculated first. This lets us get a_{i+1} by calculating the force from the new position and dividing by the planet's mass. I implemented this algorithm in C++ as follows:

```
void SolarSystem::Verlet(){
  _solved = true;
  // make force variables in this scope so they persist through loops
  vector<double> Fx,Fy,Fz;
  vector<double> fx,fy,fz;
  double f_x,f_y,f_z;
  for( unsigned int fi = 0; fi < GetN(); fi++ ){
    TotalForce(fi,0,f_x,f_y,f_z);
    Fx.push_back(f_x);
    Fy.push_back(f_y);
    Fz.push_back(f_z);
    fx.push_back(f_x);
    fy.push_back(f_y);
    fz.push_back(f_z);
  }

  for( int i = 1; i <= _Ntime; i++){
    for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){
      if(_vPlanets.at(iPlanet).Fixed()){
        _vPlanets.at(iPlanet).AddCoordinates(i*_step,
                                             _vPlanets.at(iPlanet).X(i-1),
```

```
                                        _vPlanets.at(iPlanet).Y(i-1),
                                        _vPlanets.at(iPlanet).Z(i-1),
                                        _vPlanets.at(iPlanet).Vx(i-1),
                                        _vPlanets.at(iPlanet).Vy(i-1),
                                        _vPlanets.at(iPlanet).Vz(i-1));
  }
  else{
    // advance spatial coordinates by one time step
    // set velocity to zero for now
    fx.at(iPlanet) = Fx.at(iPlanet);
    fy.at(iPlanet) = Fy.at(iPlanet);
    fz.at(iPlanet) = Fz.at(iPlanet);
    _vPlanets.at(iPlanet).AddCoordinates(i*_step,
                                        _vPlanets.at(iPlanet).X(i-1)+
                                        _step*_vPlanets.at(iPlanet).Vx(i-1)+
                                        _step*_step*fx.at(iPlanet)/(2.*_vPlanets.at(i
                                        _vPlanets.at(iPlanet).Y(i-1)+
                                        _step*_vPlanets.at(iPlanet).Vy(i-1)+
                                        _step*_step*fy.at(iPlanet)/(2.*_vPlanets.at(i
                                        _vPlanets.at(iPlanet).Z(i-1)+
                                        _step*_vPlanets.at(iPlanet).Vz(i-1)+
                                        _step*_step*fz.at(iPlanet)/(2.*_vPlanets.at(i
                                        0.,0.,0.);
  }
}
for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++){
```

```
      if( !_vPlanets.at(iPlanet).Fixed() ){
        // calculate force using new position coordinates
        TotalForce(iPlanet,i,Fx.at(iPlanet),Fy.at(iPlanet),Fz.at(iPlanet));
        // advance velocity coordinates by one step
        _vPlanets.at(iPlanet).SetVx(_vPlanets.at(iPlanet).Vx(i-1)+
                                    _step*(fx.at(iPlanet)+Fx.at(iPlanet))/(2.*_vPlanets.at
        _vPlanets.at(iPlanet).SetVy(_vPlanets.at(iPlanet).Vy(i-1)+
                                    _step*(fy.at(iPlanet)+Fy.at(iPlanet))/(2.*_vPlanets.at
        _vPlanets.at(iPlanet).SetVz(_vPlanets.at(iPlanet).Vz(i-1)+
                                    _step*(fz.at(iPlanet)+Fz.at(iPlanet))/(2.*_vPlanets.at


      }
    }
  }
}
```

Note that the equations are for the x components of the position and velocity only. As can be seen in the code, the other coordinates can be found using analogous equations.

4 The Solar System

Armed with these algorithms, I was able to model several different systems. The simplest case is the Earth-Sun system, ignoring the other planets, assuming the sun is stationary and the orbit of Earth is circular. In the next case, I determined the velocity needed for a planet starting at 1 AU from the sun to escape from the solar

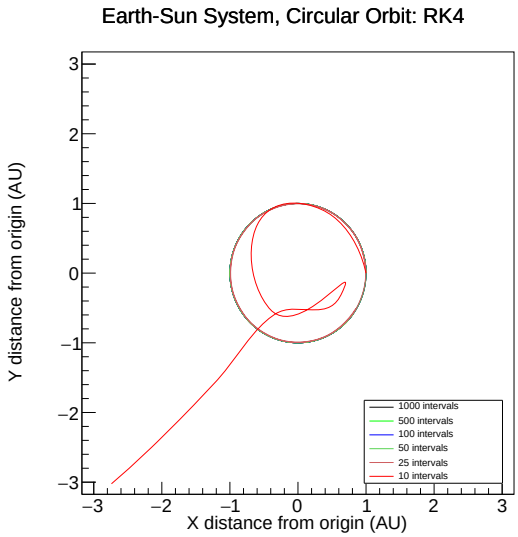

Figure 1: Plot of Earth's position in Earth-Sun system for different numbers of time steps over 1.9 years using the RK4 algorithm.

system. By adding the forces on each planet due to all other bodies component-wise, I am then able to move onto three and more bodied systems culminating with all major bodies of the solar system. Note that for all calculations, I used the units of solar masses ($M_\odot$) for mass, AU (mean distance from Earth to the Sun) for distance, and years for time. For easier comparison, I have placed many of the resulting plots in the appendix.

4.1 Earth-Sun only

Figures 1 and 2 were calculated so that Earth has a circular orbit. In appendix A, additional plots of the angular momentum and kinetic and potential energy of the system are included.

