# The Effects of Jupiter on High-Speed Asteroid Trajectories Using Runge-Kutta Approximation

Ieun Choo, Richeek Dutta, Myles Gong, Parth Shrotri, Evan Yu
*University of Illinois Urbana-Champaign, Dept. of Aerospace Engineering*

**Past studies have shown that Jupiter's gravitational well has a significant impact in redirecting smaller, slower-moving asteroids on a potential collision course with Earth. However, the effect of Jupiter on the trajectories of large, high-speed asteroids has not been the subject of many comprehensive studies with regard to planetary defense. These bodies are important to analyze, as their high speeds give ground-based operations little time to react in the event of a sudden redirection. This paper investigates the impact of Jupiter on large, high-speed asteroids by comparing the simulated trajectories of 15 such asteroids with and without the influence of the gravitational effects of Jupiter. The objective of this study is to determine the possibility of a high-speed interstellar asteroid undergoing a gravitational assist upon close pass with Jupiter, which would increase the asteroid's kinetic energy and may place the body on a collision course with Earth. Data generated of the simulated asteroids' trajectories using a Runge-Kutta 4 solver is analyzed to quantify the effect that Jupiter has in terms of the difference in trajectory when the asteroids reach Earth. Ultimately, it is found the trajectories of asteroids performing steep close passes with Jupiter at high velocities are not significantly altered over small time spans. This report also analyzes the efficacy of the Runge-Kutta 4 method used to solve this $n$-body gravitational problem. To validate the solver, known two-body and three-body systems are simulated and compared to real data from the NASA JPL Horizons dataset. Finally, the team has created a general framework for analyzing $n$-body problems with tools including trajectory visualization that make simulating asteroid dynamics modular and easy to analyze.**

## I. Nomenclature

| | | |
|---|---|---|
| AU | = | astronomical unit ($1.496\times10^{8}$ km) |
| EARTh | = | **E**valuation of **A**steroid **R**edirection **Th**reats |
| $G$ | = | universal gravitational constant ($6.6743\times10^{-11}$ $\frac{\mathrm{m}^3}{\mathrm{kg}\cdot\mathrm{s}^2}$) |
| ICRF | = | International Celestial Reference Frame |
| $m$ | = | mass of object (kg) |
| $\boldsymbol{r}$ | = | distance from Jupiter (m) |
| RK4 | = | Runge-Kutta $4^{\text{th}}$ order approximation |
| TAYLOR Series | = | **T**ool for **A**nal**Y**sis of ce**L**estial **O**rbits using **R**K4 in **Series** |
| $\Delta t$ | = | time-step (s) |

## II. Introduction

This report discusses the effect of Jupiter on the trajectory of massive asteroids performing high-speed, high-angle passes near the planet. Being the most massive planet in the solar system, Jupiter wields significant gravitational influence and has the potential to redirect smaller asteroids either to a stable orbit with itself at the Sun-Jupiter Lagrange points 4 and 5 [1] changing their orbit away from Earth's, or it can alter the trajectory of the asteroid towards Earth. The case of Lexell's demonstrates both effects well. This comet passed by Earth in 1770 and was first observed by Charles Messier as it made its closest approach at 0.015 AU. Messier calculated its orbital period at 5.58 years and that it must have been perturbed by Jupiter, causing it to be sent toward Earth. Despite having an extremely low period, the comet has not been observed since. Messier and Laplace determined that it must have been perturbed a second time during another close approach with Jupiter when the comet and Earth were in solar conjunction. That work has since been verified using modern computational methods [2].

These two opposing consequences of Jupiter's gravitational influence necessitate a precise analysis of the effects of Jupiter on the asteroids. The team analyzed Jupiter's influence on asteroids simulated using various initial positions and velocities. The system utilized in the simulation includes the Sun, eight planets including Earth, and 15 asteroids that are randomly distributed within a specifically defined region. This report analyzes the trajectories of the simulated asteroids with and without Jupiter's gravitational influence to quantify the total effect that the planet has on redirected massive, high-speed asteroids. The dynamical model utilized in this study involves all the gravitational effects of the planets in the solar system and implemented a numerical method to solve the state of the most relevant bodies: Earth, Jupiter, and the asteroids. In order to condense simulation time, the positions of the rest of the planets in the solar system were determined using data from the JPL Horizons dataset [3], with the Sun considered fixed at the origin. The problem modeled in this study will henceforth be referred to as the **E**valuation of **A**steroid **R**edirection **Th**reats (EARTh) problem.

Section III discusses the details of the numerical method used to solve the EARTh problem. Section III.A presents a discussion on the design of the 4-step Runge-Kutta method to solve the defined initial value problem. Validation of the method's accuracy for a two-body and three-body problem is presented in Section III.B. Section III.C addresses the application of the numerical method to general $n$-body problems, the development of the **T**ool for **A**nal**Y**sis of ce**L**estial **O**rbits using **R**K4 in **Series** (TAYLOR Series), and how it is used for the EARTh problem.

Section IV presents the results of the study, demonstrating the difference in the asteroids' trajectories with and without the presence of Jupiter. These are quantified both graphically and numerically. Further discussion of possible sources of error are included as well.

Finally, Section V closes with a summary of methods and results, along with potential areas for future development or optimization.

## III. Methods

### A. Development of the Numerical Method

A 4-step Runge-Kutta (RK4) method has been chosen to simulate the gravitational effects of Jupiter on asteroids entering the inner Solar System. The RK4 method is an explicit method, meaning that no information about future time-steps is required. It is also a single step method, so only data from one point is needed. The 4th order Runge-Kutta method provides a good balance of accuracy and computational expense, as it is the highest order method that requires the same number of steps as the order of the accuracy. Instead of taking a simple average of the slopes, the RK4 method takes the weighted average of the slopes going through the iterations, making the numerical solution more accurate.

*1. The Development of RK4 Method*

The RK4 method propagates the integral using the following equation:

$$y_{k+1} = y_k + \left(\frac{\Delta t}{6}\right)(k_1 + 2k_2 + 2k_3 + k_4) \tag{1}$$

in which $y_{k+1}$ is the value of the function one time-step in the future, $y_k$ is the value of the function at the current time-step, $\Delta t$ is the size of the-time-step, and $k_i$ for $i = 1, \ldots, 4$ are defined as:

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right) \\ k_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t k_1}{2}\right) \\ k_3 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t k_2}{2}\right) \\ k_4 &= f\left(t_n + \Delta t, y_n + \Delta t k_3\right) \end{aligned}$$

where

$$\frac{dy}{dt} = f(t, y).$$

This method was used twice to calculate position from acceleration to solve the $n$-body problem.

