Response Paper for 3rd Project

Jialin Liu
Thanaphong Phongpreecha

April 10, 2016

Abstract

In this report, we solve the position and velocity of each planet's orbit in the solar system, which is essentially coupled first order equations (with an assumption of circular orbit), by using Verlet and Runge-Kutta 4 method. To these approaches, we have investigated the significance of initial velocity, the stability of each function, testing the conservation of energy and angular momentum, and the effect of relativistic correction. Using distances from the Sun and trajectory, we found that Runge-Kutta 4 is more stable than Verlet as one would expect from $O(h^4)$ as opposed to $O(h^2)$. The total energy and angular momentum were confirmed to be conserved (with derivation) as they should be, although with a consistent fluctuation. The escape velocity was derived to be $2\sqrt{2}\pi$ and were proved to be correct by showing the Earth's escaped trajectory. The mass of, at least, Jupiter plays an important role in the stability of the system. And lastly, the relativistic correction resulted in prediction of precession of Mercury's perihelion.

1 Introduction

In this report, we arranged and named the results and discussion in order of the problem statement given due to numbers of questions asked. Presenting in this way would allow readers to check easier. Overall, this report investigate the application of Verlet and Runge-Kutta method to solve ordinary differential equation pertaining to the solar system (with a circular orbit assumed). In this report, we have 1) discretize the differential equations 2) write codes to solve those equations 3) investigating the limitation and advantages of each solving method 4) analyzing the resulting properties from the estimation, including energy, and momentum 5) investigating the effect of initial velocity and 6) exploring the perihilion precession of the Mercury by adding general relativistic correction.

Introduction to the Solar system problem

In solar system, the interaction between planets are defined by closed form expression of gravitational force under classical situation. The interaction can be expressed by a set of coupled partial differential equation. Although they can not be solved explicitly, by supplying the initial condition, their numerical solutions can be calculated using different approximation method. In general, most numerical solution methods are different way of estimating first and second order derivative at each point by manipulating Taylor expansion. Two of the most important and vastly used algorithms are Verlet method and Runge-Kutta method.

Introduction to Verlet

Most molecular dynamic program implemented Verlet method. It gives a truncation error of h^3.

We have the Taylor expansion of position:

$$x_{i+1} = x_i + hx_i^{(1)} + h^2 x_i^{(2)} + O(h^3),$$

where x is the position, h is the step length and i is the step index.

The velocity can be approximated to be:

$$v_{i+1} = v_i + \frac{h}{2}(v_{i+1}^{(1)} - v_i^{(1)}) + O(h^3)$$

Introduction to Runge-Kutta 4 (RK4)

The RK4 method is a numerical technique used to solve ordinary differential equation of the form:

$$\frac{dy}{dx} = f(x, y), y(0) = y_0$$

Essentially, the algorithm estimate the next point of y by employing a set of four slopes (k_1 - k_4). The first slope (k_1) is based on the initial point i. Using the slope of k_1, the k_2 is evaluated at half step ($x_{i+0.5}$). The k_2 is then used to estimate k_3 from the initial point x_i to point $x_{i+0.5}$. The forth slope is found using the same way that k_3 was found except using k_3 as the initial slope from point x_i to point x_{i+1} instead. After finding all these four k's, they are combined and weight-averaged to find y_{i+1}. The algorithm goes as:

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

where,

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$
$$k_3 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h)$$
$$k_4 = f(x_i + h, y_i + k_3 h)$$

A similar algorithm is used for velocity resulting in,

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

where,

$$k_1 = v_i$$
$$k_2 = v_i k_1 \frac{h}{2}$$
$$k_3 = v_i k_2 \frac{h}{2}$$
$$k_4 = v_i k_3 h$$

2 Methods

The codes were written separately, one in MATLAB (Verlet) and another in Python (RK4).

For RK4, object-oriented coding was used. Overall the code works as defining planets in a particular class (planet)–> adding planets to another bigger class (solarsystem) where other functions are defined –> Solver function is called which return values such as distances, positions, and energy –> the selected return values can be plot using function defined in the bigger class (solarsystem).

The code starts with defining a class "Planet", where each planets/orbiting objects' properties can be defined.

```
class planet:

    def __init__(self, name, mass, x, y, z, vx, vy, vz):
        self.name = name
        self.mass = mass
        self.x = x
        self.y = y
        self.z = z
        self.vx = vx
        self.vy = vy
        self.vz = vz
```

Following the planet class, each of these planets can be compiled and interact together by adding to another class "s+olarsystem", which contain many functions within it, below we give an example of 1) adding planets 2) showing planets 3) compile all planets' properties into one matrix 4) finding potential energy and 5) plotting potential energy. There are many more functions defined within the class solarsystem.

```
class solarsystem:

    count = 0

    def __init__(self):
        self.planets = []

    def add(self, a_planet):
        self.planets.append(a_planet)
        solarsystem.count += 1

    def compilePlanet(self):
        y_i = np.zeros([solarsystem.count,7])
        ii = 0
        for a_planet in self.planets:
            y_i[ii,6] = a_planet.mass
            y_i[ii,5] = a_planet.vz
            y_i[ii,4] = a_planet.vy
            y_i[ii,3] = a_planet.vx
            y_i[ii,2] = a_planet.z
```

```
            y_i[ii,1] = a_planet.y
            y_i[ii,0] = a_planet.x
            ii += 1
    return y_i

    def potential(self, vec, i):
        G = 4*np.pi**2
        PE = -2*G*vec[i,6]/(vec[i,0]**2+vec[i,1]**2
                +vec[i,2]**2)**0.5
        return PE

    def PEPlot(self, PE, h, tmax):
        plt.figure()
        m = range(int(tmax/h)+1)
        for i in range(solarsystem.count):
            if i != 0:
                plt.plot(m, PE[i]+PE[0])
        plt.xlabel("Time steps", **axis_font)
        plt.ylabel("Potential Energy", **axis_font)
        plt.title("Potential Energy", **title_font)
        plt.tick_params(axis = 'x', labelsize = 15)
        plt.tick_params(axis = 'y', labelsize = 15)
        plt.show()
```

