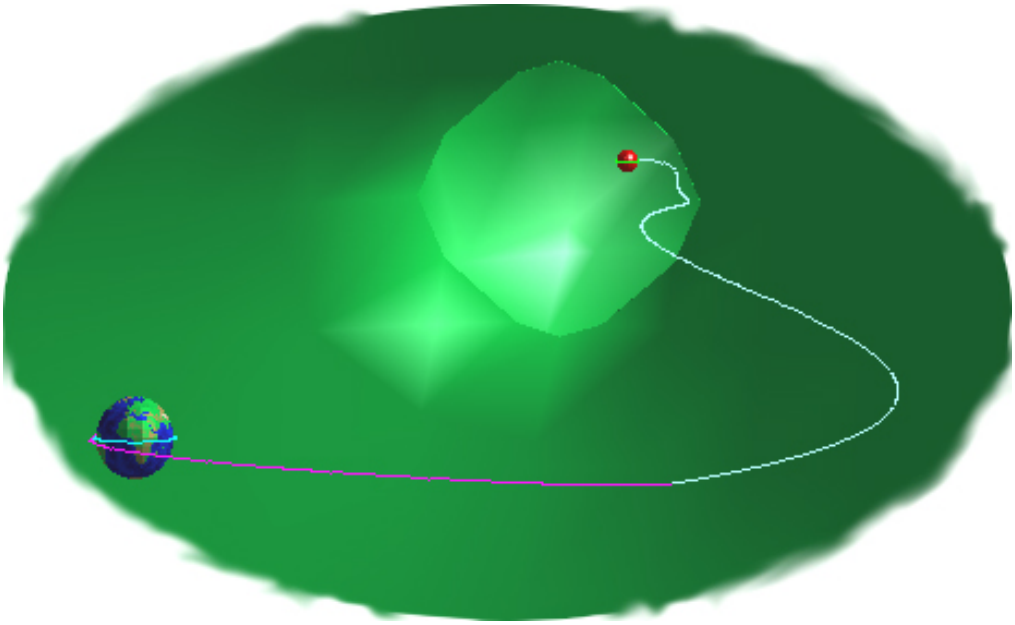
Restricted Three-Body Problem

Kevin Daugherty

Fall 2004



Contents

Chapter 1

Introduction

The Fall 2004 independent study covered Three-Body orbit problems. The faculty advisor for this study was Dr. Hanspeter Schaub. Regular meetings and progress reports, in addition to homework assignments made up the grade for this independent study. The three-body problem topics included:

- Lagrange's Three-Body Solution
  - Equilateral Triangle Arrangement
  - Collinear Arrangement
  - Circular Orbits
- Circular Restricted Three-Body Problem
  - The Jacobi Integral
  - Zero Relative Velocity Surfaces
  - Lagrange Libration Point Stability

The information on these topics can be found in:

> H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*, AIAA Education Series, Reston, VA, 2003.

This report contains the individual reports and MATLAB code written during the semester, beginning with a "cookbook" explaining how to find the initial setup and conditions for the only stable arrangements of 3 massive bodies.

Chapter 2

"Cookbook" For Finding the Initial Setup and Conditions

2.1 Finding the Locations and/or Masses

The first step to solving a three body problem is to figure out where all the masses are at an initial time. Since there are only two possible types of configurations which will work, that makes things a little bit easier. The two possible ways, as shown on page 428 in the book, are equilateral triangle and collinear arrangements.

2.1.1 Equilateral Triangle

For an equilateral triangle, all three masses must be given the position of at least 2 of the masses, and the velocity vector of one of these. The shape of the initial setup does not depend on the the mass of the bodies, only on the positions. Given 2 of the positions relative to some inertial origin ($\boldsymbol{x}_1$ and $\boldsymbol{x}_2$), we can find the distance between these points, and therefore the length of each side of the equilateral triangle, ρ, by first finding the vector r_{12} pointing from body 1 to body 2.

$$\boldsymbol{r}_{12} = \boldsymbol{x}_2 - \boldsymbol{x}_1 \tag{2.1}$$

This is the vector along the one known side of the triangle. The length of this vector is ρ; the length of the sides of the equilateral triangle.

