A Method of Asteroid Intercept Trajectory through Computation and Modeling

Ed Chen

May 7, 2019

Abstract

Considering the process and methods of finding an algorithm to derive the orbit for an interceptor to any potentially hazardous asteroids or space debris.

1 Initial Problem

Consider the following problem. Given a simplified solar system model, with only 4 major celestial bodies (Sun, Earth, Jupiter, Saturn), devoid of all other celestial objects, we start at a day, March 10th 2018. At this date, there is an asteroid exactly 1.5 billion kilometers away from the center of our solar system model. How do we determine whether this asteroid will potentially risk the near-earth are, and if it does, what would the trajectory of an intercepting missile be?

Firstly, I set the exact parameters under which I'll be trying this problem. The year starts at an initial date March 10th, 2018, when the asteroid is 1.5 billion kilometers away from the center of the solar system, making it reach just outside the orbit of Saturn. I plan to only model this with the simplified solar system, only including the Sun, Earth, Jupiter, and Saturn, but I may later add Mars, Venus, and our Moon to try the potential possibility of gravity assists to expedite the process. The asteroid in question will have approximately a velocity between 10 m/s to 70 m/s and a mass of approximately 3×10^9 kg to assume reasonable conditions.

2 Initial Method Summary

From the basic consideration of the problem, it will be obvious that we need to develop a general model of the solar system. This can be developed using several application of Celestial Mechanics, Astrodynamics, and Orbital Mechanics. Namely, I will explore the following

1. Gravitational Potential of a Contiguous Sphere
2. n-Body Equations of Motion
    - n-body Equations of Motion
    - 3 body Equation of Motion
3. n-Body Ephemeris
4. Interception Methods
    - Hohmann Orbit Transfers
    - Clohessy Wiltshire Equations
5. Numerical Integration for Modeling
    - RK4 Methods
6. Interception Modeling
7. Gravitational Assists and Further Advances

In order to model the solar system, we will have to develop either ways to link together numerous 3 body orbit transfer equations such as the Hohmann transfer or the Clohessy-Wiltshire Equations, or develop a full n-body ephemeris of the solar system using Newton's n-body Equations of Motion.

Also, for the actual interception of the asteroid, I will have to consider Rocket Dynamics as well as formulas such as the Clohessy Wiltshire Equations to intercept an already orbiting object.

Essentially, by using a general set of equations derived off Newton's Gravitational Law, we are able to model the relative interactions between different celestial bodies, which can be manipulated to form a relatively accurate Solar System model, which in concept is a n-Body ephemeris. With this model, we can use different iterations and equations derived from these gravitational law to determine the path of an intruding object, essentially just another object in the n-body system. Using methods such as Numerical Integration, I can use coding languages such as Matlab, C++, or Python to execute such a simulation. Extremely helpful to this process, I'll use the JPL Horizons library to obtain specific values on the various Celestial bodies I will be working with.

Following this, I will use different applications of Rocket Dynamics to effectively find an intercept trajectory to this potentially hazardous asteroid, since it cannot be fully incorporated into the n-body model, as it is affected by its own thrust factor, instead of just the relative gravitational pull of the other objects.

Modeling the aforementioned system will be beneficial for determining the point where the trajectories of the asteroid and intercept missile intersect, utilizing various methods of Numerical Analysis to effectively approximate the solutions to the system, which has no closed form.

3 Gravitational Potential of a Contiguous Sphere

Firstly, in order to model any of the gravitational interactions between multiple celestial bodies, we obviously need to assert the gravitational pull between different celestial bodies. Although Newton's universal law of gravitation is common knowledge, his proof of the Shell Theorem is much less well known, and is considerably helpful in considerations of these situations. A bit of textbook digging gives us

Given that we have 2 point masses, we already know that the gravitational force between them is defined by $F = G\frac{m_1 m_2}{r^2}$. Now, given that instead of point masses we have contiguous, constant density spheres, then what the gravitational force between them would be.

First, let us assume that there are a number N particles grouped around a general vicinity, forming the total mass

$$M = \sum_{N}^{i=1} m_i$$

Defining the position vector from any point mass m_i to m as

$$\hat{u}_i = \frac{\vec{r}_i}{\|r_i\|}$$

where $\vec{r}_i$ is the position vector between a particle m_i and m, making $\hat{u}_i$ the force of gravity on m by the set of points m_i equal to

$$F_i = -\frac{Gmm_i}{\|r_i\|^2}\hat{u}_i = -\frac{Gmm_i}{\|r_i\|^3}\vec{r}_i \tag{1}$$

simply by using Newton's law of Gravitation. Plugging in the definition for Gravitational Potential Energy, essentially, using the gravitational force law to find the change in work, we get

$$PE_i = -G\frac{mm_i}{\|r_i\|} \tag{2}$$

Thus, if we go back to considering the whole system we find that since the net potential energy of the system is equal to sum of all the individual potential energies of the point masses, the gravitational force would equal

$$F = -\nabla PE = -\left(\frac{\partial PE}{\partial x}\hat{i} + \frac{\partial PE}{\partial y}\hat{j} + \frac{\partial PE}{\partial z}\hat{k}\right) \tag{3}$$

However, our problem seeks to solve for contiguous spheres, not collections of point masses. Using the following diagram we can effectively

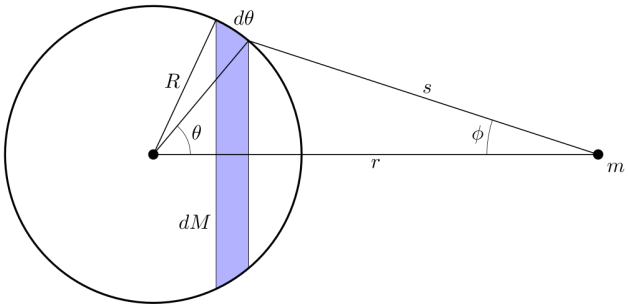


Figure 1: A solid sphere and a point mass m

Consider a similar situation, but instead of point masses we have a solid contiguous sphere with a combined mass M and a constant mass density ρ, making the mass of the point masses

$$M = \iiint_{v} \rho\, dv$$

This, makes the equation for Gravitational Potential Energy $dPE = -G\frac{mdM}{s} = -Gm\frac{dv}{s}$ through differentiating by i. Thus, by this equation the total sum of the Potential Gravitational Energy from each of the now contiguous N particles is

$$PE = V = -Gm \iiint_{v} \frac{dv}{s}$$

Now, we seek to evaluate M so we can substitute it into V and drastically reduce the equation. Since the sphere, is in fact, a sphere, we will use a triple integral in spherical coordinates to evaluate M as it is much easier than doing it in say, Cartesian coordinates. The volume in spherical coordinates is