The solver code algorithm follows that of RK4 as introduced in the introduction, then the returned positions and velocities at each point are used to calculated the properties, such as energy, momentum, etc.

```
def solverRK4(h, tmax):
    y_temp = np.zeros([solarsystem.count, 7])
  position = np.zeros((solarsystem.count,7,int(tmax/h)+1))
    KE = np.zeros((solarsystem.count, int(tmax/h)+1))
    PE = np.zeros((solarsystem.count, int(tmax/h)+1))
    L = np.zeros((solarsystem.count, int(tmax/h)+1))
   distance = np.zeros((solarsystem.count, int(tmax/h)+1))
    k1 = np.zeros([solarsystem.count, 7])
    k2 = np.zeros([solarsystem.count, 7])
    k3 = np.zeros([solarsystem.count, 7])
    k4 = np.zeros([solarsystem.count, 7])
    y_i = SS.compilePlanet()
    t = 0.0
    n=0

    while t < tmax:
        derivative(y_i, k1, solarsystem.count)
        sum_matrix(y_temp, 1, y_i, 0.5*h, k1,
        solarsystem.count)
        derivative(y_temp, k2, solarsystem.count)
        sum_matrix(y_temp, 1, y_i, 0.5*h, k2,
```

```
        solarsystem.count)
        derivative(y_temp, k3, solarsystem.count)
        sum_matrix(y_temp, 1, y_i, h, k3,
        solarsystem.count)
        derivative(y_temp, k4, solarsystem.count)

        for i in range(solarsystem.count):
            if i!=0:
                for j in range(6):
                    y_i[i,j] = y_i[i,j] + h*(k1[i,j]
                    + 2*k2[i,j] + 2*k3[i,j] + k4[i,j])/6
                    position[i,j,n] = y_i[i,j]
                    distance[i,n] = SS.distance(y_i,i)
                    KE[i,n] = SS.kinetic(y_i,i)
                    PE[i,n] = SS.potential(y_i,i)
                    PE[0,n] = SS.potential(y_i,i)
                    /y_i[i,6]*y_i[0,6]
                    if i != 0:
                        L[i,n] = SS.angMomentum(y_i,i)
        t += h
        n += 1

# fixing minor bugs
    position[:,:,int(tmax/h)] = position[:,:,int(tmax/h)-1]
    KE[:,int(tmax/h)] = KE[:,int(tmax/h)-1]
    PE[:,int(tmax/h)] = PE[:,int(tmax/h)-1]
    distance[:,int(tmax/h)] = distance[:,int(tmax/h)-1]

    return [position, KE, PE, distance, L]
```

Verlet method in Matlab followed same flow chart with RK4. In order to understand all the algorithm, none of advanced Matlab functions are used in Matlab. Only those can be found in native python are used in the code. Also, Verlet method is implemented under objective orientation scheme.

3 Results and Discussion

3.1 Problem A

Rewrite the following second order differential equations:

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{Earth}}$$

$$\frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{Earth}}$$

into first order differential equations:

$$\frac{dx}{dt} = v(x,t), \frac{v}{dt} = F(x,t)/m = a(x,t) = \frac{GM_{sun}}{r_x^2}$$

$$\frac{dy}{dt} = v(y,t), \frac{v}{dt} = F(y,t)/m = a(y,t) = \frac{GM_{sun}}{r_y^2}$$

To discretize the above equation and implement Verlet method:
We know,

$$x_i^{(1)} = v_i, x_i^{(2)} = a_i = \frac{GM_{sun}}{r_x^2}.$$

Substitute them into the equation in Introduction part, we have:

$$x_{i+1} = x_i + hv_i + h^2 \frac{GM_{sun}}{r_{xi}^2} + O(h^3),$$

and

$$v_{i+1} = v_i + \frac{h}{2}(v_{i+1}^{(1)} - v_i^{(1)}) + O(h^3)$$

3.2 Problem B

The verlet algorithm is implemented using Matlab with object orientation programming. Two classes, planet and solarSystem, are created. The planet class is used for storing initial position, velocity and mass of different planets. The main algorithm is implemented in solarSystem class, including add planet to solar system, settings of simulation parameters, calculate the trajectory of planets and various output. In the future, to have a better data flow and programming logical, we should store all planet related information in planet class such as the position and velocity at each step.

The detail of the code for Runge-Kutta 4 can be found in the method part. But in short, we attempted to write it in a object-oriented manner. The classes used include "Planet" where planets/suns/other orbiting objects can be added. The other class is SolarSystem, where the planets added from class Planet can be interacted to calculate force, potential energy, plotting, and many more. The function defined is Runge-Kutta 4 which is called upon the SolarSystem class.

