Solar System Modeling with Numerical ODE Solvers

Nicolas Dronchi
Michigan State University - Computational Physics - PHY480
Dronchin@msu.edu
Git Hub link: https://github.com/dronchin/Project3
(Dated: April 3, 2018)

This project set out to model planetary interactions in our solar system using differential equation solvers. Both Euler's method and the Velocity Verlet method were used and compared. A focus of this project was also on the object orientation of the code which allowed it to be more flexible. The main results from this study was in the stability, escape velocity accuracy, and the accuracy of the Earth's predicted location. The stability of the Velocity Verlet method had much better long term stability than Euler's method because it conserved energy but they were equally stable in the short term. The escape velocity of earth was determined to be 8.8636 AU/yr which has a 0.2% error compared to the expected value. Finally, the precision of the Earth's position after a year was calculated to be a 0.000468% error for the x position and a 9.363% error in the y position compared to the HORIZONS program developed by NASA.

Usage: Numerical differential equations solvers. Planetary motion/trajectories.

I. INTRODUCTION

The solar system we live in has eight planets (nine if you feel inclined to add Pluto) and one central star that we call the sun. But if you wanted to be more accurate, you would include moons for each planet, maybe the asteroid belt, and don't forget any comets passing by. While this N-body problem seems like a lot, each member only interacts though gravity.

The law of universal gravitation is what describes their interaction.

$$F = \frac{GM_1m_2}{r^2}$$

Using the force, we enter into the laws of motion using newtons law.

$$\sum F = ma$$

Once you take into account all of the potential sources of a gravitational force, you get the acceleration and can therefore solve for the positions of each body.

Solving for the numerical solutions to this problem is a common calculation carried out in the field of astrophysics. Applications to astrophysics include having to calculate the trajectories of spacecraft or probes in missions as well as being able to predict the outcome after a steering burn. This can also be used on the closer scale with launching satellites into orbit and checking for collisions between them. Finally, this is a useful tool for the prediction of where in the night sky celestial bodies sit and could even be extended to predicting asteroid impacts.

In this project, I go on to discus the methods used for calculating the differential equations determining their motion. Then we take a quick technical look at the code and tests with it as well as a look into programming for a N-body through object oriented programming. Finally, I review the results and conclusions of the system in terms of stability, accuracy of the escape velocity, and percent error compared to Nasa's calculation of where earth should be a year from now.

II. METHODS

A. Setting Up the Differential Equations

In the case of this project, I model the solar system in 2D. With all the planets sitting in the same plane. It is easy to added the 3rd dimension as you just repeat the following for the z direction.

We start with a simple model of the Sun and Earth. Using only Newton's law, the acceleration is split into the x and y components.

$$a_x = \frac{F_x}{M_E} = \frac{Fcos(\theta)}{M_E}; \quad a_y = \frac{F_y}{M_E} = \frac{Fsin(\theta)}{M_E}$$

With the universal law of gravity, as stated in the intro, the force is substituted in and the acceleration of the Earth is calculated [1].

$$a_x = \frac{-GM_\odot M_E}{r^2} * \frac{rcos(\theta)}{rM_E} = \frac{-GM_\odot x}{r^3}; \quad a_y = \frac{-GM_\odot y}{r^3}$$

For both directions of acceleration, we can pick units of distance to be AU and units of time to be years. When we set the initial rotational velocity of the Earth to be 1 $rotation/yr$, the velocity is 2π AU/yr. Thus set $GM_\odot = v^2r = 4\pi^2$ AU^3/yr^2

So far, what has been set up is the acceleration. This is needed to to find the position because we know that

$a = \frac{dv}{dt} = \frac{d^2x}{dt^2}$. Thus we can use the acceleration to find the velocity and the velocity to find the position using the ODE solvers found later in the methods.

This forms an initial condition problem where the behavior of the system in the long term depends on the starting values of all the planets in the system. For the "Real Planets" system, the initial conditions were taken from the HORIZONS Web-Interface program which can be found at https://ssd.jpl.nasa.gov/horizons.cgi#top [2]. Our initial conditions were taken from the date of 3/23/2018.

B. Differential Equations Solvers

Euler's method is one of the simplest ODE solvers. While simple, it still gives great accuracy as long as you take a sufficiently small step size. For the initial value problem,

$$\frac{dy}{dt} = f(t, y); \quad y(t_0) = y_0$$

we are looking for values of y at later moments of time. To do this, we use the derivative $f(t, y)$ as the tangent to the line and approximate y_1 being the point on that tangent at $t_1 = t_0 + h$, where h is a small time step.

