Comparison of the Orbits of Heavenly bodies using Velocity Verlet and Fourth Order Runge-Kutta computational Methods

Bryan Isherwood

April 7, 2016

Abstract

In this project we numerically solve the n-body orbital equations using two approximation methods: Velocity Verlet and Fourth order Runge-Kutta. Using these algorithms we analyze the movements of the Sun, Earth, and Jupiter around their respective orbits. We find that Runge-Kutta preforms the most accurate and stable computation for many body systems while the Velocity Verlet algorithm preforms well in two body systems when gravitational forces are small compared to the Sun's.

1 Introduction

Since the inception of physics as a mathematical discipline scientists have attempted to map and calculate the movements of the planets in our solar system. While the calculus of Issac Newton and his universal law of gravity generated great strides in this endeavour, the mathematics of the n-body problem often proves too complicated to solve by analytic calculation alone. In order to accomplish this feat we then turn to numerical computation to approximate the solution of planetary trajectories.

This paper will review and implement two methods of approximation: Velocity Verlet (VV) and Fourth order Runge-Kutta (RK4). The Verlet algorithm provides a good approximation for our trajectories despite its simplicity. As a result it will be the first algorithm analyzed in this paper. RK4 is a fast yet more complicated algorithm that provides greater accuracy over VV and acts as the work horse for various calculations we will use to prove our calculation obeys the standard physical laws of orbital mechanics.

We will begin with a brief review of Newtonian orbital mechanics, Kepler's laws, and the analytic theory behind both approximation methods. Next we will analyze the actual algorithm implemented for our various calculations. In analyzing our given solar system we look at several physical quantities and observe how they change as more planets are added or existing planets are altered e.g. increases in mass.

2 Theory

2.1 Orbital Mechanics

Any discussion of orbital mechanics must begin with Newton's universal law of gravity:

$$m_1\ddot{\mathbf{r}} = -\frac{Gm_2m_1}{r^2}\hat{\mathbf{r}} \tag{1}$$

Where G is the gravitational force constant, m_2 is the mass of the object creating a gravitational potential felt by m_1, and r is the relative distance between the two objects. We can then split this equation up into a force law in the x- and y- planes individually:

$$\begin{aligned}\ddot{x} &= -\frac{GMx}{r^3}\\ \ddot{y} &= -\frac{GMy}{r^3}\end{aligned} \tag{2}$$

We can then create a set of coupled linear equations using the velocity equation:

$$\begin{aligned}v_x &= \dot{x}\\ v_y &= \dot{y}\\ \dot{v_x} &= -\frac{GMx}{r^3}\\ \dot{v_y} &= -\frac{GMy}{r^3}\end{aligned} \tag{3}$$

We can further manipulate our equations by recalling astronomical units and setting the average radius of earths orbit as 1 AU, allowing us to say:

$$GM_{sun} = 4\pi^2\frac{AU^3}{yr^2} \tag{4}$$

when we assume a circular orbit of the earth around the sun [1]. Equations (2) and (3) assume a large mass positioned at the center of mass of the system with only one other body orbiting it, with a much smaller mass. In order to determine how multiple bodies affect the orbit of our planet of interest we can write:

$$\begin{aligned}\ddot{x_i} &= -\sum_{k\neq i}\frac{4\pi^2(x_i - x_k)}{|\mathbf{r}_i - \mathbf{r}_k|^3}\frac{M_k}{M_{sun}}\\ \ddot{y_i} &= -\sum_{k\neq i}\frac{4\pi^2(y_i - y_k)}{|\mathbf{r}_i - \mathbf{r}_k|^3}\frac{M_k}{M_{sun}}\end{aligned} \tag{5}$$

We also know from Kepler that there are a set of laws that follow from the assumption of Newton's force law:

$$\begin{aligned}&1^{st} : All\ planets\ follow\ elliptical\ orbits\ with\ the\ sun\ at\ one\ focus\\ &2^{nd} : The\ area\ covered\ by\ a\ planets\ elliptical\ orbit\ over\ a\ period\ of\ time\ is\ constant\\ &3^{rd} : The\ orbital\ period\ of\ a\ planet\ is\ given\ by : \tau^2 = \frac{4\pi^2}{GM_{sun}}R^3\end{aligned} \tag{6}$$

further details and proofs for these laws are given in [2]. This will act as the starting point for our investigation. Due to the nature of these coupled differential equations it is difficult if not impossible to solve the general n-body equation by hand. As a result we require computational methods in order to analyze the system of interest. For this project, we will use the Velocity Verlet and the fourth order Runge-Kutta methods of approximation.

2.2 Velocity Verlet

As demonstrated in [1], the Verlet algorithm is derived from a Taylor expansion on position and velocity:

$$\begin{aligned} x_{i+1} &= x_i + h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i + O(h^3) \\ v_{i+1} &= v_i + h\dot{v}_i + \frac{h^2}{2}\ddot{v}_i + O(h^3) \end{aligned} \tag{7}$$

we can then rewrite the velocity component to look like:

$$v_{i+1} = v_i + \frac{h}{2}(\dot{v}_{i+1} + \dot{v}_i) \tag{8}$$

A full proof is given in [1].

2.3 Runge-Kutta

The Fourth order Runge-Kutta formalism is far more complicated than the velocity verlet algorithm, but it is also more accurate. An amended proof is given here, for a full proof please refer to [1]. To begin suppose the formalism of:

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt. \tag{9}$$

we can then approximate this integral as:

$$\int_{t_i}^{t_{i+1}} f(t, y)dt \approx hf(t_{i+1/2}, y_{i+1/2}) + O(h^3). \tag{10}$$

making eq. (9) of the form:

$$y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}) + O(h^3). \tag{11}$$

Using the midpoint rule for integration allows us to create an intermediate step in our calculation, this will ultimately allow for greater accuracy over the verlet method. At the moment though, we do not know the value of the intermediate point. To determine it we use Euler's approximation method starting at our initial point:

$$y_{(i+1/2)} = y_i + \frac{h}{2}\frac{dy}{dt} = y(t_i) + \frac{h}{2}f(t_i, y_i). \tag{12}$$

we can then define the quantities:

$$\begin{aligned} k_1 &= hf(t_i, y_i) \\ k_2 &= hf(t_{i+1/2}, y_i + k_1/2) \end{aligned} \tag{13}$$

with the final value

$$y_{i+i} \approx y_i + k_2 + O(h^3). \tag{14}$$

Although we can contine to improve upon this value by defining:

$$\begin{aligned} k_1 &= hf(t_i, y_i) \quad k_2 = hf(t_i + h/2, y_i + k_1/2) \\ k_3 &= hf(t_i + h/2, y_i + k_2/2) \quad k_4 = hf(t_i + h, y_i + k_3) \end{aligned}$$

leading us to our final value of:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^4) \tag{15}$$