3.3 Problem C

Which initial velocity gives a circular orbit?

In order to achieve and maintain a circular orbit, we know that the Earth-Sun gravitational force and the Earth's centripetal force are equivalent at any given direction, position, and time. From,

$$v_{Earth} = \frac{distance}{\Delta t} = n\frac{2\pi R}{\Delta t}$$

given that the Sun-to-Earth distance is equal to 1 A.U. and n is a number of year. Then for a period of 1 year,

$$v_{Earth} = 2\pi$$

Since, the velocity of the Earth in circular orbit is constant (explained in latter subsection), the initial velocity must also be 2π. Figure 1 shows the circular path of Earth's orbit around the Sun, although when zoom in we found that the distance is constantly fluctuating around 0.999, which is an acceptable value.

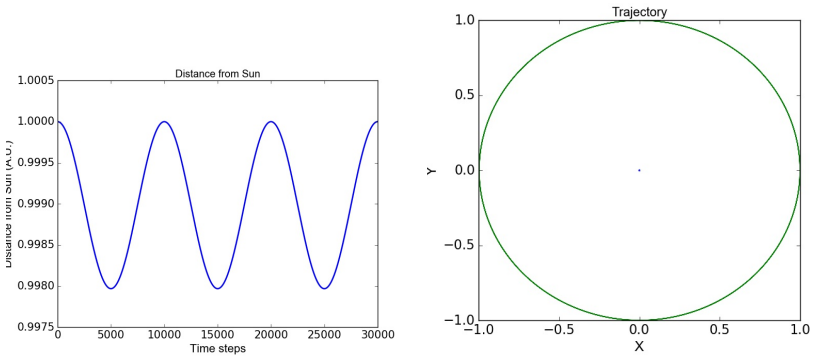

Figure 1: Distances from Earth to Sun and its orbit in a 3 year cycle using time step of E-04.

Testing the stability of the algorithm as function of different time steps Δt

For Runge-Kutta 4:

From Figure 2 the step size that causes visible deviation in 100 year period is at h = 0.03. The step of 0.01 seems to be fluctuating at a certain constant, a similar behavior we have seen throughout other more stable step sizes. The effect become more pronounced when increase the step to 0.05. The trajectory of the earth clearly becomes thickens from the accumulated error. As RK4 has a global error of $O(h^4)$, we will see that the lease amount of error accumulated, i.e. not taken into account other derivation operation, over a hundred year is 0.0125, 0.0027, and 0.0001, a huge difference for h = 0.05, 0.03 and 0.01, respectively.

Apparently, this implies that the step size of 0.01 also accumulates some tiny amount of error. Although it is not visible within 100 years, expanding it to 10,000 years clearly shows that the Earth's orbit does deviate, although very slowly (Figure 3). This insinuates that the stability depends on both step size and total amount of years. No matter what step size, an infinitesimal amount of error will accumulate, and it is only a matter of how many years are to be projected to see if those error will be visible.

For Verlet:

In order to compare with RK4, same time step and simulation time are used. Verlet have a lower order glabal error of $O(h^2)$ comparing with RK4, so a smaller time step of 0.001 is also used. The trajectory of different time step is shown in Figure 4. The thickness of the trajectory gradually increased with increasing of time step.

Investigating kinetic energy, potential energy, and angular momentum

The Law of Conservation of Energy states that the total energy of a closed system remains constant. In the case of Earth-Sun system. Figure 5 shows the kinetic, potential, and total energy of Earth and Sun. It can be seen that these values are constant and conserved, although fluctuating a tiny bit. This fluctuating issue is exclusive to Runge-Kutta 4 method and persists even if we lock

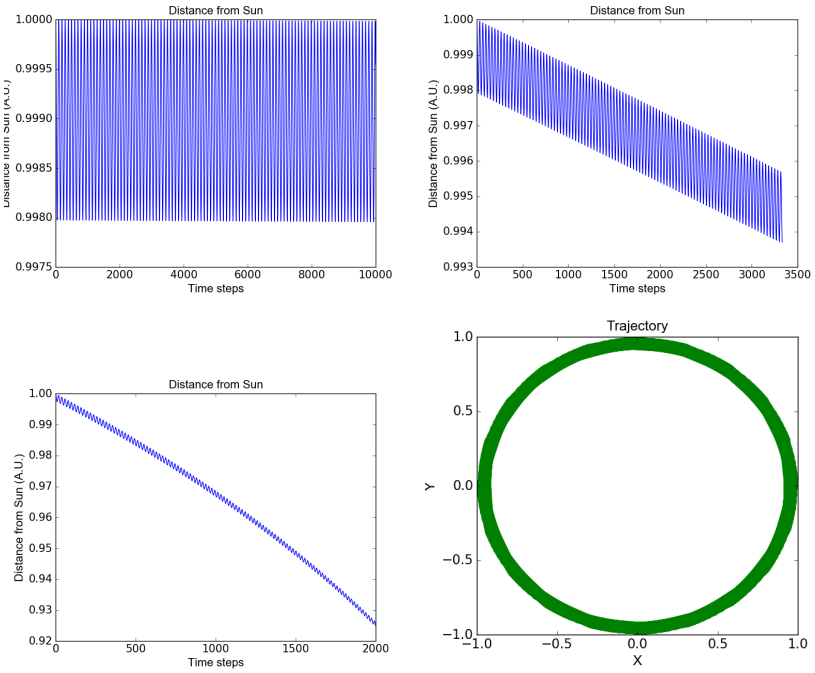

Figure 2: Distances from the Sun in different step sizes (top left = 0.01, top right = 0.03, bottom left = 0.05 for the period of 100 years. The bottom right picture is the destabilized orbit of the time step 0.05.