$$dv = \prime^2 sin\ \ d\ \ d\prime d$$

making the equation for M with a radius R become

$$M = \int_{=0}^{2} \int_{\prime=0}^{R} \int_{=0} \prime^2 sin\ \ d\ \ d\prime d$$

by integrating around both the and the parameters, and setting the radius to R. Bringing out the inside elements of the triple integral, we can separate and simplify the d and d integrals, leaving us with

$$M = (2\ \ )(2) \left(\int_{0}^{R} \prime^2 d\prime \right)$$

Substituting this back into the equation for potential energy, it yields

$$V = -Gm \int_{0}^{R} \int_{0} \frac{sin\ \ d}{s} \prime^2 d\prime \Bigg)$$

Next, in order to solve for s, we use the law of cosines and differentiate it with respect to to yield

$$sin\ \ d\ = \frac{sds}{\prime r}$$

which works out to $\frac{2}{r}$, and once substituted into the equation for M, and then V, yields the formula for Gravitational Potential Energy

$$\boxed{V = \frac{-2\ \ GmM}{4} \frac{2}{r} = -\frac{GmM}{r}} \tag{4}$$

Essentially, this equation gives the force of a sphere on a nearby point mass. This, in practice, shows that the gravitational effect of 1 celestial body on another is equal to the gravitational effects on each other by a point mass in the center of the respective bodies.

4 N-Body Equations of Motion

Next, we will seek to understand the equations that govern the relative movements due to gravitational force of a system of n celestial bodies. Provided with position, velocity, and time, of a number n celestial bodies, we seek to model and predict true orbit motions with their gravitational interactions.

4.1 General Formulas for N-body systems

First off, we consider a number n point masses each representing a celestial body, since it is shown that the gravitational force between a solid sphere and another point is equivalent to the gravitational force between the point masses in the center of these spheres (by Section 3). Every point mass m_i has a position vector q_i, which by Newton's Second Law yields that ma equals the sum of the forces affecting a mass

$$F = ma = m_i \frac{d^2 q_i}{dt^2} \tag{5}$$

and thus by Newton's Gravitational Law says that

$$F_{ij} = \frac{G m_i m_j (q_j - q_i)}{\|q_j - q_i\|^3} \tag{6}$$

where F_{ij} denotes the gravitational force exerted on i by j, so it naturally follows that $F_{ij} = -F_{ji}$. Thus by summing all the gravitational forces on a specific object i using the formula (6), we can equate it to equation (5) to get

$$m_i \frac{d^2 q_i}{dt^2} = \sum_{j=1; j \neq i} \frac{G m_i m_j (q_j - q_i)}{\|q_j - q_i\|^3} \tag{7}$$

using the Gravitational Potential Energy equation from earlier, (2), we can see that the equation (7) is also equal to

$$-\frac{\partial V}{\partial q_i} \qquad V = -\sum_{1 < j \; n} \frac{G m_i m_j}{\|q_j - q_i\|}$$

by equation (3). In addition, momentum is defined as $p = mv$, so in this case

$$p_i = m_i \frac{dq_i}{dt}$$

Through Hamiltonian mechanics, the equations of motion become

$$\frac{dq_i}{dt} = \frac{\partial E}{\partial p_i} \qquad \frac{dp_i}{dt} = -\frac{\partial E}{\partial q_i} \tag{8}$$

where E is the energy function ($E = T + U$), and T represents the kinetic energy which is represented by

$$T = \sum_{i=1} \frac{\|p_i\|^2}{2m_i}$$

which holds the Hamilton Equations of motion true, through the basic principle of Conservation of Energy. Essentially, this proves that the n-body problem is essentially a system of $6n$ first-order differential equations. Using the initial conditions of position, mass, and orbital tilt through the NASA JPL Horizons library, we are able to determine a model of the Solar System with a Numerical Method using approximation methods such as the Euler Method or Runge-Kutta Formulas.

4.2 Restricted 3-Body Equations of Motion

In order to further explore a simplified model of this, the 3 body Equations of Motion which can be solved if simplified with certain assumptions, hence the title Restricted 3 Body Equations of Motion. Firstly, by Equation (6) we get

$$\begin{aligned} F_{12} &= -F_{21} = \frac{Gm_1m_2(q_2 - q_1)}{\|q_2 - q_1\|^3} \\ F_{13} &= -F_{31} = \frac{Gm_1m_3(q_3 - q_1)}{\|q_3 - q_1\|^3} \\ F_{23} &= -F_{32} = \frac{Gm_2m_3(q_3 - q_2)}{\|q_3 - q_2\|^3} \end{aligned} \tag{9}$$

where every celestial body i has a mas m_i, a position vector q_i, and an acceleration vector $a_i = \ddot{q_i}$, where $\ddot{}$ denotes the second derivative. Thus, by the property described in equation (7) gives us that the equation modeling the motion of body 1 as

$$F_{12} + F_{13} = m_1a_1 \tag{10}$$

substituting in the formulas for the interactive forces from Equations (9) we get the following equations for the acceleration of the first celestial body

$$a_1 = \frac{Gm_2(q_2 - q_1)}{\|q_2 - q_1\|^3} + \frac{Gm_3(q_3 - q_1)}{\|q_3 - q_1\|^3} \tag{11}$$

Applying the same logic to the other 2 celestial bodies, we get

$$a_2 = \frac{Gm_1(q_1 - q_2)}{\|q_1 - q_2\|^3} + \frac{Gm_3(q_3 - q_2)}{\|q_3 - q_2\|^3} \tag{12}$$

$$a_3 = \frac{Gm_1(q_1 - q_3)}{\|q_1 - q_3\|^3} + \frac{Gm_2(q_2 - q_3)}{\|q_2 - q_3\|^3} \tag{13}$$

remember that the position vectors are related to the accelerations by

$$\frac{d^2q_i}{dt^2} = a_i$$

which, when plugged in to equations (11 to 13) yields

$$\ddot{q_1} = \frac{Gm_2(q_2 - q_1)}{\|q_2 - q_1\|^3} + \frac{Gm_3(q_3 - q_1)}{\|q_3 - q_1\|^3} \tag{14}$$

$$\ddot{q_2} = \frac{Gm_1(q_1 - q_2)}{\|q_1 - q_2\|^3} + \frac{Gm_3(q_3 - q_2)}{\|q_3 - q_2\|^3} \tag{15}$$

$$\ddot{q_3} = \frac{Gm_1(q_1 - q_3)}{\|q_1 - q_3\|^3} + \frac{Gm_2(q_2 - q_3)}{\|q_2 - q_3\|^3} \tag{16}$$

a system of Second Order Ordinary Differential Equations with respect to time.

