Development of CR3BP, ER3BP and N-Body Orbit Simulations Using Matlab

a project presented to
The Faculty of the Department of Aerospace Engineering San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Andrew Torricelli

December 2017

approved by

Prof. Jeanine Hunter
Faculty Advisor

Development of CR3BP, ER3BP and N-Body Orbit Simulations Using Matlab

A Torricelli[1]
San Jose State University, San Jose, CA, 95192

The Three-Body and N-body Problem has confounded the greatest physicists and mathematicians for centuries. In the many attempts for an elegant solution, this deceptively difficult problem has led to numerable advancements in mathematics. However, since its formulation by Newton, no closed-form solution has been found. Presently it is accepted that no such solution to the general three-body problem exists and never will. However, with some restrictions and computational tools, accurate simulations are possible. This paper will delve into the history of the Three-body problem and introduce mathematical concepts for solving the Circular Restricted Three-Body Problem (CR3BP), the Elliptic Restricted Three-Body Problem (ER3BP), and develop methods for solving an unrestricted N-Body Problem. Solutions to the restricted cases as well as the N-body are applied to Apollo missions for evaluation. The N-body goes a step further, simulating a wide range of orbital systems in various reference frames for evaluation.

Nomenclature

$\vec{r}_i$	=	Position vector of object *i*
R	=	Radius
R_E	=	Earth Radius
R_S	=	Sun Radius
a	=	Semi-major axis
G	=	Gravitational Constant
D	=	Distance from Earth to Moon
ρ	=	Mass ratio
i	=	Inclination
dt	=	Time step
ω	=	Argument of Perigee
P	=	Orbit Period
e	=	Eccentricity
θ	=	Angular Velocity
M	=	Mean Anomaly
n	=	Mean Motion
μ	=	Gravitation Parameter
ν	=	True Anomaly
$\vec{V}_i$	=	Velocity vector of object *i*
ΔV	=	Delta-V, Change in velocity
p_0	=	Initial position
V_0	=	Initial velocity
t	=	Time
m_i	=	Mass
$\lvert\vec{r}_1 - \vec{r}_2\rvert$	=	Distance between objects 1 & 2

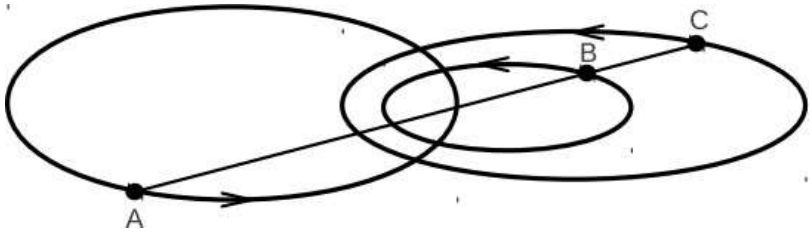
[1] SJSU MSAE Student, Aerospace Engineering, 1 Washington Square, San Jose, CA 95112

I. Introduction

THE three-body problem seeks to determine movements of three bodies in space under mutual gravitational interaction. A solution hopes to determine all future and past spatial locations of three bodies based solely on positions and velocities from a single instant in time. A general solution for the problem known as the *general three-body problem*, would describe these movements in 3 dimensions with no restrictions in mass, initial position or velocity. While the two-body problem has been solved completely from its inception, including an additional body to the problem has proven to be much more difficult. For centuries, physicists and mathematicians have been unsuccessful in their attempts to discover a closed-form solution, and as it is now known, no closed-form solution exists for the general three-body problem. Three-body motion is considered generally unpredictable. However, with some restrictions, namely one mass taken to be negligible, the problem becomes much easier. This is called the *restricted three-body problem*. There are two paths that can be taken in solving the restricted three-body problem, the simpler *circular restricted three-body problem* (CR3BP) where the two larger masses have circular orbits around their shared center of mass, or the more complicated but accurate *elliptic three-body problem* (ER3BP) where the larger masses move in elliptical orbits around their shared center of mass.

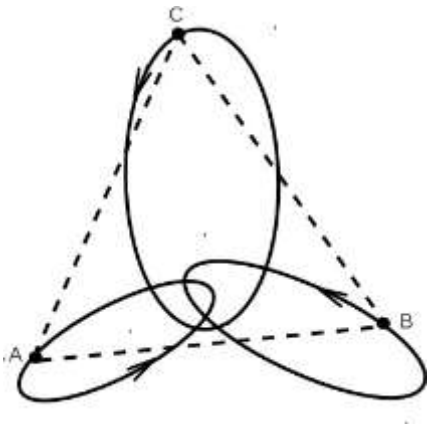
The three-body problem was first formulated by Isaac Newton in 1687 in the *Principia*. He extensively studied the motion of the Earth around the Sun and the Moon around the Earth, but found the problem difficult. Newton was only able to obtain approximate solutions to within 8% of known observations.

Later in 1767, Euler proposed a special form of the general three-body problem where the three bodies were places in a straight line. With sufficient initial conditions, the three bodies would move in elliptical orbits while preserving the straight line positions seen in 9Figure 1. Euler was also the first to study the three-body problem in a co-rotating, or synodic, reference frame by placing the origin at the barycenter. This was an important step in the eventual development of the circular restricted three-body problem.



[9]**Figure 1.** Special form solution developed by Euler where three bodies are in a line.

Soon after, in 1772, Lagrange discovered another special class of orbits. When the positions of three bodies formed an equilateral triangle with a certain set of specified initial velocities, the equilateral configuration stayed consistent over time. This configuration is shown in9Figure 2. Lagrange also greatly contributed to the CR3BP when he discovered 5 positions in a circular orbit where the gravitational force equaled the centrifugal force for negligible masses. These positions are now called the Lagrange points, where L1, L2, and L3 were determined unstable and L4 and L5 we determined to be stable. Figure 3 shows the positions of the five Lagrange points.



[9]**Figure 2.** Lagrange's special solution with 3 objects in an equilateral triangle.

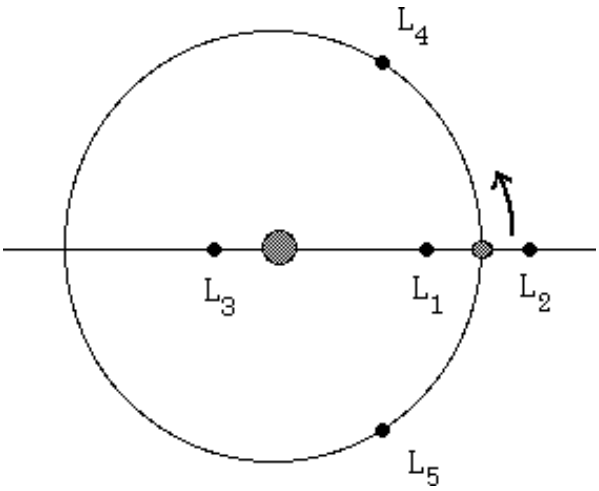


Figure 3. Lagrange Points for a circular orbit

Then in 1836, using the synodic barycenter coordinate system developed by Euler, Carl Gustav Jacob Jacobi was able show an integral of motion exists for a three body system. Now called the Jacobi integral, it is the only known conserved value in the circular restricted three-body problem. The constant of integration discovered by Jacobi is shown below,

$$C_J = n^2(x^2 + y^2) + 2\left(\frac{\mu_1}{r_1} + \frac{\mu_2}{r_2}\right) - (\dot{x}^2 + \dot{y}^2 + \dot{z}^2)R = -\omega_B L \tag{1}$$

where n is the mean motion, μ is the GM or gravitational constant and mass multiple and r_1 and r_2 are distances from the two large masses. The integral equation is,

$$\dot{x}\ddot{x} + \dot{y}\ddot{y} + \dot{z}\ddot{z} = \frac{\delta U}{\delta x}\dot{x} + \frac{\delta U}{\delta y}\dot{y} + \frac{\delta U}{\delta z}\dot{z} = \frac{dU}{dt} \tag{2}$$

After integration of Eq. (2) the formula becomes,

$$\dot{x}^2 + \dot{y}^2 + \dot{z}^2 = 2U - C_J \tag{3}$$

This integral was essential to George William Hill in 1878 when he applied it to the motion of the infinitesimally small masses of asteroids. This led to Hill conceptualizing zero velocity curves as a visualization tool still in use today. An example of zero velocity curves or surfaces is shown in Figure 4. This led Hill to his eventual formulation of a version of Lunar Theory also still in use today. Hill's Lunar Theory approached the circular restricted three-body problem from a new angle by analyzing perturbations on special cases of lunar orbits to find positions to a relative high degree of accuracy. Applying perturbations to special orbits and analyzing the results became an important avenue for later CR2BP contributions.

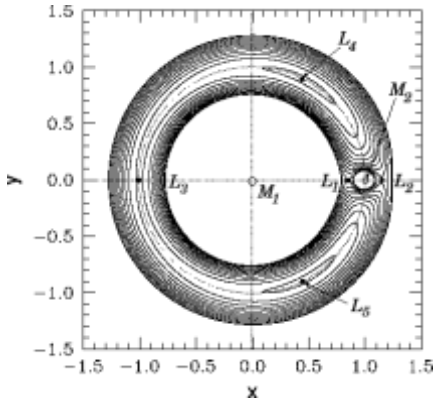


Figure 4. Zero Velocity Curves or Surfaces developed by George William Hill

One of the most important contributions to the three-body problem came from Poincaré between 1892 and 1899. Poincaré published a series of books on methods for solving differential equations. More importantly, he developed methods for identifying systems of equations that were non-integrable. These new methods allowed Poincaré to identify the three-body problem as unpredictable or unsolvable which changed the focus of the problem to techniques used today. Poincaré later developed this farther into a theory of chaos.

With this new knowledge, finding a closed form solution became highly unlikely. Future efforts were therefore focused on solutions using infinite series. Sundman in 1912 was able to find a complete solution to the three-body problem using a power series, however, this solution converged very slowly and restricted its use from any reasonable applications. Despite Sundman's power series implying an overarching solution exists for the three-body problem, his method only gave results indirectly so the three-body problem's unpredictability was preserved. Computational methods with series continued to develop through the first half of the 20th century, and as computers continued to advance into the latter half of the 20th century, numerical solutions of the three-body problem became easier and faster. Solutions to greater degrees of accuracy were calculated at increasing speeds allowing precise and accurate trajectories to be found for trajectories to the moon and beyond.

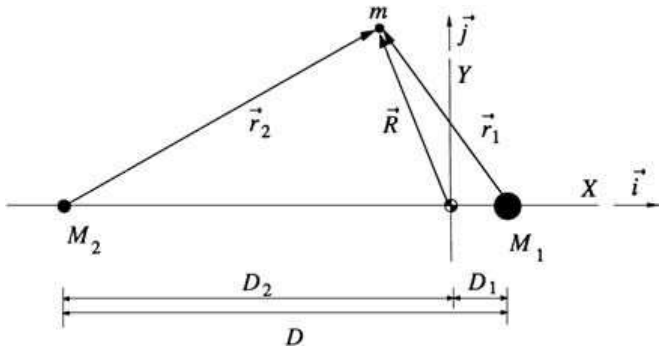
II. The Circular Restricted Three-Body Problem

	Earth		Moon
Mass	$5.9723 \times 10^{24}\, kg$	Mass	$7.346 \times 10^{22}\, kg$
Equatorial Radius	$6378.1\ km$	Equatorial Radius	$1738.1\ km$
Semimajor Axis	$149.60 \times 10^{6}\, km$	Semimajor Axis	$0.3844 \times 10^{6}\, km$
Mean orbital velocity	$29.78\ km/s$	Mean orbital velocity	$1.022\ km/s$
GM	$0.39860 \times 10^{6}\, km^3/s^2$	GM	$0.00490 \times 10^{6}\, km^3/s^2$
Perihelion	$147.09 \times 10^{6}\, km$	Perigee	$0.3633 \times 10^{6}\, km$
Aphelion	$152.10 \times 10^{6}\, km$	Apogee	$0.4055 \times 10^{6}\, km$
Orbit inclination	$0.000\ deg$	Revolution period	$27.3217\ days$
Orbit eccentricity	0.0167	Synodic period	$29.53\ days$
Sidereal rotation period	$23.9345\ hrs$	Inclination to ecliptic	$5.145\ deg$
Inclination of equator	$23.44\ deg$	Inclination to Earth equator	$18.28 - 28.58\ deg$
		Orbit eccentricity	0.0549
		Distance from Earth	$3.78 \times 10^{5}\, km$

Table 1. Earth and Moon Parameters used in the proceeding chapters

The Three-Body Problem can be simplified into the restricted three-body problem if one of the masses is infinitesimally small. This simplification can be readily applied to the motion of satellites in the Earth-Moon system. The problem can be simplified even farther if the orbits of the two massive bodies are nearly circular, that is to say, the eccentricity is nearly 0. This is called the Circular Restricted Three-Body Problem and reasonably accurate results can be obtained for low eccentricity systems. In the Earth-Moon system, the Moon has an eccentricity of 0.0549 and can be considered nearly circular. However, this approximation is not satisfactory for certain orbits due to increasing resonate perturbations, but in many occurrences this simplification suffices.

The circular restricted three body problem in a rotating barycenter frame was first developed and utilized by Euler in 1772. His efforts focused on studying the Moon's motion around the Earth, however, this section will center on satellite motion for which the same methods can be easily applied.



[1]**Figure 5.** Typical Circular Restricted Three-Body Problem Geometry

Since we are restricting the elliptical motion of the Earth and Moon to circular orbits about their barycenter, the angular velocity is simply constant.

$$n = \sqrt{\frac{G(M_1 + M_2)}{D^3}} \qquad (4)^1$$

M_1 and M_2 are the masses of the Earth and the Moon respectively and D is the distance between the two. For the circular restricted three-body problem, D will also remain constant in time. G is the gravitational constant and is taken to be $6.67408 \times 10^{-11}\ m^3 kg^{-1} s^{-2}$.

The position of the spacecraft is expressed in Cartesian coordinates in the barycenter frame of reference. The origin is therefore the combined center of mass of the Moon and the Earth

$$\vec{R} = X\,\vec{\imath} + Y\,\vec{\jmath} + Z\,\vec{k} \tag{5}[1]$$

The spacecraft's inertial acceleration or the 2nd derivative of Eq. (5) is expressed in equation,

$$\ddot{\vec{R}} = (\ddot{X} - 2n\dot{Y} - n^2X)\,\vec{\imath} + (\ddot{Y} + 2n\dot{X} - n^2Y)\,\vec{\jmath} + \ddot{Z}\,\vec{k} \tag{6}[1]$$

$2nY$ and $2nX$ are Coriolis accelerations and n^2X and n^2Y are centrifugal accelerations that arise from the rotating non-inertial frame. The equation of motion due to gravitational interactions is given by,

$$m\ddot{R} = -\frac{GM_1 m}{r_1^3}\vec{r}_1 - \frac{GM_2 m}{r_2^3}\vec{r}_2 \tag{7}[1]$$

where m is the mass of the spacecraft and r_1 and r_2 are the radial magnitudes to the spacecraft from the Earth and the Moon respectively. Vectors $\vec{r}_1$ and $\vec{r}_2$ are defined as,

$$\begin{aligned}\vec{r}_1 &= (X - D_1)\,\vec{\imath} + Y\,\vec{\jmath} + Z\,\vec{k}\\ \vec{r}_2 &= (X - D_2)\,\vec{\imath} + Y\,\vec{\jmath} + Z\,\vec{k}\end{aligned} \tag{8}[1]$$

In Eq. (7) the mass m of the spacecraft can be eliminated from the calculation by dividing it from both the left and right sides.

By combining Eqs. (6) and (7), the equations of motion for the circular restricted three-body problem are established. The equations of motion are displayed in the following:

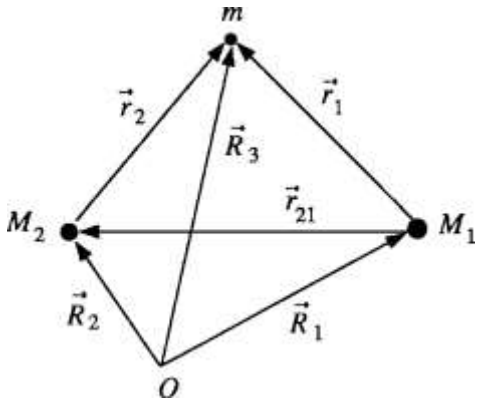
$$\ddot{X} - 2n\dot{Y} - n^2X = -\frac{GM_1(X - D_1)}{r_1^3} - \frac{GM_2(X - D_2)}{r_2^3} \tag{9}[1]$$

$$\ddot{Y} + 2n\dot{X} - n^2Y = -\frac{GM_1Y}{r_1^3} - \frac{GM_2Y}{r_2^3} \tag{10}[1]$$

$$\ddot{Z} = -\frac{GM_1Z}{r_1^3} - \frac{GM_2Z}{r_2^3} \tag{11}[1]$$

III. The Elliptic Restricted Three-Body Problem

The effects of eccentricity are ignored in the circular restricted three-body problem, but perturbations from even small eccentricities can have larger influences than radiation pressure and gravity of the sun. Therefore, disbarring special cases, the circular restricted three-body problem is generally inaccurate. To account for these eccentric perturbations, formerly constant variables of distance and angular velocity are implemented in their dynamic form. The Moon's relatively small eccentricity ($e_{moon} = 0.05490$) can drastically perturb satellites over time and complicates calculations significantly. 1Figure 6 shows the general geometry of a three-body system but with an arbitrary point of origin. We begin calculations from the most general depiction.



[1]**Figure 6.** Three Body System

Adding the gravitational influences of the two massive bodies gives the satellites equation of motion,

$$m\ddot{\vec{R}}_3 = -(GM_1m/r_1^3)\,\vec{r}_1 - (GM_2m/r_2^3)\,\vec{r}_2 \quad (12)^1$$

whereby the satellites mass m can be removed from both sides of the equation. The result is the equation of acceleration,

$$\ddot{\vec{R}}_3 = -(GM_1/r_1^3)\,\vec{r}_1 - (GM_2/r_2^3)\,\vec{r}_2 \quad (13)^1$$

In order to find the equation of acceleration for M_1 we must first define the position vector of mass M_2 with respect to mass M_1,

$$\vec{r}_{21} = \vec{R}_2 - \vec{R}_1 \quad (14)^1$$

Similar to the acceleration vector of the satellite, the acceleration equation of M_1 is,

$$\ddot{\vec{R}}_1 = -(GM_2/r_{21}^3)\,\vec{r}_{21} + (Gm/r_1^3)\,\vec{r}_1 \quad (15)^1$$

We can now combine the acceleration equations of masses m and M_1 into the relative motion of mass m with respect to mass M_1 using $\vec{r}_{21} = \ddot{\vec{R}}_3 - \ddot{\vec{R}}_1$.

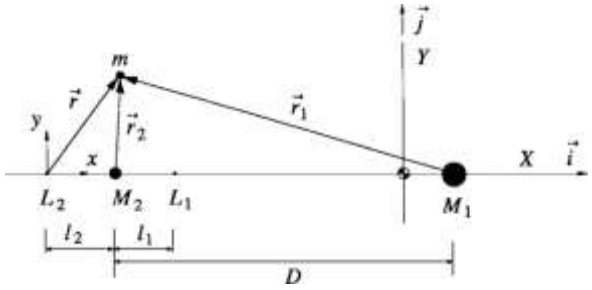
$$\ddot{\vec{r}}_1 = -G(M_1 + m)\frac{r_1}{r_1^3} - GM_2\left\{\frac{r_2}{r_2^3} + \frac{r_{21}}{r_{21}^3}\right\} \quad (16)^1$$

Again using the same procedure outlined above, the relative motion of the infinitesimal mass m with respect to mass M_2 is given by,

$$\ddot{\vec{r}}_2 = -G(M_2 + m)\frac{r_2}{r_2^3} - GM_1\left\{\frac{r_1}{r_1^3} + \frac{r_{21}}{r_{21}^3}\right\} \quad (17)^1$$

If you recall, in the circular restricted three-body problem above, the angular velocity was considered constant because the distances between the large masses M_1 and M_2 were also constant. In the elliptic three-body problem this is not the case, the farther in orbit an object is, the slower its orbital speed and angular velocity. An object at apoapsis moves slower than at its periapsis. Therefore the rotational speed or angular velocity is a dynamic quantity changing

over time. However, a position constant can be found from the ratio of the distance D and the libration points l_1 and l_2. 1Figure 7 shows the libration points and the typical elliptic restricted three-body system setup.



[1]**Figure 7.** Elliptic Three-Body Problem

The instantaneous distance D with either of the instantaneous libration points l_1 or l_2 is constant,

$$l_1/D = \gamma = constant \tag{18}^1$$

We can now define the position of the spacecraft with respect to libration point 1 or 2 in order to eventually describe its motion.

$$\vec{r} = x\,\vec{\imath} + y\,\vec{\jmath} + z\,\vec{k} \tag{19}^1$$

Relating 1Figure 6 and 1Figure 7, and using Eqs. (18) and (19), we can describe the position vectors in the following new formulas:

$$\vec{r}_{21} = -D\,\vec{\imath} \tag{20}^1$$

$$\vec{r}_1 = [-(1+\gamma)D + x]\,\vec{\imath} + y\,\vec{\jmath} + z\,\vec{k} \tag{21}^1$$

$$\vec{r}_2 = (-\gamma D + x)\,\vec{\imath} + y\,\vec{\jmath} + z\,\vec{k} \tag{22}^1$$

Please note that in the three equations above, the distance D is not constant, changing at every instance during orbit. We then solve for the 2nd derivative, or acceleration of $\vec{r}_1$ shown below,

$$\begin{aligned}\ddot{\vec{r}}_1 = {} & \{-(1+\gamma)\ddot{D} + \ddot{x} - \ddot{\theta}y - 2\theta\dot{y} - \theta^2[-(1+\gamma)D + x]\}\,\vec{\imath} \\ & + \{\ddot{y} - \ddot{\theta}(1+\gamma)D - 2\dot{\theta}(1+\gamma)\dot{D} + \ddot{\theta}x + 2\dot{\theta}\dot{x} - \dot{\theta}^2 y\}\,\vec{\jmath} + \ddot{z}\,\vec{k}\end{aligned} \tag{23}^1$$

Please also note that the angular velocity θ is not constant during orbit for the elliptic restricted three-body problem. Lastly, before writing the equations of motion, ρ is introduced as the mass ratio of the two large bodies.

$$\begin{aligned}\rho &= M_2/(M_1 + M_2) \\ 1 - \rho &= M_1/(M_1 + M_2)\end{aligned} \tag{24}^1$$

Finally the equations of motion for the spacecraft can be obtained in non-dimensional form by removing gravitational constant G and combining Eqs. (16) and (23) in the following:

$$\ddot{x} - \theta y - 2\theta\dot{y} - \theta^2[x - (1+\gamma)D] = (1+\gamma)D - (1-\rho)[x - (1+\gamma)D]/r_1^3 - \rho(x - \gamma D)/r_2^3 + \rho/D^2 \quad (25)^1$$

$$\ddot{y} - \theta(1+\gamma)D - 2\theta(1+\gamma)D + \theta x + 2\theta\dot{x} - \theta^2 y = -2\theta[\dot{x} - (1+\gamma)D] - (1-\rho)y/r_1^3 - \rho y/r_2^3 \quad (26)^1$$

$$\ddot{z} = -(1-\rho)z/r_1^3 - \rho z/r_2^2 \quad (27)^1$$

To reiterate, r_1 and r_2 are the magnitudes of vectors $\vec{r}_1$ and $\vec{r}_2$ respectively and are defined as

$$r_1 = \sqrt{[x - (1+\gamma)D]^2 + y^2 + z^2}$$
$$r_2 = \sqrt{(x - \gamma D)^2 + y^2 + z^2} \quad (28)^1$$

It's important to recognize that Eqs.
(25), (26), and (27) are non-dimensional. The distances x, y, z, r_1, r_2 *and* D are in units of the semimajor axis a, the angular velocity θ is in units of mean angular rate n and time is in units of n^{-1}.