$$\rho = \sqrt{\boldsymbol{r}_{12} \cdot \boldsymbol{r}_{12}} \tag{2.2}$$

Since the angle between the sides of the triangle are known to be 60 degrees, it is easiest to rotate the vector pointing from body 1 to body 2 by 60 degrees about an axis perpendicular to the plane of motion ($\hat{\boldsymbol{\imath}}_h$) to get $\boldsymbol{r}_{13}$. In order to find this vector we must be given either a velocity of one of the bodies, or a vector which is known to either lie in, or perpendicular to the plane of of motion. Given a vector perpendicular to the plane of motion, $\boldsymbol{hvec}$, the unit vector $\hat{\boldsymbol{\imath}}_h$ is easily found by dividing the vector by its magnitude

$$\hat{\boldsymbol{\imath}}_h = \frac{\boldsymbol{hvec}}{\sqrt{\boldsymbol{hvec} \cdot \boldsymbol{hvec}}} \tag{2.3}$$

Finding $\hat{\boldsymbol{\imath}}_h$ from the velocity vector or vector in the plane of motion, and positions can be done with a simple cross product, since cross products yield vectors orthogonal to both of the vectors.

$$\hat{\boldsymbol{\imath}}_h = \frac{\boldsymbol{r}_{12} \times \boldsymbol{v}_1}{|\boldsymbol{r}_{12} \times \boldsymbol{v}_1|} \tag{2.4}$$

Remember that the magnitude of the cross product can be found by taking the result of the cross product, dotting it with itself and then taking the square root:

$$|\boldsymbol{r}_{12} \times \boldsymbol{v}_1| = \sqrt{(\boldsymbol{r}_{12} \times \boldsymbol{v}_1) \cdot (\boldsymbol{r}_{12} \times \boldsymbol{v}_1)} \tag{2.5}$$

Now that the $\hat{\boldsymbol{\imath}}_h$ vector is known we can also find $\hat{\boldsymbol{\imath}}_{r_{12}}$ and $\hat{\boldsymbol{\imath}}_\theta$. The goal is to break the vector up into this new coordinate frame and then rotate it by 60 degrees around $\hat{\boldsymbol{\imath}}_h$ to find the $\boldsymbol{r}_{13}$ vector, as seen in Figure 2.1. This new $\boldsymbol{r}_{13}$ gives the position of the third body relative to the first body.

The unit vector pointing from body 1 to body 2, $\hat{\boldsymbol{\imath}}_{r_{12}}$, is easily found via:

$$\hat{\boldsymbol{\imath}}_{r_{12}} = \frac{\boldsymbol{r}_{12}}{|\boldsymbol{r}_{12}|} \tag{2.6}$$

Then since we have 2 of the unit vectors for the coordinate frame, the third ($\hat{\boldsymbol{\imath}}_\theta$) can be found using

$$\hat{\boldsymbol{\imath}}_\theta = \hat{\boldsymbol{\imath}}_h \times \hat{\boldsymbol{\imath}}_{r_{12}} \tag{2.7}$$

The $\boldsymbol{r}_{12}$ vector can be written in this new coordinate frame as

$$\boldsymbol{r}_{12} = \rho \hat{\boldsymbol{\imath}}_{r_{12}} \tag{2.8}$$

Using the definition for a rotation matrix which rotates a vector by an angle θ around the third axis found as Equation (3.31c) in the book also shown

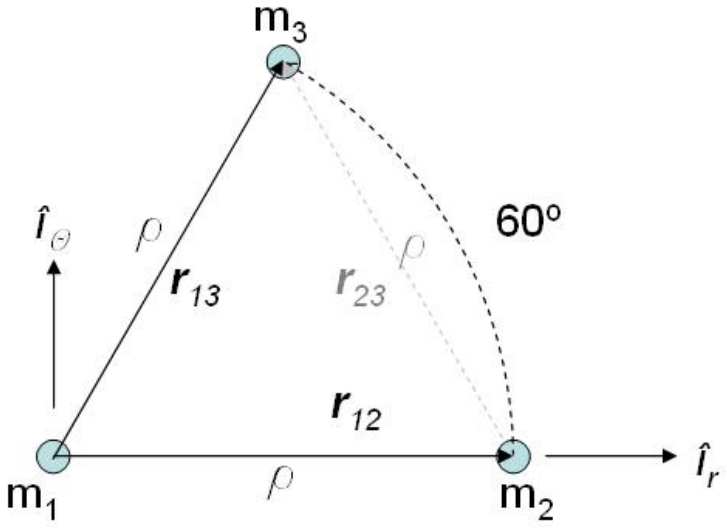


Figure 2.1: Rotating $\boldsymbol{r}_{12}$ by 60° around $\hat{\boldsymbol{\imath}}_h$ to get $\boldsymbol{r}_{13}$

here

$$M_3(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

In this coordinate frame r_{13} can be found using this rotation matrix using

$$\boldsymbol{r}_{13} = [M_3(\theta)]\boldsymbol{r}_{12} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \rho \\ 0 \\ 0 \end{bmatrix} \tag{2.10}$$