Note that the orbits in the Verlet plot of figure 2 begin to noticeably diverge at 50 intervals, while the RK4 algorithm doesn't seem to diverge until somewhere between 10 and 25 intervals. Additionally, the kinetic energy, potential energy, and

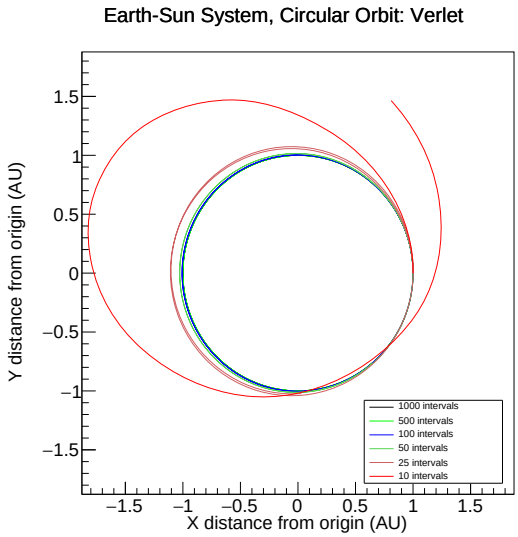

Figure 2: Plot of Earth's position in Earth-Sun system for different numbers of time steps over 1.9 years using the Verlet algorithm.

the angular momentum all converge to being constant at higher numbers of intervals. This is expected because a circular orbit has constant velocity and is at a constant distance from the sun.

4.2 Escape Velocity

4.3 Three Body System: Stationary Sun

4.4 Three Body System: Dynamic Sun

4.5 All Major Solar System Bodies

5 Conclusion

Both algorithms give good approximations of orbits in single and many body cases given sufficient time steps. The RK4 method is more exact that the Verlet method at the expense of being more difficult to utilize and some extra floating point operations

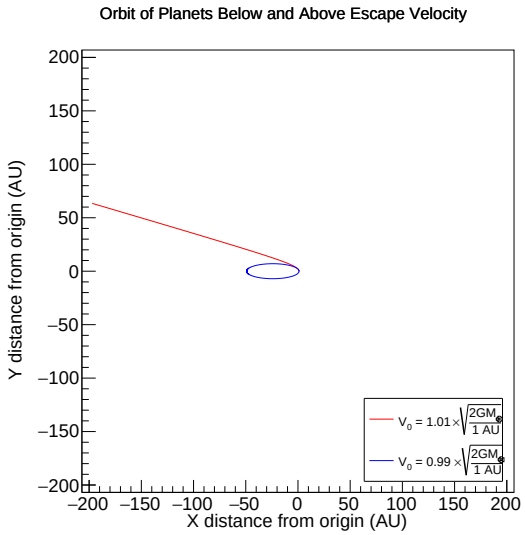

Figure 3: Plot of planet with starting position of 1 AU and starting mass of one Earth mass with initial velocity just above and below escape velocity. Orbital period found by trial and error to be 126 years for the planet just below escape velocity. 20000 time steps were used for accurate orbit modeling.

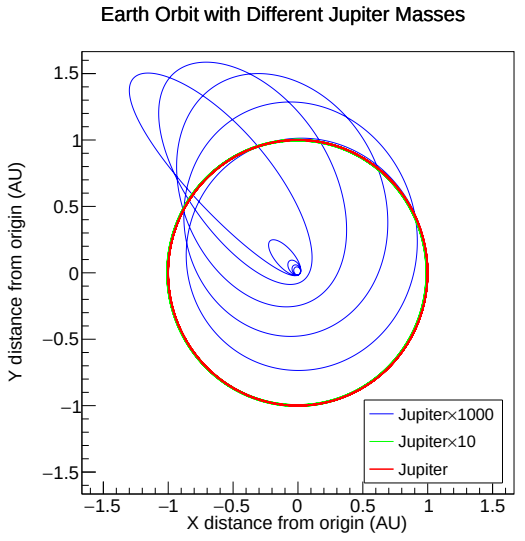

Figure 4: Plot of the position of earth in three body system with stationary sun and different Jupiter masses. Used a time length of 13 years with 13000 time steps with the RK4 algorithm.

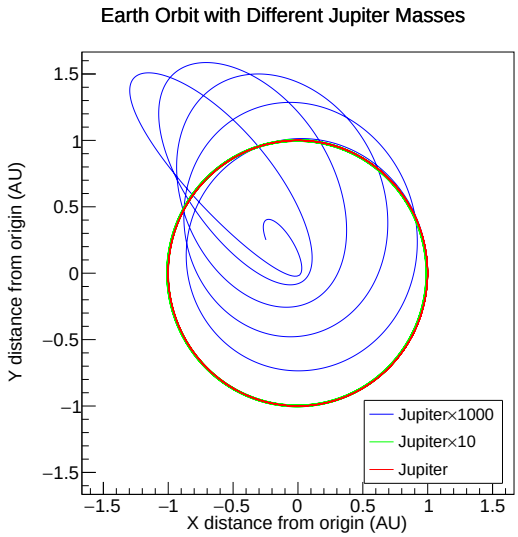

Figure 5: Plot of the position of earth in three body system with stationary sun and different Jupiter masses. Used a time length of 13 years with 13000 time steps with the Verlet algorithm.