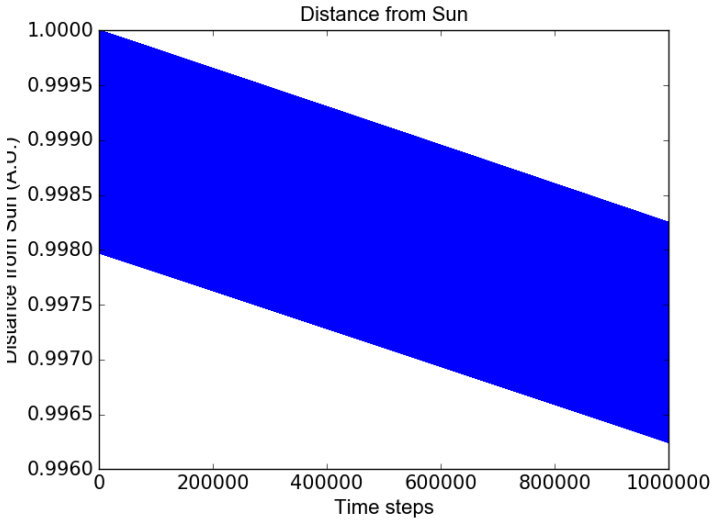

Figure 3: Distances from the Sun in different step sizes of 0.01 for the period of 10,000 years.

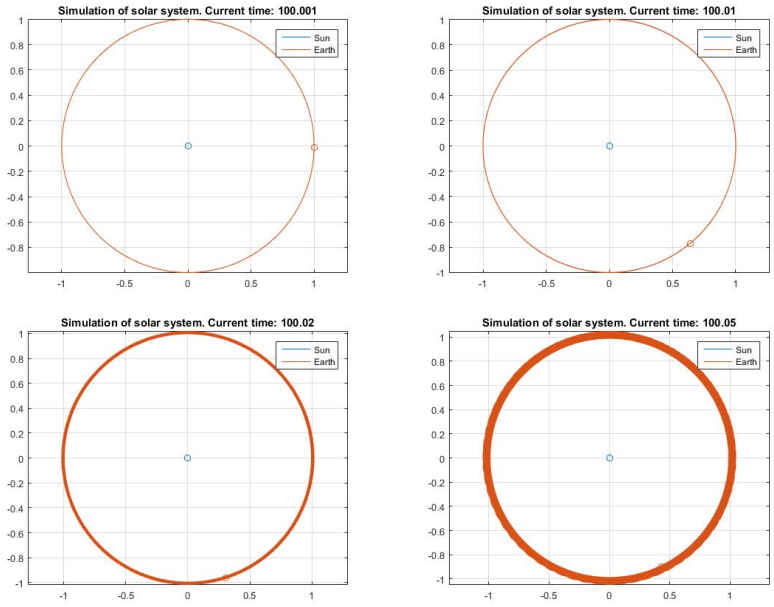

Figure 4: Trajectory of planets with different time step (top left = 0.001, top right = 0.01, bottom left = 0.03, bottom right = 0.05) for the period of 100 years.

the Sun position at [0,0,0]. The total energy, and angular momentum, as with other isolated systems, remained constant. The loss in kinetic energy results in equivalent increase in potential energy counterpart. For this particular case, a simple explanation to why these values are conserved is because the circular path suggests that at any point the centripetal force and the gravitational force towards the Sun are equivalent, i.e.,

$$\frac{GM_1M_2}{R^2} = \frac{M_2v^2}{R}$$

As radius and masses are constant, so does v. This ultimately makes the kinetic energy of Earth in this ideal circular orbit a constant. The constant R also suggests constant potential energy.

For verlet, the potential energy, kinetic energy and total energy as function of time when time step = 0.001 is shown in Figure 6. Also, the absolute error accumulated in 5 years on total energy is for different time step is summarized. Since the position of Sun is not fixed, the position of Sun will also move in a circle. And the kinetic and potential energy will oscillate in opposite phase. The total energy is increasing very slow (1E-5 relative error in 5 years), which can be treated as a constant. The absolute error increased with increasing of time step.