3 Algorithm

In order to demonstrate the mentality behind the given algorithm we will discuss it in three parts: Acceleration calculation, VV implementation, and RK4 implementation. Firstly is important to note that the initial conditions and mass of each orbiting body have been stored in a class object know as **Planet**. Four *n x m* matrices have been created to track the position and velocity information of each planet. If n is the number of step sizes and m is the number of planets in our system, these arrays allow us to keep track of the trajectory of each of our planets over time. The pre-compiled linear algebra library **Lib.cpp** was used to create these matrices. It is also important to note this project normalizes the solar system's mass by the mass of the sun. To that effect the mass of the sun is then $M_{sun} = 1.0$ *solar masses*, and all other masses are scaled accordingly.

3.1 Acceleration calculation

Following from eq. (5) we can see that the net acceleration in the x- and y-directions are given by a summation over the interaction between the planet of interest and each of the other planets. To preform this calculation we create a simple nested loop over all planets of interest and each other planet, written as such:

```
$for(j = 0; j < m; j++)$
   $a_x = a_y = 0$
  $for(k = 0; k < m; k++)$
    $if(k \neq m)$
      $a_x += \frac{4\pi^2(x_j - x_k)m_k}{[(x_j - x_k)^2 + (y_j - y_k)^2]^{3/2}}$
      $a_y += \frac{4\pi^2(y_j - y_k)m_k}{[(x_j - x_k)^2 + (y_j - y_k)^2]^{3/2}}$        (16)
    $else$
      $a_x, a_y += 0$
$return \quad a_x, a_y$
```

The given algorithm will calculate the x- and y- direction accelerations individually at each point in time. Please note that this does not include the direction of the force, but only its magnitude. The loop over j allows for the calculation of each planet's acceleration at t = *ih.* The loop over k calculates the acceleration due to each other planet in the system. Using this algorithm we can then implement our VV and RK4 algorithms by referring to the acceleration calculation at each step.

3.2 Verlet Calculation

As given by eq.s (7) and (8) above, this calculation is fairly straightforward. We can determine the position of our planet at t = *(i+1)h* by:

$$\begin{aligned} X[i+1][j] &= X[i][j] + hV_x[i][j] - \frac{h^2}{2}a_{x,i,j} \\ Y[i+1][j] &= Y[i][j] + hV_y[i][j] - \frac{h^2}{2}a_{y,i,j} \end{aligned} \tag{17}$$

which can be read as: the position of the object at t = *(i+1)h* for the j th planet is the sum of its initial position, the distance covered by its velocity, and the distance covered by its acceleration all at t = *ih.* It is important to note here and for later sections that the acceleration calculated here is for the i th time step. In order to determine the velocity at the next step we need to know the acceleration at the next step. In order to do this it is advised to update the position for each planet before moving on to the velocity calculation. This requires two separate loops over j for each quantity. After determining the acceleration at each time step the calculation can be finished with:

$$\begin{aligned} V_x[i+1][j] &= V_x[i][j] - \frac{h}{2}(a_{x,i+1,j} + a_{x,i,j}) \\ V_y[i+1][j] &= V_y[i][j] - \frac{h}{2}(a_{y,i+1,j} + a_{y,i,j}) \end{aligned} \tag{18}$$

3.3 Runge-Kutta Calculation

The process for using RK4 properly in a calculation requires more micromanaging than the previous steps, mostly due to the necessity of successive approximations. To simplify this we will break up the algorithm into six parts that will be given explicitly below. Before then it is worth while to go over some of the finer intricacies of how the program for this project was run. First, every time the acceleration is calculated for one planet it must be calculated for every planet. To demonstrate the reasoning behind this suppose that we forgo this and only determine the acceleration of the *j+1st* after the *jth*. If you will recall from eq. () that we "advance" the position of our planet of interest three times, two of which are at half step intervals. If the position of every other planet is not know at those half step intervals, then the net acceleration at those points becomes invalid. The determination of x_{i+1} will inevitably assume that a constant acceleration through the entire time step, even though its position updated twice prior to the final calculation. Thus, each position calculation introduces increasing amounts of computational error per iteration. On the other hand, if we determine the position information for each planet simultaneous we avoid accumulating errors in our acceleration calculation and decrease our computational error.

When updating position information, it may be worth while to create four new matrices that can store the position and velocity information in the x-y plane at half step intervals. This way the information about the initial position is not lost and while still keeping the information on where the half step positions occur. We also require four arrays of length m that will store our "k" values at each step. With these things in mind, we can present the full algorithm for the RK4 calculation used for this project.

3.3.1 Step 1: Advance position and velocity by one half step

$$\begin{aligned}
&k_{vx1}[j] = a_{x,i,j} \\
&k_{vy1}[j] = a_{y,i,j} \\
&k_{x1}[j] = V_x[i][j] \\
&k_{y1}[j] = V_y[i][j] \\
&X_{half}[i][j] = X[i][j] + \frac{h}{2}k_{x1}[j] \\
&Y_{half}[i][j] = Y[i][j] + \frac{h}{2}k_{y1}[j] \\
&V_{x,half}[i][j] = Vx[i][j] - \frac{h}{2}k_{vx1}[j] \\
&V_{y,half}[i][j] = Vy[i][j] - \frac{h}{2}k_{vy1}[j]
\end{aligned} \tag{19}$$

3.3.2 Step 2: Determine acceleration at the first half step

$$\begin{aligned}
k_{vx1}[j] &= a_{x,i+1/2,j} \\
k_{vy1}[j] &= a_{y,i+1/2,j} \\
k_{x1}[j] &= V_{y,half}[i][j] \\
k_{y1}[j] &= V_{y,half}[i][j]
\end{aligned} \tag{20}$$

3.3.3 Step 3: Reevaluate position at half step

For this step it may be best to overwrite the previous half step position, as it is the more accurate mathematical value. Regardless, the position in this step will be denoted as "primed" to distinguish them from step 1:

$$\begin{aligned}
&X'_{half}[i][j] = X[i][j] + \frac{h}{2}k_{x2}[j] \\
&Y'_{half}[i][j] = Y[i][j] + \frac{h}{2}k_{y2}[j] \\
&V'_{x,half}[i][j] = Vx[i][j] - \frac{h}{2}k_{vx2}[j] \\
&V'_{y,half}[i][j] = Vy[i][j] - \frac{h}{2}k_{vy2}[j]
\end{aligned} \tag{21}$$