*2. RK4 Method for the N-body Problem*

The $n$-body problem, defined by Eq. (2) uses the net gravitational force vector on each body to calculate the acceleration analytically. The calculated acceleration is then used, by means of RK4, to determine the approximate velocity at the next time-step. Using the approximate velocity at a point in time, the approximated position at the next time-step is also determined using RK4. This process is repeated for every body in the system at every time-step to fully define the dynamics.

$$m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} = \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{G m_i m_j \left(\boldsymbol{r}_j - \boldsymbol{r}_i\right)}{\left\|\boldsymbol{r}_j - \boldsymbol{r}_i\right\|^3} \tag{2}$$

## B. Validation of the 4-step Runge-Kutta Method

*1. Validation Method*

To validate that the RK4 method is appropriate for the analysis of the effect that Jupiter has on the asteroids and to demonstrate the correctness of the implementation, the method is tested on well-known two-body and three-body problems for which the numerical solution can be compared to the NASA JPL Horizons System dataset [3] which we consider ground-truth. The two-body problem is a model of the Earth's orbit around the Sun, with the primary analysis being on the Earth's orbit. The three-body problem involves the orbit of the Earth around the Sun and the orbit of the Moon around the Earth. Both problems consider the Sun fixed at the origin and define the coordinate system according to the International Celestial Reference Frame (ICRF) [4]. The initial position and velocity of the Earth and Moon are set according to the data from January 1, 2024 obtained from the NASA JPL Horizons System [3]. The initial acceleration is set as the zero vector. This acceleration is immediately recalculated at the first time-step. The RK4 method is then applied as described in Section III.A to solve the corresponding initial value problem with different time-steps. The final data points of the resulting approximate trajectories are then compared to the corresponding ground-truth data to quantify the global error. Then, the plots of the true trajectory and the approximate trajectory are compared to qualitatively verify the correctness of the implementation. Also, the final data point of the approximate trajectory is compared to the final data point of the trajectory calculated using a baseline time-step of 30s to get the local truncation error. This local truncation error value is also plotted with respect to $\Delta t$ to verify that the local truncation error in RK4 scales with $O(\Delta t^4)$.

The convergence plots comprise the global error for simulation runs for 365 days on the set of 20 different time-steps spaced evenly between 30 seconds and 1 hour on a logarithmic scale. The error of the final position vector as a proportion of the magnitude of the ground-truth position vector is calculated and plotted for each time-step size. This is a method of demonstrating that the numerical solution converges to the true solution at the expected rate. Moreover, since the RK4 method is an explicit method, it requires a smaller time-step for the method to be absolutely stable than a comparable implicit method. The convergence plot also provides a method of determining optimal time-steps to solve the EARTh problem. The designated criterion for validity asserts that the error be smaller than 1% for the three-body problem. The run-time is also plotted for each simulation for the same set of time-steps, which is used to evaluate the sensitivity of simulation efficiency to time-step size. This plot is then used to determine the time-step size that optimizes the trade-off between run-time and error. With the chosen time-step of 60 seconds, a plot is generated to show the trajectory of the bodies with respect to the Sun and compared with the true trajectory obtained from NASA JPL Horizons System [3].

*2. Validation Results*

The convergence test of the two-body problem shown in Fig. (1a) demonstrates that the error of the numerical solution converges where it decreases as the time-step decreases. The convergence test of the three-body problem shown in Fig. (2a) also demonstrates that the error decreases as the time-step decreases. However, from the time-step of size 175 seconds (the eighth time-step), the error does not seem to follow the expected convergence rate. Upon further analysis, it was determined that this is due to the nature of truncation error at large time-steps. Thus, these tests confirm that the RK4 implementation is correct.

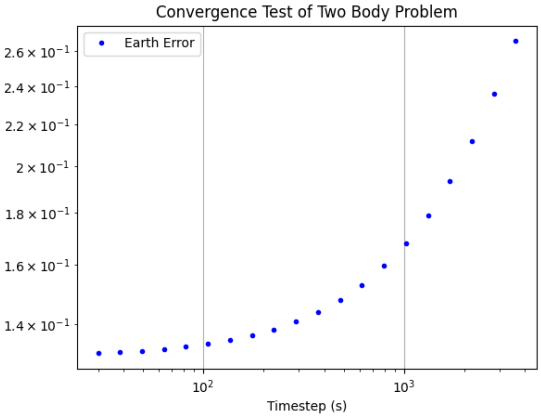
To determine the size of the valid time-step that we use to solve the EARTh problem, we considered the errors over time-steps for both two-body and three-body problems together. While the three-body problem shows that time-steps smaller than or equal to 290 seconds (the tenth time-step) generate error less than 1%, the global error for the two-body problem is greater than 1% even with the smallest time-step of 30 seconds. This introduces an interesting phenomenon.

There is a significant decrease in error between the two-body and three-body problem analyses. This is likely because of the oversimplified model that the two-body system assumes. The two-body system used in testing does not account for the effects of the Moon on the Earth's orbit [5]. This led the team to believe that adding more bodies would improve the accuracy, especially if these bodies are significant to the system, such as the Moon. Since a time-step under 290 seconds generates a sufficiently small error with the three-body problem as discussed, we could confirm that time-steps in that range will generate a sufficiently small error with even more bodies to demonstrate the numerical solution to be accurate for the $n$-body problem analyzing the effect of Jupiter on numbers of asteroids.

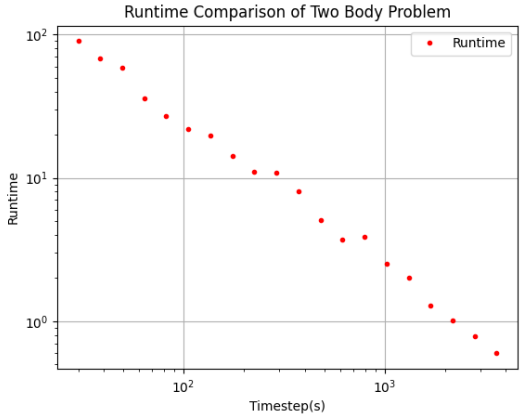
We consider the run-time in addition to global error for a better choice of valid time-step. Fig. (1b) and Fig. (2b) show that the run-time is nearly linearly proportional to the inverse of a time-step, meaning, decreasing the time-step might result smaller error but will take longer computational time. Also comparing the run-time plots of two problems shows that adding more bodies requires more time for the simulations to run. Since runs take even longer for the full $n$-body problem, run-time is also an important factor to consider. For these reasons, though it would be best to choose 30 seconds for the least error as in Fig. (1a), we chose 60 seconds. While the simulation still took a significant amount of time (2 hours each for the system with and without Jupiter), the guarantee of accuracy was more important to the team.