Now that we have obtained these equations, note that there is no closed-form solution for this system, but can be approximated to a impressively high accuracy using Numerical Methods through brute force solving for each

$$q_i = \begin{matrix} 8 \\ < \\ . \\ . \end{matrix} \begin{matrix} 9 \\ x_i = \\ y_i \\ z_i \end{matrix} ;$$

which will be looked into later. However, if we further make assumptions about the 3 masses, we are able to solve certain variations of the 3-body equations of motion. These smaller scenarios will be helpful in intercepting the specific asteroid as it approaches its near earth orbit, where the system can be modeled as one of the restricted 3-body equations.

5 Interception Methods

As for the methods to intercept this potentially hazardous asteroid, we have a couple options. Included in the n body model, it would have another Force element of thrust as it is not purely operating on gravitational pull as the other objects are. However, its mass would be so insignificant compared to the other bodies, that if we isolated it into a separate 3 body system, we could get a solution to a certain type of the restricted 3 body equation, which is described as follows.

5.1 Clohessy-Wiltshire Equations

Clohessy-Wiltshire Equations are the parametric equations modeling the velocity of a chaser spacecraft aiming to hit an object in circular around a point mass. Essentially, they model the relative motion of the the chaser and the satellite, and are particular used to determine logistics for satellite rendezvous.

$$\ddot{x} = 3n^2x + 2n\dot{y}$$
$$\ddot{y} = -2n\dot{x}$$
$$\ddot{z} = -n^2z$$

These equations are considered to be the first simple model of the motion of chaser spacecraft, and greatly contributed to more detailed models for future space projects. In short, the basic model that the Clohessy-Wiltshire equations describes is a stepping stone for further advanced models used to eventually propagate space rendezvous for future aerospace missions.

In this case, these equations would be helpful in order to model the first stage of the Rocket interception process, as we would need an initial missile in orbit to set launch as opposed to directly shooting at the intrusive asteroid.

5.1.1 Derivation

First, we begin with the equations for a three-body motion in space. Our 3 bodies in this case are the orbiting object, the chaser, and the central mass around which they orbit. From the three-body model, taking into account the Observed, Coriolis, Euler, and Centripetal accelerations that effect celestial bodies, we are able to expand the formula for acceleration, and thus

$$v = \frac{dr}{dt} + \omega \times r$$
$$a = \frac{d^2r}{dt^2} + 2\omega \times \frac{dr}{dt} + \frac{d\omega}{dt} \times r + \omega \times (\omega \times r)$$

Now assuming that one of the masses m is infinitesimally small, the solution of the restricted three-body problem by equation (7) yields that

$$m\left(\frac{d^2r}{dt^2} + 2\omega \times \frac{dr}{dt} + \omega \times (\omega \times r)\right) = -\frac{Gmm_1}{p_1^3}p_1 - \frac{Gmm_2}{p_2^3}p_2 \tag{17}$$

$$p_1 = r - r_1 \qquad p_2 = r - r_2 \qquad r = r_3 = \frac{m_1r_1 + m_2r_2}{m_1 + m_2} = \mathbf{i} + \mathbf{i} + \mathbf{i} \tag{18}$$

$$\omega = \omega\,\mathbf{i} \qquad w^2 = \frac{G(m_1 + m_2 + m)}{r_{12}^3} \approx \frac{G(m_1 + m_2)}{r_{12}^3} \tag{19}$$

where $(i_x; i_y; i_z)$ and $(\mathbf{i}\,;\mathbf{i}\,;\mathbf{i}\,)$ are a set of orthogonal unit vectors, so that any vector v can be equated to $r = \mathbf{i} + \mathbf{i} + \mathbf{i}$. Angular velocity is given by ω, Gravity by G, the position vectors of $(m; m_1; m_2)$ by $(r; r_1; r_2)$ respectively, and $(p_1; p_2)$ are the relative distances from m to the center of rotation of $(m_1; m_2)$ respectively.

To setup these equations to fit a spacecraft scenario, m denotes the mass of the chase spacecraft, m_1 denotes the target spacecraft, and m_2 denotes the central body of mass. Thus, as m_1 and m_2 are on the same axis since the target spacecraft is already in orbit around the central body of mass,

$$r_1 = r_1 i \quad \text{and} \quad r_2 = r_2 i$$

Additionally, since the masses m and m_1 are both negligible in the presence of the central body of mass m_2, they will both be assumed to be infinitesimals. As a result, with the spacecrafts treated as point masses, only the gravitational force G that the central body has on both the spacecraft is taken into account, since the gravitational effects of the 2 spacecraft are small enough to be ignored. In scope of real life applications, these assumptions are quite reasonable, as the satellites shot into space dwarf the Earth itself.

As such, the center of the system will be grounded at the central mass itself, making $r_2 = 0$, so that $r = p_2$ by the equality stated in 18. Furthermore, the vector p_1 will become the position of the chase spacecraft relative to the target spacecraft. Thus, by equation 19, angular velocity is

$$!^2 = \frac{Gm_2}{r_1^3} \qquad !^2 r_1^3 = Gm^2$$

plugging all these equalities into equation 17, we find that the equation governing the motion of the chase spacecraft can be described by

$$\frac{d^2p}{dt^2} + 2! \times \frac{dp}{dt} + ! \times (! \times (p + r_1)) = -\frac{! r_1^3}{r^3} r \tag{20}$$

factoring, and taking the first few terms of the Taylor series expansion,

$$\frac{r_1}{r} = (1 + 2i \ \cdot i_{r_1} x + x^2)^{\ \frac{1}{2}} = 1 - i \ \cdot i_{r_1} x + \cdots \tag{21}$$

plugging in the first few terms of the Taylor Series, we see that

$$\frac{d^2p}{dt^2} + 2! \times \frac{dp}{dt} + ! \times (! \times p) = -!^2 p + 3!^2 (i \ \cdot p) i \ + O(p^2) \tag{22}$$

then it follows that since $p = \ i \ + \ i \ + \ i \ $ and $! \times (! \times p) = -!^2 (\ i \ + \ i \ )$, the equation modeling the motion of the chase spacecraft relative to the target spacecraft is

$$\frac{d^2p}{dt^2} + 2! \times \frac{dp}{dt} = -!^2 \ i \ + 3!^2 \ i \ + O(p^2) \tag{23}$$