Due to the dynamic nature of values D and θ in the elliptic restricted three-body problem, an equation is necessary to represent their changing values. To simplify these equations and easily solve for their derivatives, a series expansion is used in terms of eccentricity and the mean anomaly $M = t - t_p$. The time of perigee passage is t_p. It is common to assume that $t_p = 0$ for simplification, meaning the start of an orbit is at periapsis. The nondimensional series expansions for distance D between masses M_1 and M_2 and the radial velocity θ are shown below:

$$\begin{aligned} D = 1 &+ \frac{1}{2}e^2 + \left(-e + \frac{3}{8}e^3 - \frac{5}{192}e^5 + \frac{7}{9216}e^7\right)\cos M \\ &+ \left(-\frac{1}{2}e^2 + \frac{1}{3}e^4 - \frac{1}{16}e^6\right)\cos 2M \\ &+ \left(-\frac{3}{8}e^3 + \frac{45}{128}e^5 - \frac{567}{5120}e^7\right)\cos 3M \\ &+ \left(-\frac{1}{3}e^4 + \frac{2}{5}e^6\right)\cos 4M \\ &+ \left(-\frac{125}{384}e^5 + \frac{4375}{9216}e^7\right)\cos 5M \\ &- \frac{27}{80}e^6\cos 6M - \frac{16807}{46080}e^7\cos 7M + \cdots \end{aligned} \quad (29)^1$$

$$\begin{aligned} \theta = 1 &+ \left(2e - \frac{1}{4}e^3 + \frac{5}{96}e^5 + \frac{107}{4608}e^7\right)\cos M \\ &+ 2\left(\frac{5}{4}e^2 - \frac{11}{24}e^4 + \frac{17}{192}e^6\right)\cos 2M \\ &+ 3\left(\frac{13}{12}e^3 - \frac{43}{64}e^5 + \frac{95}{512}e^7\right)\cos 3M \\ &+ 4\left(\frac{103}{96}e^4 - \frac{451}{480}e^6\right)\cos 4M \\ &+ 5\left(\frac{1097}{960}e^5 - \frac{5957}{4608}e^7\right)\cos 5M \\ &+ 6\frac{1223}{960}e^6\cos 6M + 7\frac{47273}{32256}e^7\cos 7M + \cdots \end{aligned} \quad (30)^1$$

IV. The N-Body Problem

The n-body problem is the historically difficult problem of mathematically modeling the motion of 3 or more bodies in mutual interaction. This type of interaction, which is typically gravitational but can be electrical or other forms, is generally considered chaotic and unpredictable. However with the help of computers and increasingly fast computational speed, a numerical estimation is possible to a desired degree of accuracy. The problem is formulated by creating N functions with $N - 1$ terms each, where N is the number of bodies in the system. This can be shown in simplistic form by the following equation,

$$F_i = -\sum_{\substack{j=1 \\ i \neq j}}^{N} G \frac{m_i m_j}{|\vec{r}_i - \vec{r}_j|^2} \hat{r}_i - \hat{r}_j \qquad i = 1,2, \dots, N \tag{31}$$

where, r_i and r_j are the position vectors of objects i and j. G is the gravitational constant and $m_{i,j}$ are the masses of objects i and j. F_i is the force on object i which is equal to $F_i = a_i m_i$ or $F_i = m_i (d^2\vec{r}_i/dt^2)$. This expression allows the mass m_i to be divided from both sides of Eq. (31) to obtain,

$$\frac{d^2\vec{r}_i}{dt^2} = -\sum_{\substack{j=1 \\ i \neq j}}^{N} G \frac{m_j \vec{r}_{ij}}{|\vec{r}_{ij}|^3} \qquad \vec{r_{ij}} = \vec{r_i} - \vec{r_j} \quad \& \quad i = 1,2, \dots, N \tag{32}$$

This equation is used to develop N second order differential equations. This system is nonlinear and highly coupled causing three main difficulties for integration. First, this system is highly chaotic and no analytical solution exists, only numerical methods are capable of producing solutions. Second, due to numerical integration being necessary, the system grows in computational time by N^2 with an increase in bodies. This makes the study of large systems, such as galaxies, difficult and presents a limitation on its usefulness depending on the computational speeds available and the size of the system being modeled. Lastly, singularities and instabilities are encountered when two objects come into very close proximity, usually only encountered when an object moves inside a planets radius, except in the case of super dense objects such as neutron stars or black holes.

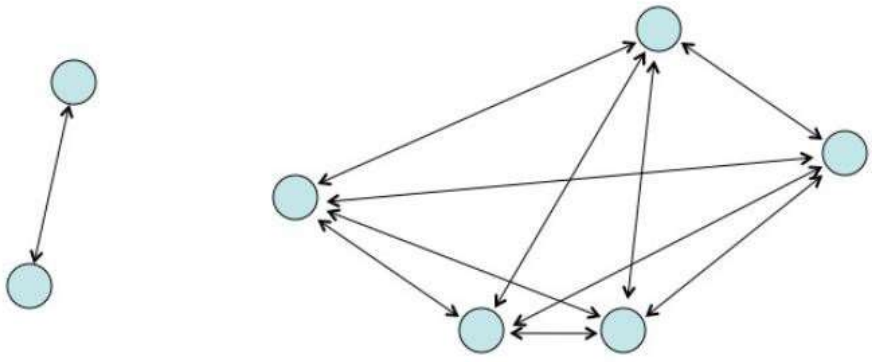


Figure 8. As the number of objects increase, the number of calculations increases to N^2, greatly increasing the computational time to reach a solution.

As an example, for 3-bodies you would obtain 3 equations of motion with 6 terms or interactions. You have 3 total terms on the left and 6 total terms on the right so you have 9 terms in total to numerically solve for. The equations of motions for 3-bodies is shown here.

$$\ddot{\vec{r}}_1 = -G\left(\frac{m_2\vec{r}_{12}}{|\vec{r}_{12}|^3} + \frac{m_3\vec{r}_{13}}{|\vec{r}_{13}|^3}\right)$$
$$\ddot{\vec{r}}_2 = -G\left(\frac{m_1\vec{r}_{12}}{|\vec{r}_{12}|^3} + \frac{m_3\vec{r}_{23}}{|\vec{r}_{23}|^3}\right) \quad (33)$$
$$\ddot{\vec{r}}_3 = -G\left(\frac{m_1\vec{r}_{13}}{|\vec{r}_{13}|^3} + \frac{m_2\vec{r}_{23}}{|\vec{r}_{23}|^3}\right)$$

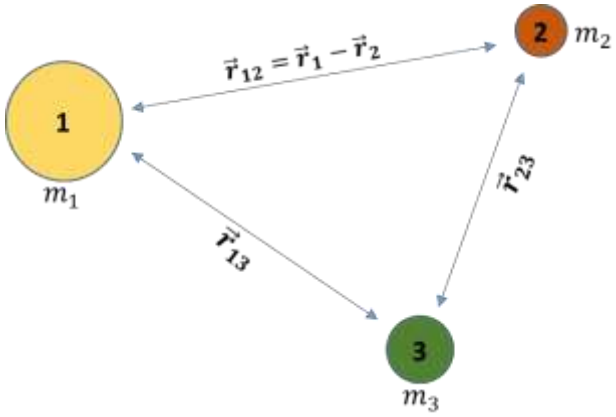


Figure 9. 3-body system where each line represents 2 interactions for a total of 6 interactions, or 2 for each of the 3 bodies.

You can then infer that 4 bodies will require 4 equations of motions with 3 interactions for each individual body. Therefore the number of terms that must be solved for are 4 on the left and 12 on the right of Eq. (34) so 16 in total or N^2 for $N = 4$. As one can see, adding additional bodies demands exponentially more computational time.

$$\ddot{\vec{r}}_1 = -G\left(\frac{m_2\vec{r}_{12}}{|\vec{r}_{12}|^3} + \frac{m_3\vec{r}_{13}}{|\vec{r}_{13}|^3} + \frac{m_4\vec{r}_{14}}{|\vec{r}_{14}|^3}\right)$$
$$\ddot{\vec{r}}_2 = -G\left(\frac{m_1\vec{r}_{12}}{|\vec{r}_{12}|^3} + \frac{m_3\vec{r}_{23}}{|\vec{r}_{23}|^3} + \frac{m_4\vec{r}_{24}}{|\vec{r}_{24}|^3}\right) \quad (34)$$
$$\ddot{\vec{r}}_3 = -G\left(\frac{m_1\vec{r}_{13}}{|\vec{r}_{13}|^3} + \frac{m_2\vec{r}_{23}}{|\vec{r}_{23}|^3} + \frac{m_4\vec{r}_{34}}{|\vec{r}_{34}|^3}\right)$$
$$\ddot{\vec{r}}_4 = -G\left(\frac{m_1\vec{r}_{14}}{|\vec{r}_{14}|^3} + \frac{m_2\vec{r}_{24}}{|\vec{r}_{24}|^3} + \frac{m_3\vec{r}_{34}}{|\vec{r}_{34}|^3}\right)$$

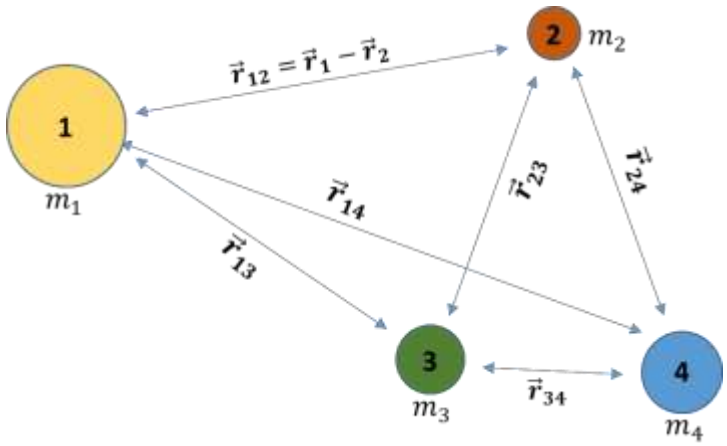


Figure 10. 4-body system with 12 interactions, or 3 for each of the 4 bodies.

V. Translunar Injection Orbit Parameters and Initial Conditions

Translunar Injection[1]

	Apollo 8	Apollo 10	Apollo 11	Apollo 12	Apollo 13	Apollo 14	Apollo 15	Apollo 16	Apollo 17
GET	002:56:05.51	002:39:20.58	002:50:13.03	002:53:13.94	002:41:47.15	002:34:33.24	002:56:03.61	002:39:28.42	003:18:37.64
KSC Date	21 Dec 1968	18 May 1969	16 Jul 1969	14 Nov 1969	11 Apr 1970	31 Jan 1971	26 Jul 1971	16 Apr 1972	07 Dec 1972
GMT Date	21 Dec 1968	18 May 1969	16 Jul 1969	14 Nov 1969	11 Apr 1970	31 Jan 1971	26 Jul 1971	16 Apr 1972	07 Dec 1972
KSC Time	10:47:05 a.m.	03:28:20 p.m.	12:22:13 p.m.	02:15:13 p.m.	04:54:47 p.m.	06:37:35 p.m.	12:30:03 p.m.	03:33:28 p.m.	03:51:37 a.m.
Time Zone	EST	EDT	EDT	EST	EST	EST	EDT	EST	EST
GMT Time	15:47:05	19:28:20	16:22:13	19:15:13	21:54:47	23:37:35	16:30:03	20:33:28	08:51:37
Altitude (ft)	1,137,577	1,093,217	1,097,229	1,209,284	1,108,555	1,090,930	1,055,296	1,040,493	1,029,299
Altitude (n mi)	187.221	179.920	180.581	199.023	182.445	179.544	173.679	171.243	169.401
Earth Fixed Velocity (ft/sec)	34,140.1	34,217.2	34,195.6	34,020.5	34,195.3	34,151.5	34,202.2	34,236.6	34,168.3
Space-Fixed Velocity (ft/sec)	35,505.41	35,562.96	35,545.6	35,389.8	35,538.4	35,511.6	35,579.1	35,566.1	35,555.3
Geocentric Latitude (deg N)	21.3460	-13.5435	9.9204	16.0791	-3.8635	-19.4388	24.8341	-11.9117	4.6824
Geodetic Latitude (deg N)	21.477	-13.627	9.983	16.176	-3.8602	-19.554	24.9700	-11.9881	4.7100
Longitude (deg E)	-143.9242	159.9201	-164.8373	-154.2798	167.2074	141.7312	-142.1295	162.4820	-53.1190
Flight Path Angle (deg)[2]	7.897	7.379	7.367	8.584	7.635	7.480	7.430	7.461	7.379
Heading Angle (deg E of N)	67.494	61.065	60.073	63.902	59.318	65.583	73.173	59.524	118.110
Inclination (deg)	30.636	31.698	31.383	30.555	31.817	30.834	29.696	32.511	28.466
Descending Node (deg)	38.983	123.515	121.847	120.388	122.997	117.394	108.439	122.463	86.042
Eccentricity	0.97553	0.97834	0.97696	0.96966	0.9772	0.9722	0.9760	0.9741	0.9722
C3 (ft^2/sec^2)	-15,918,930	-14,084,265	-14,979,133	-19,745,586	-14,814,090	-18,096,135	-15,643,934	-16,881,439	-18,152,226

[1] Compiled from Saturn V launch vehicle flight evaluation reports and mission reports.
[2] Flight path angle and heading angle are 'space-fixed' for these measurements.

[2]**Table 2.** Translunar Injection Initial Conditions from Apollo By The Numbers, Richard Orloff

In order to determine the translunar injection (TLI) orbit parameters and to benchmark the initial flight path before the Moon's influence is significant, some orbit calculations must be made from the above parameters. We first want to retrieve the accurate radius of the TLI from the altitudes listed above. The altitude represents the distance from the spacecraft to the surface of the Earth as a Fischer Ellipsoid. To find the radius of the Earth at the point of TLI, the following equation is used,

$$R_{ellipsoid} = \frac{ab}{\sqrt{(b\cos\psi)^2 + (a\sin\psi)^2}} \quad (35)^2$$

where a and b are the equatorial and polar radii of earth respectively. ψ is the Geocentric Latitude (deg N) of the point on the ellipsoid directly below the spacecraft. ψ can be found in 2Table 2. The radius magnitude of the spacecraft from the center of the Earth is simply,

$$r_{SC} = R_{ellipsoid} + Altitude \quad (36)$$

The perigee and apogee radii is then calculated using,

$$R_{p,a}/r = -C \pm \frac{\sqrt{C^2 - 4(1-C)(-\cos^2\phi)}}{2(1-C)} \quad (37)^2$$

where ϕ is the flight path angle from 2Table 2, $C = 2GM/(rV^2)$ and V is the space-fixed velocity from 2Table 2. The eccentricity is given by,

$$e = \sqrt{\left(\frac{rV^2}{GM} - 1\right)^2 \cos^2\phi \sin^2\phi} \quad (38)$$

or is given by 2Table 2. The true anomaly is calculated using,

$$\nu = \tan^{-1}\frac{\left(\frac{rV^2}{GM}\right)\cos\phi\sin\phi}{\left(\frac{rV^2}{GM}\right)\cos^2\phi - 1} \quad (39)^2$$

The semi-major axis can be calculated from two formulas,

$$a = \frac{1}{\left(\frac{2}{r} - \frac{V^2}{GM}\right)} = \frac{R_p + R_a}{2} \quad (40)$$

Next we need a way to transform coordinates between planes. This can be accomplished with the following equations:

$$\tan\alpha = \frac{\sin\lambda\cos i - \tan\beta\sin i}{\cos\lambda}$$
$$\sin\delta = \sin\beta\cos i + \cos\beta\sin i\sin\lambda \quad (41)^2$$

where α is the equatorial longitude, δ is the equatorial latitude, λ is the orbital longitude, β is the orbital latitude, and i is the orbital plane inclination. Note, in this case we are ignoring the influence of the moon, so β is equal to 0 since the spacecraft will not deviate from its original orbital plane. Therefore the equations can be rewritten as,

$$\tan\alpha = \tan\lambda\cos i$$
$$\sin\delta = \sin i\sin\lambda \quad (42)$$

In Table 1 above, we are given δ and i as the geocentric latitude and inclination respectively. With the above two equations we have two unknowns which can be solved for, α, the equatorial longitude and λ, the orbital longitude. With the orbital longitude in hand, the argument of perigee ω can be found with,

$$\omega = \lambda - \nu \tag{43}$$

Then, the time of perigee passage is calculated. We will need to calculate the time between the spacecraft's true anomaly at translunar injection and the last perigee. To do this, we first need the Eccentric Anomaly, given by,

$$E = \cos^{-1}\left(\frac{e + \cos\nu}{1 + e\cos\nu}\right) \tag{44}$$

Then the Mean Anomaly is found using,

$$M = E - e\sin E \tag{45}$$

Which allows us to calculate the Mean Motion n, given by

$$n = \sqrt{\frac{GM}{a^3}} \tag{46}$$

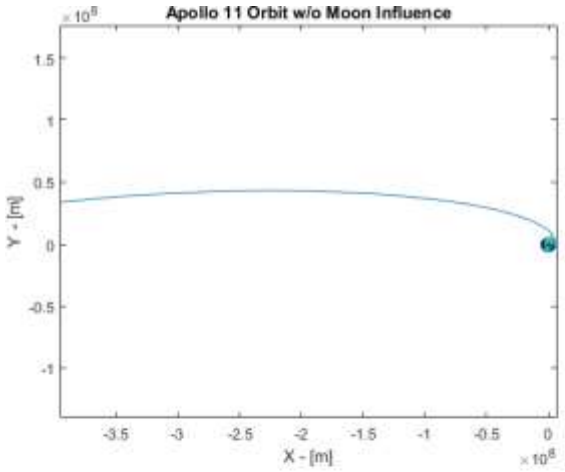
Finally we can solve for the time from perigee. The Mean Anomaly is in units of radians and the mean motion is in units of radians per second. So dividing the Mean Anomaly by the Mean Motion gives the time from perigee,

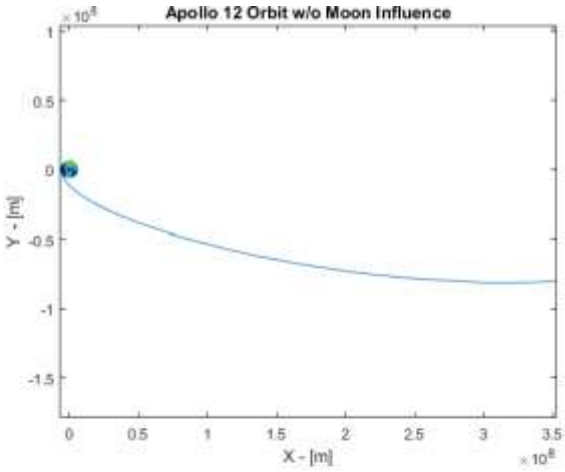
$$\Delta t_{\,parigee} = M/n \tag{47}$$

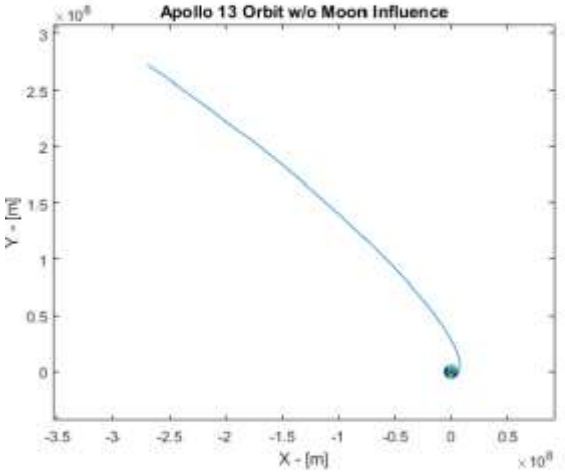
With this, all the orbital parameters of the Apollo missions at translunar injection are known. The MATLAB code used for finding these parameters is in Appendix A. These parameters will be used as the initial conditions for the circular restricted three-body problem and the elliptic restricted three-body problem discussed in the following sections. The main orbital elements for Apollo 11 through 17 are shown in the table below and the top-down view of each Apollo mission's orbit without the influence of the moon is shown.

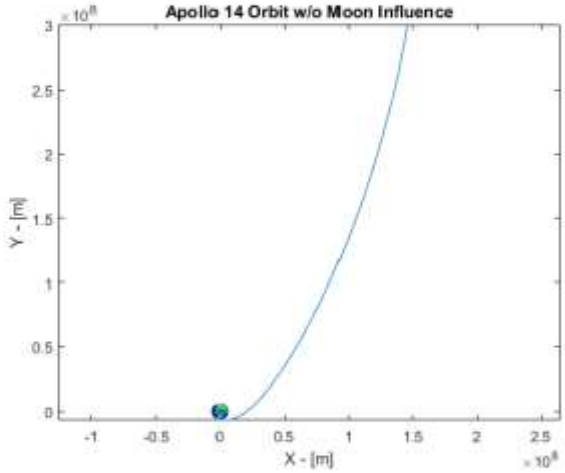
	Apollo 11	**Apollo 12**	**Apollo 13**	**Apollo 14**	**Apollo 15**	**Apollo 16**	**Apollo 17**
Semi-Major Axis (m)	286534624	217313692	291363850	237209195	274456600	254303965	236452232
Eccentricity	0.976965	0.969664	0.977362	0.972206	0.976016	0.974125	0.972173
Inclination (deg)	31.383	30.555	31.817	30.834	29.696	32.511	28.466
Argument of Perigee (deg)	4.4102	15.573	-22.791	-55.664	42.928	-37.705	-5.1089
Longitude of Ascen. Node (deg)	358.380	159.004	341.843	302.899	354.851	335.249	147.315
Time from perigee (sec)	158.95	186.71	164.66	161.78	159.82	160.64	159.07

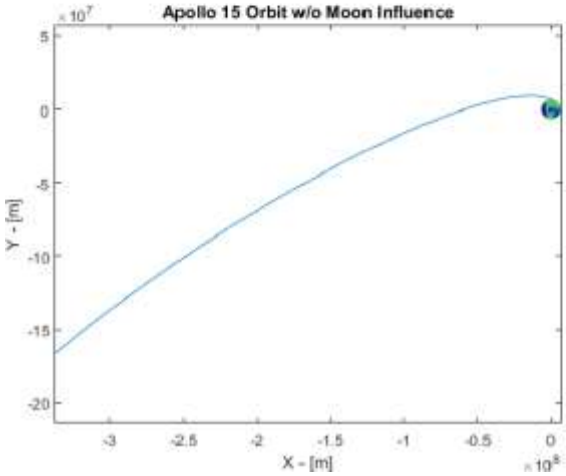
Table 3: Calculated Orbit Elements of Apollo 11 through 17

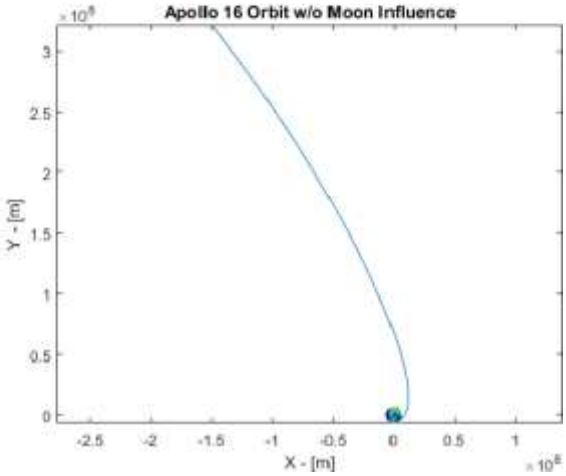

Apollo 11 Orbit w/o Moon Influence
Y - [m]
X - [m]


Apollo 12 Orbit w/o Moon Influence
Y - [m]
X - [m]


Apollo 13 Orbit w/o Moon Influence
Y - [m]
X - [m]


Apollo 14 Orbit w/o Moon Influence
Y - [m]
X - [m]


Apollo 15 Orbit w/o Moon Influence
Y - [m]
X - [m]


Apollo 16 Orbit w/o Moon Influence
Y - [m]
X - [m]

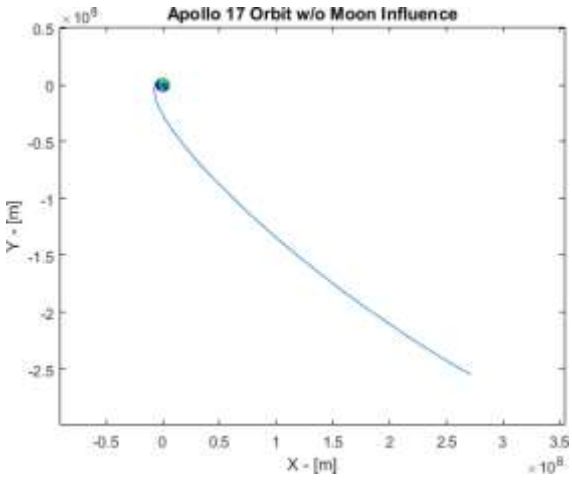


Figure 11. The trajectories of Apollo missions 11-17 without moon influence in cartesian coordinates derived from celestial ccoordinates. These are derived using the common orbital equations in this section so that a comparison to the initial trajectories from the restricted three-body problems can be matached for benchmarking.

