Analysis and Comparison of the Efficiency and Accuracy of Two Algorithms in N-body Problem

Changlin Su, Yao Kuang
CSC336 Optional Course Project

1 Introduction

N-body problem is an important problem in classic physics. This problem originated from the understanding and prediction of planetary trajectories. Formally, it can be state that

> Calculate the interactive forces of a group of objects and predicate their motion in the future, by its current orbital properties.

The two body problem ($n = 2$) has been solved completely. However, it is far more complex for the case that $n \geq 3$.

In this project, we try to use two different algorithms, velocity verlet integration method and Runge-Kutta 4th order method, to perform numerical simulation and numerical calculation on n-body problem, and then analysis and compare the efficiency and accuracy of the algorithms.

We designate the solar system as our simulation object, which contains one star, the Sun, and eight planets bound to it by gravity. So, we will simulate the motion of nine bodies numerically ($n = 9$).

We need orbital properties of these night bodies, including mass, aphelion and orbital velocity on aphelion (minimum orbital velocity) of planets. These data are referred from NASA's website.

In addition, there is many limitations in our simulation, so it cannot show the real universe. At first, due to the lack of data (real-time location), we assume that the aphelion points of all planets are on the same straight line. So we can regard the sun as the origin, and then the initial positions of all planets are on the x-axis, with the coordinates of the aphelion. Then, for convenience, our simulation will be on the plane without considering the Z axis, and our simulation will ignore the influence of relativity.

After the numerical simulation, the running time of two algorithms for simulating the motions in the same period of time will be the efficiency criterion, and the energy change of the entire system will be the accuracy criterion.

2 Mathematical models

The heart of modeling trajectory of objects in a N-body problem is the analysis of newton's second law. Finding the forces that is acting on the objects are crucial to the accuracy of our solution.

The main forces acting on the planets is the universal gravitation, which can be calculated by Newton's law of universal gravitation:

$$F_g = \frac{Gm_1m_2}{r^2}$$

where m_1 and m_2 are the mass of the objects and G is a universal physical constant and r is the distance between the two objects.

Then we can expand our formula to calculate the forces of N other bodies acting on our target:

$$\vec{F_g} = \sum_{i=1}^{n} \frac{Gm_tm_i}{\vec{r}^2}$$

where m_t denote the mass of our target and m_i denote the mass of the other N bodies.

Now, we have calculated the forces acting on a body, and we need to study it's effect on that body's trajectory, which can be calculated by Newton's second law:

$$\vec{F} = m\vec{a}$$

Let $\vec{a}(t)$ denote the acceleration at time t, and let $\vec{v}_0$ denote our initial velocity. We can apply calculus to calculate the velocity of the object at time t:

$$\vec{v}(t) = \vec{v}_0 + \int \vec{t}\vec{a}(t)dt$$

and finally the displacement of the object:

$$\vec{x} = \int \vec{v}(t)dt$$

Since we are performing numerical integration, the process of integration is discrete. The displacement equation becomes:

$$\vec{x} = \sum (\vec{t}v_0 + \sum \vec{a}(t)\Delta t)\Delta t$$

So if we add x with our initial coordinate x_0, we will obtain our new coordinate after a small time slice Δt by these formulas.

3 Numerical/Computational Methods

3.1 Algorithm I: Velocity Verlet Integration Method

The verlet integration methods arise from the original newton's equation. It is an symplectic integrator, this means that this integration algorithm is designed to seek solution for Hamiltonian systems. A Hamiltonian system has a well know property of conserving at least one of its quantity. i,e, one of the property stays constant in the system. In the domain of our problem, the total energy of the system (kinetic + potential energy). In other words, ideally the verlet integration will simulate a trajectory that conserves total energy in our solar system. A simple varlet integration iteration consists of 3 main steps: The first step is to calculate the objects position $\vec{x}(t + \Delta t)$ from the velocity data $\vec{v}(t)$:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2$$