in scalar form, this becomes

$$\frac{d^2}{dt^2} - 2! \frac{d}{dt} - 3!^2 \ = 0 \tag{24}$$

$$\frac{d^2}{dt^2} + 2! \frac{d}{dt} = 0 \tag{25}$$

$$\frac{d^2}{dt^2} + !^2 \ = 0 \tag{26}$$

to simplify, we will express position as

$$p = r = x i \ + y i_r - z i_z \quad i_{r_1} = i_r \quad ! = -! i_z$$

and as such, we arrive at the Clohessy-Wiltshire Equations

$$\frac{d^2x}{dt^2} + 2! \frac{dy}{dt} = 0$$

$$\frac{d^2y}{dt^2} - 2! \frac{dx}{dt} - 3!^2 y = 0$$

$$\frac{d^2z}{dt^2} + !^2 z = 0$$

5.1.2 Solution

$$\ddot{x} = 3n^2x + 2n\dot{y} \tag{27}$$

$$\ddot{y} = -2n\dot{x} \tag{28}$$

$$\ddot{z} = -n^2z \tag{29}$$

For the Clohessy Wiltshire equations, the angular velocity denoted by $!$ can be expanded to $n = ! = \sqrt{\frac{\mu}{r^3}}$, where r is still denoting the position vector of the chaser spacecraft. However, for a circular target orbit, it is unnecessary to expand $!$. Later, for Numerical Methods, n can be obtained through the NASA JPL Horizons database

To start, we notice that equation 28 can be expressed as

$$\frac{d}{dt}(\dot{y} + 2n\ x) = 0 \tag{30}$$

$$\dot{y} = C_1 - 2n\ x \tag{31}$$

and substituting this into 27 gives

$$\ddot{x} + n^2\ x = 2nC_1 \tag{32}$$

this equation, when solved and differentiated gives the x parametric of the relative velocity of the chaser spacecraft

$$x = \frac{2}{n}C_1 + C_2sin(nt) + C_3cos(nt) \tag{33}$$

$$\dot{x} = C_2ncos(nt) - C_3nsin(nt) \tag{34}$$

Substituting 33 into 31 gives us the y component of the relative velocity vector of the spacecraft

$$\dot{y} = -3C_1t + 2C_2nsin(nt) - 2C_3ncos(nt) \tag{35}$$

and integrating this with respect to time yields

$$y = -3C_1t + 2C_2cos(nt) - 2C_3sin(nt) + C_4 \tag{36}$$

now, to find the constants, we use the initial conditions that

$$\text{When } t = 0 \quad x = x_0 \quad y = y_0 \quad \dot{x} = \dot{x}_0 \quad \dot{y} = \dot{y}_0$$

Evaluating equations 33 to 36 with these given initial conditions, at $t = 0$,

$$\frac{2}{n}C_1 + C_3 = x_0 \qquad C_2n = \dot{x}_0$$
$$-3C_1 - 2C_3n = \dot{y}_0 \qquad 2C_2 + C_4 = y_0$$

Solving these equations, the constants are determined to be

$$C_1 - 2n\ x_0 + \dot{y}_0 \qquad C_2 = \frac{1}{n}\ \dot{x}_0 \qquad C_3 = -3\ x_0 - \frac{2}{n}\ \dot{y}_0 \qquad C_4 = -\frac{2}{n}\ \dot{x}_0 + y_0$$

Finally, for the z axis we need to solve the last equation. Its solution is of the same format as 33, but $C_1 = 0$

$$z = C_5sin(nt) + C_6cos(nt)$$

differentiating this, we get the velocity relative to the existing orbit

$$\dot{z} = C_5 n\cos(nt) - C_6 n\sin(nt)$$

likewise, the initial conditions are $z = z_0$ and $\dot{z} = \dot{z}_0$ at $t = 0$, which yields

$$C_5 = \frac{\dot{z}_0}{n} \qquad C_6 = z_0$$

Finally, to get the relative trajectory of the chaser spacecraft, we need to substitute all the C_n values into the equations, and yields a closed form solution

$$x = 4\left(x_0 + \frac{2}{n}\dot{y}_0\right) + \frac{\dot{x}_0}{n}\sin(nt) - \left(3x_0 + \frac{2}{n}\dot{y}_0\right)\cos(nt)$$

$$y = y_0 - \frac{2}{n}\dot{x}_0 - 3(2nx_0 + \dot{y}_0)t + 2\left(3x_0 + \frac{2}{n}\dot{y}_0\right)\sin(nt) + \frac{2}{n}\dot{x}_0\cos(nt)$$

$$z = \frac{1}{n}\dot{z}_0\sin(nt) + z_0\cos(nt)$$

Plugging this in to the original set of equations, it does indeed solve them. From this solution, it is determined that only y has a term which grows linearly, and the other two components continuously oscillate. Thus, the linear term in y, $2nx_0 + \dot{y}_0$ must be equal to 0 or else the chaser satellite would simply go farther and farther from the target spacecraft. Expressed as a matrix, the solution of the system is given by

$$\begin{bmatrix} x(t) \\ y(t) \\ \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = \begin{bmatrix} 4-3\cos(nt) & 0 & \frac{1}{n}\sin(nt) & \frac{2}{n}(1-\cos(nt)) \\ 6(\sin(nt)-nt) & 1 & -\frac{2}{n}(1-\cos(nt)) & \frac{1}{4}(4\sin(nt)-3nt) \\ 3n\sin(nt) & 0 & \cos(nt) & 2\sin(nt) \\ -6n(1-\cos(nt)) & 0 & -2\sin(nt) & 4\cos(nt)-3 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ \dot{x}_0 \\ \dot{y}_0 \end{bmatrix}$$

$$\begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} \cos(nt) & \frac{1}{n}\sin(nt) \\ -n\sin(nt) & \cos(nt) \end{bmatrix} \begin{bmatrix} z_0 \\ \dot{z}_0 \end{bmatrix}$$

6 Numerical Integration of N-Body Systems

In order to effectively model an N-Body System, note that there are no closed-form solutions for the system, but however can be approximated to a high accuracy using Numerical Methods to manipulate data given by NASA JPL Horizons' Database. Firstly, recall that for a 3 body problem, the system of ODEs that models the system are given by (14 to 16)

$$\ddot{q_1} = \frac{Gm_2(q_2 - q_1)}{\|q_2 - q_1\|^3} + \frac{Gm_3(q_3 - q_1)}{\|q_3 - q_1\|^3} \tag{37}$$

$$\ddot{q_2} = \frac{Gm_1(q_1 - q_2)}{\|q_1 - q_2\|^3} + \frac{Gm_3(q_3 - q_2)}{\|q_3 - q_2\|^3} \tag{38}$$

$$\ddot{q_3} = \frac{Gm_1(q_1 - q_3)}{\|q_1 - q_3\|^3} + \frac{Gm_2(q_2 - q_3)}{\|q_2 - q_3\|^3} \tag{39}$$