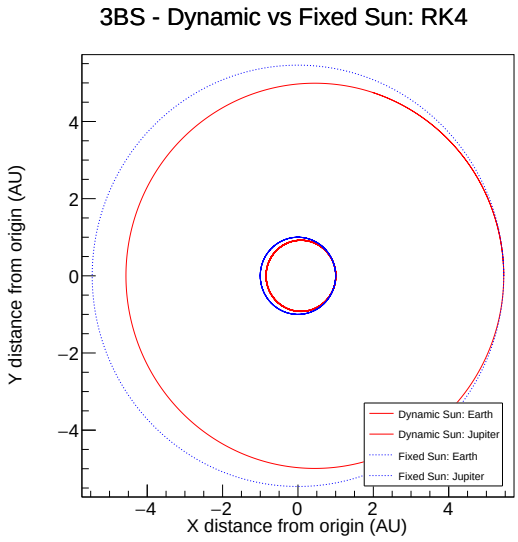

Figure 6: Plot comparing 3 body systems (Earth, Jupiter, and the Sun) with a stationary sun vs a system revolving around a common center of mass. Note the mass of the sun in the dynamic case was increased by 10% to increase the visual effect. Time: 13 years, Time Steps:13000, Algorithm: RK4

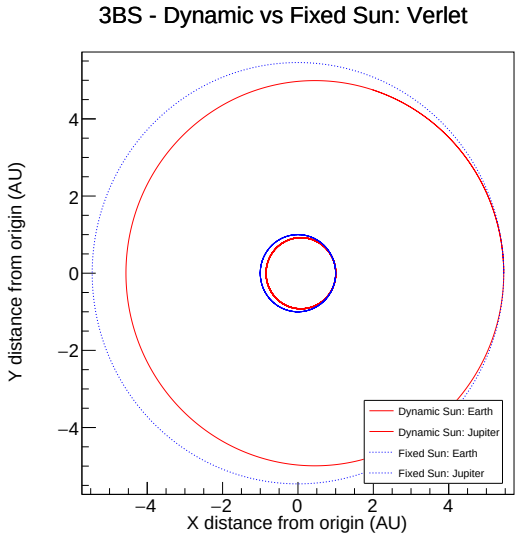

Figure 7: Plot comparing 3 body systems (Earth, Jupiter, and the Sun) with a stationary sun vs a system revolving around a common center of mass. Note the mass of the sun in the dynamic case was increased by 10% to increase the visual effect. Time: 13 years, Time Steps:13000, Algorithm: Verlet

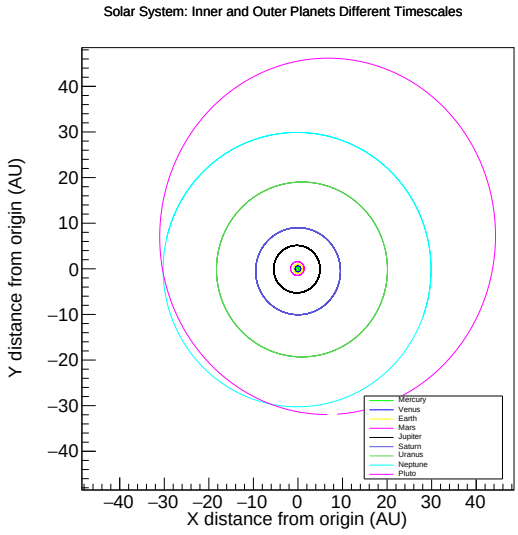

Figure 8: Plot of all major bodies with inner and outer planets calculated separately. Inner Planets - Time: 5 years, Time Steps: 5000, Algorithm: RK4. Outer Planets - Time: 250 years, Time Steps: 25000, Algorithm: RK4

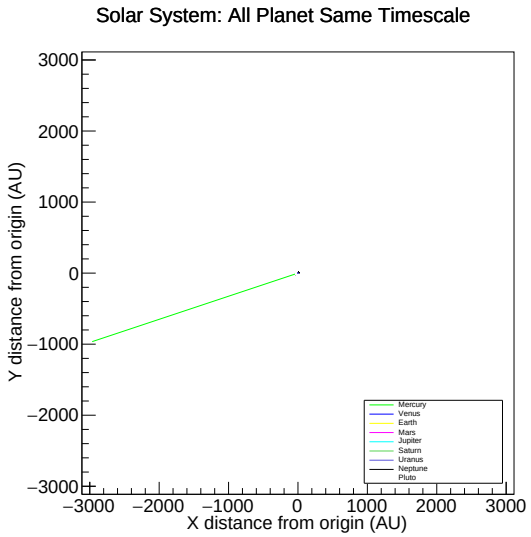

Figure 9: Plot of all major bodies calculated at the same time. Note that Mercury leaves the solar system. Time: 250 years, Time Steps: 25000, Algorithm: RK4

(both algorithms only depend on the previous step, so they only increase in floating point operations linearly). The many body case is limited by systems that move at many different timescales, since a relatively short time step for a body with a long period is a long time step for a body with a short period. The algorithms also fail to take into account effects due to general relativity. Both of these shortcomings can be seen in figure 9 with Mercury.

The different time scales can be remedied by calculating the orbits of bodies on different time scales separately. This is the method used in figure 8. Additionally, discrepancies due to general relativity could be remedied by including a relativistic correction to the calculation of the force on relevant bodies.

Appendix A

Earth-Sun Plots

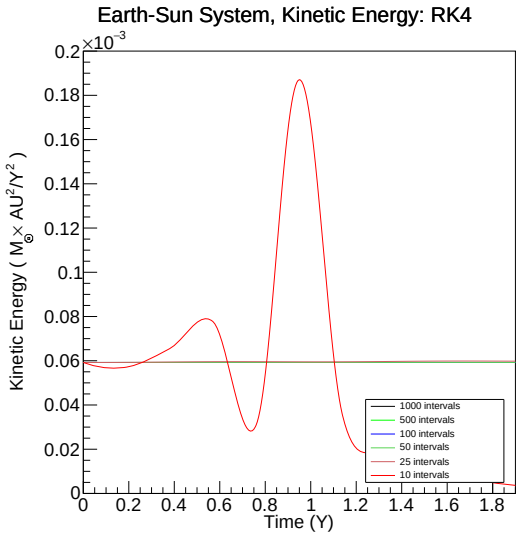

Figure A.1: Plot of Earth-Sun system kinetic energy for different numbers of time steps over 1.9 years using the RK4 algorithm.