VI. CR3BP Simulation

The MATLAB code for the Circular Restricted Three-Body Problem can be found in Appendix B. In this code, the methods found in Section 2 are used to find the Equations of motion. The equations of motion are written into a function, called cr3bp3.m. In this function, variables are separated into new variables in a similar fashion to a Multi-Input Multi-Output (MIMO) for Ordinary Differential Equations (ODE). This format can be used by the ODE45 numerical solver to calculate the position and velocity of the satellite incrementally from initial to final time with a specified step size.

Rotations are implemented to the initial position and velocity vectors found in Section IV to account for the barycenter coordinates. In this instance, the barycenter or synodic coordinate system places the moon on the negative x-axis. However, the position of the translunar injection (TLI) is rotated around the Z-axis so that the spacecraft intercepts or rendezvous with the Moon as it traverses its orbit path in time. The purpose of the rotations on the initial values from Section IV are to match the TLI with the phase difference of the Moon's location.

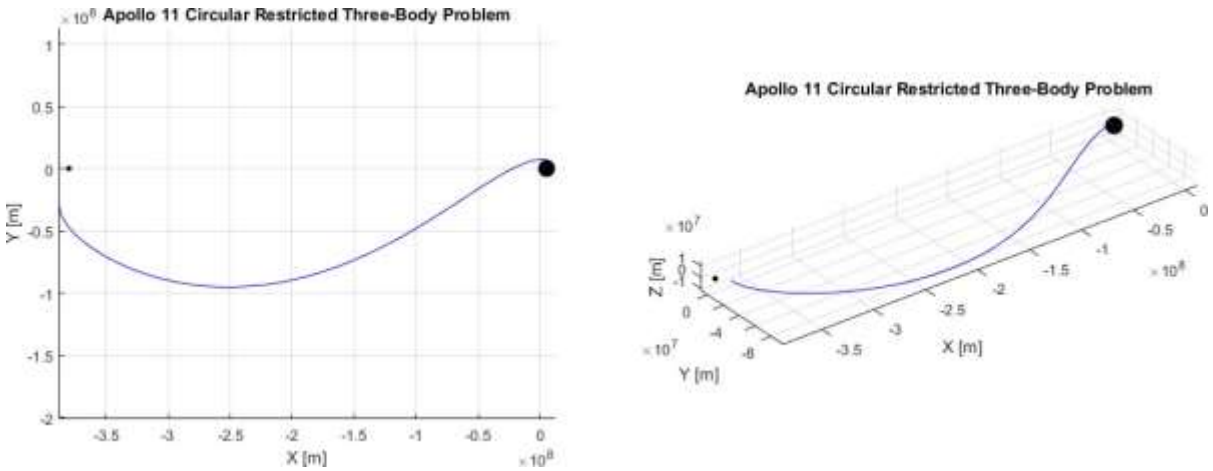


Figure 12. The Apollo 11 translunar injection orbit in the circular restricted three-body problem

VII. ER3BP Simulation

The MATLAB code for the Elliptic Restricted Three-Body Problem is displayed in Appendix C. The primary difference between the CR3BP and the ER3BP is found inside the function used by the ODE45 solver. The initial ODE45 parameters in both circular and elliptic restricted three-body problems is formatted identically as x_0.

$$x_0 = [X, Y, Z, V_x, V_y, V_z] \tag{48}$$

The first three elements of the array represent the Cartesian coordinates of the spacecraft in the non-inertial, Earth-Moon barycenter frame. The last three elements represent the component velocities of the spacecraft in the same rotating frame.

Reviewing the methods and steps outlined in Section III for the elliptic restricted three-body problem, decidedly more complicated, dynamic equations of motion are presented. No longer are the mass distances and radial velocities constant. Closer inspection of the ER3BP equations of motion reveal single and double derivatives of the dynamic values of D and θ. Fortunately, D and θ can be approximated by a series in eccentricity, making differentiation simpler. However, symbolic solvers are computationally taxing in MATLAB, so the pre-derived equations of Eqs. (29)[1] and (30)[1] are included in the ODE45 function at the bottom of Appendix C for faster results. After implementing the derivatives into the equations of motion, solving the ODEs is very similar to solving the CR3BP.

However, certain graphical challenges are presented when displaying a dynamic system and the code displays changing graphical plots over time. Obviously, displaying dynamic plots is not possible in a paper, so some graphs display snapshots of the Moon's position in multiple locations to hopefully give the impression of a dynamic and moving plot.

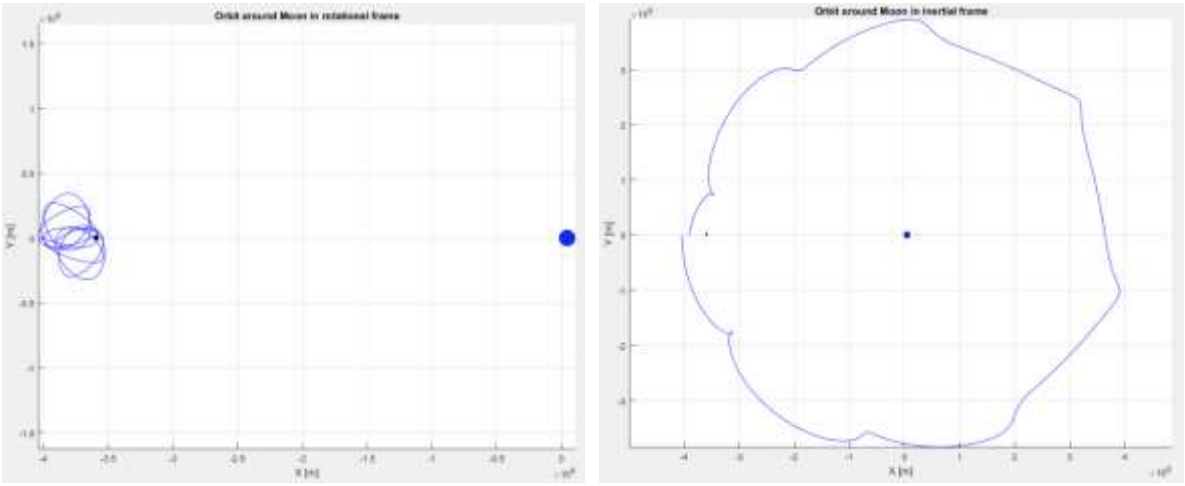


Figure 13. This is a random satellite orbit of the Moon. The rotating barycenter frame is on the left, and the inertial barycenter frame is on the right. This shows the stark difference in visualization a frame of reference can make.

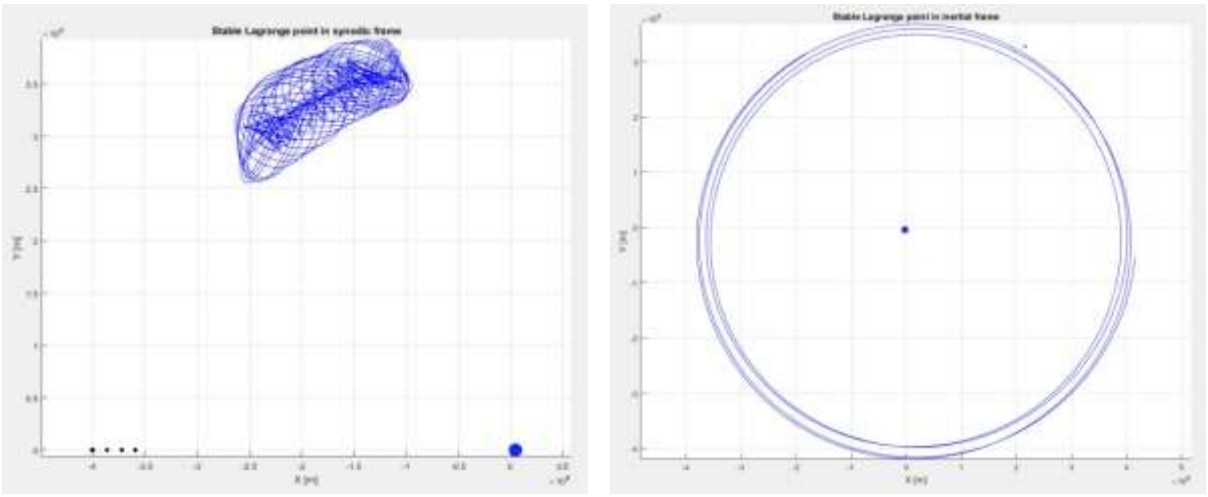


Figure 14. Here the satellite is positioned near one of the stable Lagrangian points. As can be seen in the left graph, the satellite never strays far from the location over many orbits. The Moon's eccentric orbit can also be seen in the lower left corner at positions between apogee and perigee. The graph on the right shows the same satellite with less orbits but in an inertial frame. The end of the blue line is the satellite position and the Moon is in the top right. In this particular position, the satellite stays at a similar position from the moon during the entirety of orbit motion. You can also see the eccentric orbit of the Moon from the carrying distances of the satellites path.

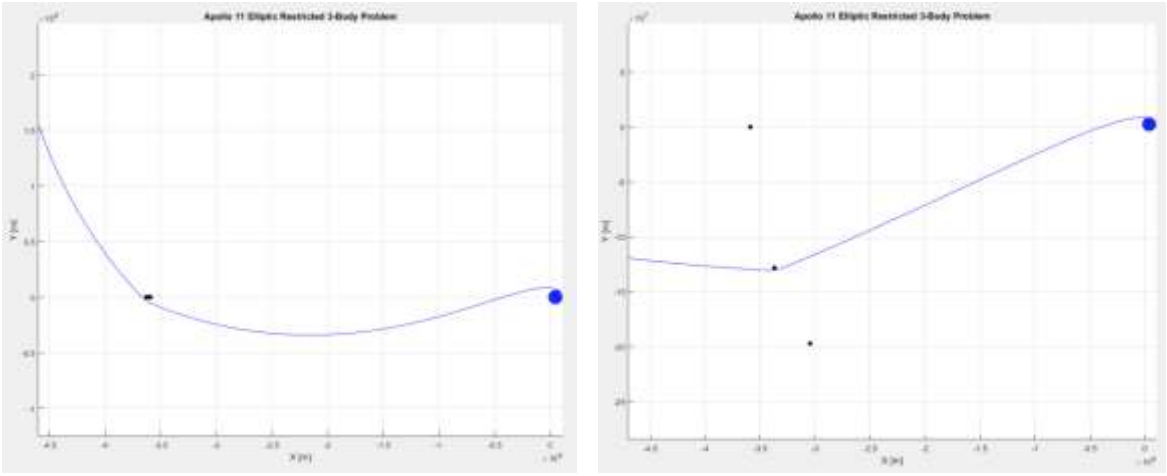


Figure 15. Here we see the rotating frame of Apollo 11's translunar orbit on the left and the inertial frame of the same orbit on the right. Using the initial conditions from Section IV and rotational transformations for proper orientation with the Moon, we can see that the initial conditions in a ER3BP simulation get the spacecraft close enough to the Moon to have a major trajectory change. In the actual Apollo 11 mission, a retrograde burn would slow the S/C down to orbit the Moon once it's in close proximity. On the Right, The Moon moves in a counterclockwise direction. There initial position, rendezvous position and the final position are shown so the trajectory of Apollo 11 is clear.

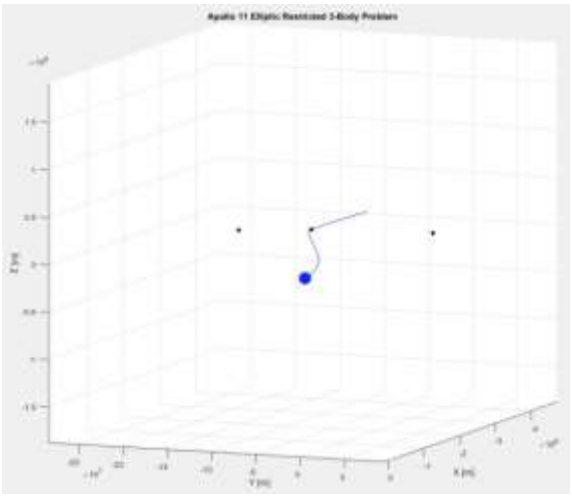


Figure 16. Here the same Apollo 11 orbit trajectory from the right plot in Figure 15 is shown in 3 dimensions. The path of the vehicle as it leaves the earth is much clearer and its rendezvous with the Moon more dramatic.

VIII. N-Body Simulation

This section will introduce a method for modeling the n-body problem in MATLAB, however the same techniques could apply to other coding languages as well. As stated in previous sections, the n-body problem, or any system of 3 or more bodies, has no analytical solution and must be solved numerically. Therefore a numerical solver, sometimes called a numerical integrator, is required. There are many different types of solvers available but it is not difficult to write one's own solver once the concept is understood. Each solver has advantages and disadvantages that can greatly influence the speed and accuracy of a solution's results. For this paper's code, the family of Runge-Kutta solvers are primarily used, particularly the RK89. However, many types of solvers exist that are optimized for specific types of problems. One such example is presented in the paper by Ahmad[11] which describes a more efficient method of numerical integration for the n-body problem. A more in-depth overview of solvers will be discussed in Section C. Methods for coding the equations of motion (EOM) for any chosen number of bodies is discussed next.

A. Creating Dynamic Equations of Motion (EOM) Using For-Loops

For a relatively small number of bodies, one could simply hard-code the equations of motion, such Eqs. (33) and (34), into MATLAB and be able to solve any orbital problems with the same number of bodies. This would very quickly become a tedious method as the number of bodies, N increase. Every unique value of N-bodies would require a unique set of equations. If you never required a solution to anything other than 5-body systems, then it may be more efficient to just use the EOM for 5-bodies. However, the spirit of the n-body problem is to have a solution that works for any numbers of objects. To have a robust orbit simulator, it's required to build the equations of motion automatically. Probably the most obvious method would be to use for-loops. This method is shown in the following MATLAB function.

```
%%%%% N-body Eq of Motion (EOM) Construction Function (For-Loop Method)%%%%%
N = length(mu); % number of bodies

    function ss_vec = nBodyFunc(t,xv)
        pos = reshape(xv(1:3*N),3,N); % Separate/reshape position vectors
        acc_RHS = zeros(3,size(pos,2)^2); % Allocate space to RHS terms
        for i = 1:size(pos,2) % Reference body
            for j = 1:size(pos,2) % Interacting bodies
                if i ~= j
                    r_ij = pos(:,i) - pos(:,j); % Distance vector
                    acc_term = -mu(j)*r_ij/norm(r_ij)^3; % Acceleration term
                else
                    acc_term = [0;0;0]; % Zero acceleration for i=j
                end
                 acc_RHS(:,i*j) = acc_term; % Acceleration terms on RHS
            end
        end
        acc_Sum = sum(reshape(acc_RHS,3*N,N),2); % Sum of acceleration on RHS
        ss_vec = [xv(3*N+1:end);acc_Sum]; % state-space vector
    end
```

Here we have a function with inputs of time t and combined position velocity vector xv. The variable t is not used in the function but is required for the integrator to incrementally solve the system over time. The vector xv is simply an array of the initial position and velocity coordinates in the format of Equation (49).

$$xv = [x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_N, y_n, z_N, v_{x_1}, v_{y_1}, v_{z_1}, v_{x_2}, v_{y_2}, v_{z_2}, \dots, v_{x_N}, v_{y_N}, v_{z_N}] \tag{49}$$

Combining the state vectors of position and velocity is not required, it simply reduced 2 arrays into one in a previous section of code. The first line of the function reshapes the array into 3 rows of $[x_i;\ y_i;\ z_i]$ and N columns. Space is then allocated for variable acc_RHS (accelerations on Right Hand Side). A nested for-loop is created for i equations with j-1 terms. The situation of i = j computationally the acceleration of body i acting on itself, which is 0. The distance vector is calculated and input into the acceleration equation for variable acc_term. Here you can make out components of Newton's equation for gravitational acceleration with $mu(j) = G\,m_j$, $r\_ij = \vec{r}_{ij}$ and $norm(r\_ij)^3 = |\vec{r}|^3$. This is one term from the sum in Eq. (32). This is saved in acc_RHS before repeating the inner loop to find the next acceleration term using object j+1. After finishing the inner loop, the process starts again for body i+1, finding the individual accelerations from every other body. This process repeats until N equations with $N - 1$ acceleration terms (recall that a_{xyz} is set to 0 when i = j) in x y z components, are saved in array acc_RHS. acc_RHS is reshaped into a 3N x N matrix, with xyz components lined vertically and acceleration terms summed horizontally. The result is a column vector of the total acceleration of each xyz component for every system object.

$$acc\_Sum = [a_{x_1}, a_{y_1}, a_{z_1}, a_{x_2}, a_{y_2}, a_{z_2}, \dots, a_{x_N}, a_{y_n}, a_{z_N}] \tag{50}$$

The differential solvers are designed to expect a function output in $[r'; r'']$ format or in State-Space format,

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad where, \quad \begin{matrix} x_1 = x \\ x_2 = x' \end{matrix} \tag{51}$$

In short x'_1 *and* x'_2 are equal to velocity and acceleration respectively. Thus the initial state vector velocities from function input xv are combined with the sum of acceleration terms to obtain array ss_vec,

$$\text{ss\_vec} = [v_{x_1}, v_{y_1}, v_{z_1}, v_{x_2}, v_{y_2}, v_{z_2}, \ldots, v_{x_N}, v_{y_N}, v_{z_N}, \\ a_{x_1}, a_{y_1}, a_{z_1}, a_{x_2}, a_{y_2}, a_{z_2}, \ldots, a_{x_N}, a_{y_n}, a_{z_N}] \tag{52}$$

Equation (52) is the result equating to the left side of Equation (51) and used in solving for the next iteration's position and velocity $[x_1\ x_2]^T$.

B. Creating Dynamic Equations of Motion (EOM) Using Vectorization

```
%%%%% N-body Eq of Motion (EOM) Construction Function (Vectorization Method)%%%%%
N = length(mu);

x123   = repmat(1:N,1,N);
x23    = x123(logical(ones(N) - eye(N)));
x12_34 = reshape(1:N*(N-1),N-1,N)';
x11    = reshape(repmat(1:N,N-1,1),1,N*(N-1));

    function [ss_vec] = nBodyFunc(t,xv)
        pos     = reshape(xv(1:3*N),3,N);          % Separate/reshape position vectors
        r_ij    = pos(:,x23) - pos(:,x11);         % Distance vector
        r3      = diag(1./sqrt(sum(r ij.^2)).^3); % |r ij|^3
        mu_diag = diag(mu(x23));                   % Diagonal Matrix of Grav Param
        acc_RHS = r_ij*r3*mu_diag;                 % Acceleration terms on RHS
        acc_Sum = sum(reshape(acc_RHS(:,x12_34(:)),3*N,N-1),2); % Acceleration Sum RHS
        ss_vec = [xv(3*N+1:end);acc_Sum];          % state-space vector
    end
```

Another method for building the n-body's EOM is to vectorize the function. Although for-loops are simple to understand and manipulate, they can be computationally slow for a large number of iterations. Vectorization is essentially creating arrays and matrices before the calculation which recreate the looping process so for-loop processes can be avoided. From the MathWorks (MATLAB) webpage on Vectorization,

> MATLAB® is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called vectorization. Vectorizing your code is worthwhile for several reasons:
>
> - Appearance: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.
> - Less Error Prone: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.
> - Performance: Vectorized code often runs much faster than the corresponding code containing loops.

This is done by creating vectors of pointers repeated and arranged in such a way as to recreate the i and j values in the previous section's for-loops. Two useful functions in MATLAB, repmat() and reshape(), are invaluable for vectorization. The function repmat() is short for 'repeat matrix' and simply repeats the desired array or matrix vertically and/or horizontally a specified number of times. The function reshape() creates an equal element sized but differently shaped matrix, i.e. a 3x4 into a 6x2 or 12x1 matrix. If we evaluate 3 bodies, or $N = length(mu) = 3$, then x123, x1214, x23, and x11 from the above code are,

$$\begin{aligned} \text{x123} &= [1\ \ 2\ \ 3\ \ 1\ \ 2\ \ 3\ \ 1\ \ 2\ \ 3] \\ \text{x23} &= [2\ \ 3\ \ 1\ \ 3\ \ 1\ \ 2] \\ \text{x12\_34} &= [1\ \ 2;\ 3\ \ 4;\ 5\ \ 6] \\ \text{x11} &= [1\ \ 1\ \ 2\ \ 2\ \ 3\ \ 3] \end{aligned} \tag{53}$$

Here x11 would represent the value of i and x23 the value of j in every loop of the previous section. So it would be the force on object 1 from object 2, then the force on object 1 from object 3, then the force on object 2 from object 1, then the force on object 2 from object 3 and so on. Vector x123 is just the list of objects (in this case 3) repeated in an array so that the values where i=j, can be removed to create x23. The formation of x23 can be complicated but it is shown here for clarity.

```
(ones(N) - eye(N))
```

$$\begin{matrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{matrix} - \begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix} = \begin{matrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{matrix} \tag{54}$$

```
logical(ones(N) - eye(N))
```

$$\text{logical}\begin{matrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{matrix} = \begin{matrix}\text{False} & \text{True} & \text{True}\\ \text{True} & \text{False} & \text{True}\\ \text{True} & \text{True} & \text{False}\end{matrix} \tag{55}$$

```
x123(logical(ones(N) - eye(N)))
```

$$x123\begin{matrix}\text{False} & \text{True} & \text{True}\\ \text{True} & \text{False} & \text{True}\\ \text{True} & \text{True} & \text{False}\end{matrix} = [2 \quad 3 \quad 1 \quad 3 \quad 1 \; 2] = x23 \tag{56}$$

Therefore x23 is only the instances where i≠j, avoiding a redundant calculation or loop not required when i=j. For 3 objects, saving this step can be trivial, but for a large number of bodies and iterations, this step can save a significant amount of computational time.

Lastly, x12_34 is the vector of forces in acc_Sum reshaped so that all forces acting on a single body are oriented into one row. So all forces on body 1 are collected in row 1, then all the forces on body 2 are collected in row 2 and so on. Ss_vec is then the sum of all the columns of each row to give the total force on each body given in a single column vector in one instance of time or one iteration of the numerical solver.