Then, we will use the updated position $\vec{x}(t + \Delta t)$ and universal law of gravity to obtain the acceleration $\vec{ta}(t + \Delta t)$ at the new time step. And finally, we will update the velocity $\vec{v}(t + \Delta t)$ at the new time step with the averages of acceleration estimates on both the old time step and new time step:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{1}{2}(\vec{a}(t) + \vec{a}(t + \Delta t))\Delta t$$

By performing these updates iterative, we are able calculate an object's trajectory under the gravitational influence of other objects in space. And since verlet integration is a symplectic integrator, it will conserve total energy better than other non-symplectic integrator of the same order.

3.2 Algorithm II: Runge-Kutta 4th Order Method

Runge-Kutta algorithm is a 4th order integration method is a integration algorithm developed by German mathematicians Carl Runge and Wilhelm Kutta in the 1900s. From now on, we will refer to this method as RK4 in the report. Let $\frac{dy}{dx} = f(t, y)$ and $y(t_0) = y_0$. For each iteration of the RK4 the steps are shown below:

$$y(t + \Delta t) = y(t) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1 \Delta t}{2}))$$

$$k_3 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2 \Delta t}{2}))$$

$$k_4 = f(t_n + \Delta t, y_n + k_3 \Delta t))$$

In our implementation, since we are solving 2nd order differential equations, we uses RK4 method two times to solve for velocity and position. Note that it is possible to convert the this second order differential equation to a system of first order equations and solve them more efficiently. Below is a pesudo code for the RK4 method in our implementation, for simplicity, we are only showing the calculation for only one dimension:

```
procedure Single_RK4_Step(position, velocity, dt):
    k1a = compute_acceleration(position)
    k1v = velocity_x

    k2a = compute_acceleration(position + (dt/2) * k1v)
    k2v = velocity + (dt/2) * k1a

    k3a = compute_acceleration(position + (dt/2)*k2v)
    k3v = velocity + (dt/2) * k2a

    k4a= compute_acceleration(position + dt*k3v)
    k4v = velocity + dt * k3a

    new_position = position +
                    (dt/6)*(k1v + 2*k2v + 2*k3v + k4v)

    new_velocity = velocity +
                    (dt/6)* (k1a + 2*k2a + 2*k3a + k4a)
```

4 Simulation Model and Data

In this project, we will use two algorithm, which are introduced in Section 3, to perform numerical simulation on n-body problem ($n = 9$). The simulation object of our project is the sun and its eight planets.

For convenience, and the lack of real-time data, we will use a simplified model of the solar system. We use a two-dimensional coordinate system to show the trajectories of eight planets. In the initial state, the sun is on the origin and eight planets is on the x-axis, with the coordinates of the aphelion. And we let the velocity of the sun be 0 and that of others be their minimum orbital velocity respectively. Therefore, all data we need is their mass, minimum orbital velocity and aphelion.

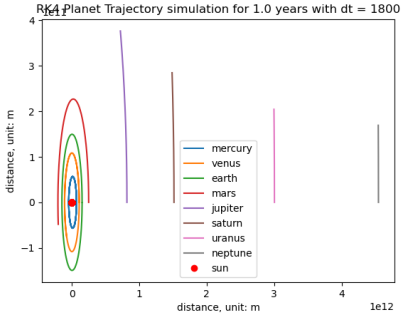
Then we do the simulation with the time slice one hour and half an hour, $\Delta t = 3600s$ and $\Delta t = 1800s$. And we simulate the trajectories of these planets in 1 year and 165 years, which are the orbital periods of the earth and Neptune, the planet farthest from the sun among the eight planets, respectively.