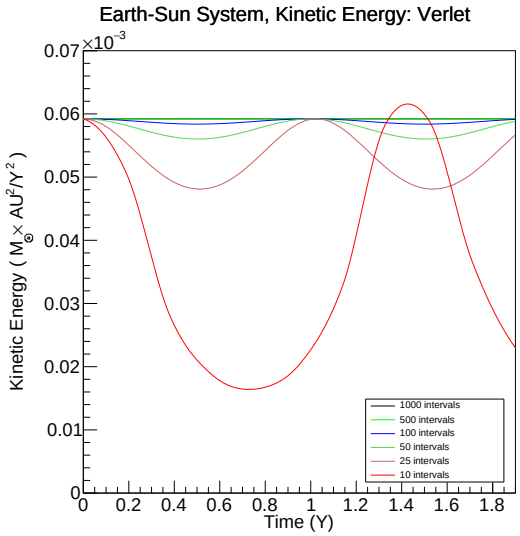

Figure A.2: Plot of Earth-Sun system kinetic energy for different numbers of time steps over 1.9 years using the Verlet algorithm.

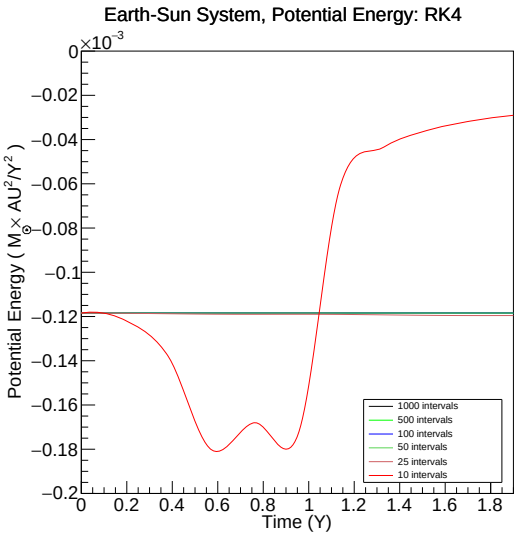

Figure A.3: Plot of Earth-Sun system potential energy for different numbers of time steps over 1.9 years using the RK4 algorithm.

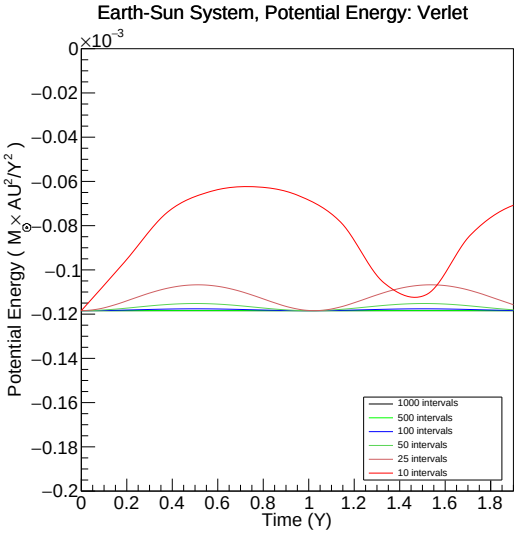

Figure A.4: Plot of Earth-Sun system potential energy for different numbers of time steps over 1.9 years using the Verlet algorithm.

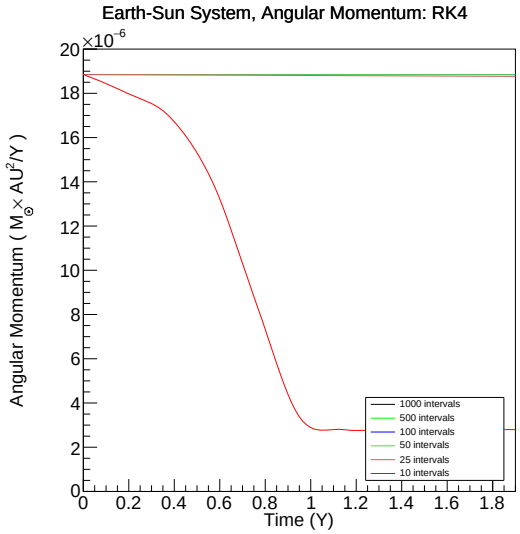

Figure A.5: Plot of Earth-Sun system angular momentum for different numbers of time steps over 1.9 years using the RK4 algorithm.

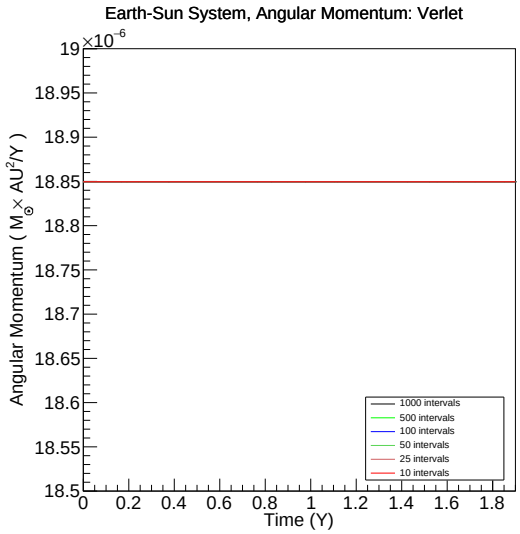

Figure A.6: Plot of Earth-Sun system angular momentum for different numbers of time steps over 1.9 years using the Verlet algorithm.