Using the previously determined time-step, for each of these problems, we plot the approximate trajectory of Earth around the Sun and of the Moon and Earth around the Sun as shown in Fig. (4). The approximate trajectory obtained from the numerical solution nearly overlaps that of the true trajectory as expected with a sufficiently small time-step. The plot also confirms that the dynamics and our RK4 method can be successfully implemented to the $n$-body initial value problem.

The plots showing local truncation error versus time-step as well as global error versus time-step demonstrate a correct implementation of the RK4 solver for our system. It is evident that for both the two-body and the three-body system, the local truncation error scales with $\Delta t^4$, as the slope of the two graphs are approximately the same.

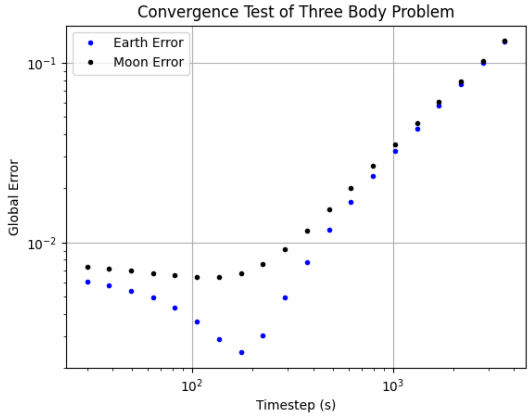



(a) Convergence plot for two-body problem.

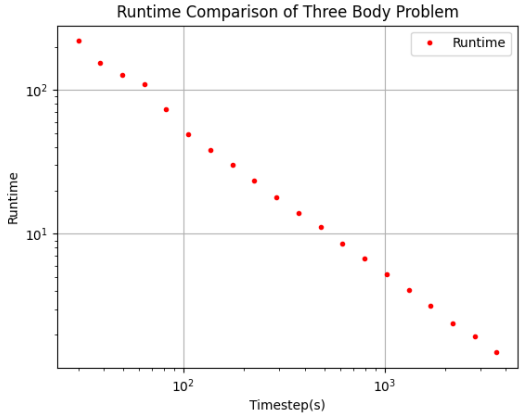



(b) Run-time over time for two-body problem.

**Fig. 1 Plots for validation of RK4 method on two-body problems.**

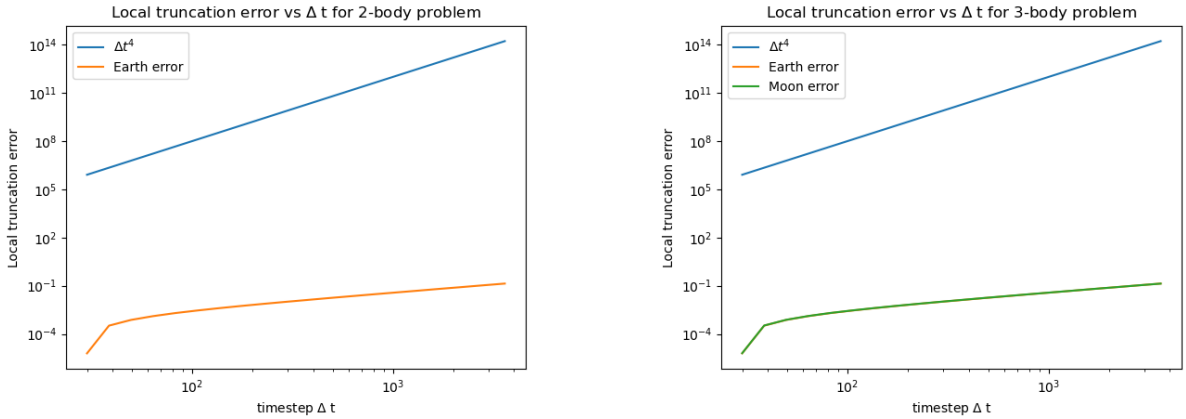



(a) Convergence plot for three-body problem.



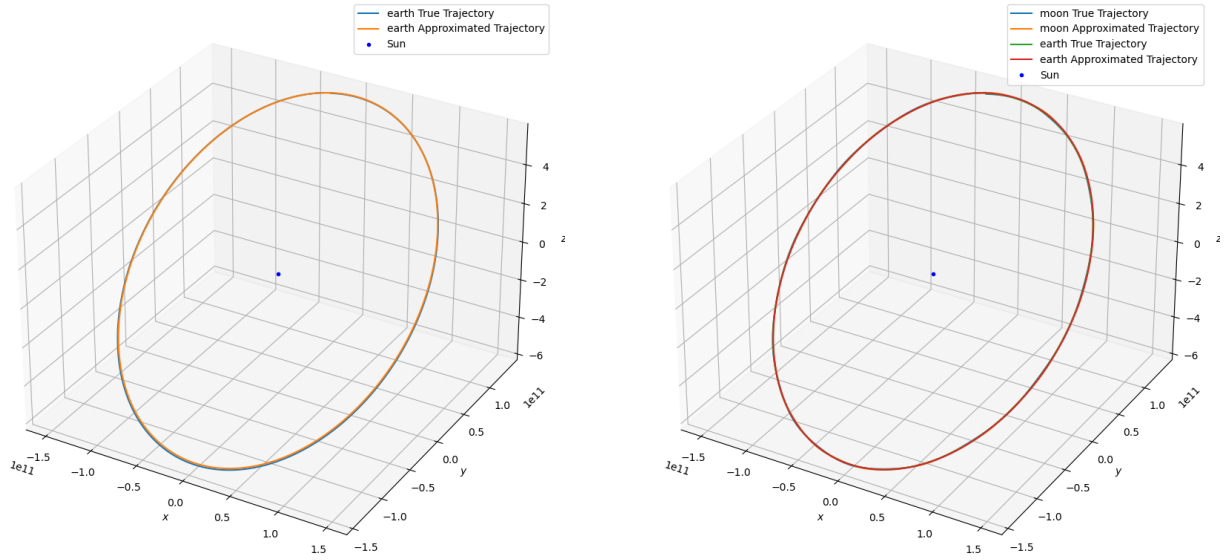

(b) Run-time over time for three-body problem.