3.3.4 Step 4: Determine acceleration at revised half step

$$\begin{aligned}
k_{vx3}[j] &= a_{x',i+1/2,j} \\
k_{vy3}[j] &= a_{y',i+1/2,j} \\
k_{x3}[j] &= V_{y',half}[i][j] \\
k_{y3}[j] &= V_{y',half}[i][j]
\end{aligned} \tag{22}$$

3.3.5 Step 5: Evaluate the half step position and acceleration again

$$\begin{aligned}
X''_{half}[i][j] &= X[i][j] + \frac{h}{2}k_{x3}[j] \\
Y''_{half}[i][j] &= Y[i][j] + \frac{h}{2}k_{y3}[j] \\
V''_{x,half}[i][j] &= Vx[i][j] - \frac{h}{2}k_{vx3}[j] \\
V''_{y,half}[i][j] &= Vy[i][j] - \frac{h}{2}k_{vy3}[j] \\
k_{vx4}[j] &= a_{x'',i+1/2,j} \\
k_{vy4}[j] &= a_{y'',i+1/2,j} \\
k_{x4}[j] &= V_{x'',half}[i][j] \\
k_{y4}[j] &= V_{y'',half}[i][j]
\end{aligned} \tag{23}$$

3.3.6 Step 6: Determine final position for full step length

$$\begin{aligned}
X[i+1][j] &= X[i][j] + \frac{h}{6}(k_{x1}[j] + 2(k_{x2}[j] + k_{x3}[j]) + k_{x4}[j]); \\
Y[i+1][j] &= Y[i][j] + \frac{h}{6}(k_{x1}[j] + 2(k_{y2}[j] + k_{y3}[j]) + k_{y4}[j]); \\
Vx[i+1][j] &= Vx[i][j] - \frac{h}{6}(k_{vx1}[j] + 2(k_{vx2}[j] + k_{vx3}[j]) + k_{vx4}[j]); \\
Vy[i+1][j] &= Vy[i][j] - \frac{h}{6}(k_{vy1}[j] + 2(k_{vy2}[j] + k_{vy3}[j]) + k_{vy4}[j]);
\end{aligned} \tag{24}$$

4 Results

Using the program given at the bottom of the paper, we can calculate the orbits of the Sun, Earth, and Jupiter. To start with we will analyze the binary Sun-Earth system for a circular orbit. In order to have a stable circular orbit, given our set of units, we require the velocity of earth to be $4\pi^2 \frac{AU}{yr}$ at a distance of 1 AU from the sun. For this system we assume that the Sun exists in a fixed position with enough mass to not be affected by the gravity of earth. When we use VV we can see that the radius is stable around r $\approx$ 1.0, although it does diverge away slightly before coming back to its initial condition [figure 1]. Using RK4 gives a better result, as the radial position of the earth does not change over time, remaining at r = 1 [figure2]. Rk4 clearly conserves the total energy of the system. Considering the potential energy $U(\mathbf{r}) = -\frac{GMm}{|r|}$, it is clear that the energy is conserved as the every value in the calculation acts as a constant. It can also be shown that when $\delta t = 0.002$ years for a 1 year domain, that the total velocity is conserved over all time steps. It is clear to see in this case that there is computational error associated with this calculation. By calculating the total energy of the system we can see that the numerical imprecision of the VV algorithm causes energy conservation to be broken, at least to a small value. This makes sense for the given system considering that gravity is considered to be a conservative force. This means that the change in potential energy of an object as it moves between two points must be the same regardless of the path it takes to get there. Meaning that for any small deviation across it's orbit, so long as the Earth can form a closed loop over the entire orbit, the total energy will be conserved.

Using the Earth as an example, we can now determine under what conditions an unbounded orbit will occur. Beginning with the Earth at 1 AU the velocity was increased slowly until $\dot{r}$ became negative. Through this exercise it was found that the velocity necessary to create an unbounded orbit was $v = 2\pi + 2.5885754$.

We can now add another planet to our solar system. Adding Jupiter creates a force perturbation on the Earth, skewing its orbit slightly. For our purposes, the initial velocity of Jupiter was determined to ensure that without the Earth it would follow a circular orbit around the Sun. These initial conditions are: $r = 5.2AU$ and $v_0 = \frac{2\pi}{\sqrt{5.2}} \approx 2.76\frac{AU}{yr}$. Using Kepler's third law, we can determine that $\tau_{Jup} \approx 11.8yr$. As a result, all simulations including Jupiter exist over a domain of 12 years. Figures (3) and (4) show the differences between the orbits when using VV and RK4, respectively. It is clear to see that the orbits are slightly perturbed in both cases, as the system now attempts to move into an elliptical plane. Jupiter's orbit is far less perturbed than Earth's, which makes sense considering how much larger Jupiter's mass is by comparison as well as it's smaller speed.

Figures (5)-(6) and (7)-(8) show what happens to Earth's orbit as the mass of Jupiter is increased by factors of 10 and 1000 for both VV and RK4, respectively. When Jupiter's mass was increased by a factor of 10, there were no major differences Earth's orbit, but the factor of 1000 increase causes an extreme perturbation. Being stuck between two massive gravitational body forces Earth's orbit to precess, following the orbit of Jupiter. Using VV shows a wild spiraling of Earth outwards. Using RK4 by comparison, shows a more controlled precession, almost as if the origin had become a strange attractor.

Finally, we can use real astronomical data to act as an initial condition for both Jupiter and Earth. Using a center of mass calculation and conservation of momentum, we can calculate the initial conditions of the Sun in this three-body system. The values were input into our program and allowed to run. Figures (9) and (11) we can see how these three orbits compare with one another. For both VV and RK4, the Sun remains very close to the CM position of the system. When looking at the Earth and Jupiter on the other hand, there are some disturbing differences between VV and RK4 computational methods. The course updates of VV causes the orbits of Earth and Jupiter to slowly shift outwards. Using RK4 cleans these orbits up. Rather than changing the orbit dramatically, RK4 predicts a widening of the orbits, while still keeping them elliptical and controlled. The stability of the system is increased greatly with RK4, while VV predicts the Earth slowly moving away from it's initial orbit.

While a full computation was not completed, it was determined that both formulas were very stable.