In order to effectively solve for the acceleration vectors, and thus the position and velocity vectors, we start off by decomposing both the position and velocity vectors to its $x; y; z$ components. Thus, the position vector and velocity vector are

$$q_i = \begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix}, \qquad v_i = \begin{Bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{Bmatrix},$$

Plugging into Equations 37 to 39 yields

$$a_1 = \begin{Bmatrix} \ddot{x}_1 \\ \ddot{y}_1 \\ \ddot{z}_1 \end{Bmatrix} = \begin{Bmatrix} \frac{Gm_1(x_2 - x_1)}{q_{12}^3} + \frac{Gm_3(x_3 - x_1)}{q_{13}^3} \\ \frac{Gm_1(y_2 - y_1)}{q_{12}^3} + \frac{Gm_3(y_3 - y_1)}{q_{13}^3} \\ \frac{Gm_1(z_2 - z_1)}{q_{12}^3} + \frac{Gm_3(z_3 - z_1)}{q_{13}^3} \end{Bmatrix},$$

from Equation 37, and follows for the equations modeling the other 2 objects as well. Forming a vector y of all the desired values, we can get

$$\dot{y} = f = [v_1; v_2; v_3; a_1; a_2; a_3]$$

which in turn can be used to model the relative motions of the celestial objects in question, using approximation methods such as Runge Kutta or the Euler Method.

Returning back to the original n-Body model in question, we can essentially use the same method shown above to calculate the values of the 3 body system to model more complex ones. Given the general formula for the acceleration equation that models the general motion of an n-body system, recall that Equation 7

$$m_i \frac{d^2 q_i}{dt^2} = \sum_{j=1; j \neq i} \frac{Gm_i m_j (q_j - q_i)}{\|q_j - q_i\|^3}$$

gives the system of number i 2nd order ODEs that models the motion for number i bodies. Following by the same guidelines which we used to integrate the 3 body equation, we have to take each body and determine its acceleration through the sum of the gravitational forces caused by all the other celestial bodies, not including itself. To make this a tad bit simpler, note that 7 can be simplified to

$$\frac{d^2 q_i}{dt^2} = \sum_{j=1; j \neq i} \frac{Gm_j (q_j - q_i)}{\|q_j - q_i\|^3}$$

by dividing both sides my m_i, leaving us with the formulas for acceleration. Thus, this becomes a function of the position vectors purely. To code this, since we will need to run many cycles to get a reasonably shaped approximation, we lean towards more computationally efficient languages such as Cpp. Written in code, this becomes

```
body target = m_bodies[body_index];
for (int index = 0; index< m_bodies.size(); index++)
       {
              if (index != body_index)
              {
                  body & external = m_bodies[index];
                      double dx = target.location.x - external.location.x;
                      double dy = target.location.y - external.location.y;
                      double dz = target.location.z - external.location.z;
                      double q = sqrt(dx*dx+dy*dy+dz*dz);
                      auto dm = G_const * external_body.mass / (q*q*q);
                      acceleration = acceleration + (external.location - target.location) * dm;
              }
       }
       return acceleration;
}
```

Essentially, for all the celestial bodies that are not the target body itself, we calculate the effective acceleration that results from its gravitational interactive force. Through calculating the magnitude or distance between the 2 bodies, we get a number q which is then used to calculate the right side of Equation 7 and added to the total acceleration for that specific celestial body. Following this, we can use various approximation methods with the specific accelerations to model the specific orbits

6.1 Runge-Kutta Methods

Following this, in order to approximate the position vectors from the acceleration we have determined, we can use Runge Kutta or Euler methods, both iterative methods to approximate the solutions of ODEs. Firstly, since we do have an initial value problem in the form

$$\dot{y} = f(t; y) \qquad y(t_0) = y_0$$

Now, using a step size for t, in our case which will be time we can approximate the next value by using

$$t_{n+1} = t_n + h$$

and then approximating it in 4 segments progressively,

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

$$k_2 = f(t_n + \frac{h}{2}; y_n + h\frac{k_1}{2})$$

$$k_3 = f(t_n + \frac{h}{2}; y_n + h\frac{k_2}{2})$$

$$k_4 = f(t_n + h; y_n + hk_3)$$

and then, this yields that

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

For our problem, acceleration would become y, and time would become t. Coded out, using Cpp we can calculate the formulas for k using

```
k1 = (external_body->location - target_body.location) * tmp;

velocity_update = partial_step(target_body.velocity, k1, 0.5);
location_update = partial_step(target_body.location, velocity_update, 0.5);
k2 = (external_body->location - location_update) * tmp;
```

```
velocity_update = partial_step(target_body.velocity, k2, 0.5);
location_update = partial_step(target_body.location, velocity_update, 0.5);
k3 = (external_body->location - location_update) * tmp;

velocity_update = partial_step(target_body.velocity, k3, 1);
location_update = partial_step(target_body.location, velocity_update, 1);
k4 = (external_body->location - location_update) * tmp;
```

and then, calculate the new acceleration using

```
acceleration += (k1 + k2 * 2 + k3 * 2 + k4) / 6;
```

All in all, this - when run up to an increasingly small timestep, is incredibly accurate in modeling the orbits of our solar system, running it at a timestep of 10 seconds per update for a total of 10,301 times yields