**Fig. 2 Plots for validation of RK4 method on three-body problems.**




**(a) Truncation error for two-body problem.** **(b) Truncation error for three-body problem.**

**Fig. 3 Plots of truncation error at various time-steps.**




**(a) Trajectory of the Earth around the Sun.** **(b) Trajectory of the Earth and the Moon around the Sun.**

**Fig. 4 Trajectory of bodies around the Sun for two-body and three-body problems over 365 days.**

## C. Implementation of the Method to the *N*-body Problem

*1. Code Structure*

The team created a generalized tool for simulating the dynamics of *n*-body problems - the **T**ool for **A**nal**Y**sis of ce**L**estial **O**rbits using **R**K4 in **Series** (TAYLOR Series). TAYLOR Series provides a framework for simulating *n*-body dynamical systems by allowing the creation of custom `Body` objects with different initial conditions and physical properties. The `Body` class contains all dynamical properties, performs all the force computations for the `Body`, and stores the trajectory of the `Body`. There are two possible methods of determining position, the choice of which is specified at object creation. The first is propagation by RK4, which uses a numerical estimate of the position at every point in time based on the gravitational forces applied by the other `Bodys` in the system. The second method is using a lookup from the NASA JPL Horizons dataset [3] to determine the true location at a given time. This was done for a couple of reasons. First, it decreases the error introduced by the propagation of other planets. The desired information

does not include the other planets, and the effect of the asteroids on the other planets can be assumed to be negligible due to the extremely low mass of the asteroids relative to the planets. The second reason is that it is faster to perform a lookup from a dictionary (an $O(1)$ operation) than to run the force calculation and propagate (an $O(n)$ operation) where $n$ is defined as the number of bodies in the system. The code structure is illustrated in Fig. (6). A link to the GitHub Repository containing our source code can be found in Appendix V.A.