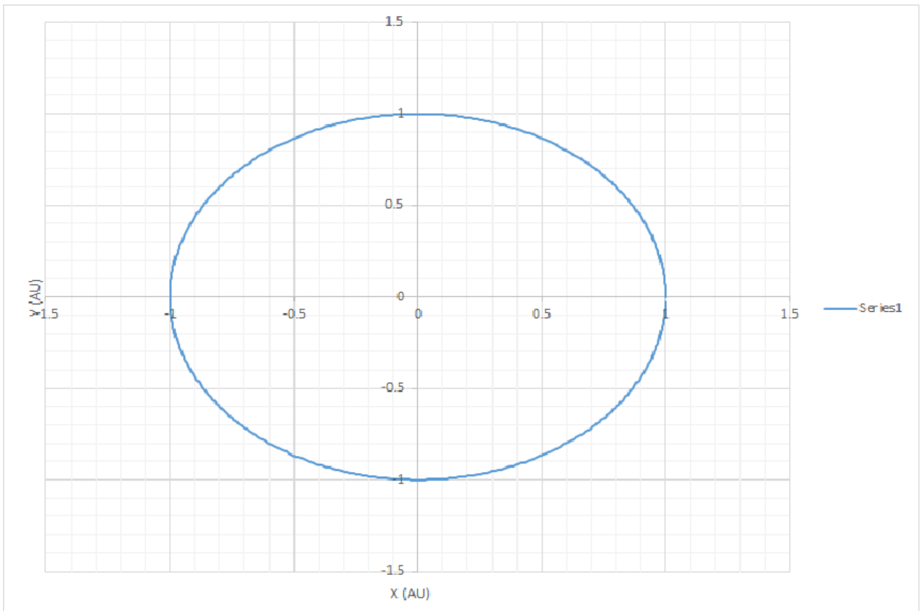

Figure 1: This plot shows the orbit of Earth using n = 500 and tmax = 1 year using VV.

5 Conclusion

We have shown how two different methods of calculating the orbits of planetary bodies predict the orbits of those bodies. While in many cases VV was a good approximation, it has been shown that RK4 does a much better job of handling full thee body calculations. This is likely due to the presence of the intermediate steps introduced by RK4. Using RK4 allows for the advantages of using a smaller time step, while removing the computational uncertainty inherent in using a small time step.

While both methods predict planetary orbits with good accuracy, RK4 clearly allows for more accurate and stable calculations as the complexity of the system of interest increases.

6 Appendix

```
#include <iostream>
#include <string>
#include <cmath>
#include <fstream>
#include <cstdlib>
#include <new>
#include "planet.h"
#include "lib.h"
//#include "time.h"
```

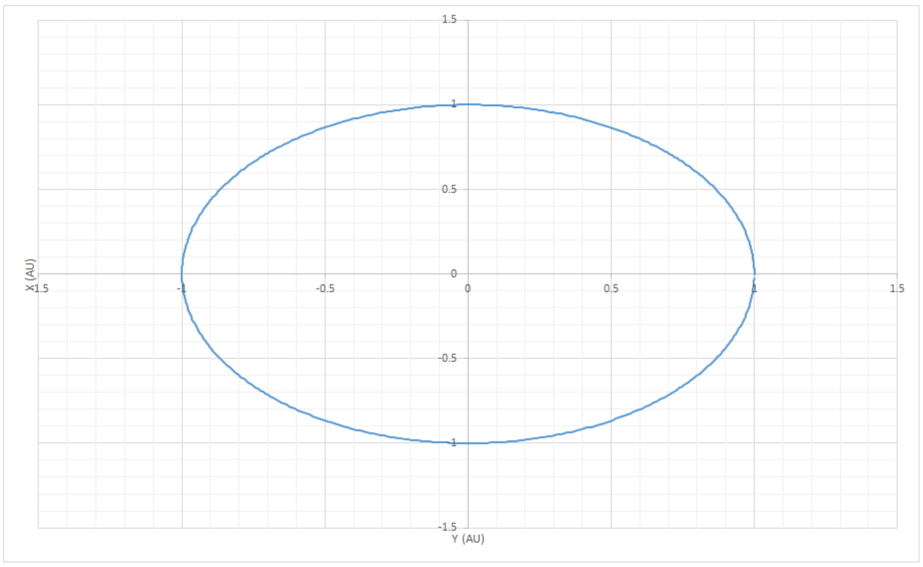

Figure 2: This plot shows the orbit of Earth using n = 500 and tmax = 1 year using RK4.

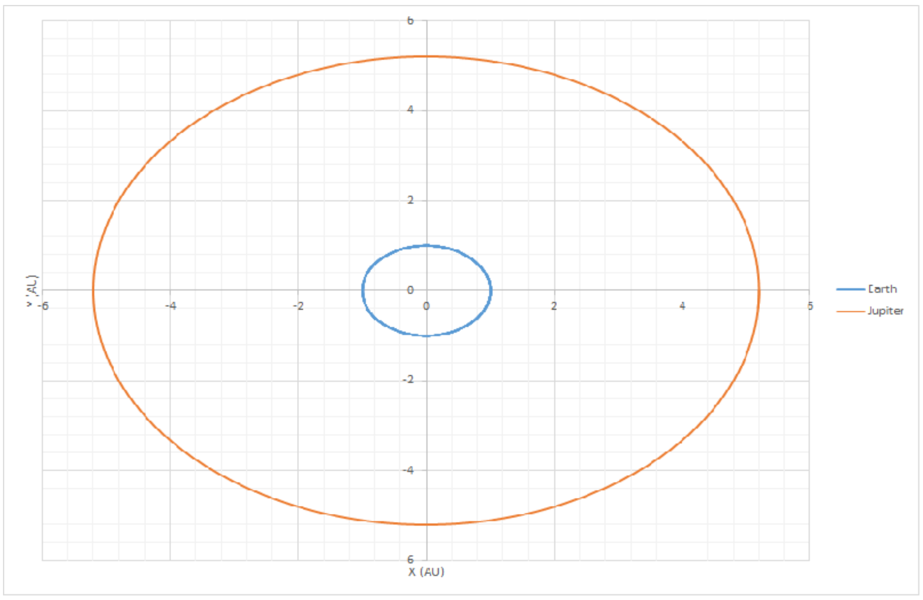

Figure 3: This plot shows the orbits of Earth and Jupiter using n = 6000 and tmax = 12 years using VV.