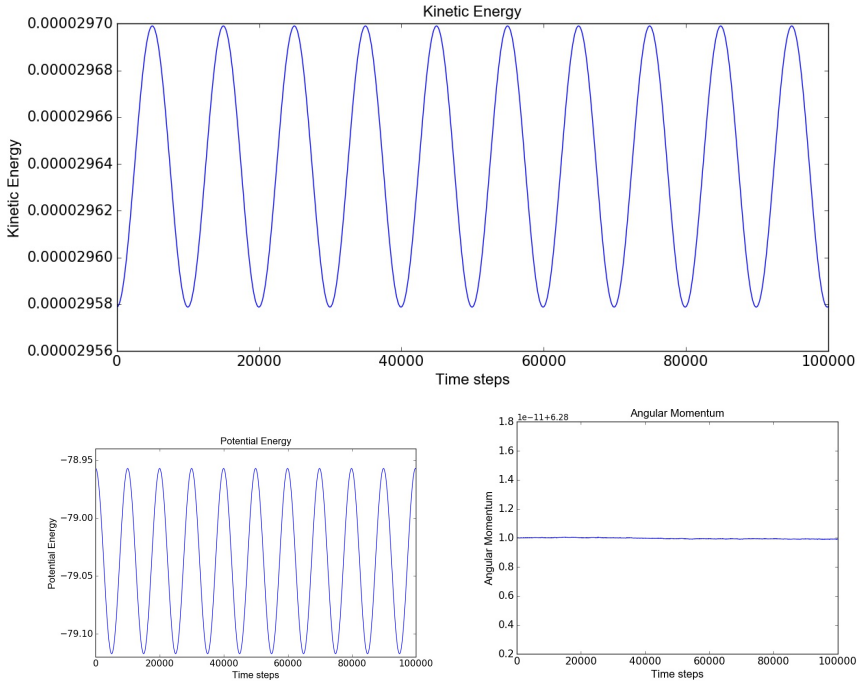

Figure 5: Total kinetic and potential energy (top and left corner) of the Earth and Sun combined.Angular momentum of the Earth (right corner). These are done with step size of 0.0001 and for a period of 10 years.

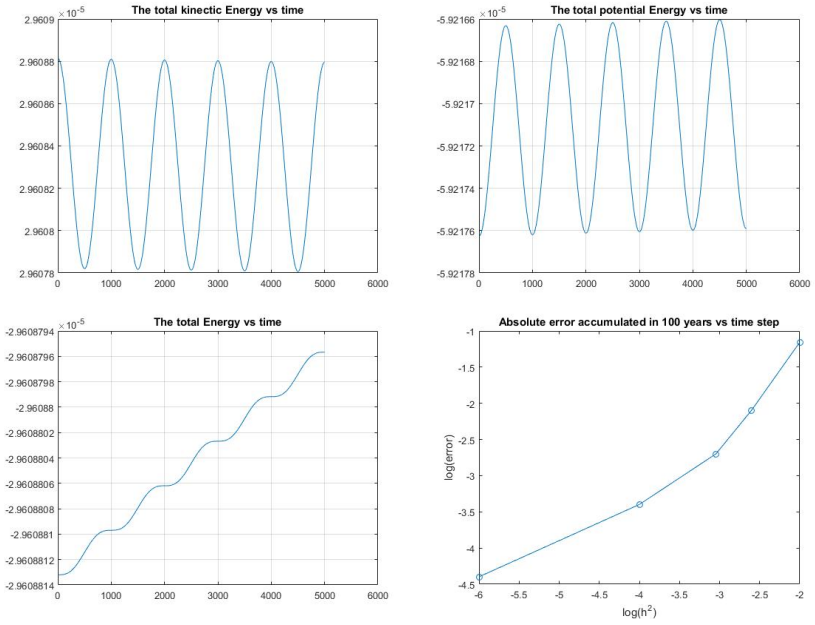

Figure 6: Total kinetic and potential energy (top and left corner) of the Earth and Sun combined.Angular momentum of the Earth (right corner). These are done with step size of 0.0001 and for a period of 10 years.

3.4 Problem D

Find out by trial and error what initial velocity is needed for Earth to escape from the sun

By varying the initial velocity from 2π to 1.5 times of 2π, the escape velocity is approximated between 1.4 and 1.5 times of 2π. The trajectories are show in Figure 7.

Finding an exact answer

Given a conservation of energy,

$$\frac{M_2 v_1^2}{2} - \frac{GM_1 M_2}{R} = 0$$

Rearranging the equation gives,

$$v_1 = \sqrt{\frac{2GM_1}{R}}$$

For Earth-Sun case, this would be,

$$v_1 = \sqrt{\frac{2(4\pi^2)(1)}{1}} = 2\sqrt{2}\pi$$

Therefore, $v_{escape} > 2\sqrt{2}\pi$.

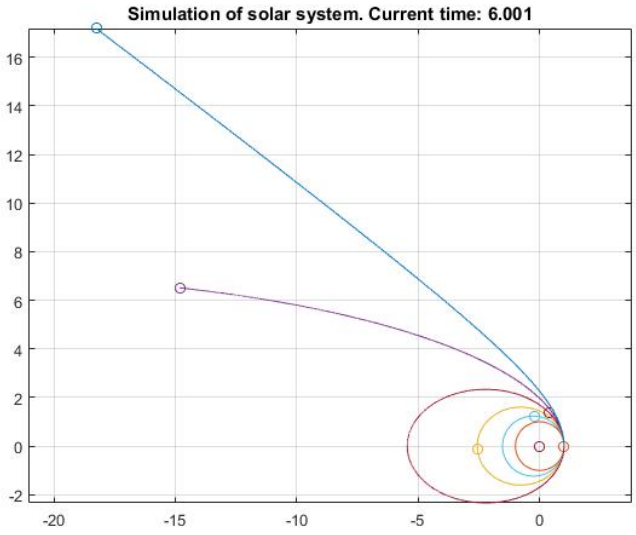

Figure 7: The trajectory of earth with different initial velocity. From inside to outside: 1.0, 1.1, 1.2, 1.3, 1.4 and 1.5 times of 2π

3.5 Problem E

For Runge-Kutta 4:

The stability of the Earth-Sun-Jupiter system is shown roughly in a form of trajectory (although an exact distance from origin should be plotted, due to huge difference in distance, this inhibits a clear distinction). In figure 8 the mass of Jupiter is according to the real mass, which is 317 folds that of Earth's. It can be seen that for a period of 100 years, the step size of 0.001 or 0.01 does not yield a significantly different trajectory. However, it is to be expected that if the period is prolonged, the step size of 0.01 will certainly shoot off first. It is also noteworthy that even with the step size of 0.001, the orbit changed a bit in every year from the fact that the line become (very) slightly thickened.