Appendix B

Three Body System: Stationary Sun Plots

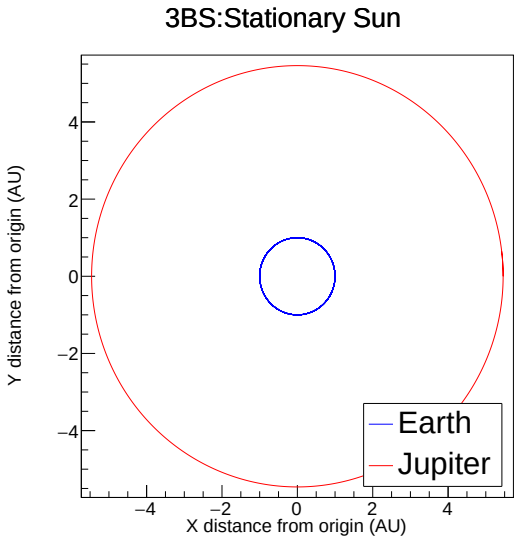

Figure B.1: Plot of 3 body system with Earth, Jupiter, and a stationary sun using regular Jupiter mass. Time interval: 13 years, Time steps: 13000, Algorithm: RK4.

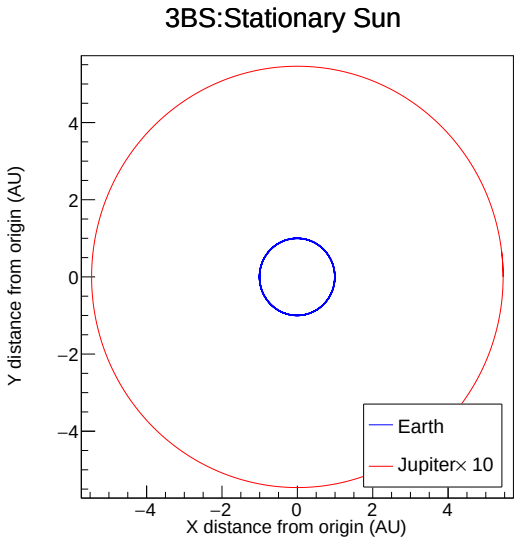

Figure B.2: Plot of 3 body system with Earth, Jupiter, and a stationary sun using $\times 10$ Jupiter mass. Time interval: 13 years, Time steps: 13000, Algorithm: RK4.

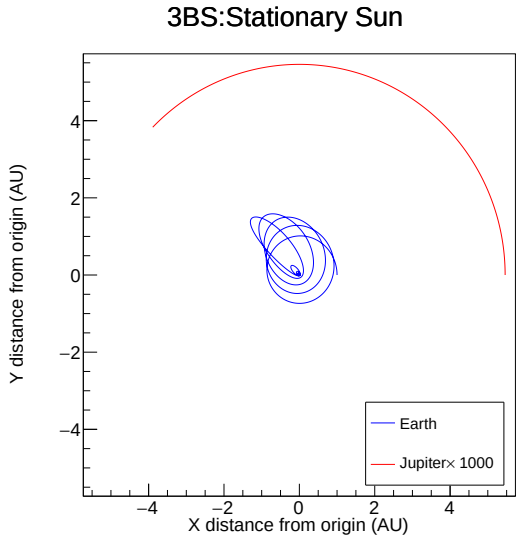

Figure B.3: Plot of 3 body system with Earth, Jupiter, and a stationary sun using $\times 1000$ Jupiter mass. Used a shorter timespan because this heavy Jupiter causes Earth to "hit the sun", then go barreling out many AU so the orbital structure is no longer visible. Time interval: 4.8 years, Time steps: 13000, Algorithm: RK4.

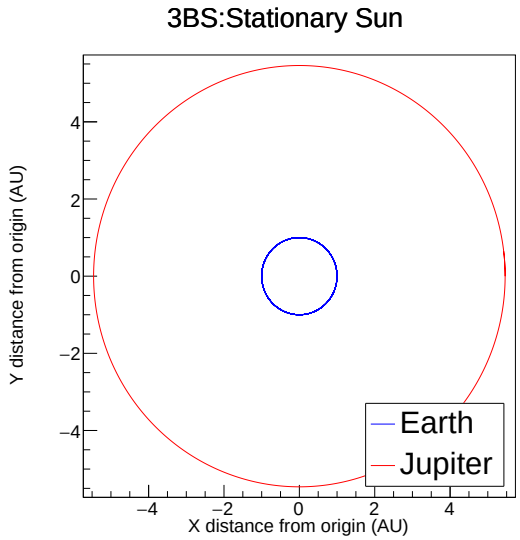

Figure B.4: Plot of 3 body system with Earth, Jupiter, and a stationary sun using regular Jupiter mass. Time interval: 13 years, Time steps: 13000, Algorithm: Verlet.

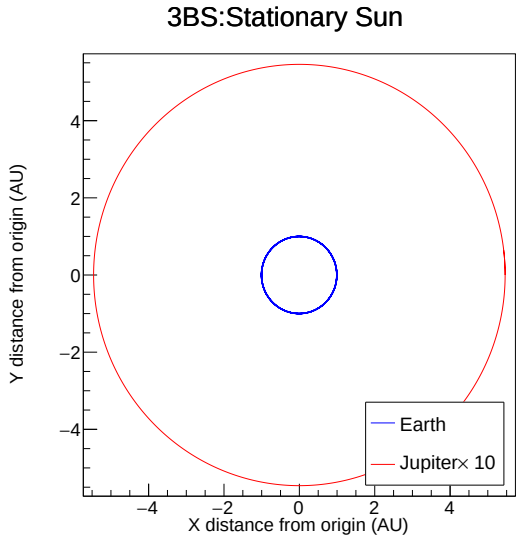

Figure B.5: Plot of 3 body system with Earth, Jupiter, and a stationary sun using ×10 Jupiter mass. Time interval: 13 years, Time steps: 13000, Algorithm: Verlet.