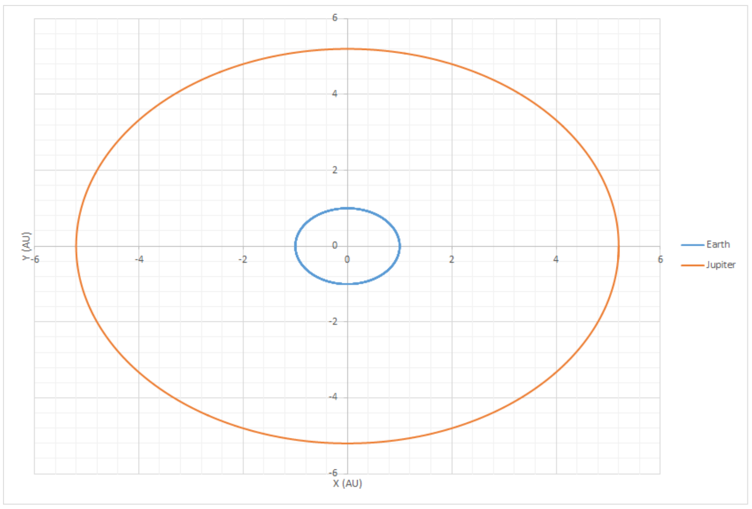

Figure 4: This plot shows the orbits of Earth and Jupiter using n = 6000 and tmax = 12 years using RK4.

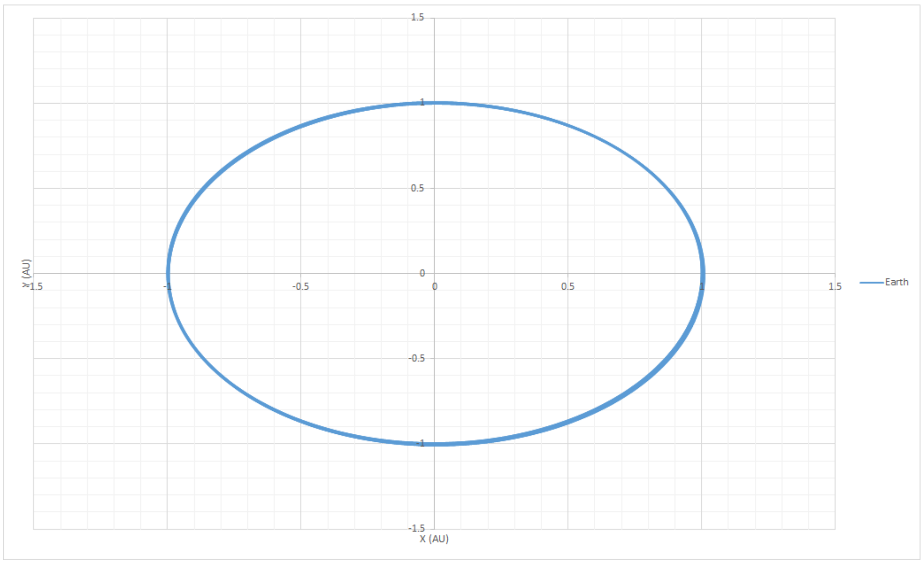

Figure 5: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using VV when the mass of Jupiter is increased by a factor of 10.

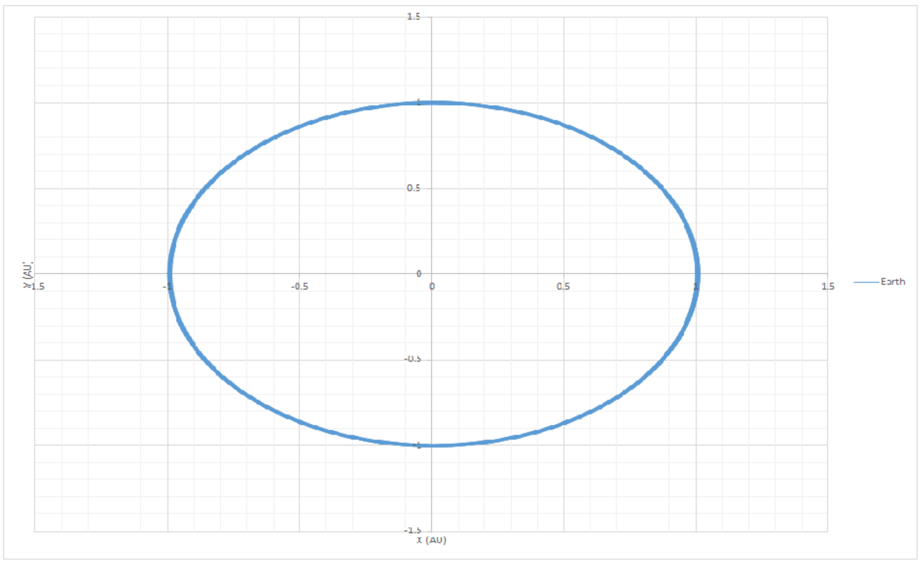

Figure 6: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using RK4 when the mass of Jupiter is increased by a factor of 10.

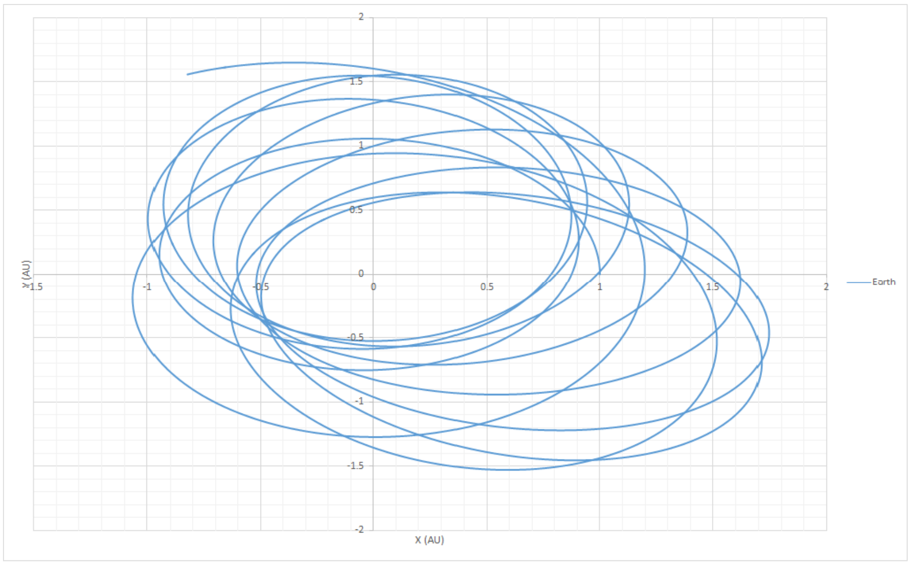

Figure 7: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using VV when the mass of Jupiter is increased by a factor of 1000.