Using the step size of 0.001, which resulted in considerable stability in Figure 8 top left, increasing the mass of Jupiter by 10x shows an obvious effect in destabilizing the orbits of both Earth, Sun, and Jupiter (Figure 9 (left)). Further increasing it to 1000x totally shoot off the Earth from the system (Figure 9 (right)).

It is interesting to see that the error expedited when the mass of Jupiter increased. It is known that the accumulate error is a function of $O(h^4)$. This implies that the coefficient in front of the error term is also a function of mass.

To further investigate the effect of Jupiter mass and step size. We plotted the trajectory for even a smaller step size (0.0001) for a 10x Jupiter's mass (Figure 10 (left)) to see if we can pull it back to stabilization. Nonetheless, this is to no avail, the trajectory is pretty much the same when using step size of 0.001 (Figure 9 (left)). Interestingly, not only increasing the mass can destabilize the orbits, reducing the mass has the same effects. Although using the same step size that was previously stable (0.01) in Figure 8 (top right), reducing the mass

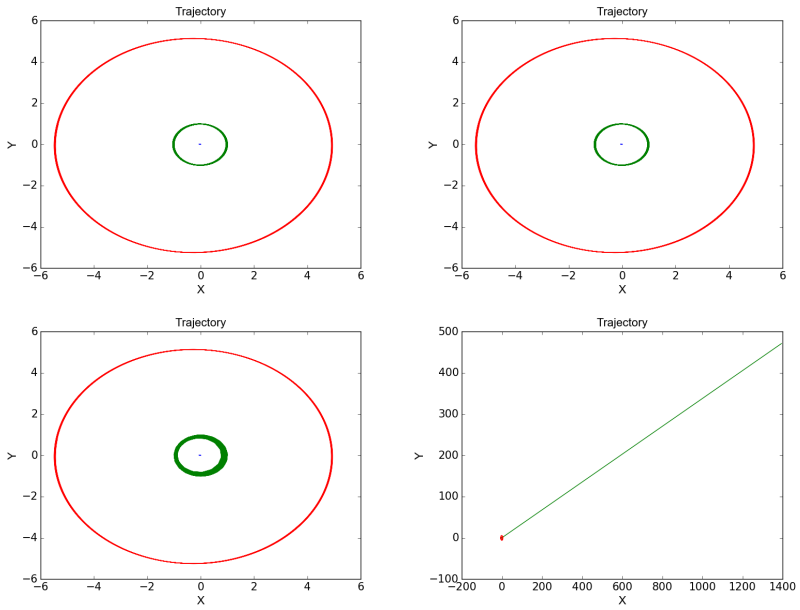

Figure 8: Trajectory of Earth, Jupiter, and Sun (too small to observe), in a period of 100 years using a step size (h) of 0.001 (top left), 0.01 (top right), 0.055 (bottom left), and 0.06 (bottom right).

by 10x causes it to stabilize too (Figure 10 (right)).

For verlet method: The interaction between Jupiter and the other two star is added in solarSystem class in verlet method. The list of all stars is stored inside solarSystem class. The calculation of acceleration is done in a for loop to sum up the gravitational force from other stars.

To test the stability of verlet method in three body problem, different time steps are used to generate the trajectory. The absolute change in energy in 100 years is plotted as a criterion for stability. As shown in Figure 11, when time step is increased, the error also increased linearly with time step. However, even with time step of 0.01, the orbits didn't deviate much from perfect circle in 1000 years as shown in Figure 11 upper left.

3.6 Problem F

Adding More Planets

Using a real data obtained from `http://ssd.jpl.nasa.gov/horizons.cgi#top`, a 3D plot of Sun, Mercury, Earth, and Mars is shown in Figure 12. Other planets are omitted for aesthetic reasons. The same initial position and velocity is also used in Verlet method. The top view and side view of trajectory for all 9 planet in 50 years is shown in Figure 13. We can see the orbit plane of Pluto is much different than the other 8 planets. Due to the limitation of simulation time, Uranus, Neptune and Pluto are not able to complete one cycle.

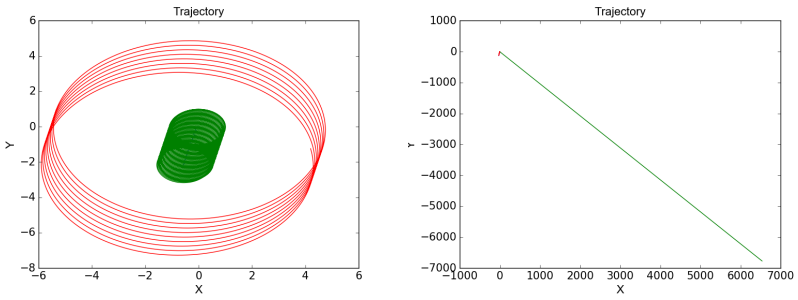

Figure 9: Trajectory of Earth, Jupiter, and Sun, in a period of 100 years using a step size (h) of 0.001 with a 10 time and 1000 time mass of Jupiter (left).