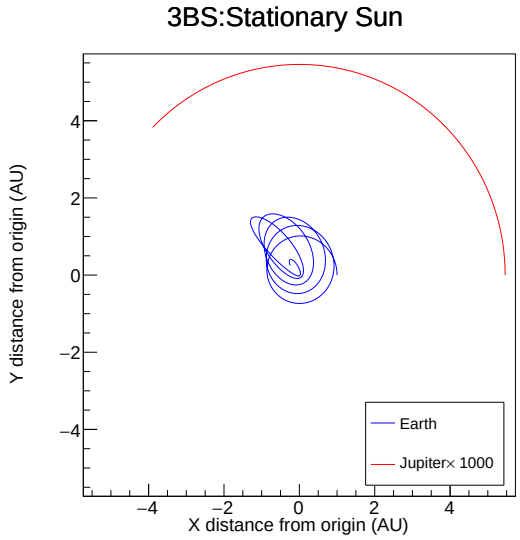

Figure B.6: Plot of 3 body system with Earth, Jupiter, and a stationary sun using ×1000 Jupiter mass. Used a shorter timespan because this heavy Jupiter causes Earth to "hit the sun", then go barreling out many AU so the orbital structure is no longer visible. Time interval: 4.8 years, Time steps: 13000, Algorithm: Verlet.

Appendix C

All Major Solar System Bodies: Plots

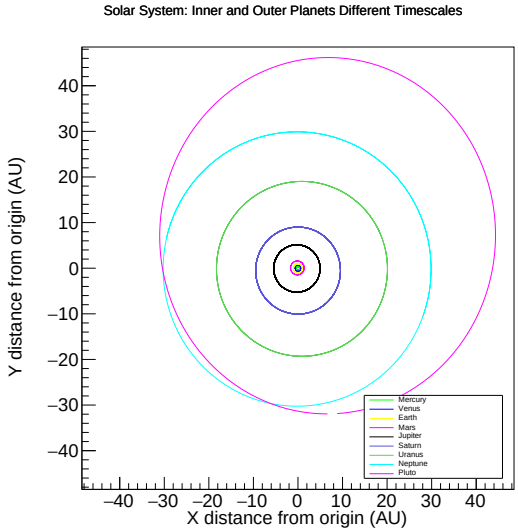

Figure C.1: Plot of all major bodies with inner and outer planets calculated separately. Inner Planets - Time: 5 years, Time Steps: 5000, Algorithm: Verlet. Outer Planets - Time: 250 years, Time Steps: 25000, Algorithm: Verlet

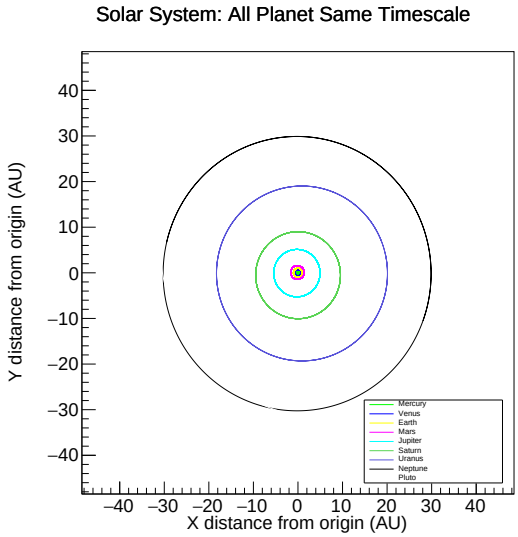

Figure C.2: Plot of all major bodies calculated at the same time. Time: 250 years, Time Steps: 25000, Algorithm: Verlet

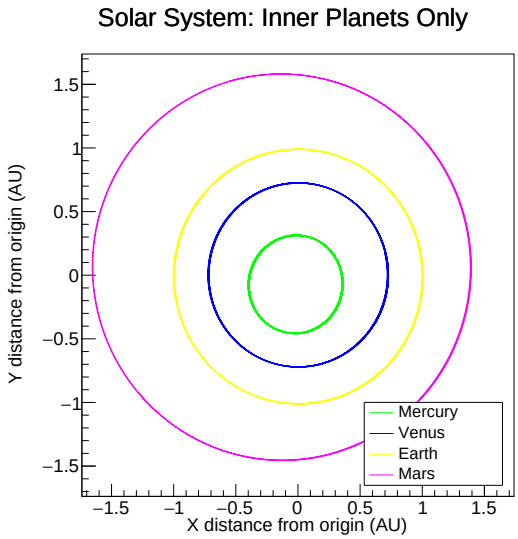

Figure C.3: Inner solar system. Time: 5 years, Time Steps: 5000, Algorithm: RK4

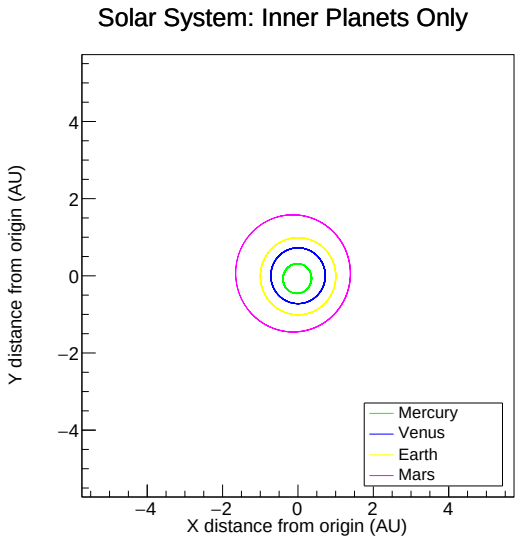

Figure C.4: Inner solar system. Time: 5 years, Time Steps: 5000, Algorithm: Verlet