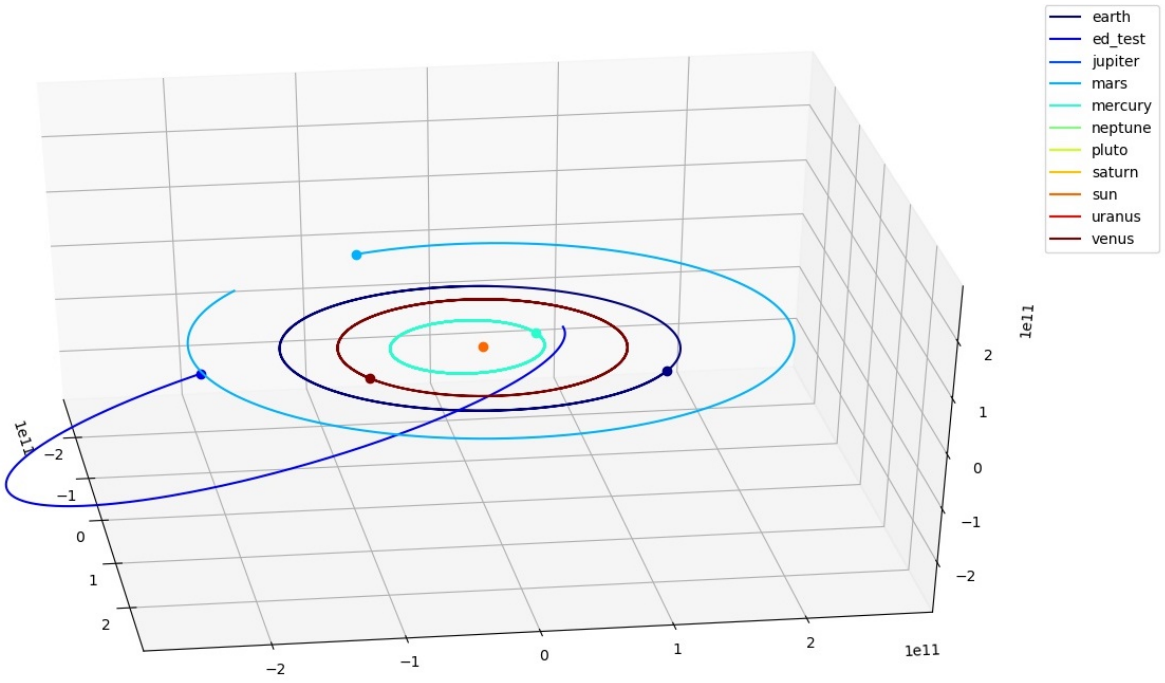


Figure 2: N-Body Simulation

7 Interception Modeling

In order to model the interception trajectory of a possible projectile to deter possibly threatening asteroids, we need to first determine what a 'hazardous asteroid' is. Due to the wide possibilities of trajectory deviation, for the purposes of this project, 'hazardous asteroid' will be considered as any asteroid that falls within 5×10^6 kilometers of Earth, roughly 2 times our moon's orbit on either side.

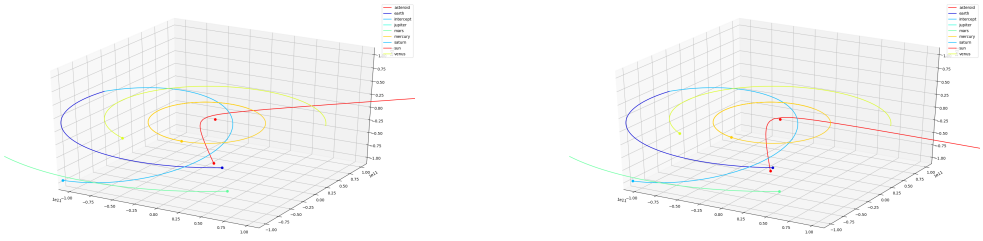
Ideally, we would want to have an equation to determine this, that we could solve an equation and find the possible locations where this asteroid would be considered hazardous. However, by using a general approximation, we cannot do that, since the way we find an orbit at a time t is by repeatedly running iterative methods such as the RK4 Method described above. Simply since they are iterative we cannot equate and evaluate them

$$\frac{d^2 q_i}{dt^2} = \sum_{j=1; j \neq i} \frac{G m_j (q_j - q_i)}{\|q_j - q_i\|^3}$$

the only way would to be computationally brute force the values. As such, I used a computational method commonly known as gradient-step, which is used mainly in applications such as AI and adaptive learning. Essentially, starting off for a fixed C in one axis, the other 2 axis could be related to each other with an equation. Off this equation, there would be 10 data points run, and then determine the shortest distance between the asteroid and the planet Earth. To do this, I used Perl

```
print "file1=$ARGV[0]\n";
print "file2=$ARGV[1]\n";
open(F,"<$ARGV[0]");
@ref_obj=();
$_=<F>;
chop;
while(<F>){
  chop;
  push @ref_obj,[split(/,/,$_)];
}
close F;
open(F,"<$ARGV[1]");
@target_obj=();
$_=<F>;
chop;
while(<F>){
  chop;
  push @target_obj,[split(/,/,$_)];
}
close F;
print "size=",scalar @ref_data,"\n";
#print "@ref_obj\n";
$min=-1;
for($i=0;$i<$#ref_obj;$i++){
  $dx=($ref_obj[$i][0]-$target_obj[$i][0]);
  $dy=($ref_obj[$i][1]-$target_obj[$i][1]);
  $dz=($ref_obj[$i][2]-$target_obj[$i][2]);
  $dist=sqrt($dx*$dx+$dy*$dy+$dz*$dz);
  if($min<0 || $dist<$min){
    $min=$dist;
    $t_min=$i;
    $min_line=join(",",@{$ref_obj[$i]});
  }
  print "t=",$i*10,", $dist ref=", join(",",@{$ref_obj[$i]}), " target=",join(",",@{$target_obj[$i]}),"\n";
}
printf "t_min=%g, dist=%g\n",$t_min*10,$min;
```

to first import the appropriate lines of data from the .dat files, and then brute calculate the distance between the 2 objects using the distance formula. By finding the shortest point between earth and the asteroid, we could effectively single out which asteroid trajectories could be considered as hazardous. In turn, the gradient step function which output the 10 possible values, singled out the one with the smallest all time shortest distance to earth, and then chose 4 more values around that, at a smaller step size to test. It gravitated towards the step size which produced the smallest distance between the asteroid and Earth, until the shortest distance would be less than 5×10^6, our 'hazardous' threshold.



(a) label 1

(b) label 2

Figure 3: Asteroid Paths

As shown above, where the red represents the asteroid, and the blue represents the orbital track of Earth, the gradient step program was quite effective in picking out the asteroids which were considered 'hazardous'. Because of the generally consistent orbital tilt in our solar system, many of the chosen asteroid trajectories were along the parallel axis of our solar system, asteroids coming in at the top/bottom could only be briefly affected by the sun's gravity, and thus rarely differentiated in trajectory paths. Mainly, the slingshot of other planets and the sun provided the main gravitational potential to hit the 'hazardous' zone.