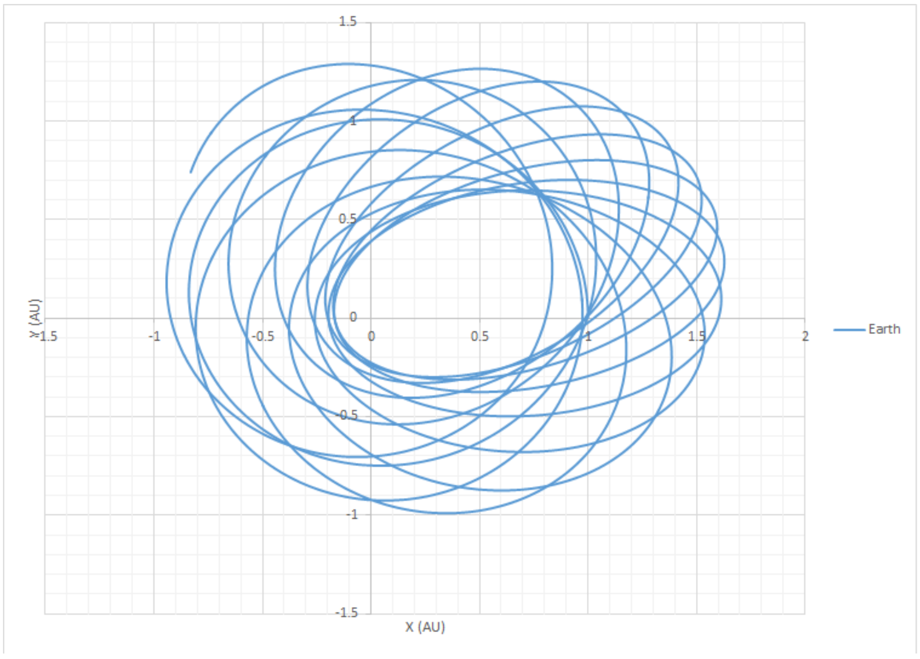

Figure 8: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using RK4 when the mass of Jupiter is increased by a factor of 1000.

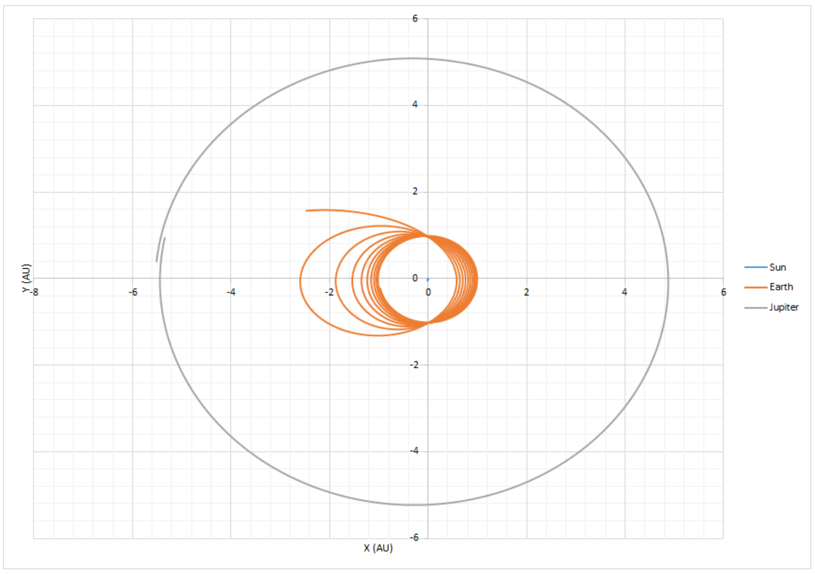

Figure 9: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using VV when initial conditions are constructed from real data.