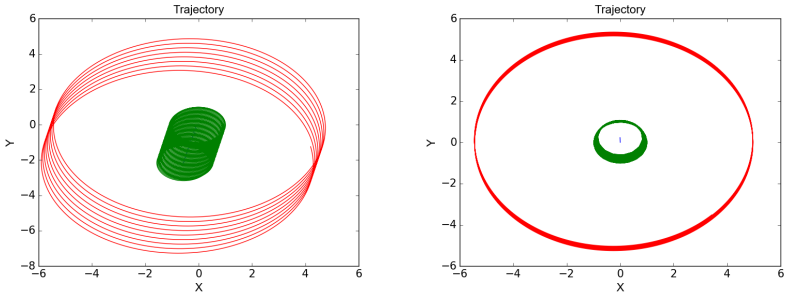

Figure 10: Trajectory of Earth, Jupiter, and Sun, in a period of 100 years using a step size (h) of 0.0001 for a Jupiter mass of 10x (left) and step size of 0.055 for a Jupiter mass of 0.1x (right).

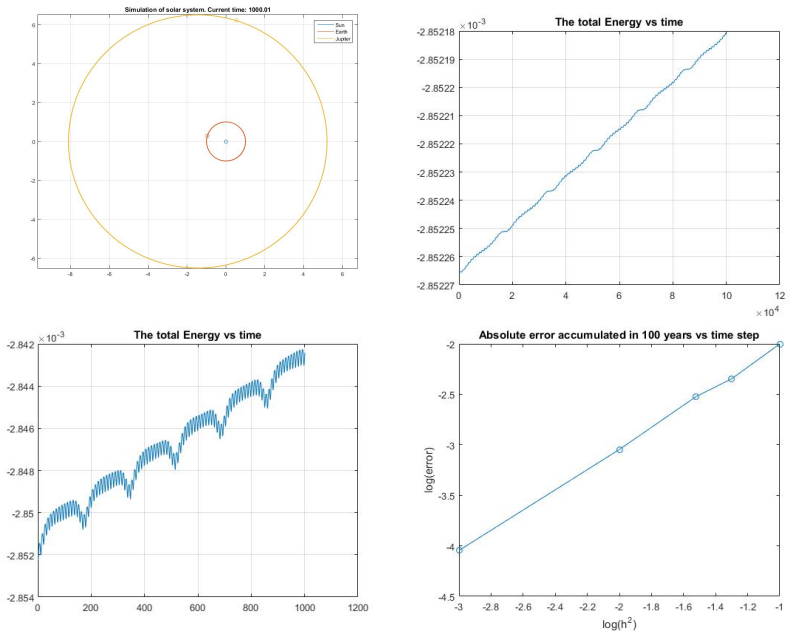

Figure 11: Trajectory of Earth, Jupiter, and Sun, in a period of 100 years using a time step (h) of 0.001. The energy change of h = 0.001 and h = 0.1 and the absolute error as a function of time step in log scale

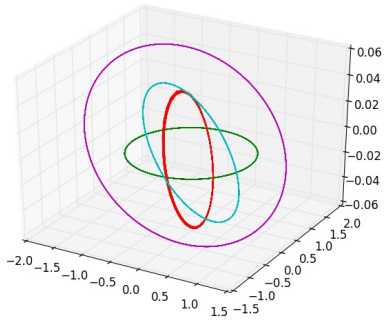

Figure 12: Orbits of Mercury, Earth, Venus, and Mars in 3D.