In our initial conditions we need both v_0, x_0, and y_0. We use Euler's method to first find the velocity using the current position of the planet to find the acceleration. The new velocity at the new point can be found because the acceleration is only dependent on the position.

$$Vx_i = Vx_{i-1} + h * a(x_i)$$

After the the velocity is updated we then use the updated velocity of the system in Euler's method again to calculate the new position [1].

$$X_i = X_{i-1} + h * Vx_i$$

While Euler's method can have good accuracy at very small values of h, it generally overshoots in the case of planetary modeling. It has a local error proportional to h^2 and has a general error proportional to h. This leads to the changing of values that are typically conserved in the system. This is explored later in the results an analysis section when looking at the stability of the system.

The main method of integration used in this project is the Velocity Verlet method. It has the advantage of not changing the conserved quantities like kinetic/potential energy or angular momentum of the system. For the general Verlet integration method, you use the Taylor expansion. In our case we use one Taylor expansion for the position and one for the velocity.

$$x_{i+1} = x_i + h * x_i' + h^2/2 * x_i'' + O(h^3)$$

$$v_{i+1} = v_i + h * v_i' + h^2/2 * v_i'' + O(h^3)$$

Calculating the position is straight forward. Ignoring the error while making substitutions for velocity and position we get:

$$x_{i+1} = x_i + h * v_i + h^2/2 * a_i$$

For velocity we run into the issue of not having an analytical expression for the value of v_i''. This is not a value that is typically known. Therefore, an estimation can be obtained looking at the difference between a_i and a_{i+1} which was calculated earlier. This gives the following for updating velocity [3].

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

which simplifies to

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

The Velocity Verlet method has local accuracy on the order of h^3 which gives it a global accuracy of h^2. This is much better than Euler's method but the error can be improved with other methods of differential equation solvers. Methods like high order Runge-Kutta methods can provide much higher levels of accuracy while adding on more computational power requirements [4]. Then there are adaptive methods of the Runge-Kutta that can improve computational time by switching between different order of the Runge-Kutta method. These were not used in this project but could be implimented in the future.

III. CODE AND TESTS

For coding this project, I used primarily object orientation in python. I also created a file called SunEarthJupiter_timing in order to compare the computational time required for object orientation vs hard coding with functions. The results of this timing study can be seen in figure 1. While the hard object oriented program takes longer, they both scale linearly. The advantage of the object orientation is the ability to scale the number of objects interacting quickly.

We start by examining the two classes used. There is a planet class and a solarsystem class. The planet class holds variables such as positions, velocities, and mass as well as all the positions that it's been in since the start of it's movement. For the planet class, there are four

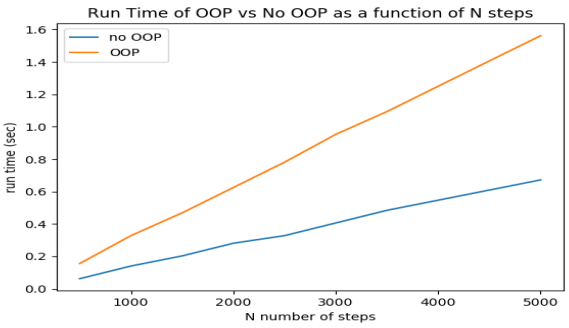

FIG. 1. Comparison study on hard coding vs object orientation

methods. These can be seen in listing 1 showing the four calculations for distance, kinetic energy, potential energy, and angular momentum. The distance method is used for acceleration calculations while the others are designed to be used as tests to see the properties of the system.

Listing 1. Planet Methods

```
def distance(self, OP):
    dis = np.sqrt((OP.x-self.x)**2
    + (OP.y-self.y)**2)
    return distance
def kineticEnergy(self):
    E = 0.5*self.mass*
    (self.vx**2+self.vy**2)
    return E
def potentialEnergy(self):
    E = -4.0*np.pi**2*self.mass/
    (2*np.sqrt(self.x**2+self.y**2))
    return E
def angularMomentum(self):
    AM = self.mass*np.sqrt(self.x**2+
    self.y**2)*np.sqrt(self.vx**2+
    self.vy**2)
    return AM
```

The solarsystem class holds the step size, number of steps, and all the planets in a list. There are two versions of this class. One uses the Euler methods of integration and the other uses the Velocity Verlet method. The Euler method is seen in listing 2 and the Verlet method in listing 3.