Following this, we can use some properties of Hohmann transfers to model interplanetary transfers between the 2 orbits, the orbit of earth, and the orbit of the asteroid. By finding the effective velocity vector of the Hohmann transfer between these 2 orbits, and then doing the computation to factor in the effect of the rest of the nBody celestial model we are able to find that the velocity vector is equivalent to

$$V_D = \sqrt{\frac{2}{sun}} \sqrt{\frac{R_2}{R_1(R_1 + R_2}}$$

where R_1 is the magnitude of the position vector of Earth at departure, and R_2 is the magnitude of the position vector of the asteroid at arrival, otherwise known as the closest point. As such, this yields the final model

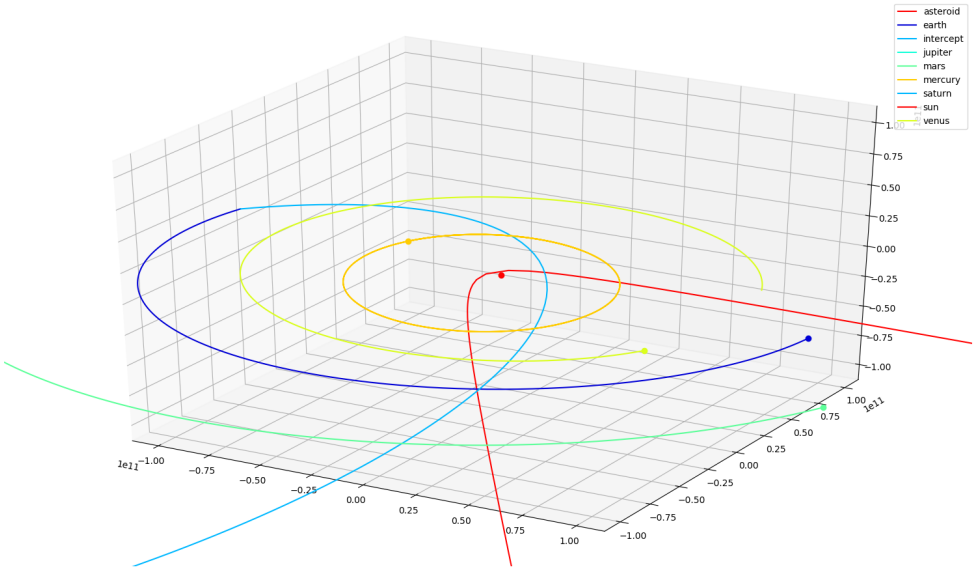


Figure 4: Asteroid Collision

where the light blue represents the intercept trajectory, dark blue represents the trajectory of earth, and the red represents the trajectory of the asteroid.