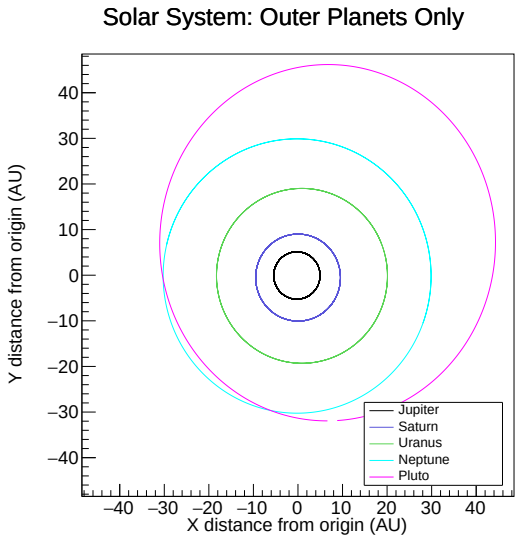

Figure C.5: Outer solar system. Time: 250 years, Time Steps: 25000, Algorithm: RK4

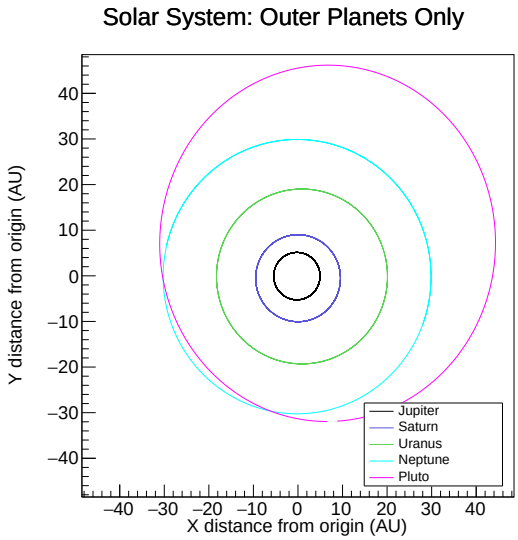

Figure C.6: Outer solar system. Time: 250 years, Time Steps: 25000, Algorithm: Verlet