Listing 2. Euler

```
#CUR is the Current planet being updated
def Euler(self, CUR):
    ax, ay = self.Accel(CUR)
    CUR.vx = CUR.vx + ax*self.h
    CUR.vy = CUR.vy + ay*self.h

    CUR.x = CUR.x + CUR.vx*self.h
    CUR.y = CUR.y + CUR.vy*self.h
```

Listing 3. Velocity Verlet

```
#CUR is the Current planet being updated
def VV(self, CUR):
    ax, ay = self.Accel(CUR)

    CUR.x += self.h*CUR.vx
          + (self.h**2/2)*ax
    CUR.y += self.h*CURt.vy
          + (self.h**2/2)*ay

    ax_new, ay_new = self.Accel(CUR)

    CUR.vx += self.h/2*(ax + ax_new)
    CUR.vy += self.h/2*(ay + ay_new)
```

The Euler takes about $12 + 20x$ FLOPS where x is the number of planets in the system. This is due to the need to calculate the acceleration which takes about 20 FLOPS per planet. The Velocity Verlet method takes about $20 + 40x$ FLOPS as it needs to calculate the acceleration twice. Timing the whole procedure of a run with both methods gives us figure 2 confirming that the Velocity Verlet method scales worse, taking about twice the time taken compared to Euler's method.

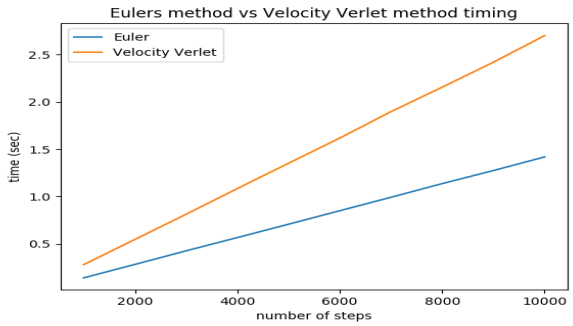

FIG. 2. timing of Euler vs Velocity Verlet method

For checks on the system, visualizations of the system were created. Graphs of the kinetic/potential energy and the angular momentum were also created. If there was an issue in the code, the planets wouldn't have kept their circular (or more elliptical) paths and the graphs of the conserved quantities wouldn't be constant. The graphs of the conserved quantities are in the results section. The graphs of the paths are seen in figures 3 and 4 bellow with distance units of AU.

IV. RESULTS

The stability of the planetary solar system calculations can be examined by looking at the tendencies of the system's conserved quantities over long runs. The conserved quantities in the system should be energy in the form of kinetic and potential, as well as the angular

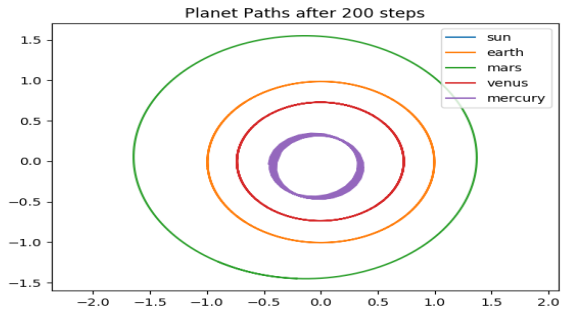

FIG. 3. Circular Orbits for planets in the inner solar system

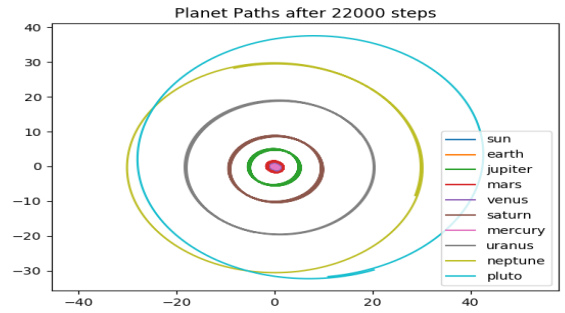

FIG. 4. Circular Orbits for planets in the outer solar system

momentum of the system.

In the first check, we look to see if the kinetic energy is conserved. Over a 100,000 steps, each of length 0.01 year we get figure 5. This gives a time span of 1,000 years.