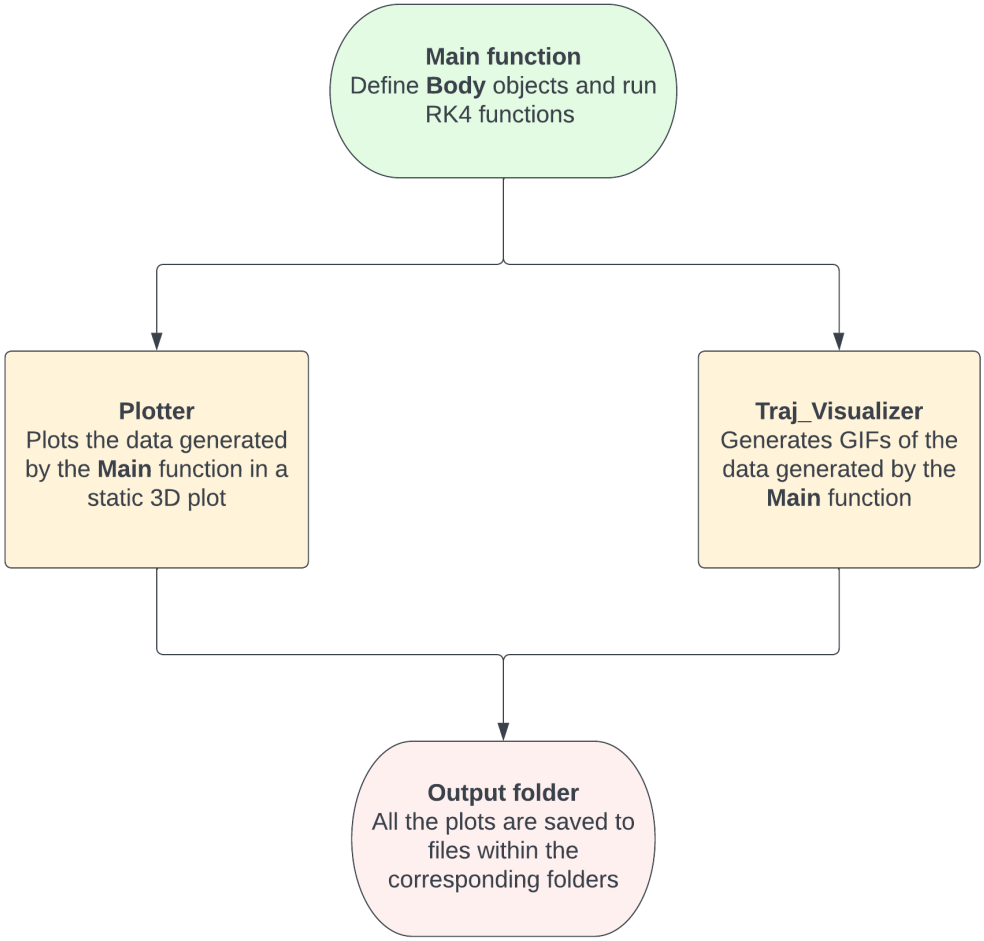



**Fig. 5 Visualization of the graphical functionality of TAYLOR Series**

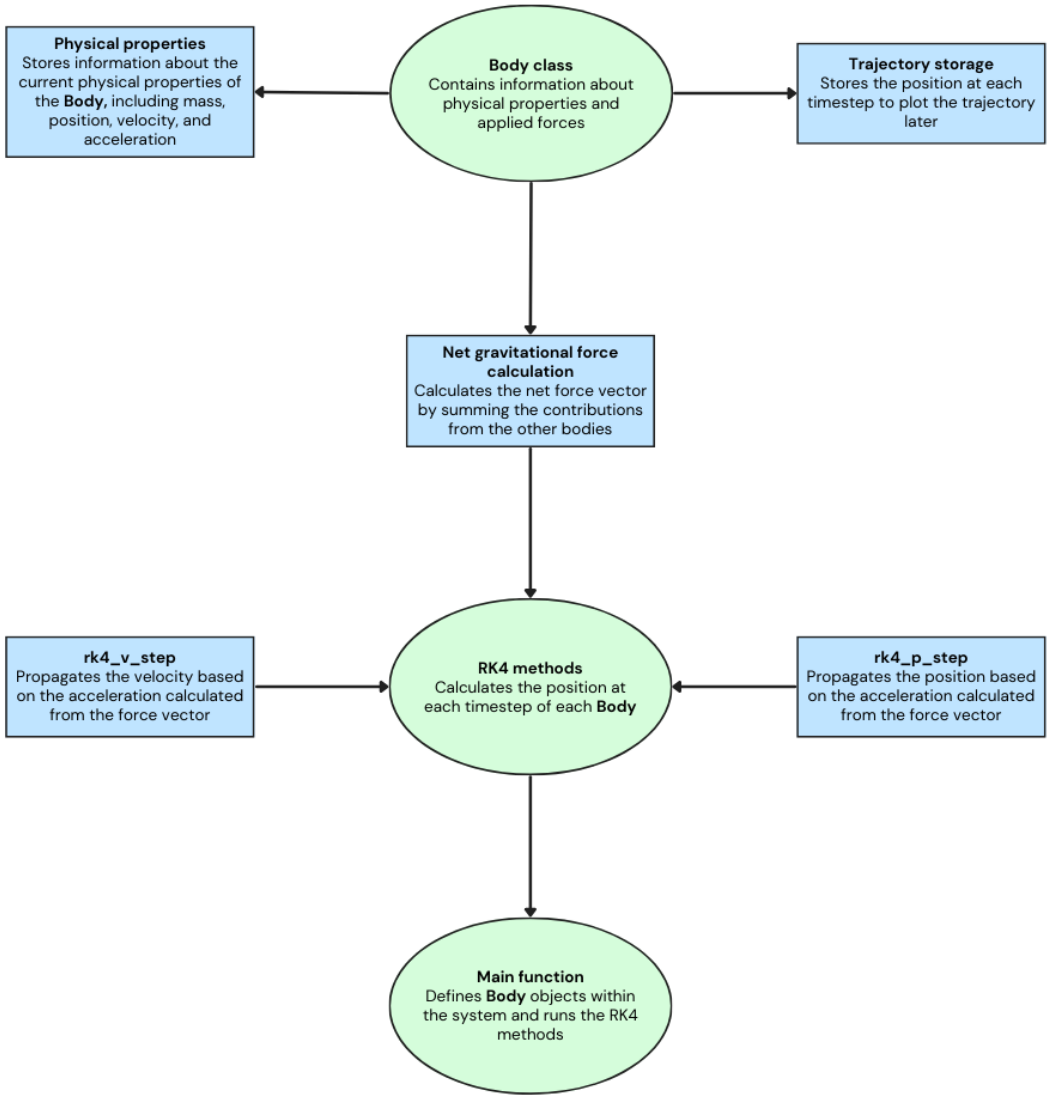



**Fig. 6 A visualization of the code structure for TAYLOR Series**

*2. Asteroid Generation*

The asteroids are generated using random variations around nominal values for various physical properties. Since the asteroids are modeled as `Body` objects, all of the physical properties defined in `Body` have to be generated. These properties are: mass, position, and velocity, where position and velocity are vectors in $\mathbb{R}^3$. The mass of each asteroid is randomly generated in the range of $2.3{\times}10^{17}$ kg to $6.9{\times}10^{17}$ kg, which includes most asteroids deemed "potentially hazardous" by NASA JPL [6]. The asteroids are spawned at a randomized initial position between the orbits of Jupiter and Saturn, with $0.125r \leq r_{\text{asteroid}} \leq 0.325r$, where $r_{\text{asteroid}}$ is the asteroid's distance from Jupiter, and $r$ is the distance from between Jupiter and Saturn. Each asteroid's velocity vector is randomized in magnitude and direction, with speeds between 9 km/s and 11 km/s. This range of generated positions and speeds is chosen to produce initial conditions approximate to those of fast-moving, interstellar asteroids passing close enough to Jupiter to potentially be redirected

by the planet's sphere of gravitational influence. The simulations used in this study generate 15 asteroids at one time, with diverse initial positions and velocity vectors in order to create a comprehensive study of many possible asteroid trajectories.

*3. Solving the EARTh Problem*

After concluding development of TAYLOR Series, the team wrote a script using the tool to evaluate the effects of Jupiter on many asteroids with various initial conditions. First, the asteroids were generated using the methods described in Section III.C.2. This ensures that the initial conditions of the asteroid are randomized but constant between runs. This mitigates any potential error that arises from the initial conditions being different. Since the planets excluding Earth and Jupiter are not important to the solution, the team decided not to propagate them and instead determine the position using the JPL Horizons Dataset [3] which as discussed in Section III.B decreases computation cost. Next, the asteroids are loaded in from the set of pre-generated asteroids and added to the list of system bodies alongside the planets. This provides the tool all the necessary information to run the simulations. Finally, the simulation is run twice: once with Jupiter and once without Jupiter. The team decided to use a time step of 60 seconds because the simulation was run overnight and run-time was not a major factor. This allowed us to use an extremely fine time step and ensure the most accurate results. After running the *n*-body simulations, we store the x, y, and z position of each relevant body and then save that data as a NumPy [7] file in a folder under the output folder. This allows us to access the relevant data for generation of plots and visualizations without running simulations gain which may result in data not being reproduced as desired as well as being extremely time intensive.

*4. Trajectory Visualization*

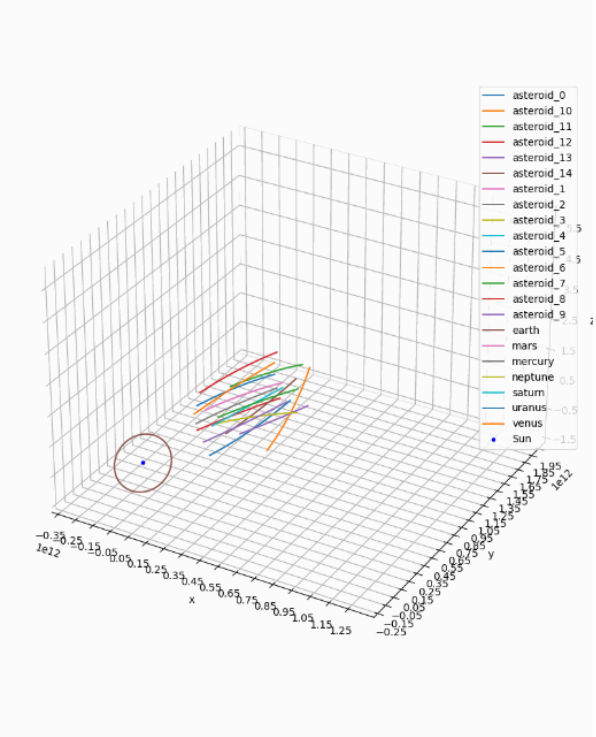
For better temporal visualization, we use the saved trajectory files to generate a video reconstruction of the trajectories of all the relevant bodies. This gives us the advantage of showing us the trajectory of the bodies as a function of time which a simple 3-dimensional plot is not capable of. To create the video we used the FuncAnimation method in Matplotlib [8]. It generates a GIF at a specified frame rate and number of new data points per object per frame, allowing the user to configure the speed of the visualization in the GIF. After generating the animated plot, it saves this plot as a GIF file in the output folder. Frames from the GIF file can be seen in Fig. (7) to demonstrate this feature.