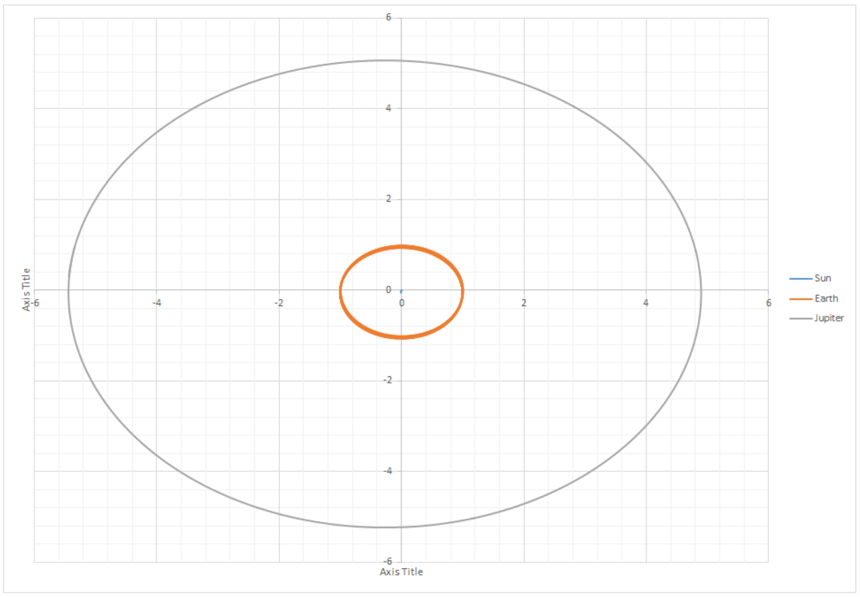

Figure 10: This plot shows the orbit of Earth using n = 6000 and tmax = 12 year using RK4 when initial conditions are constructed from real data.

```
using namespace std;

int main()
{
    double **R,**X,**Y,**Xhalf,**Yhalf,**Rhalf;
    double **Vxhalf,**Vyhalf,**Vx,**Vy;
   // double *vx,*vy,*x,*y,*r;
    //double *vxhalf,*vyhalf,*xhalf,*yhalf,*rhalf;
    double
        gravnowx,gravnowy,gravnextx,gravnexty,gravhalfx,gravhalfy,tmax,t0,h,h2,pi,gravnow,gravnext,t
    double *kx1,*kx2,*kx3,*kx4,*kvx1,*kvx2,*kvx3,*kvx4;
    double *ky1,*ky2,*ky3,*ky4,*kvy1,*kvy2,*kvy3,*kvy4;
    int i,j,n,k,m;
    t0 = 0.0;
    tmax = 12.0;
    n = 10000;
    h = (tmax - t0)/(n+1);
    h2 = h*h;
    pi = acos(-1.0);
  /*  r = new double[n];
    kvx1 = new double[n];
    kvy1 = new double[n];
    kx1 = new double[n];
    ky1 = new double[n];

    rhalf = new double[n];
    vxhalf = new double[n];
    vyhalf = new double[n];
    xhalf = new double[n];
    yhalf = new double[n];*/


       kvx1 = new double[3];
       kvy1 = new double[3];
       kx1 = new double[3];
       ky1 = new double[3];
       kvx2 = new double[3];
       kvy2 = new double[3];
       kx2 = new double[3];
       ky2 = new double[3];
       kvx3 = new double[3];
       kvy3 = new double[3];
       kx3 = new double[3];
       ky3 = new double[3];
       kvx4 = new double[3];
       kvy4 = new double[3];
       kx4 = new double[3];
       ky4 = new double[3];

    R = (double **) matrix(n,3,sizeof(double));
    X = (double **) matrix(n,3,sizeof(double));
    Y = (double **) matrix(n,3,sizeof(double));
    Rhalf = (double **) matrix(n,3,sizeof(double));
    Xhalf = (double **) matrix(n,3,sizeof(double));
    Yhalf = (double **) matrix(n,3,sizeof(double));

    Vx = (double **) matrix(n,3,sizeof(double));
    Vy = (double **) matrix(n,3,sizeof(double));
    Vxhalf = (double **) matrix(n,3,sizeof(double));
    Vyhalf = (double **) matrix(n,3,sizeof(double));




   // planet Earth(0.000003,-9.687087657503362E-01,
       -2.304478999525212E-01,0.0,3.715293823075145E-03*365.0,-1.679555302972549E-02*365.0,0.0);
   // planet
       Sun(1.0,0.00510182404,0.0008987727027,0.0,-0.0004829298902,-0.002481983714,0.0);
       //(mass,x,y,z,vx,vy,vz)
   // planet
       Jupiter(.000954,-5.344777687381148,.9428344302147893,0.0,-1.398575965353750E-03*365.0,-7.0750
    //planet all_planets[3] = {Sun,Earth,Jupiter};
planet Earth(0.000003,1.0000,0,0,0,2*pi,0);
planet Sun(1.0,0,0,0,0,0,0);
//planet CM(0,0,0,0,0,0,0);
planet Jupiter(0.000954*1000,5.2,0,0,0,-2*pi/sqrt(5.2),0);
planet all_planets[3] = {Sun,Earth,Jupiter};
m = 3;


/*x[0] = 1.0;
vy[0] = 2*pi+2.5885754;
```

```
vx[0] = 0.0;
y[0] = 0.0;*/

for(i=0;i<m;i++){
X[0][i] = all_planets[i].position[0];
Y[0][i] = all_planets[i].position[1];
Vx[0][i] = all_planets[i].velocity[0];
Vy[0][i] = all_planets[i].velocity[1];

}
  //  cout << X[0][1] << endl;





    //Verlet----------------------------------------------------------------------------------

/*    for(i = 0;i<n-1;i++){
       for(j = 0;j<m;j++){
         //  cout << R[0][1] << endl;
        R[i][j] = sqrt(X[i][j]*X[i][j] + Y[i][j]*Y[i][j]);
        gravnow = 4.0*pi*pi/pow(R[i][j],3);
        kx1[j] = 0.0;
        ky1[j] = 0.0;
           for(k=0;k<m;k++){

            if(k!=j){

               kx1[j] +=
                    4*pi*pi*(X[i][j]-X[i][k])*all_planets[k].mass/pow(pow(X[i][j]-X[i][k],2.0)+pow(Y[i
               ky1[j] +=
                    4*pi*pi*(Y[i][j]-Y[i][k])*all_planets[k].mass/pow(pow(X[i][j]-X[i][k],2.0)+pow(Y[i
               else{
               kx1[j]+= 0.0;
               ky1[j]+= 0.0;
}           }

         //  cout << gravnow*X[i][1] << " " << kx1[1] << endl;
    X[i+1][j] = X[i][j] + h*Vx[i][j] - h2/2.0*kx1[j];
    Y[i+1][j] = Y[i][j] + h*Vy[i][j] - h2/2.0*ky1[j];
       }
    //all_planets[j].position[0] = X[i+1][j];
    //all_planets[];
       for(j=0;j<m;j++){
           kx2[j] = 0.0;
           ky2[j] = 0.0;
    R[i+1][j] = sqrt(X[i+1][j]*X[i+1][j] + Y[i+1][j]*Y[i+1][j]);
    gravnext = 4.0*pi*pi/(R[i+1][j]*R[i+1][j]*R[i+1][j]);
    for(k=0;k<m;k++){
     if(k!=j){
         //cout<<X[i+1][k] << endl;
        kx2[j] +=
            4*pi*pi*(X[i+1][j]-X[i+1][k])*all_planets[k].mass/pow(sqrt(pow(X[i+1][j]-X[i+1][k],2.0)+p
        ky2[j] +=
            4*pi*pi*(Y[i+1][j]-X[i+1][k])*all_planets[k].mass/pow(sqrt(pow(X[i+1][j]-X[i+1][k],2.0)+p
        //cout << gravnextx << " " << j << " " << k << endl;
     }else{
        kx2[j]+= 0.0;
        ky2[j]+= 0.0;
}           }

    Vx[i+1][j] = Vx[i][j] -h/2.0*(kx2[j] + kx1[j]);
    test =Vy[i][j] - h/2.0*(gravnext*Y[i+1][1]+gravnow*Y[i][1]);
    Vy[i+1][j] = Vy[i][j] -h/2.0*(ky2[j] + ky1[j]);

   // cout << Vy[i+1][1] << " "<< test << endl;
}
}
    ofstream myfile;
        myfile.open("proj3_jup_circle_n=6000_t=12_VV_3plans.txt");
    for(i = 0; i<n;i++){
     myfile << i << " " << X[i][2] << " " << Y[i][2] << " "<< Vx[i][2]
         << " " << Vy[i][2] << endl;
      //"  " << i << endl;
    }*/
//----------------------------------------------------------------------------------------------------