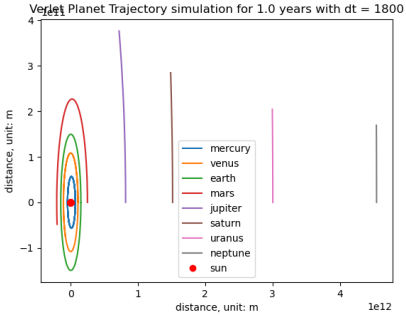
After the simulations, we will use the time cost to test the efficiency of two algorithms and use change of energy to test the accuracy.

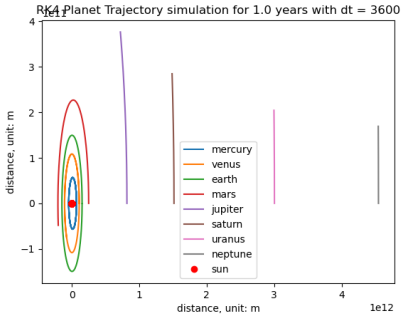
5 Results and Analysis

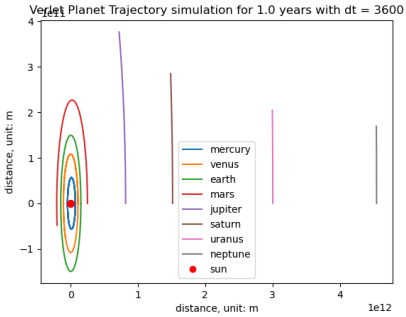
5.1 Simulations Results

- 1 year

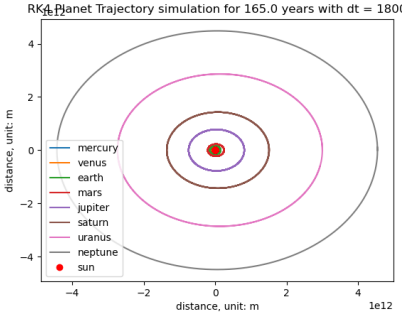


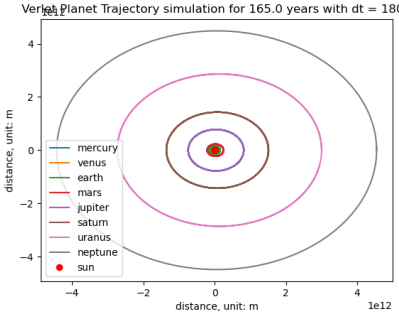


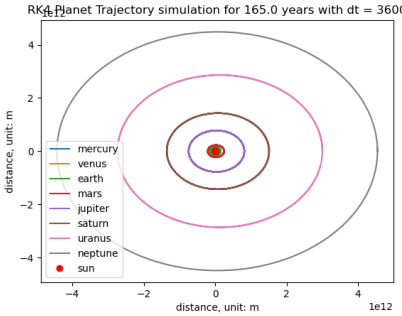


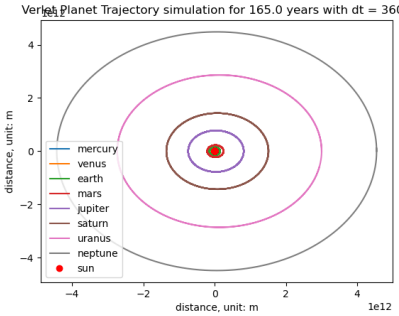


- 165 year


RK4 Planet Trajectory simulation for 165.0 years with dt = 1800
1e12
distance, unit: m
distance, unit: m
mercury
venus
earth
mars
jupiter
saturn
uranus
neptune
sun


Verlet Planet Trajectory simulation for 165.0 years with dt = 1800
1e12
distance, unit: m
distance, unit: m
mercury
venus
earth
mars
jupiter
saturn
uranus
neptune
sun


RK4 Planet Trajectory simulation for 165.0 years with dt = 3600
1e12
distance, unit: m
distance, unit: m
mercury
venus
earth
mars
jupiter
saturn
uranus
neptune
sun