## IV. Results

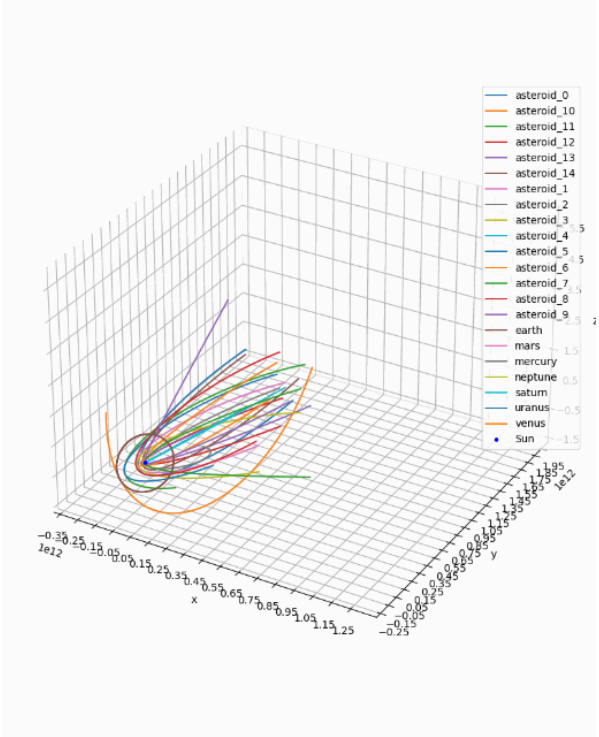
Based on the simulations, the team concludes that the effect of Jupiter on high-velocity asteroids is negligible. Although such asteroids still pose a significant threat should they collide with Earth, the presence of Jupiter will not redirect an asteroid toward Earth nor will it turn a minor threat into a potentially catastrophic one via a gravity-assisted slingshot. The data show that the difference in asteroid trajectory is essentially zero. The trajectory differences were evaluated at the point where the asteroids crossed Earth's orbit, which is the point of greatest interest to the team. The error is evaluated using the magnitude of the difference between the positions of the asteroid when they are at the same distance from the Sun. This magnitude difference is then expressed as a percentage of the magnitude of the position vector of the asteroid in the presence of Jupiter (i.e., the orbital radius of Earth).

Among the 15 asteroids that were simulated, every one of them demonstrated effectively zero change in trajectory, with the asteroid position error being exactly 0.0% or a decimal number on the order of $1\times10^{-14}$, which can be attributed to machine imprecision. This result demonstrates that asteroids of sufficiently high speed and mass to destroy Earth are unlikely to be redirected or obtain gravitational assists upon close pass at steep trajectories to Jupiter. The trajectory plots in Fig. (8a) and Fig. (8b) visually illustrate that, of the generated set of 15 asteroids with varying initial positions and trajectories, no asteroid's trajectory is significantly altered by the presence of Jupiter during the simulated close pass.

While these results offer a convincing analysis of the dynamics of asteroids within this study's selected range of initial conditions, this simulation is not a comprehensive view of the effect of Jupiter on all potentially hazardous Earth-bound asteroids. For instance, asteroids originating from the asteroid belt or interstellar asteroids traveling along slower, spiral trajectories toward Earth may potentially be redirected more by the gravitational pull of Jupiter over longer durations. Further, only large asteroids carrying the potential of world-ending impacts with Earth are analyzed in this study, though less massive asteroids may be redirected to a larger degree.

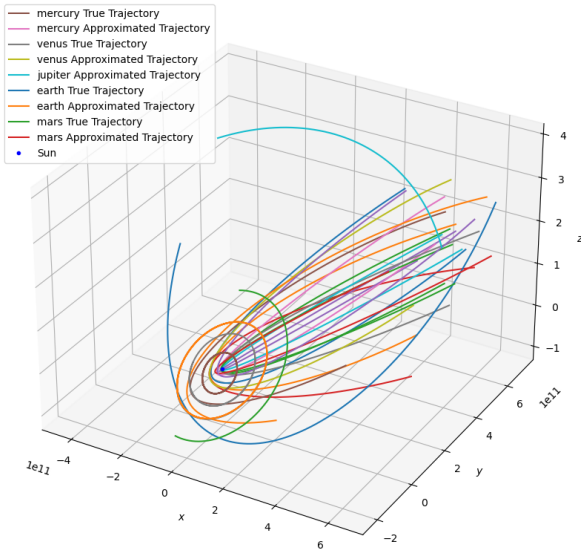



**(a) Trajectory of the asteroids, Earth, and Jupiter around the sun at some time-step 1.**

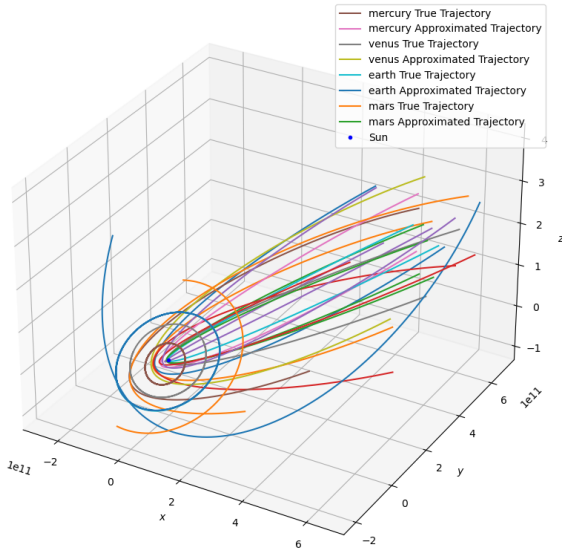



**(b) Trajectory of the asteroids, Earth, and Jupiter around the sun at some time-step 2.**

**Fig. 7 Demonstration of the trajectory visualization function.**




**(a) Plot of asteroids with Jupiter present**




**(b) Plot of asteroids without Jupiter present**

**Fig. 8 Plots of asteroid trajectories with and without Jupiter**

# V. Conclusion

Planet Earth has experienced catastrophic asteroid impacts, the most famous being the Chicxulub asteroid that wiped out the dinosaurs. While these impacts are exceedingly rare, they pose an immeasurable risk to the human race. Thus, it is important to study how seemingly harmless asteroids could become dangerous quickly. One possible way is through Jupiter's significant gravity. It is well known that Jupiter can capture and redirect asteroids away from Earth. However, Jupiter has equal potential for malevolence. The potentially harmful effect of Jupiter's gravity lies in the risk of a gravity-assisted slingshot toward Earth. Fortunately, the team has concluded that the actual effect of Jupiter on these asteroids is quite small.

Further study should be done regarding a better choice of initial conditions for asteroids entering Jupiter's sphere of influence. This involves a sensitivity analysis of the asteroid's trajectory near Earth as a function of its initial conditions. This study can also be improved upon by determining the minimum required change in trajectory that would result in such an asteroid missing Earth. This would include an analysis of launch window timing that takes into account the time required to detect a threatening asteroid.