This entire process of vectorization seems overcomplicated for 3 bodies, but is actually computationally simpler as the number of bodies increase, taking advantage of MATLAB's optimization for vector and matrix calculations. A 70 object simulation would involve 70x70 sized matrices, but matrix calculations are much faster than $70 \times 70 = 4900$ loops of individual calculations per iteration.

C. Numerical Solvers

Once the functions containing the equations of motion are developed from Section A and B, and the initial state vectors of position and velocity are saved in variables p0 and v0, one of many numerical integrators can be used to propagate the orbital system. The ones used here are variations of the Runge-Kutta methods which are a group of implicit and explicit iterative methods for approximating solutions to differential equations. These methods are extremely useful in approximately solving (to the desired accuracy) problems without analytical solutions. However, the cost of higher complexity and accuracy is required computational time. The same or very similar methods are used by NASA for their simulations, but to an extreme degree of accuracy, using super computers.

While the majority of numerical differential solvers use the same fundamental framework, they can have massively different results and run times depending on the type of problems they are applied to. For very computationally heavy problems, solvers are sometimes specifically developed to optimize the specific simulation. The solvers tried during the development of this code are shown in the code below.

```
%%%%% Choice of Numerical Solvers for Use %%%%%
switch solv
    case 1; [t,dz]    = ode113(@nBodyFunc,tt,[p0;v0]);
    case 2; [t,x,dx] = rkn86(@nBodyFunc,tt(1),tt(end),p0,v0);
    case 3; [t,dz]    = ode23(@nBodyFunc,tt,[p0;v0]);
    case 4; [t,x,dx] = rkn1210(@nBodyFunc2, tt, p0, v0);
    case 5; [t,dz]    = rk89(@nBodyFunc,[tt(1),tt(end)],[p0;v0], tol);
end
```

Solvers included in the MATLAB library are ode23 and ode113. These are considered nonstiff solvers, which are better suited for orbital mechanics equations. It is difficult to describe the difference between stiff and nonstiff problems, but one analogy is to imagine a ball rolling down a winding U-shaped slide versus a straight V-shaped slide. The ball will tend towards the center (the solution) of both slides, but disturbances in the U-slide will correct slowly and smoothly. In the V-shaped slide, a disturbance may correct quickly enough to bounce the ball back-and-forth between the sides (stable but erratic). A deflated ball (stiff solver) on the straight V-slide would reach the bottom faster than an inflated ball (nonstiff solver) bouncing erratically between the steep sides. An inflated ball would reach the bottom of winding U-shaped slide faster than a deflated ball due to less friction and a single bounce not overshooting the equilibrium point every time. The "stiffness" in this analogy refers to the straightness and steepness of the slide's walls (the equation/problem), whereas the ball is the type of solver

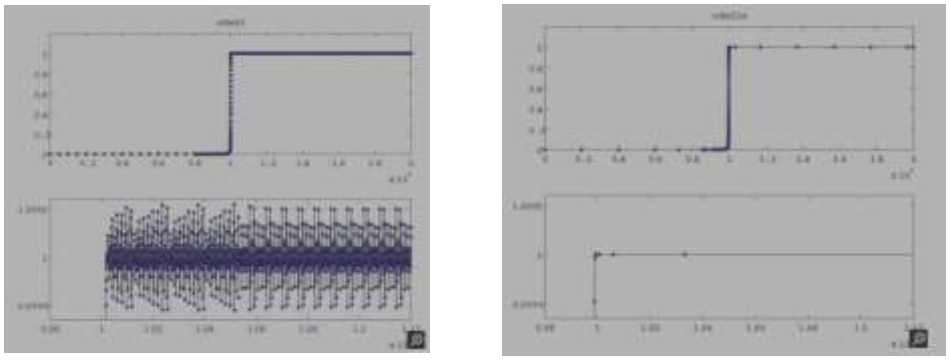


Figure 17. Nonstiff solver, ode23, used on a stiff problem (left) and stiff solver, ode23s, used on the same stiff problem (right). The bottom figures are zoomed in on the corner of vertical to horizontal transition. It shows the nonstiff solver (left) jumping around the solution while the stiff solver immediately finds the solution in much fewer steps.

For a lot of orbital mechanics problems, such as the movement of the planets in the solar system, the gravity gradient is very smooth. So a nonstiff solver such as ode113 or even ode23 is accurate and very fast over long propagations. However, orbits that come into close proximity of a planet where the gravity gradient is much higher slow down considerably or sacrifice accuracy. In these cases, a stiff solver would also be slow as the true solution is also changing rapidly near periapsis. In these cases, the rkn86, rkn1210 and rk89 are more efficient because they decrease step sizes, increasing resolution, in areas of large gradients. For rkn86 and rkn1210, the rkn stands for Runge-Kutta-Nyström, which is a slightly different method than just Runge-Kutta. The rk89 however, has proven to be the best solver for both accuracy and speed with rkn86 coming in 2nd. The rkn1210 is highly accurate but unnecessarily slow for most applications. The GMAT program developed by NASA uses the rk89 solver by default because of its balance of high accuracy and speed for most simulations conducted on personal computers. Although, the nonstiff solvers, like ode113, seem to be more efficient for large propagation times of low eccentricity orbits.

D. Dynamic Code for introducing Satellite Burns

Many orbital simulations do not have a ΔV, but for deep space probes and orbit insertions, it necessary to have a framework to add rocket burns into the equation. This is very necessary in simulating the Apollo missions as the capsule could not enter and leave lunar orbit without multiple burns taking place. For this code, all burns are

considered impulse burns, which is a reasonable approximation for the majority of chemical thrusters, which typically fire for seconds at a time.

One of the problems with simulating a burn is that it interrupts the solver with a sudden change of velocity and subsequent change of the equations of motion. There is no easy way of avoiding this interruption, so the goal is to have the burn interruption, then use the solver for the next section and then stich the section back together. For impulse burns, simulation is sectioned into pieces, where each burn is the start of a new section with a new velocity from a ΔV. Each section has a new velocity vector but uses the final position vector from the previous section. So the initial state vectors are used to propagate to the 1st burn time. Then a new simulation is initialized using the last position and new velocity vectors to propagate to the next burn, repeating until the total simulation time is reached. Summarized, each burn is a separate simulation using values from the previous simulation and then patched at the end into one combined simulation. The code for this is shown below:

```
% Burn Variable Initialization
dt = iter/time;
pos(1,:) = p0;
vel(1,:) = v0;
tob = horzcat(0,tob,time);
bvec = [bvec; 0 0 1e-10];

% Satellite Burn Segments Loop
for i = 1:length(tob)-1

    % Sets section time and iteration for each burn segment
    iter2(i) = round((tob(i+1)-tob(i))*dt);
    tt = linspace(tob(i),tob(i+1),iter2(i));

    % Burn Segment Solver/Propagation
    [t1, x1, dx1, ddx1] = nBodySolver(pos(i,:)', vel(i,:)', mu, tt', sol);

    % Set next burn segment initial values to previous end values
    % with adjusted velocity from Delta V burn
    pos(i+1,:) = x1(end,:);
    vsat = dx1(end,(3*n-2):3*n);
    dx1(end,(3*n-2):3*n) = vsat/norm(vsat)*norm(([norm(vsat) 0 0]+bvec(i,:)));
    vel(i+1,:) = dx1(end,:);

    % Pos Vel Acc Patching of solver segments
    x = [x;x1];
    dx = [dx;dx1];
    ddx = [ddx;dx];
    t = [t;t1];
end

% Removes duplicate values from patching
dup = find(hist(t,unique(t))>1);
t(dup)=[];
x(dup,:)=[];
dx(dup,:)=[];
ddx(dup,:)=[];
```

Here the time of burn, tob, and the burn vector, bvec, are specified in the function used to store the initial data, described in detail in the next section. The time of burn is simply the seconds after the start of the simulation that 1 or more burns occur in an array. The start and ending times of the simulation are tacked on to the ends as seen in the Apollo 11 example shown in Equation (57). The ΔV burn vectors are listed in columns with the coordinates listed by column $[V_x \quad V_y \quad V_z]$. Equation (58) shows the 3x3 matrix for bvec of Apollo 11.

$$\text{tob} = \begin{matrix} \text{Start} & \text{Burn 1} & \text{Burn 2} & \text{End} \\ 0 & 258119.75 & 467432.4 & 680400 \end{matrix} \tag{57}$$

$$\text{bvec} = \begin{matrix} & V_x & V_y & V_z \\ \text{Burn 1} & -0.88142064 & -0.13200888 & 0.00621792 \\ \text{Burn 2} & 0.97941384 & 0.214884 & -0.04230624 \\ \text{End} & 0 & 0 & 0 \end{matrix} \tag{58}$$

The first segment simulation segment starts at time 0 and ends at time 258119.75 sec using the initial state vectors from translunar injection. The 2nd segment starts at time 258119.75 and ends at 467432.4 sec with ΔV Burn 1 applied to the start of segment 2 and with the ending position vector of segment 1 applied to the start of segment 2. This continues until the last segment where the ending time is the end of the whole simulation. 0 ΔV always added to the last step when all the segments are completed. Each segment is combined in x and dx variables for position and velocity respectively. Lastly, because the end of one segment equals the start of the next segment, the duplicate positions and velocities are removed in the final section of the above code. The end result is a matrix of positions and a matrix of velocities that are continuous from the initial to the final simulation time with impulse ΔV burns included.

E. Inputting Initial State Vectors and Orbiting Body Info

This section is dedicated to explaining how data is input into the simulator. Unfortunately, a graphical user interface was not able to be made, so the less user friendly method is explained here. In order to input the desired data for the program required for a simulation, a function is created to store and pass on various initial values to the main code. The 6 required types of data passed are:

1. Mass
2. Object radius
3. Position vector
4. Velocity vector
5. Burn time
6. Burn ΔV vector

which are stored in vector arrays. Other types of data that can be passed on are:

1. Object color
2. Objects to omit

The purpose of storing the initial values in a function not only keeps simulations contained in their own sections, but it allows different sets of data, possibly from different sources, to be adjusted uniquely if needed in preparation for the solver. For example, state vectors from one source may be in a matrix format while another source has it in an array. The function allows different methods of preparation. An example of a function for simulating the solar system is given in the code below.

```
function [p0, v0, mu, scale, tob, bvec, cA] = StateVecInit

M    =   [ 1988500, 0.33011, 4.8675, 5.9723, 0.07346, 0.64171, 1898.19, 568.34, 86.813,
102.413, 0.01303 ]' * 1e24;
scale = [695700, 2439.7, 6051.8, 6371.008, 1737.4, 3389.5, 69911, 58232, 25362, 24622, 1187];
cA = 1:length(M); % Which objects to simulate

% Time: 1945-Jan-1 0:0:0 (Ref Sun)
p0 = [ -6.514853452736166E+04    6.923075832509800E+05     2.969644123669676E+04; % Sun
       -4.090224402811901E+07     3.092971988307618E+07     6.251586293432735E+06; % Mercury
        8.649626519565648E+07     6.558753100815241E+07    -4.093465503005635E+06; % Venus
       -2.812655902373333E+07     1.450840122031818E+08     4.736565603273362E+04; % Earth
       -2.837642934184081E+07     1.453856908325931E+08     6.006681243879348E+04; % Moon
       -4.575332432237922E+07    -2.160750372389238E+08    -3.379480787481025E+06; % Mars
       -7.937071694877126E+08     1.708772517514482E+08     1.711521054427497E+07; % Jupiter
       -1.923795039832259E+08     1.337735118153997E+09    -1.574714310247427E+07; % Saturn
        8.814142859265038E+08     2.748130246512144E+09    -1.196667562167048E+06; % Uranus
       -4.507715302467741E+09    -4.203358655158987E+08     1.125516845885302E+08; % Neptune
       -3.580075486462851E+09    4.304575097306495E+09    5.745302970305955E+08];% Pluto

v0 = [ -1.970463223071625E+00   -1.955899360310175E-01    2.369174320448657E-01; % Sun
       -4.089492060880693E+01   -3.731269883404487E+01    7.856996007483481E-01; % Mercury
       -2.308493963878707E+01    2.767558092658806E+01    1.833208420054525E+00; % Venus
       -3.168839597240482E+01   -5.981403599180973E+00    2.348307120914044E-01; % Earth
       -3.248928443141775E+01   -6.584149383830741E+00    3.229759698734869E-01; % Moon
        2.266216276795534E+01   -3.115518389430809E+00   -4.325189489577113E-01;  % Mars
       -4.873042121510281E+00   -1.236602989967739E+01    3.519065081987538E-01;  % Jupiter
       -1.205496403155248E+01   -1.599677934190488E+00    6.612639993819801E-01; % Saturn
       -8.512076610641467E+00    1.557251793473381E+00    3.284969194625559E-01; % Uranus
       -1.500221047078075E+00   -5.585955645339683E+00    3.363419003462520E-01; % Neptune
       -4.659388787869598E+00   -4.189821186825673E+00    1.441438596050844E+00];% Pluto

tob = []; % Time of Burn
bvec = []; % Burn Vector

% Position of Barycenter in reference frame
pBary = [-1.070183387116344E+06 1.335013302913338E+06 4.721272103390069E+04];
vBary = [-1.976510629978190E+00 -2.077526912802563E-01 2.371613434980l5E-01];

G    =   6.67259e-20; % [km^3/kg-s^2] gravitational constant
mu   =   M * G; % Calculates Grav Parameter
n    =   size( M, 1 ); % number of bodies

% Reshape P0 and V0 for subtraction of Barycenter state vectors
p0 = reshape(p0',[1,3*n]);
v0 = reshape(v0',[1,3*n]);

% reshape vectors into arrays
for i = 1:size(p0,2)/3
    p0(1+(i-1)*3:3+(i-1)*3) = p0(1+(i-1)*3:3+(i-1)*3) - pBary;
    v0(1+(i-1)*3:3+(i-1)*3) = v0(1+(i-1)*3:3+(i-1)*3) - vBary;
end

% Reshape p0 and v0 into vertical arrays.
p0 = reshape(p0,n*3,1);
v0 = reshape(v0,n*3,1);
end
```

All the required information for simulating the solar system is in this function. It has the Sun, all planets, the Moon and Pluto (now considered a dwarf planet). The first line is an array of all the masses in 10^{24} kg values in the order that they will appear in the state vectors. The scale variable is the radius in km for all the objects in the same order as M. The cA variable is simply an array of [1 2 3 . . .] that indicates which object data to use in the simulation. Here it shows that all objects will be simulated. Next the initial position vectors are stored in p0 as a matrix in $[X, Y, Z]$ Cartesian coordinates obtained from the JPL Horizons website. Each row is a body and each column is a coordinate. The same applies for the velocity state vectors in v0. There are no satellites in this simulation so no ΔV burns are taking place, but the variables must still be initialized. The $var = []$ initializes a

variable as NULL in MATLAB. At this point, all the externally sourced data is inputted. The lower half of the function is dedicated to calculations and matrix manipulations preparing it for the main code to read.

Due to the way in which the JPL Horizons website gives positioning information, a reference object must be chosen, in this case the Sun. However, it was desired to have the solar system barycenter as point 0. Therefore the initial state vectors of the solar system barycenter, in reference to the Sun, was found in JPL Horizons to subtract from all the planet state vectors and moving the reference frame to the barycenter. Lastly the gravitational parameters are calculated and the p0 and v0 matrices are reshaped into one vector array for output to the main code.

F. Graphical Output

For a time dependent simulation such as orbital trajectories, it is really necessary to view the movements over time. Just viewing the entire path of a satellite at the end of a simulation does not give the fill picture, as the gravitational interaction with other bodies is dependent on both the time and location at rendezvous. A still image does not provide this information. Therefore the program is designed to play through simulation in an easy to view plot. Obviously this paper is restricted to showing still images, but the true usefulness of this simulator is watching the dynamics of orbits in real time.

IX. Benchmarking and Results

In order to benchmark the code to ensure that the results are accurate, several n-body scenarios were simulated and matched with known outcomes. One way to benchmark the code was to use GMAT, a free mission analysis tool by NASA. Figure 18 shows the results of from GMAT on the left and the MATALB code on the right. Here the solar system is modeled over the same time interval and with the Earth as the reference frame. In the Earth centered frame, the planets move in spirals due to the Earth's motion around the Sun. This view led to many problematic theories before the Sun centered view was adopted, but here we can see that the results of GMAT and the MATLAB code are the same.

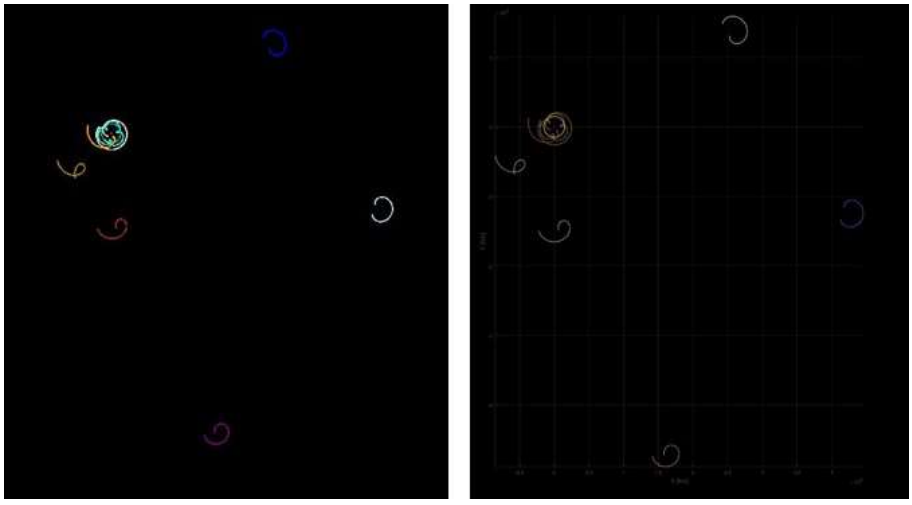


Figure 18. The motion of the planets in the earth centered ecliptic frame over 1 year. Left side shows predictions from GMAT, right side shows predictions of the MATLAB propagator

Using the same MATLAB code from the previous example but from the Sun's reference frame, we can see the movement of the solar system barycenter. Both this, and the previous example, use starting state vectors for the solar system at midnight of January 1st 1945. This date was chosen to match the same starting date of the benchmark shown in Figure 19. The barycenter is calculated using,

$$\vec{r}_{barycenter} = \frac{\sum_{i=1}^{N} \vec{r_i} M_i}{\sum_{i=1}^{N} M_i} \tag{59}$$

where $\vec{r_i}$ is the position vector of each body and M_i is the mass of each body. The barycenter is calculated in the MATLAB code,

```
% Barycenter calculation
bary = zeros(size(x,1),3);
if ApNum == 0
    for i = 1:size(x,1)
        bary(i,:) = ([dot(mu,x(i,1:3:end)), dot(mu,x(i,2:3:end)),...
            dot(mu,x(i,3:3:end))]/sum(mu));
    end
end
```

where mu is the array of gravitational parameters and x is the matrix position for each solar system object at every point in time iterated.

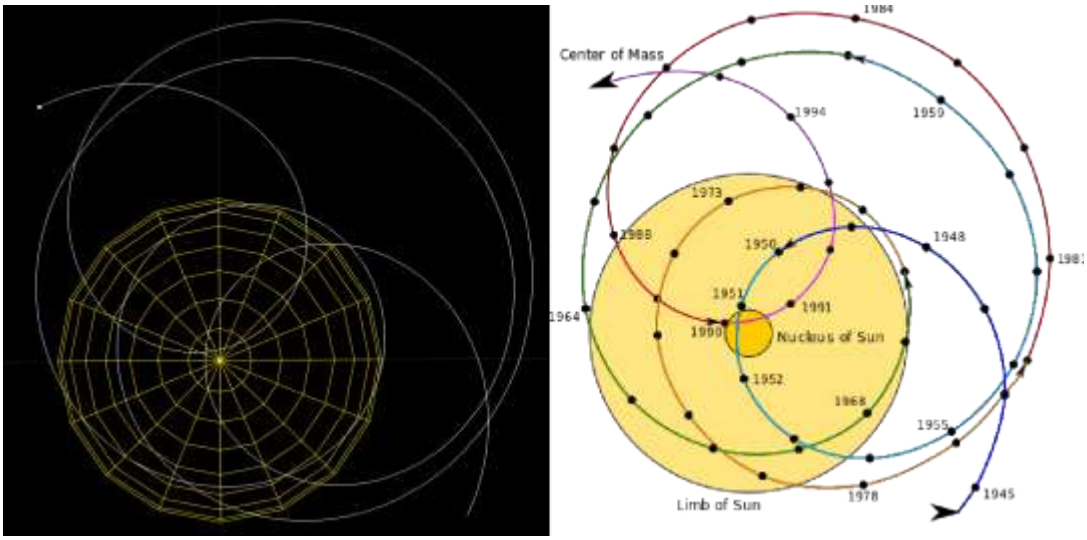


Figure 19. On the left, the barycenter and its path starting in 1945 is shown in white moving in the Sun centered frame from a simulation of the solar system. The image on the right shows the historic path of the solar system barycenter in the same time frame.

Figure 19 shows the MATLAB results for the movement of the solar system's barycenter. It can easily be seen that the results closely match the recorded movement over 50 years. It is interesting to note that the solar system's barycenter is at times inside the envelope of the Sun, even passing through the nucleus, but also at times quite far. To an observer outside of our solar system, the Sun would seem to increase and decrease its "wobble" drastically over a short time. The earliest methods for discovering extra-solar planets used Doppler spectroscopy to measure "wobble" of a star around its barycenter as an indication of a large planetary object in orbit. It is fascinating to think that far observers could be measuring the complicated movement of our Sun now! Figure 20 shows the barycenter movement over 300 years.

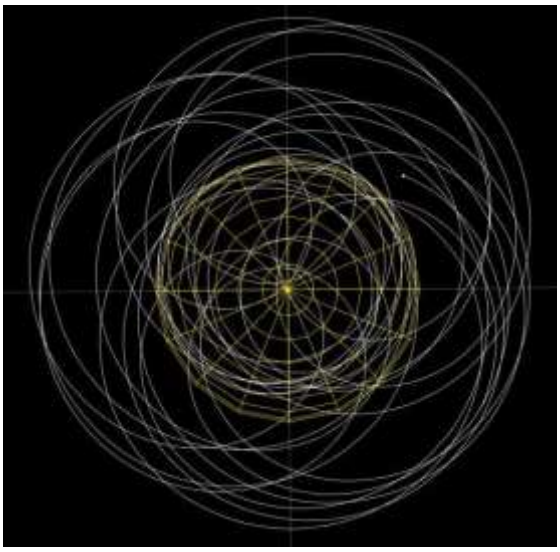


Figure 20. Solar system barycenter movement over 300 years starting in 1945.

To show that the 3-body problem of the Earth-Moon system is working correctly, a satellite was place in the 5th Lagrangian point (L5). L5 is one of the 2 stable Lagrangian points, points of neutral gravitational pull, in the Earth Moon system. A spacecraft placed at this point should stay relatively close to its original location in the Earth-Moon barycenter frame as the complicated interaction of gravitational forces of Earth and Moon create a gravitational potential valley in this region. Looking at Figure 21, we can see that the satellite moves in a complicated path, but always returns to the zone, even after 10 years, as seen in Figure 21.