-Ed Chen March. 2018

```
#include <math.h>
#include <omp.h>

#include "datastructures.h"
#include "orbit_integration.h"


point Orbit_integration::Euler::calculate_single_body_acceleration(int body_index)
{
	double G_const = 6.67408e-11; //m3 kg - 1 s - 2
	point acceleration{ 0, 0, 0 };
	body target_body = m_bodies[body_index];


// #pragma omp parallel for
	for (int index = 0; index< m_bodies.size(); index++)
	{
		if (index != body_index)
		{
			body &external_body = m_bodies[index];
			double dx = target_body.location.x - external_body.location.x;
			double dy = target_body.location.y - external_body.location.y;
			double dz = target_body.location.z - external_body.location.z;
			double r = sqrt(dx*dx+dy*dy+dz*dz);
			auto tmp = G_const * external_body.mass / (r*r*r);
		//	#pragma omp critical
			acceleration = acceleration + (external_body.location - target_body.location) * tmp;
		}
	}
	return acceleration;
}

void Orbit_integration::Euler::compute_velocity()
{
#pragma omp parallel for
	for (int i = 0; i < m_bodies.size(); i++)
	{
		point acceleration = Orbit_integration::Euler::calculate_single_body_acceleration(i);
		m_bodies[i].velocity += acceleration * m_time_step;
	}
};

void Orbit_integration::Euler::update_location()
{
//#pragma omp parallel for
	for (int i = 0; i < m_bodies.size(); i++)
	{
		body &target_body = m_bodies[i];
		target_body.location += target_body.velocity * m_time_step;
	}
}

void Orbit_integration::Euler::compute_gravity_step()
{
	omp_set_num_threads(4);
	compute_velocity();
	update_location();
}
```

```
point Orbit_integration::RK4::calculate_single_body_acceleration(int body_index)
{
	double G_const = 6.67408e-11; //m3 kg - 1 s - 2
	point acceleration{ 0, 0, 0 };
	point velocity_update{ 0, 0, 0 };
	point location_update{ 0, 0, 0 };
	body target_body = m_bodies[body_index];

	int index = 0;
	for (auto external_body = m_bodies.begin(); external_body != m_bodies.end(); *external_body++, index++)
	{
		if (index != body_index)
		{
			point k1{ 0, 0, 0 };
			point k2{ 0, 0, 0 };
			point k3{ 0, 0, 0 };
			point k4{ 0, 0, 0 };

			double r = (pow((target_body.location.x - external_body->location.x), 2) +
					pow((target_body.location.y - external_body->location.y), 2) +
					pow((target_body.location.z - external_body->location.z), 2));

			r = sqrt(r);

			auto tmp = G_const * external_body->mass / (r*r*r);

			//k1 - acceleration at current location
			k1 = (external_body->location - target_body.location) * tmp;

			//k2 - acceleration 0.5 timesteps in the future based on k1 acceleration value
			velocity_update = partial_step(target_body.velocity, k1, 0.5);
			location_update = partial_step(target_body.location, velocity_update, 0.5);
			k2 = (external_body->location - location_update) * tmp;

			//k3 acceleration 0.5 timesteps in the future using k2 acceleration
			velocity_update = partial_step(target_body.velocity, k2, 0.5);
			location_update = partial_step(target_body.location, velocity_update, 0.5);
			k3 = (external_body->location - location_update) * tmp;

			//k4 - location 1 timestep in the future using k3 acceleration
			velocity_update = partial_step(target_body.velocity, k3, 1);
			location_update = partial_step(target_body.location, velocity_update, 1);
			k4 = (external_body->location - location_update) * tmp;

			acceleration += (k1 + k2 * 2 + k3 * 2 + k4) / 6;
		}
	}
	return acceleration;
}

point Orbit_integration::RK4::partial_step(point &f, point &df, double scale)
{
	return point{
		f.x + df.x * m_time_step * scale,
		f.y + df.y * m_time_step * scale,
		f.z + df.z * m_time_step * scale
	};
```

```
};


void Orbit_integration::RK4::compute_velocity()
{
#pragma omp parallel for
	for (int i = 0; i < m_bodies.size(); i++)
	{
		point acceleration = Orbit_integration::RK4::calculate_single_body_acceleration(i);
		m_bodies[i].velocity += acceleration * m_time_step;
	}
};

void Orbit_integration::RK4::update_location()
{
	for (auto target_body = m_bodies.begin(); target_body != m_bodies.end(); *target_body++)
	{
		target_body->location += target_body->velocity * m_time_step;
	}
}

void Orbit_integration::RK4::compute_gravity_step()
{
	compute_velocity();
	update_location();
}
```

```
#include "orbit_integration.h"

template <typename Integrator>
void run_simulation(Integrator integrator, int iterations, int report_frequency)
{
	for (auto i = 0; i < iterations; i++)
	{
		if (i % report_frequency == 0)
			record_state(integrator.get_bodies());
		integrator.compute_gravity_step();
	}
	output_states(integrator.get_bodies());
}

int main(int argc, char *argv[])
{
	std::vector<body> bodies;

	bodies.push_back(solar_system::sun);
	bodies.push_back(solar_system::mercury);
	bodies.push_back(solar_system::venus);
	bodies.push_back(solar_system::earth);
	bodies.push_back(solar_system::mars);
	bodies.push_back(solar_system::saturn);
	bodies.push_back(solar_system::jupiter);
//	bodies.push_back(solar_system::uranus);
//	bodies.push_back(solar_system::neptune);
//	bodies.push_back(solar_system::pluto);
	bodies.push_back(solar_system::asteroid);
	bodies.push_back(solar_system::intercept);


	Orbit_integration::RK4 orbit(bodies, 50);

	run_simulation(orbit, (int)350072*2, 1e3);
	printf("Done\n");

}
```

```
import math
import sys
import random
import matplotlib.pyplot as plot
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
import numpy as np

line = {}
pt = {}
def update_lines(i):
    for current_body in bodies:
        line[current_body["name"]].set_data(current_body["x"][:i],current_body["y"][:i])
        line[current_body["name"]].set_3d_properties(current_body["z"][:i])
        pt[current_body["name"]].set_data(current_body["x"][i-1],current_body["y"][i-1])
        pt[current_body["name"]].set_3d_properties(current_body["z"][i-1])
    return line

def plot_output(bodies, outfile = None):
    fig = plot.figure()
    #colours = ['r','b','g','y','m','c']
    color = plot.cm.jet(np.linspace(0, 1, len(bodies)+5))
    color_assigned={"asteroid":'r', "sun":'r', "mercury": color[9], "venus":color[8], "earth":color[1], "mars":color[6],"jupiter":color[5],
                    "saturn":color[4], "uranus":color[3], "neptune":color[2],"pluto":color[1], "intercept":color[4]}
    #print(color)
    k=0
    ax = fig.add_subplot(1,1,1, projection='3d')

    for current_body in bodies:
        max_range = max(max(current_body["x"]),max(current_body["y"]),max(current_body["z"]))
        body = current_body["name"]
        line[body],=ax.plot(current_body["x"][:1], current_body["y"][:1], current_body["z"][:1], c = color_assigned[body], label = body)
        pt[body],=ax.plot(current_body["x"][:1], current_body["y"][:1], current_body["z"][:1], 'o', c = color_assigned[body])
        k+=1

    ax.set_xlim([-max_range,max_range])
    ax.set_ylim([-max_range,max_range])
    ax.set_zlim([-max_range,max_range])
    ax.legend()

    ine_ani = animation.FuncAnimation(fig, update_lines, frames=len(current_body["x"]), interval=0.2)


    if outfile:
        plot.savefig(outfile)
    else:
        plot.show()

def read_bodies(files = []):
    bodies = []
    for file in files:
        with open(file) as f:
            name = str(f.readline()).strip('\n')
            x = []
            y = []
            z = []

            for line in f:
```

```
                input = line.strip('\n').split(',')
                x.append(float(input[0]))
                y.append(float(input[1]))
                z.append(float(input[2]))
            bodies.append({"name":str(name), "x":x, "y":y, "z":z})

    return bodies

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print ("Please include list of body files as arguments")
    names = sys.argv[1:]
    bodies = read_bodies(names)
    plot_output(bodies)
```

```
#!/usr/bin/perl
#
# Sample Perl script for use with JPL's Horizons web-based batch interface.
# Written 2006-Mar-02 by Alan B. Chamberlin (JPL/Caltech)
#
# Use of this script is at your own risk.
#
#use strict;
use LWP::UserAgent;

my @data = ();

# Load the input batch file:
while (<>)
{
  #
  # Skip comments:
  next if (/^\s*!/o);
  #
  # Strip off trailing line-endind:
  s/\s*$//o;
  s/^\s*//o;
  chomp;
  #
  # Remove any spaces surrounding '=' for compactness:
  s/\s*=\s*/=/o;
  #
  # Escape special URL characters (there may be others required as well):
  s/ /%20/g;
  s/\&/%26/g;
  s/;/%3B/g;
  s/\?/%3F/g;
  #
  # Store the modified command:
  push @data, $_;
}
#print("Data==>");
#print(join(",",@data));
#print("\n");

# Assemble the URL:
my $url = 'https://ssd.jpl.nasa.gov/';
$url .= 'horizons_batch.cgi?batch=1&';
$url .= join('&', @data);

# Setup the HTTP objects.
my $ua = new LWP::UserAgent;
my $req = new HTTP::Request;

# Send the URL:
$req->method("GET");
$req->url($url);
my $res = $ua->request($req);
die "$url\n" . $res->status_line . "\n" if ( $res->is_error );

# Display the results:
#print $res->content;
my @lines = split(/\n/,$res->content);
```

```
my $ln="";
my $first_line=0;
$i=0;
foreach $ln(@lines) {
  if($first_line==1){
    #print $ln,"\n";
    @F=split(/\s*,\s*/,$ln);
    foreach $i(2,3,4,5,6,7){
      $F[$i]*=1000.0;
    }
    print(" static body $target\{ {$F[2],$F[3],$F[4]},$mass, {$F[5],$F[6],$F[7]},\"$target\" };\n");
    #print("//",(sqrt($F[2]*$F[2]+$F[3]*$F[3]+$F[4]*$F[4])/1e11)),"\n";
    last;
  }
  if($ln =~ /^\$\$SOE/o){ $first_line=1;}
  if($ln =~ /Target body name: (\S+)/o){
    $target = lc($1);
  }
  if($ln =~ /^\s*Mass .*?\(10\^(\d+) kg\s*\)\s*=\s*([.\d]+)/o){
    eval("\$mass = $2*1e$1");
  }
  if($ln =~ /Mass, 10\^(\d+)\s+kg\s+=\s+([.\d]+)/o){
    eval("\$mass = $2*1e$1");
  }
  if($ln =~ /Mass\s+\(10\^(\d+)\s+kg\)\s+~\s+([.\d]+)/o){
    eval("\$mass = $2*1e$1");
  }
}
```