With regard to the Tool, the current iteration of TAYLOR Series is remarkably powerful, but still lacks in certain areas. One potential area of improvement is in simulation speed. Currently, the simulation runs on one thread, one at a time. Since the team needed results from two different runs, the simulation time was twice that of a single run. After some experimentation with artificial multi-threading (i.e., running the same program in separate terminal instances), the performance difference is minimal when running two or three instances instead of just one. This is expected as the nature of the simulations is inherently single-threaded. If TAYLOR Series were to be adapted for a more intensive application, such as Monte Carlo methods, the addition of multi-threading would be an enormous advantage. Further improvements to sped could be found in the file formats used to store the data. Currently, TAYLOR Series uses the Numpy file format [7] which is roughly 2x faster than the standard comma-separated value format (CSV). By switching to an even faster file format, such as pickle, the plotting and data processing would be significantly faster.

Another area for potential development is determining why the trajectories TAYLOR Series produces are different from those that the Horizons dataset [3] contains. The team has contacted the Solar System Dynamics team at JPL to discuss the details of how the Horizons dataset was generated in hopes of shedding light on the causes of the discrepancies. The team hopes to gain a deeper understanding of both numerical methods and astrodynamics through the correspondence with JPL.

# Appendix

## A. Link to Source Code

TAYLOR Series Source Code Repository or `https://github.com/parthshrotri/AE370_Group_Project_1`

## B. Selections from Source Code

```
import numpy as np
from rk4 import Simulator
import sys
import os
from plotter import plot

sys.path.insert(0, os.path.abspath(
    os.path.join(os.path.dirname(__file__), '..')))

from dynamics.body import Body
from properties import prop

seconds_per_day = 24*60*60
days = 30
dt = 200

# Define system bodies
sun = Body(prop.mass_sun, np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]]), False, 'sun')
mercury = Body(prop.mass_mercury, np.array([prop.mercury_initial_position, prop.
    mercury_initial_velocity*seconds_per_day, [0, 0, 0]]), False, 'mercury')
venus = Body(prop.mass_venus, np.array([prop.venus_initial_position, prop.venus_initial_velocity*
    seconds_per_day, [0, 0, 0]]), False, 'venus')
```

```
earth = Body(prop.mass_earth, np.array([prop.earth_initial_position, prop.earth_initial_velocity*seconds_per_day, [0, 0, 0]]), True, 'earth') #everything in SI
mars = Body(prop.mass_mars, np.array([prop.mars_initial_position, prop.mars_initial_velocity*seconds_per_day, [0, 0, 0]]), False, 'mars')
jupiter = Body(prop.mass_jupiter, np.array([prop.jupiter_initial_position, prop.jupiter_initial_velocity*seconds_per_day, [0, 0, 0]]), True, 'jupiter')
saturn = Body(prop.mass_saturn, np.array([prop.saturn_initial_position, prop.saturn_initial_velocity*seconds_per_day, [0, 0, 0]]), False, 'saturn')
uranus = Body(prop.mass_uranus, np.array([prop.uranus_initial_position, prop.uranus_initial_velocity*seconds_per_day, [0, 0, 0]]), False, 'uranus')
neptune = Body(prop.mass_neptune, np.array([prop.neptune_initial_position, prop.neptune_initial_velocity*seconds_per_day, [0, 0, 0]]), False, 'neptune')
# jupiter_l4 = Body(prop.mass_L4, np.array([prop.L4_initial_position, prop.L4_initial_velocity*seconds_per_day, [0,0,0]]), True, 'jupiter_l4')
# jupiter_l5 = Body(prop.mass_L5, np.array([prop.L5_initial_position, prop.L5_initial_velocity*seconds_per_day, [0,0,0]]), True, 'jupiter_l5')

system_bodies = [sun, earth, mercury, venus, mars, jupiter, saturn, uranus, neptune]

# Asteroid generation
num_asteroids = 5

asteroid_masses = np.array(np.load(os.path.join(os.path.dirname(__file__), '..', 'data','asteroids', 'asteroid_masses.npy')))
asteroids_init_pos = np.array(np.load(os.path.join(os.path.dirname(__file__), '..', 'data','asteroids', 'asteroids_init_pos.npy')))
asteroids_init_vel = np.array(np.load(os.path.join(os.path.dirname(__file__), '..', 'data','asteroids', 'asteroids_init_vel.npy')))

for i in range(num_asteroids):
    asteroid = Body(asteroid_masses[i], np.array([asteroids_init_pos[i], asteroids_init_vel[i]*seconds_per_day, [0,0,0]]), True, 'asteroid_' + str(i))
    system_bodies.append(asteroid)


for body in system_bodies:
    body.set_system_bodies(system_bodies)
sim = Simulator()
sim.rk4_ivp(system_bodies, 0, days, dt/seconds_per_day, save_folder='../output/without_jupiter/')
# sim.rk4_ivp([sun, earth], 0, 1*day_in_sec, day_in_sec)
# plot.plot_traj([body.name for body in system_bodies if body.name != 'sun'])
```

```
import numpy as np
import sys
import os
sys.path.insert(0, os.path.abspath(
    os.path.join(os.path.dirname(__file__), '..')))

from dynamics.body import Body
from properties import prop

from tqdm import tqdm

class Simulator:
    def rk4_v_step(self, object:Body, force:np.ndarray, t_step:float):
        ''' Perform a single step of the RK4 algorithm for the velocity of an object

        Args:
            object (Body): The object to be updated
            force (np.ndarray): The net force vector on the object
            t_step (float): The time step to be used

        Returns:
            velocity (np.ndarray): The updated velocity of the object
        '''
        a = force / object.body_mass
        k1 = a
        k2 = a + 0.5 * t_step * k1
```

```
        k3 = a + 0.5 * t_step * k2
        k4 = a + t_step * k3

        v = object.get_velocity() + (t_step / 6) * (k1 + 2 * k2 + 2 * k3 + k4)

        return v, a

    def rk4_p_step(self, object:Body, force:np.ndarray, t_step:float) -> None:
        ''' Perform a single step of the RK4 algorithm for the position of an object
            Calculates both the updated position and updated velocity of the object
            Updates the state of the body object
        Args:
            object (Body): The object to be updated
            force (np.ndarray): The net force vector on the object
            t_step (float): The time step to be used

        Returns:
            None
        '''
        v, a = self.rk4_v_step(object, force, t_step)

        k1 = v
        k2 = v + 0.5 * t_step * k1
        k3 = v + 0.5 * t_step * k2
        k4 = v + t_step * k3

        p = object.get_position() + (t_step / 6) * (k1 + 2 * k2 + 2 * k3 + k4)

        object.set_acceleration(a)
        object.set_velocity(v)
        object.set_position(p)
        # print(object.get_acceleration())
        object.store_position.append(p.tolist())

    def rk4_ivp(self, objects:list, t_start:float, t_end:float, t_step:float, save_folder:str='
    ../output/') -> None:
        ''' Perform the RK4 algorithm for a system of objects

        Args:
            objects (list): A list of Body objects
            t_start (float): The start time of the simulation
            t_end (float): The end time of the simulation
            t_step (float): The time step to be used

        Returns:
            None
        '''
        t = t_start
        seconds_per_day = 24*60*60
        positions = []
        for object in objects:
            if object.name != 'sun' and object.calculate_forces == False:
                position = np.load(os.path.join(os.path.dirname(__file__), '../data/npy_files/')
    + object.name + '_trajectory.npy')
                positions.append(position)
        positions = np.array(positions)
        for t in tqdm(np.arange(t_start, t_end, t_step)):
            # Store the net force vector on each object
            force_list = []
            for i in range(len(objects)):
                force_list.append(objects[i].get_forces())

            for i in range(len(objects)):
                if objects[i].calculate_forces == True:
                    self.rk4_p_step(objects[i], force_list[i], t_step)
                elif objects[i].name != 'sun':
                    position = positions[:,i][int(t/seconds_per_day)][0:3]
                    objects[i].store_position.append(position)
```

```
            t += t_step

        path = os.path.join(os.path.dirname(__file__), save_folder)
        if os.path.exists(path) == False:
            os.mkdir(path)
        else:
            test=os.listdir(path)

            for item in test:
                if item.endswith(".npy"):
                    os.remove(os.path.join(path, item))

        for object in objects:
            if object.name != 'sun':
                filename = os.path.join(os.path.dirname(__file__), save_folder) + object.name + '_trajectory'
                np.save(filename, np.array(object.store_position), True)
```

```
import glob, os
import time
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from tqdm import tqdm

directory = os.path.join(os.path.dirname(__file__),'..', 'output', 'without_jupiter')
earth_orbit_radius = 149E12 # m

def get_trajectories_of_bodies(with_jupiter: bool):
    str = ''
    if(with_jupiter):
        str = 'with_jupiter'
    else:
        str = 'without_jupiter'
    directory = os.path.join(os.path.dirname(__file__),'..', 'output', str)
    os.chdir(directory)
    body_names = []
    appx_trajectories = {}
    for file in glob.glob("*.npy"):
        appx_trajectory = np.load(os.path.join(directory, file))
        body = file.replace('_trajectory.npy', '')
        body_names.append(body)
        appx_trajectories[body] = appx_trajectory
    return body_names, appx_trajectories

def get_min_distance_from_earth(body_traj):
    orbit_radius = np.linalg.norm(body_traj, axis=1)
    min = np.min(np.abs(orbit_radius - np.ones_like(orbit_radius)*earth_orbit_radius))
    index_of_min = np.where(orbit_radius == min)
    return min, index_of_min

def get_traj_diff_percent(min_dist_with_jupiter, min_dist_without_jupiter):
    diff = min_dist_without_jupiter - min_dist_with_jupiter
    # norm_orig = np.linalg.norm(min_dist_with_jupiter)
    # print(norm_orig)
    return diff/min_dist_with_jupiter*100

def get_difference_in_traj_near_earth():
    '''Returns the difference in trajectories of the asteroids at their nearest approach to Earth's orbit

    Returns:
        percent_diff (list): List of the percent difference in trajectories of the asteroids at their nearest approach to Earth's orbit
        body_names (list): List of the names of the bodies
    '''
    body_names, appx_pos_with_jupiter = get_trajectories_of_bodies(True)
    body_names, appx_pos_without_jupiter = get_trajectories_of_bodies(False)
```

```
    earth_traj_with_jupiter = appx_pos_with_jupiter['earth']
    earth_traj_without_jupiter = appx_pos_without_jupiter['earth']
    percent_diff = []
    for body_name in body_names:
        with_jupiter = appx_pos_with_jupiter[body_name]
        without_jupiter = appx_pos_without_jupiter[body_name]

        diff_with, ind_with = get_min_distance_from_earth(with_jupiter)
        diff_without, ind_without = get_min_distance_from_earth(without_jupiter)

        percent_diff.append(get_traj_diff_percent(diff_with, diff_without))

    return percent_diff, body_names


percent_diff, body_names = get_difference_in_traj_near_earth()
for i in range(len(percent_diff)):
    if(body_names[i].startswith('asteroid')):
        print(f'{body_names[i]} percent difference: {percent_diff[i]}')
```