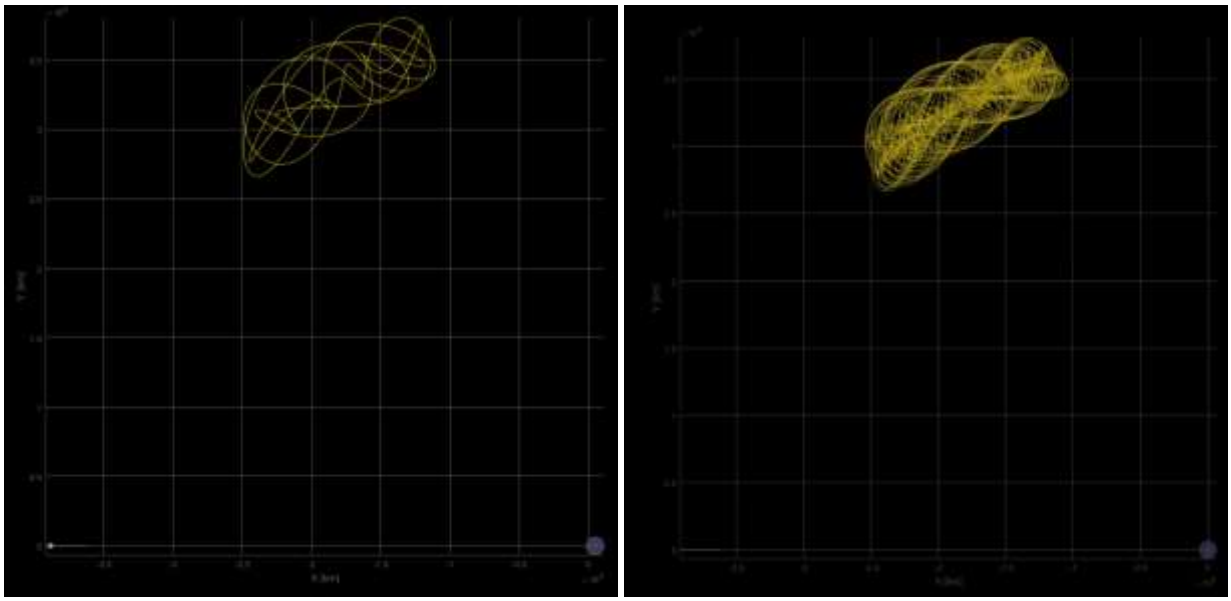


Figure 21. Movement of 1 year (left) and 10 years (right) of a satellite placed at the stable Lagrangian point L5 of the Earth Moon system.

Next, the Apollo 11 Mission starting at translunar injection (TLI), with Lunar orbit insertion, trans-earth injection (TEI) and ending at Earth atmosphere entry is shown. These images show different views centered at the E-M barycenter in the inertial ecliptic frame. Looking closely, you can see the (0, 0) coordinate which is the barycenter of the Earth and Moon is close to the surface but still inside the Earth as expected, but unlike the previous example of the Solar System barycenter moving in and out of the Sun's radius. Another thing to note is the interesting movement of Apollo 11 as it orbits the Moon. This is caused by the Moon's movement during its orbit, but in the non-inertial frame seen in Figure 25, the spacecraft moves in nearly circular orbits around the Moon.

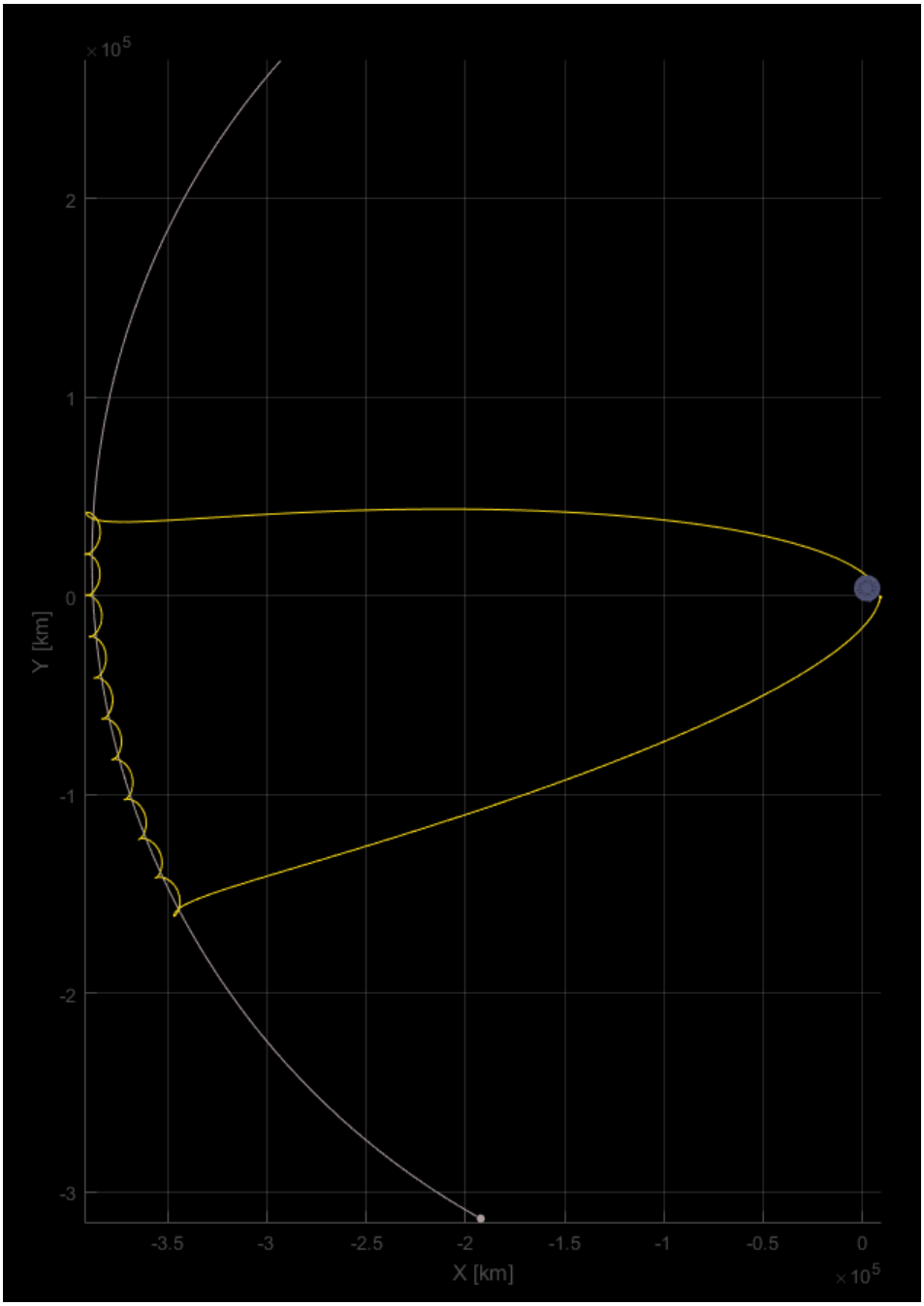


Figure 22. Apollo 11 trajectory from Earth to Lunar orbit and back again. (Top Inertial view)

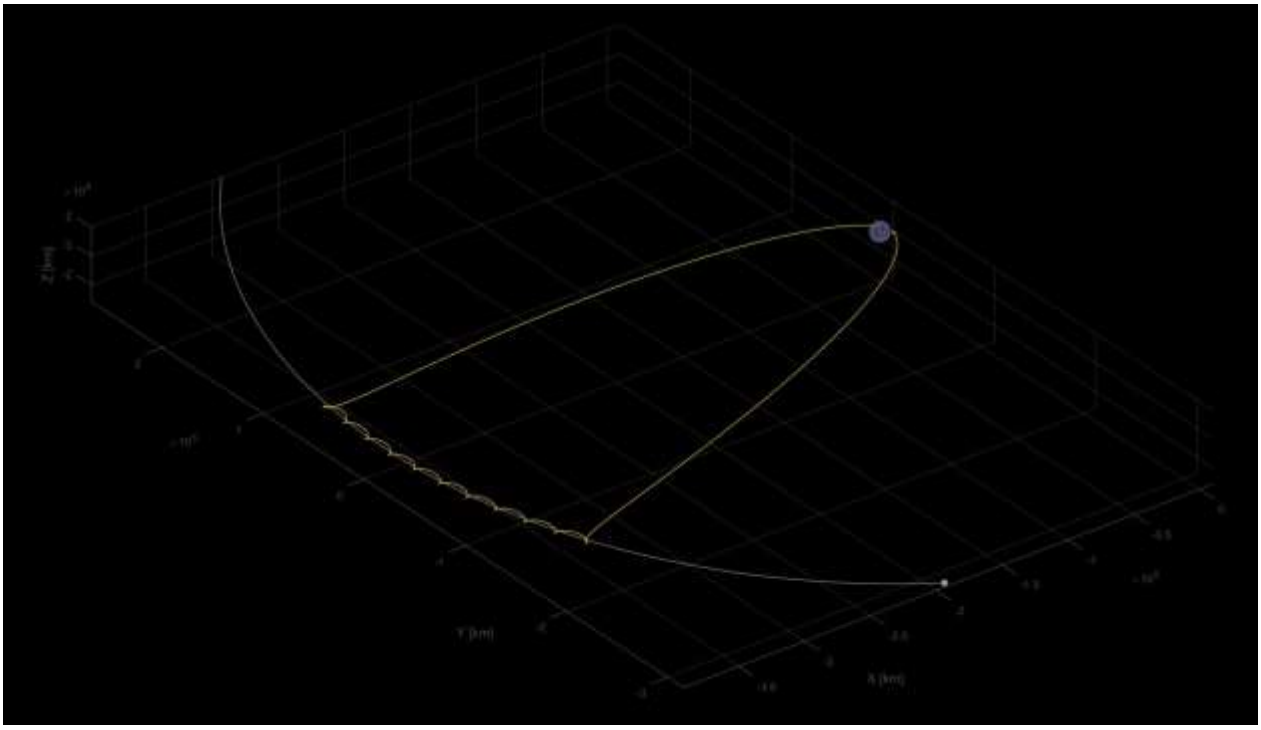


Figure 23. Apollo 11 trajectory from Earth to Lunar orbit and back again. (2nd Inertial view)

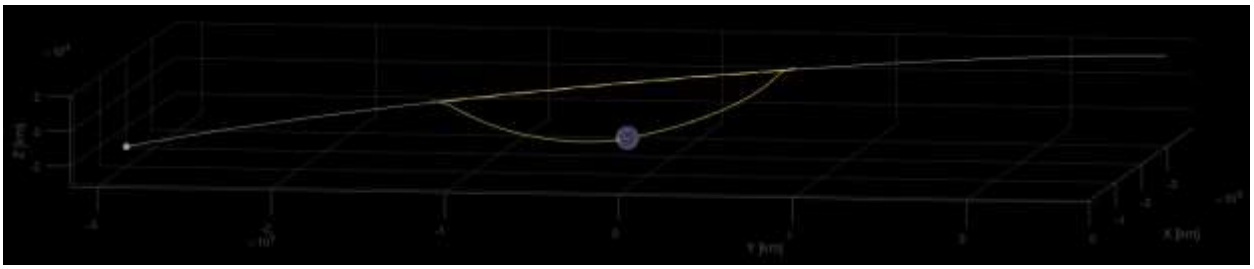


Figure 24. Apollo 11 trajectory from Earth to Lunar orbit and back again. (3rd Inertial view)

Here the Apollo 11 Mission is shown in the rotating E-M barycenter frame.

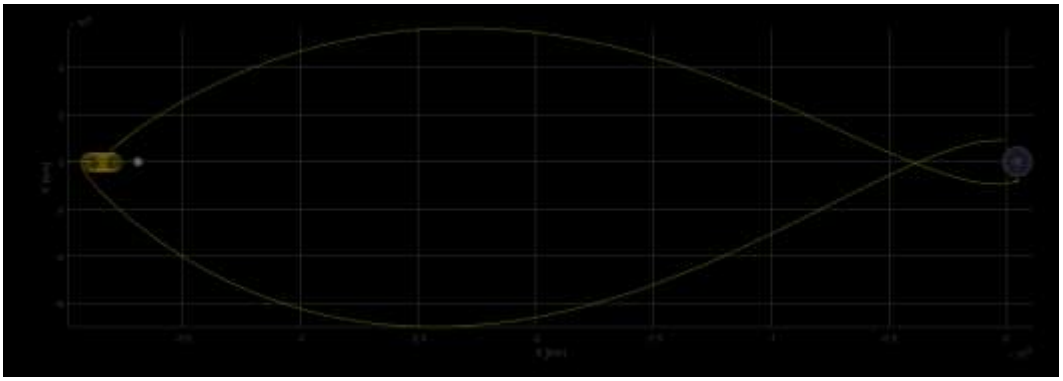


Figure 25. Barycenter Frame of Apollo 11 trajectory from Earth to Lunar orbit and back again.

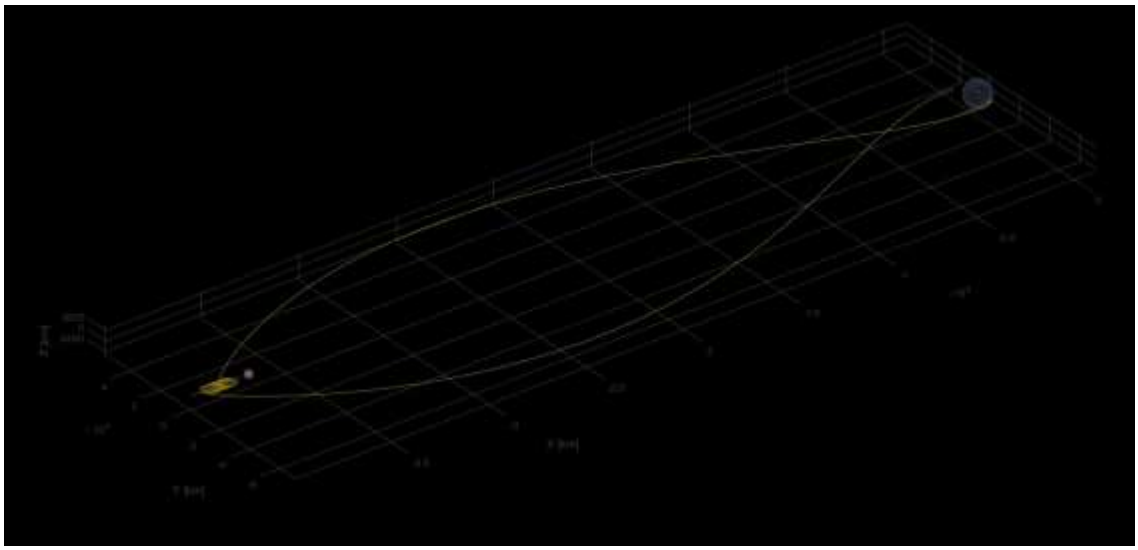


Figure 26. 2nd view of Barycenter Frame of Apollo 11 trajectory from Earth to Lunar orbit and back again.

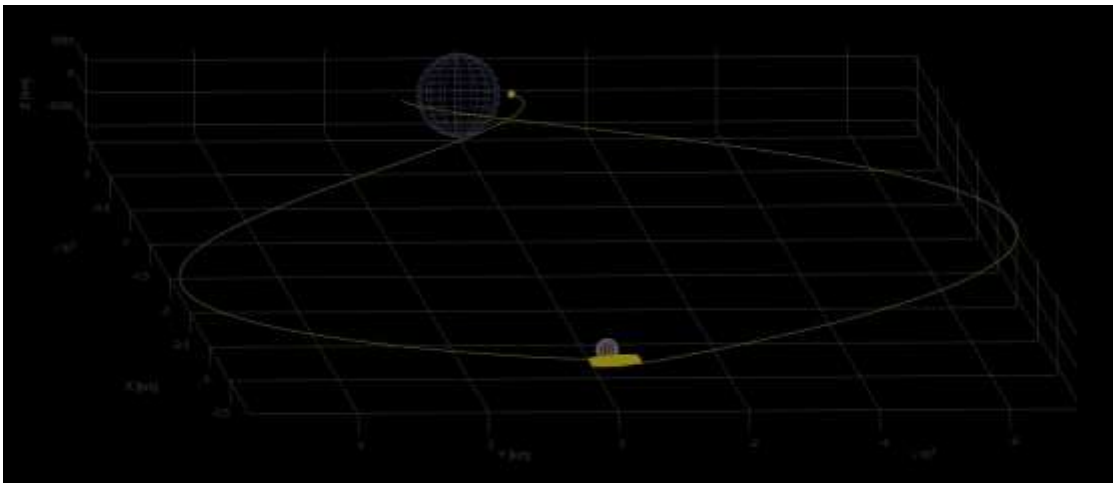


Figure 27. 3rd view of Barycenter Frame of Apollo 11 trajectory from Earth to Lunar orbit and back again.

To benchmark the Apollo 11 mission, hourly data points were obtained from JPL Horizons. The only available Apollo 11 data is for the S-IVB which was used for TLI and Lunar orbit insertion before being discarded. It is shown in red in the next figures and does not enter Lunar Orbit. So only the Apollo 11 trajectory up until Lunar orbit can be compared with.

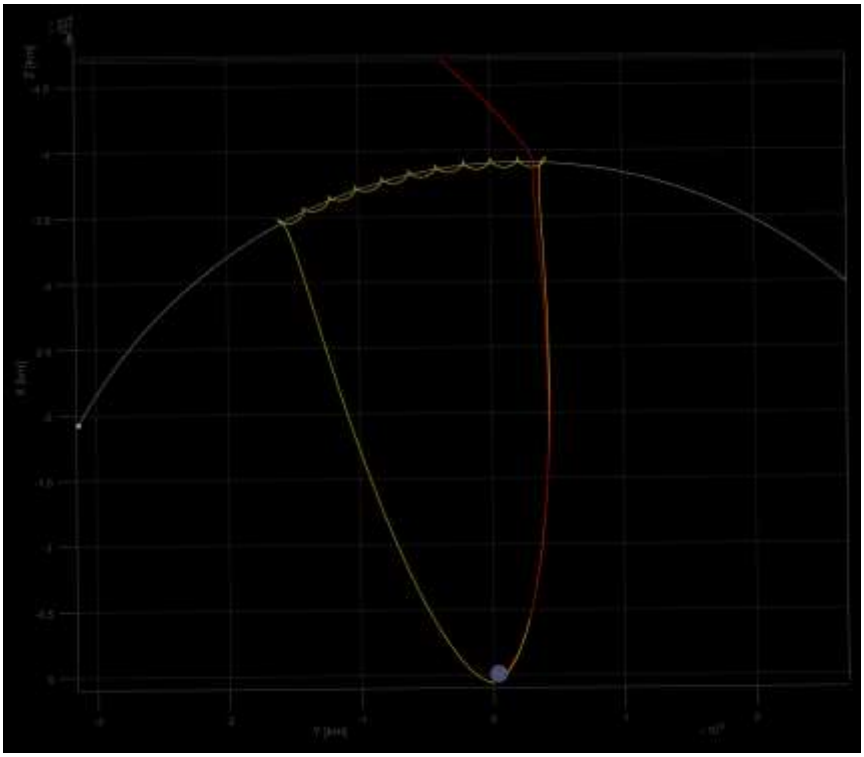


Figure 28. Apollo 11 trajectory from Earth to Lunar orbit and back again with the S-IVB location shown in red. (Top Inertial view).

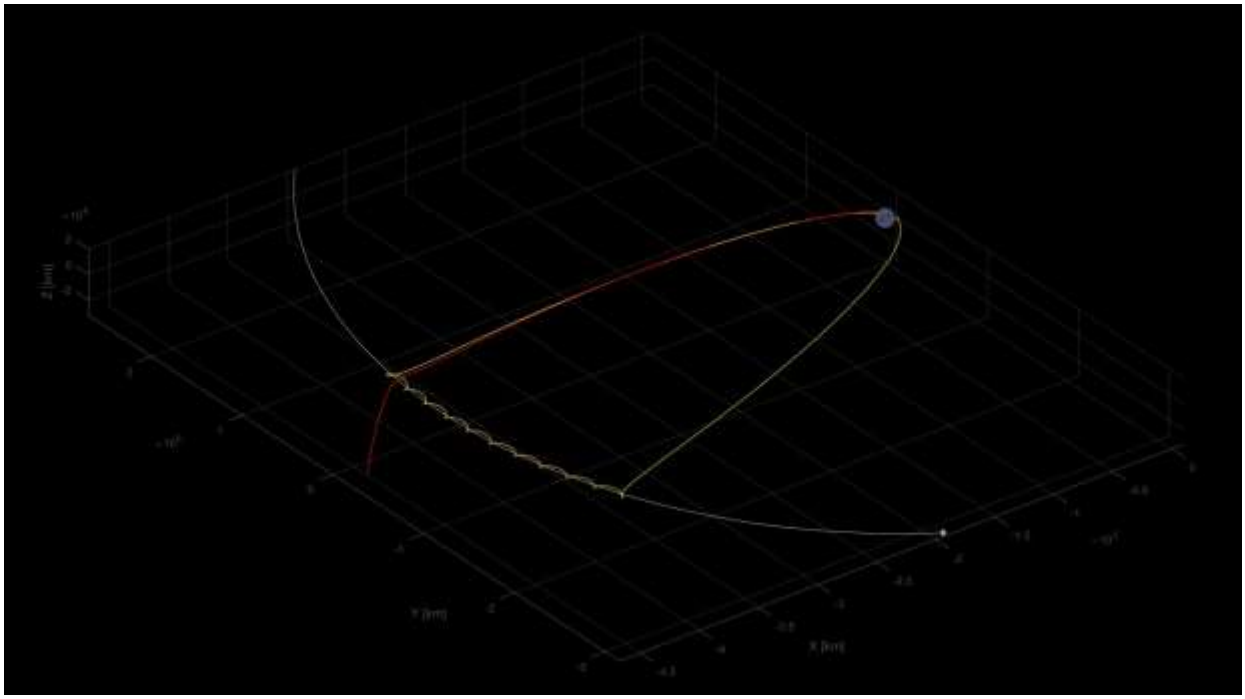


Figure 29. Apollo 11 trajectory from Earth to Lunar orbit and back again. S-IVB trajectory shown in red. (2nd Inertial view)

Figure 28 and Figure 29 show the MATLAB results very closely matching the S-IVB trajectory from JPL Horizons initially, but slightly diverging as it moves closer to the Moon. The reasoning for this is unclear. Possible explanations could be from unaccounted perturbation, such as solar wind, gravitational pull from the Sun or other solar system planets, and/or J2 perturbations during the initial translunar injection. These possibilities are considered very unlikely however, as these effects are expected to be extremely small for such a short flight.

Other, more likely reasonings could stem from a midcourse correction that was unaccounted for. Although the Apollo 11 flight plan called for a possible midcourse correction, from documents, it was stated as Nominally Zero.

TABLE 1-1 CSM BURN SCHEDULE

BURN / MANEUVER	GET: BURN TIME ΔVC	ATTITUDE (DEG): LM/LV	ATTITUDE (DEG): INERTIAL	LIGHTING	ΔV (FPS)	ULLAGE	TVC MODE	REFSMMAT	S/C WT, RESULTANT HA, HP	REMARKS
S-IVB TLI	02:44:16 5 MIN 10 SEC			BURNOUT AT 208:16	AVG: --- AVG: --- AVG: -- AV REQ: 10,451.0	----	----	PRE	WT: -- HP: -- HA: --	S-IVB BURN
TRANS S-IVB ATTITUDE MAN	03:20:44.9 2.8 SEC 15.8 FPS			DAYLIGHT	AVG: 5.1 AVG: 0.0 AVG: 14.9 AV REQ: 15.7	NOT REQUIRED	SAE AUTO	SAE	WT: 96462.3 HP: 115.8 HA: 191993.9	SPS BURN
MIDCOURSE CORRECTIONS MCC$_1$ TO MCC$_4$	11:45 26:45 53:55 70:55			----	AVG: NOMINALLY AVG: ZERO AVG: AV REQ:	NOT REQUIRED	SAE AUTO	SAE PTC PTC LOI 6378	----	TLI +5 TLI +24 LOI -20 LOI -9
LOI$_1$	75:54:28.4 5 MIN 58.9 SEC 2914.8 FPS			DAYLIGHT (AOS +4 MN 7 MIN)	AVG: 2891.8 AVG: -431.1 AVG: -20.4 AV REQ: 2924.1	NOT REQUIRED	SAE AUTO	LOI 6378	WT: 93107.4 HA: 59.2 HA: 169.8	SPS BURN
LOI$_2$	80:29:24.7 16.4 SEC			DAYLIGHT (AOS +9 MIN)	AVG: 138.2 AVG: 4.8 AVG: 73.9 AV REQ: 157.8	3 JET 10 SEC	SAE AUTO	LOI 6378	WT: 91428.6lbs HP: 53.8 HA: 65.4	SPS BURN

MIDCOURSE CORRECTIONS MCC$_1$ TO MCC$_4$	11:45 26:45 53:55 70:55		----	ΔVX: NOMINALLY ΔVY: ZERO ΔVZ: ΔV REQ:

Figure 30. Apollo 11 Burn Schedule from Flight Plan. The Midcourse correction is stated as Nominally Zero.