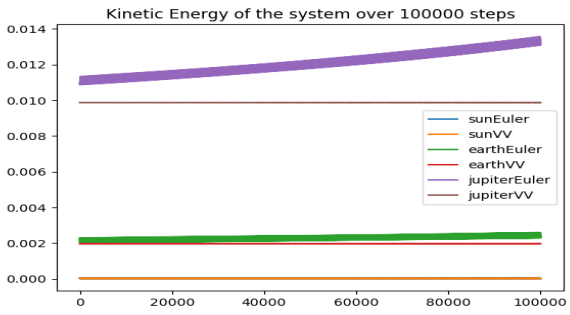

FIG. 5. long term change of the kinetic energy

It is typical that Euler's method will overestimate the velocity in this type of periodic system. The differential equation takes a cyclic shape for each independent direction. Because this system is drastically changing, the errors build up leading to the eventual overestimation of what the velocity is and therefore the position of the planet has a greater diameter [5].

The short term of kinetic energy shows a different story though. The kinetic energy is conserved at 1,000 steps, each of length 0.01 year. This can be seen in figure 6, where both Euler's method and the Velocity Verlet method stay approximately constant aside for oscillations. These same results can be seen

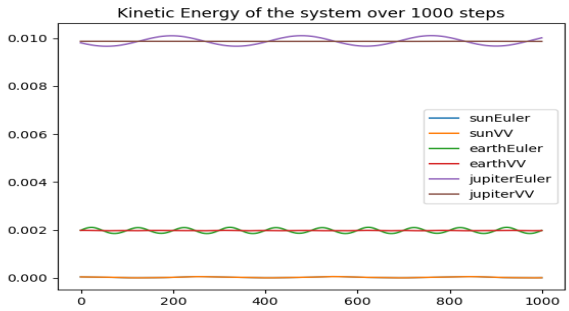

FIG. 6. short term stability of the kinetic energy

in the potential energy and angular momentum of each planet. The graphs of these conserved quantities can been see at the end in the appendix figures 7, 8, 9, 10.

This system was also used to calculate the escape velocity of orbital bodies. The analytical solution to this question is as follows [6]:

$$v_{esc} = \sqrt{\frac{2GM}{r}}$$

starting out at $(1, 0)$ and the mass of our sun being 1, The escape velocity should be:

$$v_{esc} = \sqrt{\frac{2 * 4\pi^2}{1}} \approx 8.8858 AU/yr$$

The results of the escape velocity calculation in my system was $8.8636 AU/yr$. This gives an error of only 0.2%.

The last result of the system is in comparing the position of earth determined in my program to the one determined by the HORIZONS Web-Interface program. The length of a year was extremely important in this calculation as the units they give are in AU/day. I used 265.242 days/yr in my calculations. My summary of results can be seen in table I.

V. CONCLUSION

If you look at the timings done on this set of programs, it would be easy to conclude that if you wanted speed, you would use a hard coded Euler's method. The issue

Measured	Calculated	Actual	Percent Error
X	-0.997377	-0.997373	0.000468%
Y	-0.026152	-0.023913	9.36320%

TABLE I. table of calculated postions compared to HORIZONS

with this is that hard coding was only scaled so that it modeled the Sun, Earth, and Jupiter. Scaling this to a larger system of planets might involve complications not only in code computational time but flexibility. A program that saves a little time in computation wastes a lot of time when you want to see the effect of adding one more planet. Why stop at planets though, adding other bodies of interest like moons, asteroids, or comets would take an even longer time.

The other half is using the Velocity Verlet method. Other than the advantage that it conserves energy, it provides the chance to lower the step size. Because of it's exceptional long term stability compared to Euler's method, if you wanted to study the system in even longer term events, you can increase the step size and reach further into the future. Increasing the step size in Euler's method can lead to catastrophic failures where planets have rapid shifts in energy or angular momentum and shoot off into the distance unrealistically. To get the same stability using Euler's method, you would have to make the step size small enough which cranks up the overall computational time.

Looking at the programs ability to compute the escape velocity is next. As reported in the results, the escape velocity of Earth was calculated to a 0.2% error. This has applications towards steering rockets. Escaping Earth's gravitational pull to wander out into the solar system is the goal of many NASA probes. Being able to calculate this numerically allows room to change the forces acting upon the object. Things like air resistance could be added to model to expand calculations and find out what it takes to escape earth's surface.

The accuracy of the escape velocity is also adjustable. If you narrow down where the escape velocity is, you can decrease the scanning step size so that it looks at smaller and smaller changes in velocity. Then you could also increase the threshold of what counts as an escape from gravity. Ideally, it would have to be able to get infinitely far away, but that takes an infinite amount of time. Therefor the threshold is needed to determine if it has escaped.