Verlet Planet Trajectory simulation for 165.0 years with dt = 3600
1e12
distance, unit: m
distance, unit: m
mercury
venus
earth
mars
jupiter
saturn
uranus
neptune
sun

5.2 Numerical Results

- Time Cost

Simulation for 1 year

	$\Delta t = 1800s$	$\Delta t = 3600s$
Verlet	2941ms	1466ms
RK-4	10015ms	5013ms

Simulation for 165 years

	$\Delta t = 1800s$	$\Delta t = 3600s$
Verlet	144646ms	243771ms
RK-4	1658613ms	828978ms

- Relative Energy Change

Simulation for 1 year

	$\Delta t = 1800s$	$\Delta t = 3600s$
Verlet	0.69806 %	0.69764 %
RK-4	0.69847 %	0.69847 %

Simulation for 165 years

	$\Delta t = 1800s$	$\Delta t = 3600s$
Verlet	3.79719 %	3.79933 %
RK-4	3.79507 %	3.79507 %

5.3 Analysis

From the previous results, we confirm that our simulation of the solar system is stable, we see that there are no significant drift in planetary orbit for our simulation. Another proof that our alorithm is implemented correctly is that if the time step is halved, the total simulation time increases by a factor of 2.

The result produced by the verlet method and RK4 method agree with each other on the trajectory of the planet. However, if we look at the time it takes for each algorithm, we find that RK4 took 3 times longer on average to compute a solution compare to the verlet method. This is reasonable since the RK4 method will calculate the acceleration at 4 different position to estimate the final acceleration.

In the experiment, the accuracy of the solution is measured by the difference in total energy between initial and final state, since the total energy in a closed system should not change. Note that the small relative error does not mean that the error is small, it just means the total energy at the inital state is very large compare to the error we have. The relative change in energy shows some very interesting result: The changes in step size seems to have Little effect on

the accuracy of the solution for the RK4 methods. In fact, percentage change for RK4 method is only noticeable after 6 digit behind decimals (decrease with smaller time step). We suspect that this is may be due to numerical errors (truncate and rounding error) accumulate through the integration process since in our model we are working with very large floating point numbers (around 10^{24}). Compare to RK4 method, the verlet method performs more reasonable, as the time step for the method decrease, we can see a decrease in relative energy change. This suggest that decreasing the time step will yield a significant change compare to RK4 method.

Although RK4 is a 4th order method and verlet is a 2nd order method, the simulation accuracy for both of the method are nearly the same. We suspect that this is due to the physical property of the problem we are working with. The system we are trying to simulate is a Hamiltonian system where the total energy does not change. Since verlet methods is a symplectic integration method, it has an advantage on the problem we are working with, while the RK4 method does not have such a advantage. This might be the reason for the two methods providing similar accuracy while one of them is order 2 and the order one is order 4.

5.4 Possible Improvements

There are two possible improvements that we can see in our implementations of the algorithm. The first one is the unit we are working with. Since large floating point numbers can cause issues in the from of truncation or rounding error, one way to solve this is to reformulate physical equations in our simulation to use AU (Astronomical unit) instead of meters so the numbers don't have to be as large, hence reducing rounding and truncation error during the simulation. The second possible improvement we can make is to cache the forces already calculate between the two objects so we do not have to recalculate everything for each object, hence increase the run-time of both algorithms.

6 Conclusions

From the result and analysis, we conclude that the verlet integration method is more suitable for seeking solution of a N-body system since it offers similar precision compare to RK4 (both have relative errors around 3.79% for 165 years simulation) and it is not as computationally expensive as the RK4 methods. RK4 method is around 3 times slower than the verlet method on average while offering approximately the same accuracy in the N body problem.

Reference

[1] "N-body problem," Wikipedia, [Online].
https://en.wikipedia.org/wiki/N-body_problem

[2] "Planetary Fact Sheet," NASA. [Online].
https://nssdc.gsfc.nasa.gov/planetary/factsheet/