One would have to assume a burn small enough to be stated as "Nominally Zero" would not have a noticeable difference in trajectory at arrival to the Moon. Other possibilities are slightly different values of constants, such as the gravitational constant. Another possibility is a mistake in the code, but usually a coding mistake leads to large discrepancies, not small ones where every other simulation produces expected results.

Lastly, I believe the most likely scenario is failure to match JPL Horizons data in the frame of reference used in the simulation. JPL cannot give state vectors from the Earth-Moon barycenter, it must use a body for reference. So I hypothesize that converting JPL data to the barycenter frame may have cause an issue that may stem from the Earth's movement around the barycenter during the 3 days of travel. This would explain the gradual shifting of the trajectory of the S-IVB away from the MATLAB prediction. However, attempting to account for this did not give the desired results. Finding the cause of the discrepancy should be looked into further in a future examination.

X. Conclusion

The 3-body problem has played a surprisingly important role in math and physics for centuries, beginning with the influential, Isaac Newtown and his concept of gravity itself. The seemingly simple yet deceptively complicated addition of a single body, to a fully understood 2-body problem, must have been an extremely luring challenge for every serious mathematician of their day. Over its long unsolved tenure, new math was invented and advanced in pursuit of a solution, only to be proven to be unsolvable. Such an anticlimax to a century old question, yet despite this, better and more accurate approximate solutions were still pursued. Thanks to these efforts, and the advancement of computers, we can solve the 3-body and even n-body problems today, well almost solve. With such an influential problem, it is important to understand the development and evolution of its solutions, such as the circular restricted then elliptic restricted 3BPs. These solutions, along with computers, ultimately led to the n-body

solution used in this paper. As clearly illustrated in the sections above, the n-body problem is a very powerful tool for evaluating almost any desired system. With these simulation tools, in development since 1687, we are now beginning to understand and inch our way into the 99.9̄% of everything else out there.

Appendix A - Apollo Translunar Injection Keplerian Equations

Contents

Apollo Translunar Injection orbits

```
clear all; close all; clc;
format shortg
```

Apollo 11-17 TLI parameters

```
for j = 1:7
    switch j
        case 1
            % Apollo 11
            Alt = 1097229;          % [ft] Altitude
            Alt2 = 180.581;         % [nmi]
            EF_velocity = 34195.6; % [ft/s]
            SF_vel = 35545.6;    % [ft/s]
            FP_angle = 7.367;       % [deg]
            GeoLat   =   9.9204;   %   [deg   N]
            GeodeticLong  =  9.983;  %  [deg  N]
            Longitude = -164.8373; % [deg E]
            Inclin = 31.383; % [deg]
            W = 358.380069401768;

        case 2
            % Apollo 12
            Alt = 1209284;          % [ft] Altitude
            Alt2 = 199.023;         % [nmi]
            EF_velocity = 34020.5;     % [ft/s]
            SF_vel = 35389.8;       % [ft/s]
            FP_angle = 8.584;       % [deg]
            GeoLat = 16.0791;         % [deg N]
            GeodeticLong = 16.176;    % [deg N]
            Longitude = -154.2798; % [deg E]
            Inclin = 30.555;        % [deg]
            W = 159.004272511991;

        case 3
            % Apollo 13
            Alt = 1108555;          % [ft] Altitude
            Alt2 = 182.445;         % [nmi]
            EF_velocity = 34195.3;     % [ft/s]
            SF_vel = 35538.4;       % [ft/s]
            GeoLat = -3.8635;         % [deg N]
            GeodeticLong = -3.8602;    % [deg N]
            Longitude = 167.2074; % [deg E]
            FP_angle = 7.635;       % [deg]
            Inclin = 31.817;        % [deg]
            W = 341.843467490158;
```

```
        case 4
            % Apollo 14
            Alt = 1090930;          % [ft] Altitude
            Alt2 = 179.544;         % [nmi]
            EF_velocity = 34151.5;     % [ft/s]
            SF_vel = 35511.6;       % [ft/s]
            GeoLat = -19.4388;         % [deg N]
            GeodeticLong = -19.554;    % [deg N]
            Longitude = 141.7312; % [deg E]
            FP_angle = 7.480;       % [deg]
            Inclin = 30.834;        % [deg]
            W = 302.898707272848;

        case 5
            % Apollo 15
            Alt = 1055296;          % [ft] Altitude
            Alt2 = 173.679;         % [nmi]
            EF_velocity = 34202.2;     % [ft/s]
            SF_vel = 35579.1;       % [ft/s]
            GeoLat = 24.8341;          % [deg N]
            GeodeticLong = 24.9700;    % [deg N]
            Longitude = -142.1295;  % [deg E]
            FP_angle = 7.430;       % [deg]
            Inclin = 29.696;        % [deg]
            W = 354.850726982177;

        case 6
            % Apollo 16
            Alt = 1040493;          % [ft] Altitude
            Alt2 = 171.243;         % [nmi]
            EF_velocity = 34236.6;     % [ft/s]
            SF_vel = 35566.1;       % [ft/s]
            GeoLat = -11.9117;         % [deg N]
            GeodeticLong = -11.9881;    % [deg N]
            Longitude = 162.4820; % [deg E]
            FP_angle = 7.461;       % [deg]
            Inclin = 32.511;        % [deg]
            W = 335.248934532989;

        case 7
            % Apollo 17
            Alt = 1029299;          % [ft] Altitude
            Alt2 = 169.401;         % [nmi]
            EF_velocity = 34168.3;     % [ft/s]
            SF_vel = 35555.3;       % [ft/s]
            GeoLat = 4.6824;        % [deg N]
            GeodeticLong = 4.7100;    % [deg N]
            Longitude = -53.1190; % [deg E]
            FP_angle = 7.379;       % [deg]
            Inclin = 28.466;        % [deg]
            W = 147.315250419378;
    end
```

Initial orbit elements

```
    GM = 3.986005e14;    % [m^3/s^2]
    R_eq = 6378166;      % [m]
    R_pol = 6356784.3;   % [m]
    Psi_elip = GeoLat; % [deg]

    SF_vel = SF_vel * 0.3048;    % [m/s] Space-Fixed velocity
    Re_TLI = R_eq*R_pol/sqrt( (R_pol*cosd(Psi_elip))^2 ...
        + (R_eq*sind(Psi_elip))^2); % [m] Earth Radius at TLI
    r = Re_TLI + Alt*0.3048;     % [m] Radial distance of spacecraft from Earth center

    C = 2*GM/(r*SF_vel^2);
    R_p = (-C + sqrt(C^2 - 4*(1-C)*-cosd(FP_angle)^2))/(2*(1-C))*r; % [m] Perigee distance
    R_a = (-C - sqrt(C^2 - 4*(1-C)*-cosd(FP_angle)^2))/(2*(1-C))*r; % [m] apogee distance
```

```
e = sqrt( (r*SF_vel^2/GM - 1)^2 * cosd(FP_angle)^2 + sind(FP_angle)^2); % Eccentricity
nu = atand( (r*SF_vel^2/GM)*cosd(FP_angle)*sind(FP_angle) ...
    / ((r*SF_vel^2/GM)*cosd(FP_angle)^2 - 1)); % [deg] True anomaly
a = (R_p + R_a)/2; % [m] Semi-major Axis

l_eqLat = asind( sind(GeoLat) / sind(Inclin)); % [deg] Orbital Longitude
a_eqLong = atand( tand(l_eqLat) * cosd(Inclin)); % [deg] Equitorial Longitude
w = l_eqLat - nu; % Argument of perigee

GeoLon_ascNode = Longitude - a_eqLong + 360; % [deg] Geographic Longitude of Ascending Node

E = acos( (e + cosd(nu))/(1 + e*cosd(nu))); % [rad] Eccentric anomoly
M = E - e*sin(E); % [rad] Mean Anomoly
n = sqrt(GM/a^3); % [rad/s] Mean Motion
t = M/n; % [sec] time from parigee
```

Elements past time = 0

```
numRun = 145;


Apollo = zeros(numRun,10);


for i=1:numRun+2
    if i == 1
        t_per = t+(i-1)*60;
    elseif i < 33
        t_per = t +(i-2)*15*60; %% end
    elseif i < 103
        t_per = t +(i-2)*1.5*60*15;
    elseif i < 153
        t_per = t+(i-2)*15*2*60;
        %     elseif i < 63
        %         t_per = 746+(i-2)*15*4*2*60;
        %     elseif i < 70
        %         t_per = 746+(i-2)*15*4*3*60;
    end
    M2 = t_per * n;
    syms EE
    E2 = double(vpasolve(EE == M2 + e*sin(EE),EE));
    nu2 = acosd( (cos(E2) - e) / (1 - e*cos(E2)));
    r2 = a*(1-e^2)/(1 + e*cosd(nu2));
    f2 = atan2d((e*sind(nu2)),(1+e*cosd(nu2)));
    v2 = sqrt(GM*a*(1-e^2))/(r2*cosd(f2));
    l2 = nu2 + w; % Orbit longitude
    eq_long_asc = atan2d( (sind(l2)*cosd(Inclin)),cosd(l2));
    eq_long_cel = W + eq_long_asc - 360;
    eq_lat_cel = asind(sind(l2)*sind(Inclin));

    Apollo(i,1) = t_per; % [sec] time since perigee
    Apollo(i,2) = nu2; % [deg] True Anomaly
    Apollo(i,3) = f2; % [deg] Flight path angle
    Apollo(i,4) = v2; % [m/s] velocity
    Apollo(i,5) = r2; % [m] Distance from earth center
    Apollo(i,6) = eq_long_cel; % [deg] Longitude Equitorial Coordinates celestial
    Apollo(i,7) = eq_lat_cel; % [deg] Latitude Equitorial Coordinates celestial
    Apollo(i,8) = r2*sind(90-eq_lat_cel)*cosd(eq_long_cel); % X-coordinates
    Apollo(i,9) = r2*sind(90-eq_lat_cel)*sind(eq_long_cel); % Y-coordinates
    Apollo(i,10) = r2*cosd(90-eq_lat_cel); % Z-coordinates
end
```

Data print

```
[uu,nu,wu] = sphere;


ue = uu*6371e3;


ve = nu*6371e3;
```

```
    we = wu*6371e3;



    um = uu*1737400;

    vm = nu*1737400;

    wm = wu*1737400;



    load('topo.mat','topo','topomap1');
    topo2 = [topo(:,181:360) topo(:,1:180)];
    pro.FaceColor= 'texture';
    pro.EdgeColor = 'none';
    pro.FaceLighting = 'phong';
    pro.Cdata = topo2;

    title1 = {'Apollo 11', 'Apollo 12', 'Apollo 13', 'Apollo 14', 'Apollo 15', 'Apollo 16',
'Apollo 17'};
    subt = ' Orbit w/o Moon Influence';

    % Apollo;
    % [xt,yt,zt] = sphere(20);
    % figure
    % plot(Apollo(:,1),Apollo(:,5));
    % figure
    % plot3(Apollo(:,8),Apollo(:,9),Apollo(:,10));
    % grid on
    % hold on
    % plot3(xt,yt,zt)
    figure
    plot(Apollo(:,8),Apollo(:,9))
    hold on
    h1 = surface(ue,ve,we,pro);
    colormap(topomap1);
    title(strcat(title1(j),subt));
    xlabel('X - [m]');
    ylabel('Y - [m]');
    % hold on
    % h2(i) = surf(um-384.4e6,vm,wm,'FaceColor', [.8 .8 .8]);
    axis equal

    % fprintf('a = %.9g\n',a);
    % fprintf('e = %.6g\n',e);
    % fprintf('i = %.5g\n',Inclin);
    % fprintf('w = %.5g\n',w);
    % fprintf('W = %.3f\n',W);
    % fprintf('t = %.5g\n',t);
```

```
end
```

Appendix B - Circular Restricted Three-Body Problem (CR3BP)

Contents

Circular Restricted Three-Body Problem

```
clear all; close all; clc;
format shortg
global M1 M2 R G MU MU1

M1 = 5.9723e24; % [kg] Mass of Earth
M2 = 0.07346e24; % [kg] Mass of Moon
MU = M2 / (M1 + M2); % Mass ratio of Moon
MU1 = M1 / (M1 + M2); % Mass ratio of Earth
G = 6.67408e-11; % m^3/kg-s^2
R = 384400e3; % m

theta2 = -23.4 + 5.14; % Y-rotation accounting for Moon and Earth axis
eul2 = [cosd(theta2)         0       -sind(theta2);
    0               1       0;
    sind(theta2)    0       cosd(theta2)];

theta3 = -39.485; % Z-rotation accounting for position of Moon
eul3 = [cosd(theta3)     sind(theta3)    0;
    -sind(theta3)   cosd(theta3)    0;
    0               0               1];

r1e = [-6.3851e+06 -1.7158e+06 -1.1563e+06]'; % initial position of Apollo 11
r1 = eul3*eul2*r1e; % YZ-rotations applied
r = (r1 - [R*MU 0 0]'); % radius magnitude converted to vector


v = 14.9087; % [deg] true anomaly
gamma = 7.367; % [deg] flight path angle
theta4 = -(90-v+gamma); % [deg] Velocity angle transormation
i = 31.383; % [deg] inclination

velu = [sind(90-i)*cosd(theta4); % Velocity Transform
    sind(90-i)*sind(theta4);
    cosd(90-i)];

vel_mag = 10834.3; % [m/s] Velocity
vel = eul3*eul2*velu*vel_mag; % [m/s] Velocity transformation

dt = 1500000*24*60*60;
P = sqrt((4*pi^2*R^3)/(G*(M1+M2))); % [sec] Orbit Period
tout = (2*pi*dt)/P;

% xo1 = [-.5 -sqrt(3)/2 0 0 0 0]'
xo1 = [r(1) r(2) r(3) vel(1) vel(2) vel(3)]'; % Initial conditions
tout = 259200; % final time
```

ODE45 numerical solver

```
[t, y] = ode45('cr3bp3', [0:500:tout], xo1);
```

Plots and graphing

```
figure


hold on
plot(-y(1,1),-y(1,2),'b')
xlabel('X [m]')
ylabel('Y [m]')
grid on
[u,v,w] = sphere;
hold on
u = u*6371e3;
v = v*6371e3;
w = w*6371e3;
surf(-u+MU*R,v,w)
[u,v,w] = sphere;
hold on
u = u*1737400;
v = v*1737400;
```

```
w = w*1737400;
surf(-u-(R-MU*R),v,w)
title('Apollo 11 Circular Restricted Three-Body Problem')
axis equal

for i = 2:size(y)
    hold on
    plot(-y(1:i,1),-y(1:i,2),'b')
    pause(.01)
    axis equal
end


figure
plot3(-y(1,1),-y(1,2),-y(1,3),'b')
grid on
xlabel('X [m]')
ylabel('Y [m]')
zlabel('Z [m]')
axis equal
[u,v,w] = sphere;
hold on
u = u*6371e3;
v = v*6371e3;
w = w*6371e3;
surf(-u+MU*R,v,w)
[u,v,w] = sphere;
hold on
u = u*1737400;
v = v*1737400;
w = w*1737400;
surf(-u-(R-MU*R),v,w)
title('Apollo 11 Circular Restricted Three-Body Problem')

for i = 2:size(y)
    plot3(-y(1:i,1),-y(1:i,2),-y(1:i,3),'b')
    pause(.01)
end
```

```
function [ xdot ] = cr3bp2(t, x )
%UNTITLED11 Summary of this function goes here
%   Detailed explanation goes here

global M1 M2 R G MU MU1 % Global constants

tau = 2*pi/sqrt(G*(M1+M2))*R^(3/2); % Period
w1 = 2*pi/tau; % angular velocity

r_eb_mag = R * MU; % radius magnitude to earth from barycenter
r_mb_mag = R * MU1; % radius magnitude to moon from barycenter

r_eb = r_eb_mag * [-1 0 0]'; % coversion to vector form
r_mb = r_mb_mag * [1 0 0]'; % conversion to vector form

xdot(1:3,1) = x(4:6,1); % velocity to xdot
rho = x(1:3,1); % barycenter to satalite
v = x(4:6,1); % velocity

r1 = r_eb - rho; % radius to earth from barycenter vector
r2 = r_mb - rho; % radius to moon from barycenter vector

r1_mag = norm(r1); % magnitude of earth to barycenter radius
r2_mag = norm(r2); % magnitude of moon to barycenter radius

% Equations of motion
xdd = 2*w1*v(2) + w1^2*rho(1) - (G*M1*(rho(1)+MU*R))/r1_mag^3 - G*M2*(rho(1)-MU1)/r2_mag^3;
ydd = -2*w1*v(1) + w1^2*rho(2) - (G*M1*(rho(2)))/r1_mag^3 - G*M2*(rho(2))/r2_mag^3;
zdd = - (G*M1*(rho(3) ))/r1_mag^3 - G*M2*(rho(3) )/r2_mag^3;
```

```
xdot(4:6,1) = [xdd ydd zdd]';
end
```

Appendix C - Elliptic Restricted Three-Body Problem (ER3BP)

Contents

Elliptic Restricted Three-Body Problem

```
clear all; close all; clc;
```

Main Code

```
global rho gamma

while 1
    apm = 11;


    prompt = 'Which simulation would you like to run?\nMoon Orbit[1]    Stable Langrangian
Point[2]   Apollo Missions[3]   Random Trajectory[4]\n';
    sim = input(prompt);
    if sim ~= 0:4
        fprintf(2,'\nError: Input not recognized.\n\n');
        continue
    end
    if sim == 3
        apm = input('Which Apollo mission [11, 12, 14, 15, 16, 17]: \n');
    end
    close all;


    %%%%%% Constants/Variables
    a = 384748e3;           % [m] Semi-Major Axis
    e = 0.05490;            % Eccentricity
    n = 2.661699e-6;        % [rad/s] Mean angular rate
    gammaL1 = -0.150935;    % Instantaneous libration point 1
    gammaL2 = 0.167833;     % Instantaneous libration point 2
    G = 6.67408e-11;        % [m^3/(kg s^2)] Gravitational Constant
    M1 = 5.9723e24;         % [kg] Mass of Earth
    M2 = 0.07346e24;        % [kg] Mass of Moon
    rho = M2 / (M1 + M2);   % Mass Ratio
    %%%%%% Constants/Variables
```

Apollo mission rotations and pos/vel calculations

```
    switch apm
        case 11
            % Apollo 11
            r_pos = [-6.3851e+06 -1.7158e+06 -1.1563e+06]';
            vel_mag = 10834.3;
            inc = 31.383;
            gamma_tj = 7.367;
```

```
            offs = 7.52;
            theta3 = -29.2;
            nu = 14.9087;

        case 12
            % Apollo 12
            r_pos = -[-6.4145e+06  -9.2785e+05   1.8682e+06]';
            vel_mag = 10787;
            inc = 30.555;
            gamma_tj = 8.584;
            offs = 15.573;
            theta3 = 159.004;
            nu = 17.439;

        case 13
            % Apollo 13
            r_pos = -[6.1019e+06 -2.7687e+06  -4.5252e+05]';
            vel_mag = 10832;
            inc = 31.817;
            gamma_tj = 7.635;
            offs = 22.791;
            theta3 = 341.843;
            nu = 15.448;

        case 14
            % Apollo 14
            r_pos = -[-3.6914e+05 -6.3151e+06  -2.2325e+06]';
            vel_mag = 10824;
            inc = 30.834;
            gamma_tj = 7.48;
            offs = -55.664;
            theta3 = 302.899;
            nu = 15.175;

        case 15
            % Apollo 15
            r_pos = -[3.9794e+06   4.5926e+06   2.8123e+06]';
            vel_mag = 10845;
            inc = 29.696;
            gamma_tj = 7.43;
            offs = 7.52;
            theta3 = -29.2;
            nu = 15.044;

        case 16
            % Apollo 16
            r_pos = -[4.7055e+06 -4.5567e+06  -1.3817e+06]';
            vel_mag = 10841;
            inc = 32.511;
            gamma_tj = 7.461;
            offs = 7.52;
            theta3 = -29.2;
            nu = 15.121;

        case 17
            % Apollo 17
            r_pos = -[-6.093e+06   2.7123e+06   5.4626e+05]';
            vel_mag = 10837;
            inc = 28.466;
            gamma_tj = 7.379;
            offs = 7.52;
            theta3 = -29.2;
            nu = 14.97;
    end

    theta2 = -23.44 - 5.14 + offs;

    % Tilt Offset
    eul2 = ...
        [cosd(theta2)        0       -sind(theta2);
```

```
        0               1       0;
        sind(theta2)        0       cosd(theta2)];

    % Moon Position Offset
    eul3 = ...
        [cosd(theta3)       sind(theta3)       0;
        -sind(theta3)       cosd(theta3)    0;
        0               0               1];

    r_rot = eul3*eul2*r_pos;
    r_SC = (r_rot - [a*rho 0 0]');

    theta4 = -(90-nu+gamma_tj);

    vel_uvec = ...
        [sind(90-inc)*cosd(theta4);
        sind(90-inc)*sind(theta4);
        cosd(90-inc)];

    vel = eul3*eul2*vel_uvec*vel_mag;
```

Initial conditions, Simulation Cases,

```
    syms M ti
    D1(M) = 1 + 1/2*e^2 + (-e + 3/8*e^3 - 5/192*e^5 + 7/9216*e^7)*cos(M) + (-1/2*e^2 + 1/3*e^4
...
        - 1/16*e^6)*cos(2*M) + (-3/8*e^3 + 45/128*e^5 - 567/5120*e^7)*cos(3*M) + (-1/3*e^4 +
2/5*e^6)*cos(4*M) ...
        + (-125/384*e^5 + 4375/9216*e^7)*cos(5*M) - 27/80*e^6*cos(6*M) -
16807/46080*e^7*cos(7*M);

    gamma = 0;
    Da = double(D1(0));

    %%%%%% Initial Conditions
    switch sim
        case 1
```

1 Moon Orbit

```
            dt = .001;

            days = 27.3217;

            x1 = [-1.05, 0, 0, 0, 0, 0]'

        case 2
```

2 L4/L5 Stable Lagrangian Point

```
            dt = .2;

            days = 2000;

            x1 = [-0.5, sqrt(3)/2, 0, 0, 0, 0]'*Da

        case 3
            % %%% 3 Apollo 11 Lunar Injection
            dt = .001;
            days = 2.5;
            x1 = [-r_SC(1), -r_SC(2), r_SC(3), -vel(1)/n, -vel(2)/n, vel(3)/n]'/a'
        case 4
```

Random

```
            dt = .004;
```

```
            days = 50;

            x1 = [.001, .045, 0, -6.39e3/n/a, 1.35e3/n/a, .26e3/n/a]'

        case 0
            break
    end
    x0 = x1 + [double(D1(0))*(1 - rho + gamma) 0 0 0 0 0]';
```