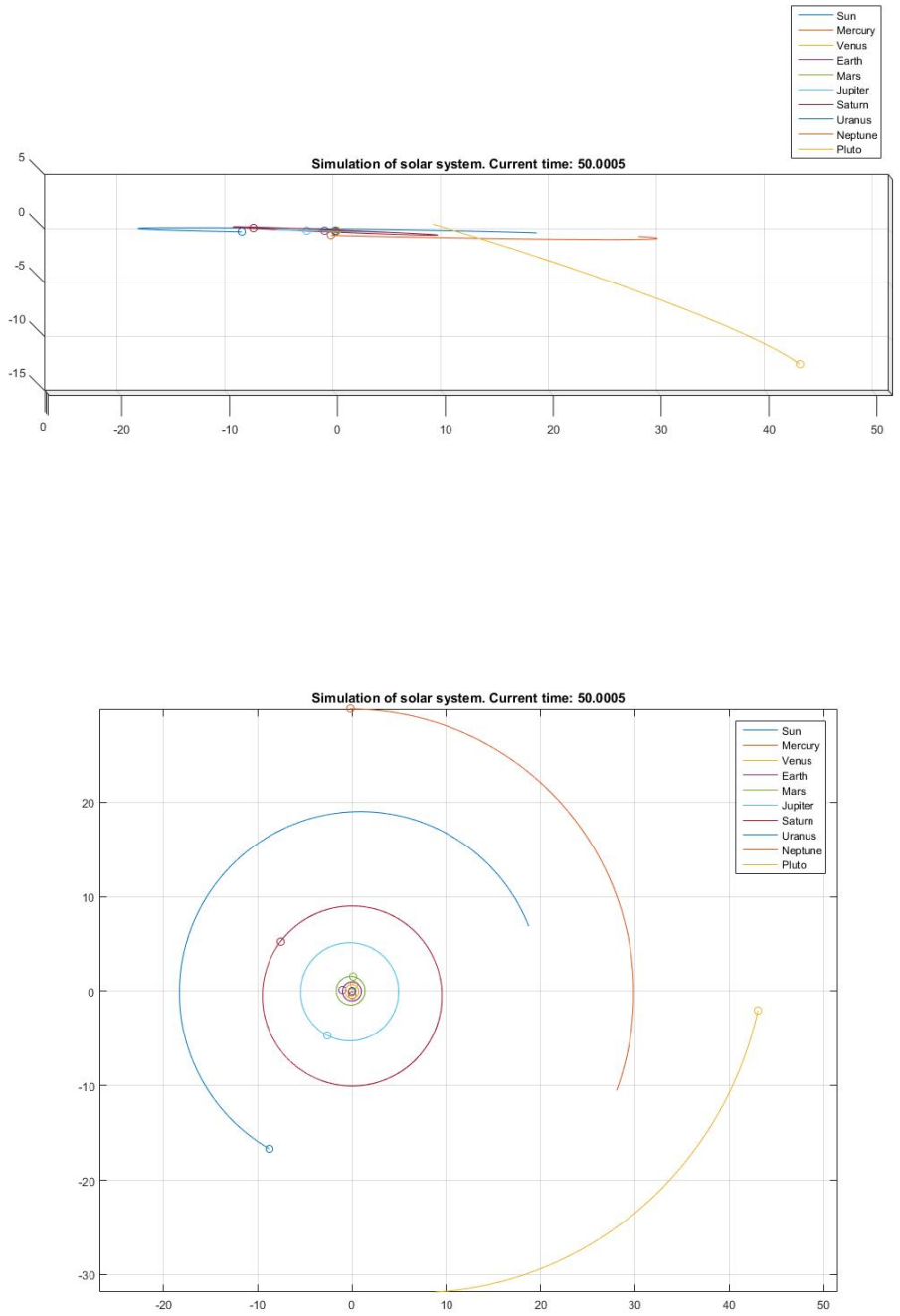

Figure 13: Side view and top view of trajectory of 9 planets calculated using Verlet method. Time step is 0.0005. Simulation time is 50 years

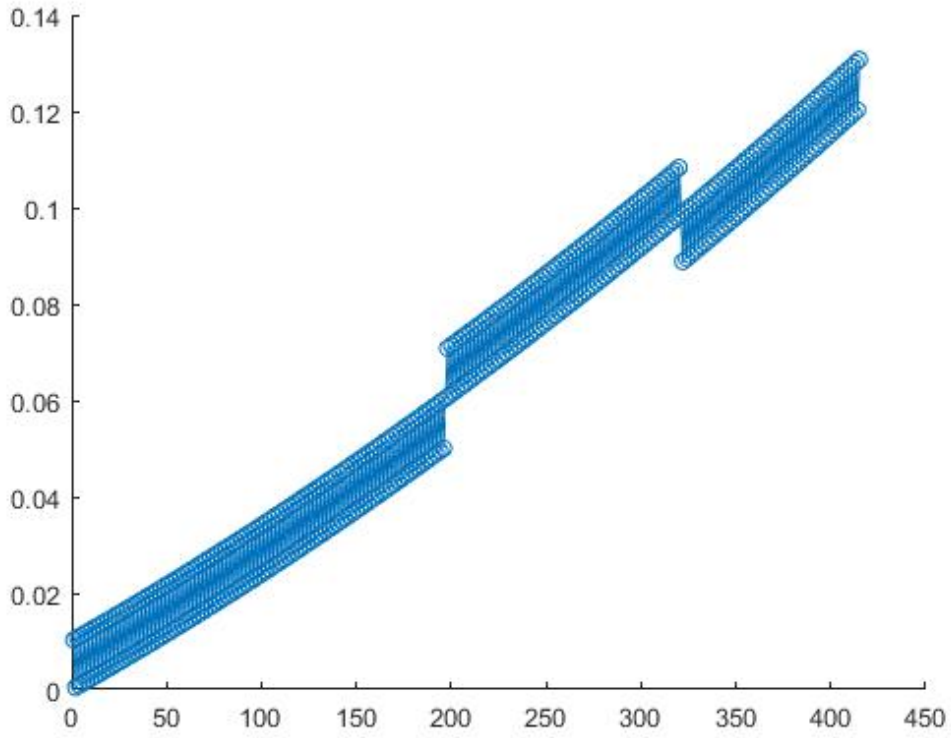

Figure 14: Perihelion angle of Mercury in a century. X axis is Mercury year.

3.7 Problem G

The relativistic correction is implemented by simply modify the interaction evaluation function in verletRT method in solarSystem class. Because the distance between Mercury and Sun is very small, a smaller time step is required. To decide a proper time step, a few running of only 5 years with different time step is tested. When time step is larger than 1E-5, the θ_p change is relative small. So time step = 1E-5 is used to plot θ_p as function of Mercury years in 100 Earth years. The results are shown in Figure 14.

4 References

1. Teodorescu, Petre P. Mechanical Systems, Classical Models: Volume 1: Particle Mechanics. Springer Science & Business Media, 2007.
2. https://www.nasa.gov/