Appendix D

All RK4 Code

```
void SolarSystem::RK4(){
  _solved = true;
  double Fx,Fy,Fz;

  for( int i = 1; i <= _Ntime; i++ ){
    vector< vector< double > > k1, k2, k3;
    for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){

      // Keep planet in same position if it is fixed
      if(_vPlanets.at(iPlanet).Fixed()){
        _vPlanets.at(iPlanet).AddCoordinates(i*_step,
                                             _vPlanets.at(iPlanet).X(i-1),
                                             _vPlanets.at(iPlanet).Y(i-1),
                                             _vPlanets.at(iPlanet).Z(i-1),
                                             _vPlanets.at(iPlanet).Vx(i-1),
                                             _vPlanets.at(iPlanet).Vy(i-1),
                                             _vPlanets.at(iPlanet).Vz(i-1));
        vector<double> placeholder;
        k1.push_back(placeholder);
      }
      else{

        // calculate k1
        vector<double> k1i;
        k1i.push_back( _step*_vPlanets.at(iPlanet).Vx(i-1) );
```

```
    k1i.push_back( _step*_vPlanets.at(iPlanet).Vy(i-1) );
    k1i.push_back( _step*_vPlanets.at(iPlanet).Vz(i-1) );

    TotalForce(iPlanet,i-1,Fx,Fy,Fz);
    k1i.push_back( _step*Fx/_vPlanets.at(iPlanet).M() );
    k1i.push_back( _step*Fy/_vPlanets.at(iPlanet).M() );
    k1i.push_back( _step*Fz/_vPlanets.at(iPlanet).M() );

    // Advance f(t_i,y_i)->f(t_i+h/2,y_i+k1/2)
    _vPlanets.at(iPlanet).AddCoordinates(i*_step,
                                         _vPlanets.at(iPlanet).X(i-1)+k1i.at(0)/2.,
                                         _vPlanets.at(iPlanet).Y(i-1)+k1i.at(1)/2.,
                                         _vPlanets.at(iPlanet).Z(i-1)+k1i.at(2)/2.,
                                         _vPlanets.at(iPlanet).Vx(i-1)+k1i.at(3)/2.,
                                         _vPlanets.at(iPlanet).Vy(i-1)+k1i.at(4)/2.,
                                         _vPlanets.at(iPlanet).Vz(i-1)+k1i.at(5)/2.);
    k1.push_back(k1i);
  }
}

for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){
  if(_vPlanets.at(iPlanet).Fixed()){
    vector<double> placeholder;
    k2.push_back(placeholder);
  }
  else{
    // calculate k2
    vector<double> k2i;
    k2i.push_back( _step*_vPlanets.at(iPlanet).Vx(i));
    k2i.push_back( _step*_vPlanets.at(iPlanet).Vy(i));
    k2i.push_back( _step*_vPlanets.at(iPlanet).Vz(i));

    TotalForce(iPlanet,i,Fx,Fy,Fz);
    k2i.push_back( _step*Fx/_vPlanets.at(iPlanet).M() );
    k2i.push_back( _step*Fy/_vPlanets.at(iPlanet).M() );
    k2i.push_back( _step*Fz/_vPlanets.at(iPlanet).M() );

    // Advance f(t_i+h/2,y_i+k1/2)->f(t_i+h/2,y_i+k2/2)
    _vPlanets.at(iPlanet).SetX(_vPlanets.at(iPlanet).X(i-1)+k2i.at(0)/2.,i);
    _vPlanets.at(iPlanet).SetY(_vPlanets.at(iPlanet).Y(i-1)+k2i.at(1)/2.,i);
    _vPlanets.at(iPlanet).SetZ(_vPlanets.at(iPlanet).Z(i-1)+k2i.at(2)/2.,i);
    _vPlanets.at(iPlanet).SetVx(_vPlanets.at(iPlanet).Vx(i-1)+k2i.at(3)/2.,i);
```

```
    _vPlanets.at(iPlanet).SetVy(_vPlanets.at(iPlanet).Vy(i-1)+k2i.at(4)/2.,i);
    _vPlanets.at(iPlanet).SetVz(_vPlanets.at(iPlanet).Vz(i-1)+k2i.at(5)/2.,i);
    k2.push_back(k2i);
  }
}

for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){
  if(_vPlanets.at(iPlanet).Fixed()){
    vector<double> placeholder;
    k3.push_back(placeholder);
  }
  else{
    // calculate k3
    vector<double> k3i;
    k3i.push_back( _step*_vPlanets.at(iPlanet).Vx(i));
    k3i.push_back( _step*_vPlanets.at(iPlanet).Vy(i));
    k3i.push_back( _step*_vPlanets.at(iPlanet).Vz(i));

    TotalForce(iPlanet,i,Fx,Fy,Fz);
    k3i.push_back( _step*Fx/_vPlanets.at(iPlanet).M() );
    k3i.push_back( _step*Fy/_vPlanets.at(iPlanet).M() );
    k3i.push_back( _step*Fz/_vPlanets.at(iPlanet).M() );

    // Advance f(t_i+h/2,y_i+k2/2)->f(t_i+h,y_i+k3)
    _vPlanets.at(iPlanet).SetX(_vPlanets.at(iPlanet).X(i-1)+k3i.at(0),i);
    _vPlanets.at(iPlanet).SetY(_vPlanets.at(iPlanet).Y(i-1)+k3i.at(1),i);
    _vPlanets.at(iPlanet).SetZ(_vPlanets.at(iPlanet).Z(i-1)+k3i.at(2),i);
    _vPlanets.at(iPlanet).SetVx(_vPlanets.at(iPlanet).Vx(i-1)+k3i.at(3),i);
    _vPlanets.at(iPlanet).SetVy(_vPlanets.at(iPlanet).Vy(i-1)+k3i.at(4),i);
    _vPlanets.at(iPlanet).SetVz(_vPlanets.at(iPlanet).Vz(i-1)+k3i.at(5),i);
    k3.push_back(k3i);
  }
}

for( unsigned int iPlanet = 0; iPlanet < GetN(); iPlanet++ ){
  if(!_vPlanets.at(iPlanet).Fixed()){
    // calculate k4
    vector<double> k4i;
    k4i.push_back( _step*_vPlanets.at(iPlanet).Vx(i));
    k4i.push_back( _step*_vPlanets.at(iPlanet).Vy(i));
    k4i.push_back( _step*_vPlanets.at(iPlanet).Vz(i));
```

```
        Fx = Fy = Fz = 0.;
        TotalForce(iPlanet,i,Fx,Fy,Fz);
        k4i.push_back( _step*Fx/_vPlanets.at(iPlanet).M() );
        k4i.push_back( _step*Fy/_vPlanets.at(iPlanet).M() );
        k4i.push_back( _step*Fz/_vPlanets.at(iPlanet).M() );

        // Advance y_(i+1) = y_i + (1/6)(k1+2k2+2k3+k4)
        _vPlanets.at(iPlanet).SetX(_vPlanets.at(iPlanet).X(i-1)
                                   +(k1.at(iPlanet).at(0)
                                   +2.*k2.at(iPlanet).at(0)
                                   +2.*k3.at(iPlanet).at(0)
                                     +k4i.at(0))/6.,i);
        _vPlanets.at(iPlanet).SetY(_vPlanets.at(iPlanet).Y(i-1)
                                   +(k1.at(iPlanet).at(1)
                                   +2.*k2.at(iPlanet).at(1)
                                   +2.*k3.at(iPlanet).at(1)
                                     +k4i.at(1))/6.,i);
        _vPlanets.at(iPlanet).SetZ(_vPlanets.at(iPlanet).Z(i-1)
                                   +(k1.at(iPlanet).at(2)
                                   +2.*k2.at(iPlanet).at(2)
                                   +2.*k3.at(iPlanet).at(2)
                                     +k4i.at(2))/6.,i);
        _vPlanets.at(iPlanet).SetVx(_vPlanets.at(iPlanet).Vx(i-1)
                                    +(k1.at(iPlanet).at(3)
                                    +2.*k2.at(iPlanet).at(3)
                                    +2.*k3.at(iPlanet).at(3)
                                      +k4i.at(3))/6.,i);
        _vPlanets.at(iPlanet).SetVy(_vPlanets.at(iPlanet).Vy(i-1)
                                    +(k1.at(iPlanet).at(4)
                                    +2.*k2.at(iPlanet).at(4)
                                    +2.*k3.at(iPlanet).at(4)
                                      +k4i.at(4))/6.,i);
        _vPlanets.at(iPlanet).SetVz(_vPlanets.at(iPlanet).Vz(i-1)
                                    +(k1.at(iPlanet).at(5)
                                    +2.*k2.at(iPlanet).at(5)
                                    +2.*k3.at(iPlanet).at(5)
                                      +k4i.at(5))/6.,i);
      }
    }
  }
}
```

Bibliography

[1] Rene Brun and Fons Rademakers. Root - an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A*, 1997. See also http://root.cern.ch.

[2] Phil Duxbury and Morten Hjorth-Jensen. Msu computational physics (phy480/905) course material. 2016.

[3] Rubin Landau, Manuel Páez, and Cristian Bordeianu. *A Survey of Computational Physics*. Princeton University Press, Princeton, NJ, 2008.