    //Runge-kutta-------------------------------------------------------------------------------------
for(i=0;i<n-1;i++){

   //First
    for(j=1;j<m;j++){
    R[i][j] = sqrt(X[i][j]*X[i][j] + Y[i][j]*Y[i][j]);
    gravnowx = 0.0;
    gravnowy = 0.0;
       for(k=0;k<m;k++){

        if(k!=j){

           gravnowx +=
               4*pi*pi*(X[i][j]-X[i][k])*all_planets[k].mass/pow(pow(X[i][j]-X[i][k],2.0)+pow(Y[i][j]
           gravnowy +=
               4*pi*pi*(Y[i][j]-Y[i][k])*all_planets[k].mass/pow(pow(X[i][j]-X[i][k],2.0)+pow(Y[i][j]
           else{
           gravnowx+= 0.0;
           gravnowy+= 0.0;
}           }



    gravnow = 4.0*pi*pi/(R[i][j]*R[i][j]*R[i][j]);


    kvx1[j]= gravnowx;
    kvy1[j] = gravnowy;
    Xhalf[i][j] = X[i][j] + h/2.0*Vx[i][j];
    Yhalf[i][j] = Y[i][j] + h/2.0*Vy[i][j];
    Vxhalf[i][j] = Vx[i][j] - kvx1[j]*h/2.0;
    Vyhalf[i][j] = Vy[i][j] - kvy1[j]*h/2.0;
    //test = gravnow*X[i][j];
    //cout << test << " " << kvx1[1] << endl;
}
    //Second

    for(j=1;j<m;j++){
    R[i][j] = sqrt(X[i][j]*X[i][j] + Y[i][j]*Y[i][j]);
    gravhalfx = 0.0;
    gravhalfy = 0.0;
       for(k=0;k<m;k++){

        if(k!=j){

           gravhalfx +=
               4*pi*pi*(Xhalf[i][j]-Xhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-Xhalf[i][k],
           gravhalfy +=
               4*pi*pi*(Yhalf[i][j]-Yhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-Xhalf[i][k],
           else{
           gravhalfx+= 0.0;
           gravhalfy+= 0.0;
}           }

        Rhalf[i][j] = sqrt(Xhalf[i][j]*Xhalf[i][j]
             +Yhalf[i][j]*Yhalf[i][j] );
        gravhalf = 4.0*pi*pi/(Rhalf[i][j]*Rhalf[i][j]*Rhalf[i][j]);
        kvx2[j] = gravhalfx;
        kvy2[j] = gravhalfy;
        kx2[j] = Vxhalf[i][j];
        ky2[j] = Vyhalf[i][j];
        //test = gravhalf*Yhalf[i][1];
        //cout << test << " " << kvy2[1] << endl;
}
    //Third
    for(j=1;j<m;j++){
        Xhalf[i][j] = X[i][j] + h/2.0*kx2[j];
        Yhalf[i][j] = Y[i][j] + h/2.0*ky2[j];
        Vxhalf[i][j] = Vx[i][j] - h/2.0*kvx2[j];
        Vyhalf[i][j] = Vy[i][j] - h/2.0*kvy2[j];}

    //Fourth

    for(j=1;j<m;j++){
    Rhalf[i][j] = sqrt(Xhalf[i][j]*Xhalf[i][j] +
         Yhalf[i][j]*Yhalf[i][j]);
    gravhalfx = 0.0;
    gravhalfy = 0.0;
       for(k=0;k<m;k++){

        if(k!=j){

           gravhalfx +=
               4*pi*pi*(Xhalf[i][j]-Xhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-Xhalf[i][k],
            gravhalfy +=
                4*pi*pi*(Yhalf[i][j]-Yhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-Xhalf[i][k],
            else{
            gravhalfx+= 0.0;
            gravhalfy+= 0.0;
}           }




        gravhalf = 4.0*pi*pi/(Rhalf[i][1]*Rhalf[i][1]*Rhalf[i][1]);
        kvx3[j] = gravhalfx;
        kvy3[j] = gravhalfy;
        kx3[j] = Vxhalf[i][j];
        ky3[j] = Vyhalf[i][j];
        //test = gravhalf*X[i][1];
        //cout << test << " " << kvx3[1] << endl;


    }


    //Fifth

    for(j=1;j<m;j++){
    Xhalf[i][j] = X[i][j] + h*kx3[j];
        Yhalf[i][j] = Y[i][j] + h*ky3[j];
        Vxhalf[i][j] = Vx[i][j] - h*kvx3[j];
        Vyhalf[i][j] = Vy[i][j] - h*kvy3[j];
        Rhalf[i][j] = sqrt(Xhalf[i][j]*Xhalf[i][j]
            +Yhalf[i][j]*Yhalf[i][j] );
               gravhalf =
                   4.0*pi*pi/(Rhalf[i][j]*Rhalf[i][j]*Rhalf[i][j]);


               gravhalfx = 0.0;
               gravhalfy = 0.0;
                 for(k=0;k<m;k++){

                  if(k!=j){

                    gravhalfx +=
                        4*pi*pi*(Xhalf[i][j]-Xhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-X
                    gravhalfy +=
                        4*pi*pi*(Yhalf[i][j]-Yhalf[i][k])*all_planets[k].mass/pow(pow(Xhalf[i][j]-X
                    else{
                    gravhalfx+= 0.0;
                    gravhalfy+= 0.0;
            }       }
```

```
                kvx4[j] = gravhalfx;
                kvy4[j] = gravhalfy;
                kx4[j] = Vxhalf[i][j];
                ky4[j] = Vyhalf[i][j];
                //test = gravhalf*X[i][j];
                //cout <<test << " " << kvx4[1] << endl;
}

    //Sixth
    for(j=1;j<m;j++){
        X[i+1][j] = X[i][j] + h/6.0*(Vx[i][j] +2.0*(kx2[j]+kx3[j]) +
            kx4[j]);
        Y[i+1][j] = Y[i][j] + h/6.0*(Vy[i][j] +2.0*(ky2[j]+ky3[j]) +
            ky4[j]);
        Vx[i+1][j] = Vx[i][j] - h/6.0*(kvx1[j] +2.0*(kvx2[j]+kvx3[j]) +
            kvx4[j]);
        Vy[i+1][j] = Vy[i][j] - h/6.0*(kvy1[j] +2.0*(kvy2[j]+kvy3[j]) +
            kvy4[j]);
        R[i+1][j] = sqrt(X[i+1][j]*X[i+1][j] + Y[i+1][j]*Y[i+1][j]);

}

    //cout << r[i] << endl;

//cout << R[i][1] << endl;

}

    ofstream myfile;
        myfile.open("proj3_earth1000_circle_n=6000_t=12_RK4_2plans.txt");
    for(i = 0; i<n;i++){
   cout << X[i][1] << " " << Y[i][1]
                << endl;
      //"  " << i << endl;
    }
//cout << r[n-1] << " " << x[n-1] << " " << y[n-1] << " " << vx[n-1] <<
    " " << vy[n-1] - 2*pi << endl;

    //---------------------------------------------------------------------------------------
    return 0;
}
```