## Acknowledgments

The authors of this paper would like to thank the course staff in their support for developing this complex problem. We would also like the Illinois Space Society's Spaceshot Avionics Guidance, Navigation, and Controls Team for providing inspiration for the code base. We would finally like to thank NASA JPL for providing the data that made this project possible.


## References


[1] Robutel, P., and Souchay, J., *Dynamics of Small Solar System Bodies and Exoplanets*, Lecture Notes in Physics, Springer, 2010, p. 197.

[2] Quan-Zhi Ye, e. a., "Finding Long Lost Lexell's Comet: The Fate of the First Discovered Near-Earth Object," *The Astronomical Journal*, Vol. 4, 2018, pp. 155–163.

[3] Giorgini, J., and Group, J. S. S. D., "NASA/JPL Horizons On-Line Ephemeris System," , 2022. URL `https://ssd.jpl.nasa.gov/horizons/`.

[4] Ma, C., and Feissel, M., "Definition and realization of the International Celestial Reference System by VLBI astrometry of extragalactic objects," *Central Bureau of IERS - Observatoire de Paris*, 1997, p. 282.

[5] Peale, S. J., "Celestial Mechanics," , 2022. URL `https://www.britannica.com/science/celestial-mechanics-physics`.

[6] NASA JPL, C. f. N. E. O. S., "Asteroid Watch: Keeping an Eye on Near-Earth Objects," , 2018. URL `https://www.jpl.nasa.gov/asteroid-watch`.

[7] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., "Array programming with NumPy," *Nature*, Vol. 585, No. 7825, 2020, pp. 357–362. https://doi.org/10.1038/s41586-020-2649-2, URL https://doi.org/10.1038/s41586-020-2649-2.

[8] Hunter, J. D., "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, Vol. 9, No. 3, 2007, pp. 90–95. https://doi.org/10.1109/MCSE.2007.55.