Finally, the Velocity Verlet system with all of the planets is useful for predicting the position of our planet to great perversion. With comparison to HORIZON's prediction of where the earth would be in one year, my program was able to calculate earth's position in one year with 0.000468% error in the x directional measurement. The y directional measurement was higher in percent error but when you look at the actual difference in the calculated vs actual, the results are equally impressive.

There are many things that this model achieves but there are still even more ways to improve it. Future improvements for the work include adding on the 3rd dimension. This would allow for better tracking of orbitals for far out planets that don't exactly match the plane of orbit that the sun is in. Other improvements include using a more robust ODE solver. Solvers like the Runge-Kutta 5 or higher order could be used to improve the numerical accuracy of the positions. Addaptive methods could also be used to increase the accuracy as planets closer to the sun need to be updated more frequently than planets further out in the solar system.

VI. CITATIONS

[1] Hjorth, Morten (2015) Computational Physics Lecture Notes Fall 2015 can be found on github at https://github.com/CompPhysics/

[2] HORIZONS Web-interface program data https://ssd.jpl.nasa.gov/horizons.cgi#top

[3] Hjorth, Morten (2018) Project3 description can be found on github at https://github.com/CompPhysics/ComputationalPhysicsMSU/tree/master/doc/Projects/2018/Project3/pdf

[4] Application of numerical integration to solve the n-body problem for celestial mechanics modeling http://ccar.colorado.edu/asen5050/projects/projects_2010/docurro/problem.html

[5] Paul Dawkins, "Differential Equations - Notes", http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx

[6] http://hyperphysics.phy-astr.gsu.edu/hbase/vesc.html

VII. APPENDIX

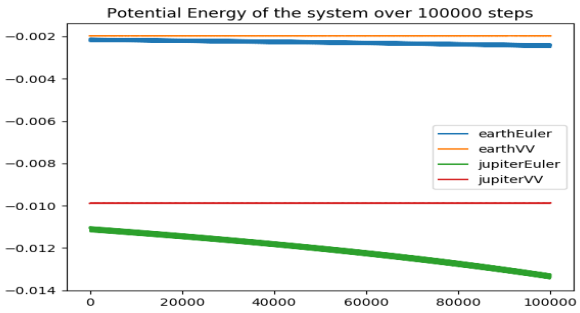

FIG. 7. long term change of the potential energy

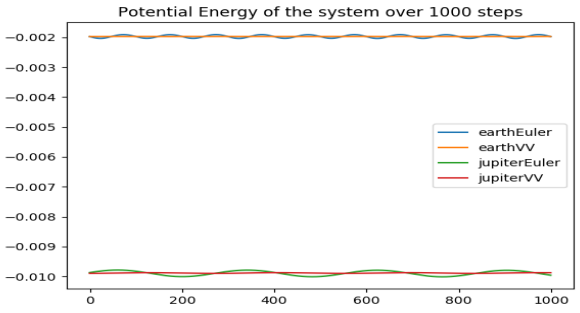

FIG. 8. short term stability of the potential energy

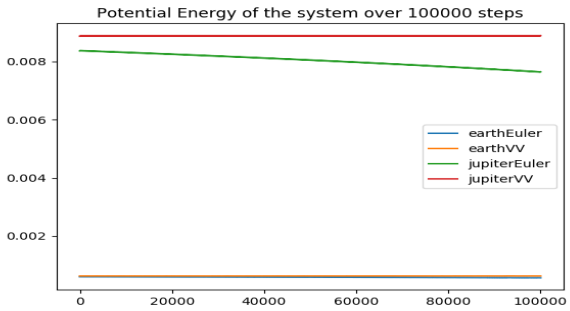

FIG. 9. long term change of the angular momentum

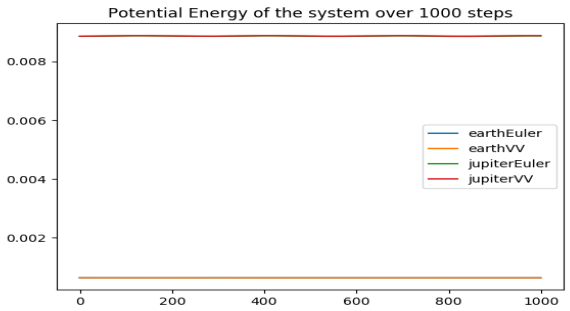

FIG. 10. short term stability of the angular momentum