ODE45 Function call

```
    %%%%%% ODE45 Calculation and Time
    tf = days*24*3600*n;
    [t, y] = ode45('er3bp2', 0:dt:tf, x0);
```

Graphing and Calculations

```
    y = y*a;


    %%%%%% Rotational Matrix
    rot(ti) = [cos(n*ti) -sin(n*ti) 0; sin(n*ti) cos(n*ti) 0; 0 0 1];

    %%%%%% Planet Sphere Initialization
    [u,n1,w] = sphere;

    ue = u*6371e3;
    ve = n1*6371e3;
    we = w*6371e3;

    um = u*1737400;
    vm = n1*1737400;
    wm = w*1737400;


    %%%%%% Orbit Plotting
    figure
    xlabel('x')
    ylabel('y')
    axis equal
    grid on

    h1 = zeros(length(y));
    h2 = zeros(length(y));
    y_adj = zeros(length(y),3);

    %%%%%% Earth Topography
    load('topo.mat','topo','topomap1');
    topo2 = [topo(:,181:360) topo(:,1:180)];
    pro.FaceColor= 'texture';
    pro.EdgeColor = 'none';
    pro.FaceLighting = 'phong';
    pro.Cdata = topo2;
```

Plots 2D in Rotational Frame of Barycenter

```
    for i = 1:size(y)
        if i > 1
            delete(h1(i-1));
            delete(h2(i-1));
            delete(h3(i-1));
        end

        t2 = (i-1)*dt;
        Dt = 1.001507005 - 0.000003017133411*cos(4.0*t2) - 0.0000001616320167*cos(5.0*t2) ...
            - 0.0000618757641*cos(3.0*t2) - 0.000000009240763254*cos(6.0*t2) -
0.0000000005482569438*cos(7.0*t2) ...
            - 0.05483796206*cos(t2) - 0.001503978626*cos(2.0*t2);
        Dt = Dt*a;
```

```
        hold on
        h1(i) = surface(ue+(Dt)*(rho),ve,we,pro);
        colormap(topomap1);
        rotate(h1(i), [0.47839,0,0.87815], (7.292115855377074e-005-
0.00000265868)*t2/n*180/pi,[(Dt)*(rho),0,0]);

        hold on
        h2(i) = surf(um-(Dt)*(1 - rho),vm,wm,'FaceColor', [.8 .8 .8]);

        hold on
        y_adj(i,:) = [y(i,1)-Dt*(1 - rho + gamma), y(i,2), y(i,3)]';
        h3(i) = plot(y_adj(1:i,1),y_adj(1:i,2),'b', 'linewidth', 0.2);
        title('Apollo 11 Elliptic Restricted 3-Body Problem');
        xlabel('X [m]');
        ylabel('Y [m]');

        drawnow limitrate
    end
```

Plots 3D in Inertial Frame of Barycenter

```
    figure


    xlabel('x')
    ylabel('y')
    zlabel('z')
    axis equal
    grid on
    for i = 1:size(y)
        if i > 1
            delete(h1(i-1));
            if i ~= 365 && i ~= 2
                delete(h2(i-1));
            end
            delete(h3(i-1));
        end

        t2 = (i-1)*dt;
        Dt = 1.001507005 - 0.000003017133411*cos(4.0*t2) - 0.0000001616320167*cos(5.0*t2) ...
            - 0.0000618757641*cos(3.0*t2) - 0.000000009240763254*cos(6.0*t2) -
0.0000000005482569438*cos(7.0*t2) ...
            - 0.05483796206*cos(t2) - 0.001503978626*cos(2.0*t2);
        Dt = Dt*a;

        hold on
        uue = [+(Dt)*(rho) 0 0]';
        uue = double(rot(t2/n)*uue);
        h1(i) = surface(ue+uue(1),ve+uue(2),we+uue(3),pro);
        colormap(topomap1);
        rotate (h1(i), [0.47839,0,0.87815], 7.292115855377074e-005*t2/n*180/pi,uue');

        hold on
        uum = [-Dt*(1 - rho) 0 0]';
        uum = double(rot(t2/n)*uum);
        h2(i) = surf(um+uum(1),vm+uum(2),wm+uum(3),'FaceColor', [.8 .8 .8]);

        hold on
        y_adj(i,:) = double(rot(t2/n))*[y(i,1)-Dt*(1 - rho + gamma) y(i,2) y(i,3)]';
        h3(i) = plot3(y_adj(1:i,1),y_adj(1:i,2),y_adj(1:i,3),'b', 'linewidth', 0.2);
        title('Apollo 11 Elliptic Restricted 3-Body Problem');
        xlabel('X [m]');
        ylabel('Y [m]');
        zlabel('Z [m]');

        drawnow limitrate
    end
end
```

```
function [ rdot] = er3bp2( t, x )
%Elliptic restricted three-body problem
%   D and theta-dot derivatives calculated ahead of time for speed

global rho gamma D2 Dd2 Ddd2 thd2 thdd2

% J2 = 1.08265e-3;
% thdd = -3/2*n*J2*

rdot(1:3,1) = x(4:6,1);

v = x(4:6,1);
r = x(1:3,1);


M = t;
D = 1.001507005 - 0.000003017133411*cos(4.0*M) - 0.0000001616320167*cos(5.0*M) ...
    - 0.0000618757641*cos(3.0*M) - 0.000000009240763254*cos(6.0*M) -
0.000000000548256943 8*cos(7.0*M) ...
    - 0.05483796206*cos(M) - 0.001503978626*cos(2.0*M);

Dd = 0.003007957252*sin(2.0*M) + 0.00001206853364*sin(4.0*M) + 0.0000008081600836*sin(5.0*M) ...
    + 0.0001856272923*sin(3.0*M) + 0.000000055444457952*sin(6.0*M) +
0.000000003837798606*sin(7.0*M) + 0.05483796206*sin(M);

Ddd = 0.006015914503*cos(2.0*M) + 0.00004827413458*cos(4.0*M) + 0.000004040800418*cos(5.0*M) ...
    + 0.0005568818769*cos(3.0*M) + 0.0000003326674771*cos(6.0*M) + 0.00000002686459025*cos(7.0*M)
+ 0.05483796206*cos(M);

thd = 0.007526702614*cos(2.0*M) + 0.00003888369655*cos(4.0*M) + 0.000002839773804*cos(5.0*M) ...
    + 0.000536770327*cos(3.0*M) + 0.0000002092861752*cos(6.0*M) + 0.00000001542080711*cos(7.0*M)
+ 0.1097586587*cos(M) + 1.0;

thdd = - 0.01505340523*sin(2.0*M) - 0.0001555347862*sin(4.0*M) - 0.00001419886902*sin(5.0*M) ...
    - 0.001610310981*sin(3.0*M) - 0.000001255717051*sin(6.0*M) - 0.0000001079456497*sin(7.0*M) -
0.1097586587*sin(M);

r1 = sqrt((-(1+gamma)*D+r(1))^2 + r(2)^2 + r(3)^2);
r2 = sqrt((-gamma*D+r(1))^2 + r(2)^2 + r(3)^2);

xdd = 2*thd*v(2) + thdd*r(2) + thd^2*(-(1+gamma)*D + r(1)) + (1 + gamma)*Ddd - (1 - rho)*(-(1 +
gamma)*D + r(1))/r1^3 - rho*(-gamma*D + r(1))/r2^3 + rho/D^2;
ydd = -thdd*(-(1 + gamma)*D + r(1)) + thd^2*r(2) - 2*thd*(-(1 + gamma)*Dd + v(1)) - (1 -
rho)*r(2)/r1^3 - rho*r(2)/r2^3;
zdd = -(1 - rho)*r(3)/r1^3 - rho*r(3)/r2^3;
rdot(4:6,1) = [xdd ydd zdd]';
end
```

Appendix D – N-Body Problem

N-Body Main

Contents

- Animation Loop for Orbit Visualization

N-Body Problem

```
clear all;
close all;
clc;
```

Input

```
% iterations
iter   = 1e5;

anim = 2;
rePlay = 0;
while 1
    try
        ApNum = input('Which Simulation: '); % Simulation number input
    catch ME
        fprintf(2,'%s\n', ME.message);
        continue
    end
    if isempty(ApNum) % Usable value test
        fprintf(2,'Error: input not recognized.\n'); % Error notification
        continue
    elseif ApNum >= 10 && ApNum <= 12 && rem(ApNum,1) == 0 || ApNum == 5 ...
            || ApNum == 0 || ApNum == 66 || ApNum == 6 || ApNum == 777 || ApNum == 3
        break
    else
        fprintf(2,'Error: input not recognized.\n'); % Error notification
        continue
    end
end

if ApNum ~= 0
    while 1
        try
            ref_Frame = input('Frame [Inertial(1), Rotational(2)]: '); % rot or inertial frame
        catch ME
            fprintf(2,'%s\n', ME.message);
            continue
        end
        if isempty(ref_Frame) % Usable value test
            fprintf(2,'Error: input not recognized.\n'); % Error notificati1on
            continue
        elseif ref_Frame == 1 || ref_Frame == 2
            break
        else
            fprintf(2,'Error: input not recognized.\n'); % Error notification
            continue
        end
    end
else
    ref_Frame = 1;
end
```

Time Parameters

```
tob = [];


bvec = [];
```

Initial State Vectors and Masses

```
global tol
tol = 1e-19;
gravFab = 0;
centObj = 0;
barycen = 0;
```

```
if ApNum >= 10 && ApNum <= 12 && rem(ApNum,1) == 0;
    if ApNum == 11; x11=apolloHor(); end
    [p0, v0, mu, n, scale, cA, tob, bvec] = ApolloCoords(ApNum);
    pFrame = 4:6;
    yrs = 0; days = 7; hrs = 21; sec = 0;
    spd = 2; solv = 5; blk = 0; gravFab = 0; centObj = 1;

elseif ApNum == 5;
    [p0, v0, mu, n, scale, cA] = LagrangeP(ApNum);
    pFrame = 4:6;
    yrs = 1; days = 0; hrs = 0; sec = 0;
    spd = 3; solv = 2; blk = 0;

elseif ApNum == 66;
    [p0, v0, mu, n, scale, cA] = Rings2();
    pFrame = 4:6;
    yrs = 0; days = 5; hrs = 0; sec = 0;
    spd = 1; solv = 3; blk = 0;

elseif ApNum == 777;
    [p0, v0, mu, n, scale, cA] = Rings3();
    pFrame = 4:6;
    yrs = 0; days = 10; hrs = 0; sec = 0;
    spd = 2; solv = 3; blk = 1;

elseif ApNum == 6;
    [p0, v0, mu, n, scale, cA, tob, bvec, xvec] = CassiniCoord;
    pFrame = 10:12;
    yrs = 0; days = 2; hrs = 0; sec = 211939200;
    spd = 4; solv = 2; blk = 0; centObj = 0;
    xvec = [xvec; 0 0 1e-10];

elseif ApNum == 3;
    [p0, v0, mu, n, scale, cA] = FallSolEr;
    [p0, v0, mu, n, scale, cA] = FallNoP;
    pFrame = 1:3;
    yrs = 0; days = 0; hrs = 0; sec = 0;
    spd = 1; solv = 5; blk = 0;
else
    [p0, v0, mu, n, scale, cA] = StateVecInit;
    yrs = 300; days = 0; hrs = 0; sec = 0;
    spd = 1; solv = 1; centObj = 1; blk = 0; barycen = 1;
end

time = ((yrs*365.25+days)*24+hrs)*3600+sec;
```

G. N-Body Function for Numerical Integration Solver

```
tic


fprintf('\n working ...\n\n');

% Numerical Solver Function
t=[];
x=[];
dx=[];
ddx=[];

% Burn Variable Initialization
dt = iter/time;
pos(1,:) = p0;
vel(1,:) = v0;
tob = horzcat(0,tob,time);
bvec = [bvec; 0 0 0];

% Satellite Burn Segments Loop
for i = 1:length(tob)-1
```

```
    % Sets section time and iteration for each burn segment
    iter2(i) = round((tob(i+1)-tob(i))*dt);
    tt = linspace(tob(i),tob(i+1),iter2(i));

    % Burn Segment Solver/Propagation
    [t1, x1, dx1, ddx1] = nBodySolver(pos(i,:)', vel(i,:)', mu, tt', solv);

    % Set next burn segment initial values to previous end values
    % with adjusted velocity from Delta V burn
    if i ~= 0 && ApNum == 6;
        x1(end,(3*n-2):3*n) = xvec(i,:);
    end
    pos(i+1,:) = x1(end,:);
    vsat = dx1(end,(3*n-2):3*n);
    if i ~= 0 && ApNum == 6
        dx1(end,(3*n-2):3*n) = bvec(i,:);
    else
        dx1(end,(3*n-2):3*n) = vsat/norm(vsat)*norm(([norm(vsat) 0 0]+bvec(i,:)));
    end
    vel(i+1,:) = dx1(end,:);

    % Pos Vel Acc Patching of solver segments
    x = [x;x1];
    dx = [dx;dx1];
    ddx = [ddx;dx];
    t = [t;t1];
end

% Removes duplicate values from patching
dup = find(hist(t,unique(t))>1);
t(dup)=[];
x(dup,:)=[];
dx(dup,:)=[];
ddx(dup,:)=[];

figN = 1;
if ref_Frame == 2


    xR = zeros(size(x));
    dxR = zeros(size(x));
    ddxR = zeros(size(x));
    Arot = vrrotvec2mat(vrrotvec(x(1,pFrame),[-1 0 0]));
    for i = 1:size(x,2)/3

        ip = 1+(i-1)*3;
        fp = 3+(i-1)*3;
        for j = 1:spd:size(x,1)

            v1x = x(j,pFrame);
            vec = x(1,pFrame);

            Arot2 = Arot*vrrotvec2mat(vrrotvec(v1x,vec));
            xR(j,ip:fp) = Arot2*x(j,ip:fp)';
            dxR(j,ip:fp) = Arot2*dx(j,ip:fp)';
            % ddxR(j,ip:fp) = Arot2*ddx(j,ip:fp)';

        end
    end

    Arot = vrrotvec2mat(vrrotvec([0, xR(1,8:9)],[0 1 0]));
    for i = 1:size(xR,2)/3

        ip = 1+(i-1)*3;
        fp = 3+(i-1)*3;
        for j = 1:spd:size(xR,1)

            v1x = xR(j,pFrame);
            vec = xR(1,pFrame);
```

```
            Arot2 = Arot*vrrotvec2mat(vrrotvec(v1x,vec));
            xR(j,ip:fp) = Arot2*xR(j,ip:fp)';
            dxR(j,ip:fp) = Arot2*dxR(j,ip:fp)';
            % ddxR(j,ip:fp) = Arot2*ddxR(j,ip:fp)';

        end
    end


    x = xR;
    dx = dxR;
    % ddx = ddxR;

end
toc
```

```
  working ...



Elapsed time is 8.187602 seconds.
```

H. Graphical Setup

```
% Barycenter calculation
bary = zeros(size(x,1),3);
if ApNum == 0
    for i = 1:size(x,1)
        bary(i,:) = ([dot(mu,x(i,1:3:end)), dot(mu,x(i,2:3:end)),...
            dot(mu,x(i,3:3:end))]/sum(mu));
    end
end

% Change of reference frame
x = horzcat(x,bary);
if centObj ~=0
    objA = centObj*3-2:centObj*3;
    mDiff = x(:,objA);
    x = x - repmat(mDiff,1,size(x,2)/3);
    bary = bary - repmat(mDiff,1,size(bary,2)/3);
end


while 1
    while 1
        try
            anim = input('Animation [Yes(1), No(2)]: '); % Animate or show last frame?
        catch ME
            fprintf(2,'%s\n', ME.message);
            continue
        end
        if isempty(anim) % Usable value test
         fprintf(2,'Error: input not recognized.\n'); % Error notificatilon
            continue
        elseif anim == 1 || anim == 2
            break
        else
         fprintf(2,'Error: input not recognized.\n'); % Error notification
            continue
        end
    end
%     anim = 2;
    if anim == 1
        plst = 1;
    elseif anim == 2
        plst = size(x,1);
    end
```

```
wmax = max(max(x(:,:))) * 1.05;
wmin = min(min(x(:,:))) * 1.05;
if abs(wmax)>abs(wmin);rnge=abs(wmax); else rnge=abs(wmin);end
ax_range = [-rnge,rnge, -rnge,rnge, -rnge, rnge];
if ishandle(figN) == 0

    [u1,n1,w1] = sphere(16);
    us = u1;
    vs = n1;
    ws = w1;

    set(0,'defaultfigurecolor', [0 0 0])
    figure
    set(gcf,'position',[2561 219 1680 979]);
    axis equal
    if blk == 1; axis(ax_range,'square'); end
    grid on
    set(gca,'Color', [0 0 0])
    ax2 = gca;
    alpha(1)

    xlabel('X [km]');
    ylabel('Y [km]');
    zlabel('Z [km]');
    rotate3d on
end
```

I. Animation Loop for Orbit Visualization

```
if ApNum == 11
    plot3(x11(:,1), x11(:,2),...
        x11(:,3), 'r-', 'markeredgecolor', 'r',...
        'markers', .01); hold on
end

pColor = SetColor(n);
plotStep = plst:spd:size(x,1);
h1 = zeros(plotStep(end),n);
h2 = zeros(plotStep(end),n);
h0 = zeros(plotStep(end),1);
h01 = zeros(plotStep(end),1);
m = 0;
loops = size(plotStep,2);
F(loops) = struct('cdata',[],'colormap',[]);
for i = plotStep

    if barycen == 1
        h0(i) = surf(us*scale(4) + bary(i,1), vs*scale(4) +...
            bary(i,2), ws*scale(4) + bary(i,3), 'FaceColor', 'none',...
            'EdgeColor', 'w'); hold on;
        h01(i) = plot3(bary(1:spd:i,1), bary(1:spd:i,2),...
            bary(1:spd:i,3), 'w-', 'markeredgecolor', 'r',...
            'markers', .1); hold on
    end

    if n > 1
        for j = 1:n
            if i <= (round(dup/spd)+50) & i >= (round(dup/spd)-50)
                h1(i,j) = surf(us*scale(j) + x(i,1+(j-1)*3),...
                    vs*scale(j) + x(i,2+(j-1)*3),...
                    ws*scale(j) + x(i,3+(j-1)*3), 'FaceColor',...
                    'none', 'EdgeColor', pColor(cA(j),1:3)); hold on;
                h2(i,j) = plot3(x(1:spd:i,1+(j-1)*3),...
                    x(1:spd:i,2+(j-1)*3), x(1:spd:i,3+(j-1)*3),...
                    '-', pColor(cA(j),1:3), 'markeredgecolor',...
                    pColor(cA(j),1:3), 'markers', .01); hold on
            else
                h1(i,j) = surf(us*scale(j) + x(i,1+(j-1)*3),...
                    vs*scale(j) + x(i,2+(j-1)*3),...
```

```
                    ws*scale(j) + x(i,3+(j-1)*3), 'FaceColor',...
                    'none', 'EdgeColor', pColor(cA(j),1:3)); hold on;
                h2(i,j) = plot3(x(1:spd:i,1+(j-1)*3),...
                    x(1:spd:i,2+(j-1)*3), x(1:spd:i,3+(j-1)*3),...
                    '-', 'color', pColor(cA(j),1:3),...
                    'markeredgecolor', pColor(cA(j),1:3),...
                    'markers', .01); hold on
            end
        end
    end

    if ApNum == 66 || ApNum == 777
        h3 = plot3(x(i,7:3:end), x(i,8:3:end), x(i,9:3:end),...
            '.', 'markeredgecolor', pColor(1,1:3), 'markers',...
            12); hold on % Rings
    end

    % Gravitational Potential Surface
    if gravFab == 1
        [X,Y] = meshgrid(-wmin*1.5:wmin*1.5/100:wmin*1.5,...
            -wmin*1.5:wmin*1.5/100:wmin*1.5);
        Z = -(mu(1)./(sqrt((X-x(i,1)).^2+(Y-x(i,2)).^2)).^1 +...
            mu(2)./(sqrt((X-x(i,4)).^2+(Y-x(i,5)).^2).^1))*.5e5-.1e5;
        h4 = surf(X,Y,Z,'facecolor',[.3 .3 .3],'FaceAlpha',0.1,...
            'EdgeColor',[.1 .1 .1]);
    end

    grid on
    axis equal
    if blk == 1; axis(ax_range,'square'); end
    set(gca,'Color',[0 0 0])
    ax2.XColor = [.4 .4 .4];
    ax2.YColor = [.4 .4 .4];
    ax2.ZColor = [.4 .4 .4];
    ax2.GridAlpha = .4;
    ax2 = gca;
    ax2.GridColor = [1 1 1];
    alpha(1)
    xlabel('X [km]');
    ylabel('Y [km]');
    zlabel('Z [km]');

    % m = m + 1;
    % F(m) = getframe;
    % drawnow()
    pause(1e-20)


    if i ~= plotStep(end)
        if barycen == 1
            delete(h0(i))
            delete(h01(i))
        end
        delete(h2(i,:))
        if ApNum == 66 || ApNum == 777
            delete(h3)
        end
        if gravFab == 1;
            delete(h4)
        end
        if n > 1
            delete(h1(i,:))
        end
    end
  end

  % Requests user input to stop or run the animation again
  while 1
      try
rePlay = input('\nWould you like to graph again? \n Yes = 1, No = 0: ');
```

```
        catch ME
            fprintf(2,'%s\n', ME.message);
            continue
        end
        if isempty(rePlay) % Usable value test
            fprintf(2,'Error: input not recognized.\n');%Error notification
            continue
        elseif rePlay == 1

            fprintf('\n')
            if ishandle(1)
                if barycen == 1
                    delete(h0(i))
                    delete(h01(i))
                end
                delete(h2(i,:))

                if ApNum == 66 || ApNum == 777
                    delete(h3)
                end
                if n > 1
                    delete(h1(i,:))
                end
                if gravFab == 1
                    delete(h4)
                end
            end
            break
        elseif rePlay == 0
            break
        else
            fprintf(2, 'Error: Value was not 1 or 0.\n');
            continue
        end
    end

    if rePlay == 0
        break
    end
```

```
end

% myVideo = VideoWriter('CassiniRK86.avi');
% open(myVideo);
% writeVideo(myVideo, F);
% close(myVideo);
% end
```

N-Body Solver Function

```
function [t,x,dx,ddx] = nBodySolver(p0, v0, mu, tt, solv)

global tol
N = length(mu);
x123   = repmat(1:N,1,N);
x23    = x123(logical(ones(N) - eye(N)));
x12_34 = reshape(1:N*(N-1),N-1,N)';
x11    = reshape(repmat(1:N,N-1,1),1,N*(N-1));

switch solv
    case 1; [t,dz]   = ode113(@nBodyFunc,tt,[p0;v0]);
    case 2; [t,x,dx] = rkn86(@nBodyFunc,tt(1),tt(end),p0,v0);
    case 3; [t,dz]   = ode23(@nBodyFunc,tt,[p0;v0]);
    case 4; [t,x,dx] = rkn1210(@nBodyFunc2, tt, p0, v0);
    case 5; [t,dz]   = rk89(@nBodyFunc,[tt(1),tt(end)],[p0;v0], tol);
```

```
end
if solv ~= 2 && solv ~= 4
    x = dz(:,1:3*N);
    dx = dz(:,3*N+1:end);
end

xSize = size(x);
ddx = zeros(xSize);
% for i = 1:xSize(1)
%     xv = [x(i,:)';dx(i,:)'];
%     pos = reshape(xv(1:3*leng),3,leng);
%     r = pos(:,x23) - pos(:,x11);
%     r3 = diag(1./sqrt(sum(r.^2)).^3);
%     gm = diag(mu(x23));
%     dpos = r*r3*gm;
%     ddx(i,:) = sum(reshape(dpos(:,x1214(:)),3*leng,leng-1),2);
% end

    function [ss_vec] = nBodyFunc2(t,xv)
        pos = reshape(xv(1:3*N),3,N);
        r_ij = pos(:,x23) - pos(:,x11);
        r3 = diag(1./sqrt(sum(r_ij.^2)).^3);
        mu_diag = diag(mu(x23));
        acc_Sum = r_ij*r3*mu_diag;
        ss_vec = sum(reshape(acc_Sum(:,x12_34(:)),3*N,N-1),2);
    end

%%%%% N-body Eq of Motion (EOM) Construction Function (Vectorized Method)%%%%%
    function [ss_vec] = nBodyFunc(t,xv)
        pos     = reshape(xv(1:3*N),3,N); % Separate/reshape position vectors
        r_ij    = pos(:,x23) - pos(:,x11); % Distance vector
        r3      = diag(1./sqrt(sum(r_ij.^2)).^3); % |r_ij|^3
        mu_diag = diag(mu(x23)); % Diagnal Matrix of -G*m_j (Std. Grav Param)
        acc_RHS = r_ij*r3*mu_diag; % Acceleration terms on RHS
        acc_Sum = sum(reshape(acc_RHS(:,x12_34(:)),3*N,N-1),2); % Sum of acceleration on RHS
        ss_vec = [xv(3*N+1:end);acc_Sum]; % state-space vector
    end
end
```

Solar System State Vectors

```
function [p0, v0, mu, n, scale, cA, tob, bvec, pBary, M] = StateVecInit

M    =   [ 1988500, 0.33011, 4.8675, 5.9723, 0.07346, 0.64171, 1898.19, 568.34, 86.813, 102.413, 0.01303 ]'
* 1e24;
scale = [695700, 2439.7, 6051.8, 6371.008, 1737.4, 3389.5, 69911, 58232, 25362, 24622, 1187];

% Time: 1945-Jan-1 0:0:0 (Ref Sun)
p0 = [ -6.514853452736166E+04    6.923075832509800E+05    2.969644123669676E+04; % Sun
       -4.090224402811901E+07    3.092971988307618E+07    6.251586293432735E+06; % Mercury
        8.649626519565648E+07    6.558753100815241E+07   -4.093465503005635E+06; % Venus
       -2.812655902373333E+07    1.450840122031818E+08    4.736565603273362E+04; % Earth
       -2.837642934184081E+07    1.453856908325931E+08    6.006681243879348E+04; % Moon
       -4.575332432237922E+07   -2.160750372389238E+08   -3.379480787481025E+06; % Mars
       -7.937071694877126E+08    1.708772517514482E+08    1.711521054427497E+07; % Jupiter
       -1.923795039832259E+08    1.337735118153997E+09   -1.574714310247427E+07; % Saturn
        8.814142859265038E+08    2.748130246512144E+09   -1.196667562167048E+06; % Uranus
       -4.507715302467741E+09   -4.203358655158987E+08    1.125516845885302E+08; % Neptune
       -3.580075486462851E+09    4.304575097306495E+09    5.745302970305955E+08];% Pluto

v0 = [ -1.970463223071625E+00   -1.955899360310175E-01    2.369174320448657E-01; % Sun
       -4.089492060880693E+01   -3.731269883404487E+01    7.856996007483481E-01; % Mercury
       -2.308493963878707E+01    2.767558092658806E+01    1.833208420054525E+00; % Venus
       -3.168839597240482E+01   -5.981403599180973E+00    2.348307120914044E-01; % Earth
       -3.248928443141775E+01   -6.584149383830741E+00    3.229759698734869E-01; % Moon
        2.266216276795534E+01   -3.115518389430809E+00   -4.325189489577113E-01; % Mars
       -4.873042121510281E+00   -1.236602989967739E+01    3.519065081987538E-01; % Jupiter
       -1.205496403155248E+01   -1.599677934190488E+00    6.612639993819801E-01; % Saturn
       -8.512076610641467E+00    1.557251793473381E+00    3.284969194625559E-01; % Uranus
       -1.500221047078075E+00   -5.585955645339683E+00    3.363419003462520E-01; % Neptune
       -4.659388787869598E+00   -4.189821186825673E+00    1.441438596050844E+00];% Pluto

tob = []; % Time of Burn
```

```
bvec = []; % Burn Vector

% Position of Barycenter in reference frame
pBary = [-1.070183387116344E+06 1.335013302913338E+06 4.721272103390069E+04];
vBary = [-1.976510629978190E+00 -2.077526912802563E-01  2.371613343498015E-01];

G    =   6.67259e-20; % [km^3/kg-s^2] gravitational constant
mu   =   M * G; % Calculates Grav Parameter
n    =   size( M, 1 ); %
cA = 1:length(M); % Which objects to simulate

% Reshape P0 and V0 for subtraction of Barycenter state vectors
p0 = reshape(p0',[1,3*n]);
v0 = reshape(v0',[1,3*n]);

% reshape vectors into arrays
for i = 1:size(p0,2)/3
    p0(1+(i-1)*3:3+(i-1)*3) = p0(1+(i-1)*3:3+(i-1)*3) - pBary;
    v0(1+(i-1)*3:3+(i-1)*3) = v0(1+(i-1)*3:3+(i-1)*3) - vBary;
end

% Reshape p0 and v0 into vertical arrays.
p0 = reshape(p0,n*3,1);
v0 = reshape(v0,n*3,1);
end
```

Apollo 10, 11, 12 Coordinates

```
function [ p0, v0, mu, n, scale, cA, tob, bvec] = ApolloCoords( ApNum )

% Apollo 10 date: 1969-MAY-18 19:44:21.9965
% Apollo 11 date: 1969-07-16 16:40:02.7475
% Apollo 12 date: 1969-NOV-14 19:32:44.9606

G    =   6.67259e-20; % [km^3/kg-s^2] gravitational constant
fps = 0.0003048;

M_cent       = 5.9724  * 1e24;      % [kg] central body mass
M_moon       = 0.07346 * 1e24*1;    % [kg] central body mass
M_sat        = 721.9;               % [kg] satellite mass (voyager 2)

M = [ M_cent; M_moon; M_sat];       % [kg] mass vector
mu = M*G;                           % [km^3/s^2]gravitational parameter vector
n = size(M,1);                      % number of elements/bodies to calculate
scale = [6371.008, 1737.4, 500];
cA = [4 5 1];

switch ApNum
    case 10
        p0 = [
            -7.938423193298528E+07 -1.281468857961604E+08 -7.148663508873433E+04;
            -7.935307206546827E+07 -1.277448075089262E+08 -3.581297047458589E+04;
            -7.937498435283971E+07 -1.281425552209317E+08 -7.014042292705178E+04];

        v0 = [
            2.380344596873252E+01 -1.414320514423730E+01 1.774069620155609E-01;
            2.283432883606584E+01 -1.405442902850622E+01 1.834657257489427E-01;
            2.543573433098150E+01 -5.677983640890194E+00  1.495517849936535E+00];

        pMinus = [-7.938385332238917E+07 -1.281420003100482E+08 -7.105317922063172E+04];
        vMinus = [2.379167062934461E+01 -1.414212646256195E+01 1.774805795348753E-01];
        tob = []; bvec = [];
    case 11

        p0 = [
            6.218930309131721E+07 -1.381040976798286E+08 7.338742270641029E+04;
            6.189261520200559E+07 -1.378311813866535E+08 9.427421511062980E+04;
            6.219089189323087E+07 -1.380934578742084E+08  7.489060787452012E+04];

        v0 = [
```

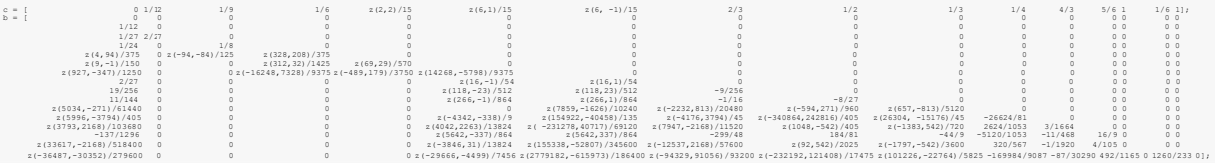
```
            2.554414977285855E+01  1.054079699013750E+01 8.710014369007490E-02;
            2.489893178548856E+01  9.810915868269419E+00 1.572353597697207E-02;
            1.974980780257676E+01 1.668804564195570E+01  8.599988484918057E-01];

        pMinus = [6.218569816011626E+07 -1.381007815874097E+08 7.364120943764597E+04];
        vMinus = [2.553630999733053E+01 1.053192850805920E+01 8.623287620315478E-02];

        % [hr min sec burnTime[hr min sec]
        b1 = ApBurn([ 75 54 28.4 0 5 58.9 4 44 44.9])+.5-.012;
        b2 = ApBurn([135 24 33.8 0 2 29.4 4 44 44.9])+.5-.012-1.33;
        tob = [b1, b2]*3600;
        bvec = [-2891.8 -433.1   20.4;
                 3213.3 705.0 -138.8]*fps;


    case 12


        p0 = [
            -6.217812098137438E+03, 1.296452154958064E+03, -5.819416748518005E+02;
            1.683955442344367E+05, -3.213509647196801E+05, -2.585085769468364E+04;
            -8.803658700360249E+03, -4.821380113666855E+03,  8.037374643977058E+03];

        v0 = [
            -1.036172498994042E-01, -4.161119718400264E-01, 1.800924936869037E-01;
            8.429580735560637E-01, 7.043707572870070E-02,  2.381643866208534E-01;
            4.959673801548565E+00, -5.256147488411352E+00,  4.908245942836967E+00];

        pMinus = [-4.096157797361373E+03, -2.623902473255802E+03, -8.889737683608131E+02];
        vMinus = [-9.211580666391250E-02, -4.102001166360834E-01, 1.807981011157126E-01];

        tob = [];
        bvec = [];


end

p0 = reshape(p0',[1,3*n]);
v0 = reshape(v0',[1,3*n]);

for i = 1:size(p0,2)/3
    p0(1+(i-1)*3:3+(i-1)*3) = p0(1+(i-1)*3:3+(i-1)*3) - pMinus;
    v0(1+(i-1)*3:3+(i-1)*3) = v0(1+(i-1)*3:3+(i-1)*3) - vMinus;
end

p0 = reshape(p0,n*3,1);
v0 = reshape(v0,n*3,1);
end
```

Lagrangia Point Simulation

```
function [ p0, v0, mu, n, scale, cA, bt] = LagrangeP( ApNum)
% Lagrangian point 5

G    =   6.67259e-20; % [km^3/kg-s^2] gravitational constant

M_cent       = 5.9724  * 1e24;     % [kg] central body mass
M_moon       = 0.07346 * 1e24;     % [kg] central body mass
M_sat        = 721.9;              % [kg] satellite mass (voyager 2)

M = [ M_cent; M_moon; M_sat]; % [kg] mass vector
mu = M*G; % [km^3/s^2]gravitational parameter vector
n = size(M,1); % number of elements/bodies to calculate
scale = [6371.008, 1737.4, 500];
cA = [4 5 1];

switch ApNum
```

```
    case 5
        p0 = [
            -6.217816180618011E+03 1.296435760194887E+03 -5.819345792284226E+02;
            1.683955774470052E+05 -3.213509619444179E+05 -2.585084831102862E+04;
            -8.803463283383866E+03 -4.821587199650016E+03  8.037568025370150E+03];

        v0 = [
            -1.036159472140258E-01 -4.161122434560039E-01 1.800926156082740E-01;
            8.429593196935455E-01 7.043690720315035E-02  2.381645166614605E-01;
            4.959706422015502E+00 -5.256073099243214E+00  4.908140946707825E+00];

        pMinus = [-4.096161426685309E+03 -2.623918635091969E+03 -8.889666449365138E+02];
        vMinus = [-9.211450466562407E-02 -4.102003869994519E-01 1.807982231357364E-01];
end

p0 = reshape(p0',[1,3*n]);
v0 = reshape(v0',[1,3*n]);

for i = 1:size(p0,2)/3
    p0(1+(i-1)*3:3+(i-1)*3) = p0(1+(i-1)*3:3+(i-1)*3) - pMinus;
    v0(1+(i-1)*3:3+(i-1)*3) = v0(1+(i-1)*3:3+(i-1)*3) - vMinus;
end

switch ApNum
    case 5
        p0(7:9) = rot(cross(p0(4:6),v0(4:6)),-pi/3)*p0(4:6)';
        v0(7:9) = rot(cross(p0(4:6),v0(4:6)),-pi/3)*v0(4:6)';
end

p0 = reshape(p0,n*3,1);
v0 = reshape(v0,n*3,1);
bt = [];
end
```

```
function [ tout, yout ] = rk89( ode_functions, tspan, init, tol )
```



```
c8 = [103/1680 0 0 0 0 0 0  -27/140 76/105 -201/280  1024/1365   3/7280  12/35 9/280    0         0];
c9 = [  23/525 0 0 0 0 0 0 0 171/1400 86/525   93/280 -2048/6825 -3/18200 39/175     0 9/25 233/4200];

if nargin < 4
    tol = 1.e-14;
end

t0 = tspan(1);
tf = tspan(2);
t = t0;
y = init;

tout = t;
yout = y';
h = (tf-t0/100);

while t < tf
    hmin = 16*eps(t);
    ti = t;
    yi = y;
    for i = 1:size(b,1)
        t_inner = ti + c(i)*h;
        y_inner = yi;
        for j = 1:i-1
            y_inner = y_inner + h*b(i,j)*f(:,j);
        end
        f(:,i) = feval(ode_functions, t_inner, y_inner);
    end


    te_max = max(abs(h*f*(c8' - c9')));


    te_allowed = tol*max(max(abs(y)),1.0);


    delta = (te_allowed/(te_max + eps))^(1/9);

    if te_max <= te_allowed
        h      = min(h, tf-t);
        t      = t + h;
        y      = yi + h*f*c9';
        tout   = [tout;t];
        yout = [yout;y'];
    end


    h = min(delta*h, 4*h);
    if h < hmin
        fprintf(['\n\n Warning: Step size fell below its minimum\n'...
                 ' allowable value (%g) at time %g.\n\n'], hmin, t)
        return
    end
end
end
```

```
function [val] = z(a,b)
val = (a+b*sqrt(6));
end
```

Runge-Kutta 86

```
function [tout, yout ,dyout,iaccept,ireject] = rkn86(FunFcn, t0, tfinal, y0, dy0, tol);
%   rkn86 Integrates a special system of ordinary differential equations using
%   an effectivelly 8-stages Runge-Kutta-Nystrom pair of orders 8 and 6.
%
%        [T,Y,DY,IA,IR] = rkn86('yprime', T0, Tfinal, Y0, DY0) integrates the special system
%        of second order ordinary differential equations of the form:
%
%   y''=f(t,y), y(t0)=y0, y'(t0)=y'0
%
%   described by the M-file YPRIME.M over the interval T0 to Tfinal.
%
%        [T,Y,DY,IA,IR] = rkn86(F, T0, Tfinal, Y0, DY0, TOL) uses tolerance TOL
%
% INPUT:
% F     - String containing name of user-supplied problem description.
%         Call: yprime = fun(t,y) where F = 'fun'.
%         t      - Time (scalar).
%         y      - Solution column-vector.
%         yprime - Returned derivative column-vector; yprime(i) = d^2y(i)/dt^2.
% t0    - Initial value of t.
% tfinal- Final value of t.
% y0    - Initial value column-vector.
% dy0   - Initial derivatives column vector
% tol   - The desired accuracy. (Default: tol = 1.e-6).
%
% OUTPUT:
% T       - Returned integration time points (row-vector).
% Y       - Returned solution, one solution column-vector per tout-value.
% DY      - Returned derivative solution,
% Iaccept - Returned number of accepted steps
% Ireject - Returned number of rejected steps
%
% The result can be displayed by: plot(tout, yout).
%
% Example: Solve two-body problem using inline
% the problem :
%              y1''=-y1/(y1^2+y2^2)^1.5, y2''=-y2/(y1^2+y2^2)^1.5
%              Initial contitions y1(0)=.5, y2(0)=0, y1'(0)=0, y2'(0)=3^0.5
% Matlab call :
%              [x,y]=rkn86(inline('[-y(1)/sqrt(y(1)^2+y(2)^2)^3;-
y(2)/sqrt(y(1)^2+y(2)^2)^3]','x','y'), ...
%              0, 20, [.5 0]',[0 sqrt(3)]', 1e-11);
% write       : plot(y(:,1),y(:,2),'-k'); % to get the elliptic orbit
%
% based on the code ODE86 by Ch. Tsitouras
%
% The coefficients of the Runge-Kutta-Nystrom pair NEW8(6) are taken from
% S. N. Papakostas and Ch. Tsitouras, "High phase-lag order Runge-Kutta and Nystrom pairs",
% SIAM J. Sci. Comput. 21(1999) 747-763.
%
% The error control is based on
% Ch. Tsitouras and S. N. Papakostas, "Cheap Error Estimation for Runge-Kutta
% methods", SIAM J. Sci. Comput. 20(1999) 2067-2088.

% Matlab version : 6.1
% Author : Ch. Tsitouras, 1996-2003.
% URL address: http://users.ntua.gr/tsitoura/
%-------------------------------------------------------------------------
% the coefficients
alpha=[0 6397/98811 12794/98811 14/37 8/13 17/22 43/46 1 1]';
```

```
beta=[[0 0 0 0 0 0 0 0]
[21738209/10373173531 0 0 0 0 0 0 0]
[81843218/29290841163 82694821/14797810534 0 0 0 0 0 0]
[286557584/4330809711 -912003620/7090326959 2215175292/16525689869 0 0 0 0 0 0]
[-1732991908/3477246155 20699018807/16215676961 -8943798416/12438207277 711229321/5458138039 0 0
0 0 0]
[10259024870/9108477419 -25149249362/9340973033 4686267579/2513053636 -326162972/7839732939
556579829/13434269006 0 0 0 0]
[-32801447959/18176875798 31592171746/6958893399 -111550006196/40089394711 3451154231/7987305225
68790340/8728368029 123716081/2797556961 0 0 0]
[62469663917/6900212338 -171339392336/7672895439 262962495363/17824923050 -
22108842829/16963055973 661764535/1698709821 -238225641/2934789434 260644226/13286668711 0 0]
[257873323/6918876884 0 1503948753/8413843957 2236434251/13285504895 1069201912/15512587877
980034039/25364950097 92941557/11497613663 0 0]]';

gamma=[[257873323/6918876884 0 1503948753/8413843957 2236434251/13285504895
1069201912/15512587877 980034039/25364950097 92941557/11497613663 0 0]
[-108540447/9734693747 0 216990433/7248923167 -693180867/13981264399 783383731/11490287817 -
639183288/13156494967 143178476/12783609495 0 0]]';

dgamma=[[257873323/6918876884 0 1885846298/9184313637 10010095879/36964622736
576314810/3215962383 378512797/2226489968 1523915682/12294818705 13956454/1038655275 0]
[-108540447/9734693747 0 300730357/8745591283 -339555838/4257328827 4673474889/26364705520 -
1278366576/5980224985 1108173697/6452782413 411153357/5767449338 -3/20]*3]';
%-------------------------------------------------------------------------
ireject=0;iaccept=0;
pow = 1/8;
if nargin < 6, tol = 1.e-6; end

% Initialization
t=t0;
y=y0;
dy=dy0;
tout = t0(:)';
yout = y0(:)';
dyout = dy0(:)';
hmax = (tfinal - t)/1;
hmin = (tfinal - t)/100000000;
f = y0*zeros(1,length(alpha));

% initial step
f(:,1) = feval(FunFcn,t,y);
h=tol^pow/max(max(abs([dy' f(:,1)'])),1e-2);
h=min(hmax,max(h,hmin));

% The main loop
   while (t < tfinal) & (h >= hmin)
      if t + h > tfinal, h = tfinal - t; end

      % Compute the  slopes
      for j = 1:length(alpha),
         f(:,j) = feval(FunFcn, t+alpha(j)*h,y+alpha(j)*h*dy+h^2*f*beta(:,j));
      end

      % Estimate the error and the acceptable error
      delta1 = max(abs(h^2*f*gamma(:,2)));
      delta2 = max(abs(h*f*dgamma(:,2)));
      delta=max(delta1,delta2)*h;

      % Update the solution only if the error is acceptable
      if delta <= tol,
         t = t + h;
         y = y + h*dy+h^2*f*gamma(:,1);
         dy = dy +h*f*dgamma(:,1);
         iaccept=iaccept+1;
         tout=[tout; t];
         yout=[yout;y'];
         dyout=[dyout;dy'];
      else
         ireject=ireject+1;
```

```
      end

      if delta ~= 0.0
         h = min(hmax, .9*h*(tol/delta)^pow);
      end
   end;

   if (t < tfinal)
      disp('SINGULARITY LIKELY.')
   end
```

Set Color Function

```
    function [color] = SetColor(BodyN)
        color = [
            249, 219, 026; % sun      1
            122, 120, 122; % mercury  2
            175, 107, 031; % venus    3
            079, 082, 115; % earth    4
            172, 159, 158; % moon     5
            140, 083, 063; % mars     6
            179, 166, 151; % jupiter  7
            206, 172, 117; % saturn   8
            185, 222, 226; % uranus   9
            059, 089, 214; % neptune  10
            171, 135, 112; % pluto    11
            249, 219, 026; % Sat1     12
            185, 222, 226; % Sat2     13
            ]/255;
        if BodyN > size(color,1)
            r = randi([0 255],BodyN-size(color,1),3)/255;
            color = vertcat(color, r);
        end
    end
```

Rotation Matrix Function

```
function R= rot(V,theta)
%This function returns the 3D rotation matrix about an arbitary vector V
%passing the origin.

V=V/norm(V);

R=[V(1)^2+(1-V(1)^2)*cos(theta),           V(1)*V(2)*(1-cos(theta))-V(3)*sin(theta), V(1)*V(3)*(1-cos(theta))+V(2)*sin(theta);
   V(1)*V(2)*(1-cos(theta))+V(3)*sin(theta),V(2)^2+(1-V(2)^2)*cos(theta),             V(2)*V(3)*(1-cos(theta))-V(1)*sin(theta);
   V(1)*V(3)*(1-cos(theta))-V(2)*sin(theta), V(2)*V(3)*(1-cos(theta))+V(1)*sin(theta), V(3)^2+(1-V(3)^2)*cos(theta)];

end
```

References

[1]Wie, B. (1998). Space vehicle dynamics and control. Aiaa.

[2]Orloff, R., & Garber, S. (2000). Apollo by the numbers: a statistical reference.

[3]Curtis, H. D. (2013). *Orbital mechanics for engineering students*. Butterworth-Heinemann.*Books*

[4]Haber, L., Haber, R. N., Penningroth, S., Novak, K., & Radgowski, H. (1993). Comparison of nine methods of indicating the direction to objects: Data from blind adults. Perception, 22(1), 35-47.

[5]Betts, J. T. (1977). Optimal three-burn orbit transfer. AIAA Journal, 15(6), 861-864.

[6]Vallado, D. A. (2001). Fundamentals of astrodynamics and applications (Vol. 12). Springer Science & Business Media.

[7]Battin, R. H. (1999). An introduction to the mathematics and methods of astrodynamics. Aiaa.

[8]Bate, R. R., Mueller, D. D., & White, J. E. (1971). Fundamentals of astrodynamics. Courier Corporation.

[9]Musielak, Z. E., & Quarles, B. (2014). The three-body problem. Reports on Progress in Physics, 77(6), 065901.

[10]Chobotov, V. A. (2002). Orbital mechanics. Aiaa.

[11]Ahmad, A., & Cohen, L. (1973). A numerical integration scheme for the N-body gravitational problem. Journal of Computational Physics, 12(3), 389-402.