Using $\theta = \pm 60$ and $\boldsymbol{r}_{12}$ in this coordinate frame, we get a convenient equation for $\boldsymbol{r}_{13}$

$$\boldsymbol{r}_{13} = 1/2\rho\hat{\boldsymbol{\imath}}_{r_{12}} \mp \sqrt{3}/2\rho\hat{\boldsymbol{\imath}}_{\theta} \tag{2.11}$$

This equation gives both of the possible vectors from the first body to the third body. For simplicity only the + is being used from here out, either can be chosen.

Replacing $\hat{\boldsymbol{\imath}}_{r_{12}}$ and $\hat{\boldsymbol{\imath}}_{\theta}$ with the vector components in the original cartesian coordinate frame allows us to find the position of the third body using the equation:

$$\boldsymbol{x}_3 = \boldsymbol{x}_1 + \boldsymbol{r}_{13} \tag{2.12}$$

We now have all the initial positions and masses.

2.1.2 Collinear

If all the masses lie in a single line, then there are two possible sets of givens. These are:

- All 3 masses are known
- The locations of 2 of the masses are known
- The location of the third is known relative to the first two...i.e. between the masses, beyond mass 1, or beyond mass 2.

or

- 2 of the masses are known
- The locations of all three masses are known

The methods for finding all the initial position and mass information is therefore broken up into two categories

2.1.2.1 All 3 masses known

To find the location of the third mass, change the numbering of the masses so that mass 1 is on one end, mass 2 is in the middle, and mass 3 is on the other end. Then use Lagrange's quintic equation, Eq. (10.52) from the book to find χ:

$$(m_1+m_2)\chi^5+(3m_1+2m_2)\chi^4+(3m_1+m_2)\chi^3 - (m_2+3m_3)\chi^2-(2m_2+3m_3)\chi-(m_2+m_3)=0 \quad (2.13)$$

Unfortunately there is no real easy way to analytically solve Lagrange's quintic equation. The best way to solve for χ is to use either a solver in a calculator or MATLAB. `sovequintic()` described in Section 6.6 can be used to solve Lagrange's quintic equation for χ.

Given χ you can now use either Eq. (10.46) or Eq. (10.47) from the book to find the relationship controlling the distances between the bodies. Each equation is easier to use for certain cases depending on which distances are already known. A combination of the following equations will give the location of the desired body for each of the 3 cases as described below the equations.

$$\boldsymbol{x}_{12}=\frac{\boldsymbol{x}_{23}}{\chi} \quad (2.14a)$$

$$\boldsymbol{x}_{12} = \frac{\boldsymbol{x}_{13}}{1+\chi} \tag{2.14b}$$

$$\boldsymbol{x}_{13} = \frac{1+\chi}{\chi}\boldsymbol{x}_{23} \tag{2.14c}$$

then use

$$\boldsymbol{x}_1 = \boldsymbol{x}_2 - \boldsymbol{x}_{12} \tag{2.15a}$$

$$\boldsymbol{x}_2 = \boldsymbol{x}_1 + \boldsymbol{x}_{12} \tag{2.15b}$$

$$\boldsymbol{x}_3 = \boldsymbol{x}_1 + \boldsymbol{x}_{13} \tag{2.15c}$$

To find the location of body 1, 2 or 3 use Eqs. 2.14a and 2.15a, 2.14b and 2.15b, or 2.14c and 2.15c respectively.

2.1.2.2 All 3 positions known

With all three positions known and 2 of the masses. Make sure the bodies are numbered with 1 on one end, 2 in the middle, and 3 on the other end. Use Eq. (10.46) to find χ.

$$\chi = \frac{|\boldsymbol{x}_3 - \boldsymbol{x}_2|}{|\boldsymbol{x}_2 - \boldsymbol{x}_1|} \tag{2.16}$$

then use Lagrange's quintic equation Eq. (10.52)

$$\begin{aligned}(m_1+m_2)\chi^5 + (3m_1+2m_2)\chi^4 + (3m_1+m_2)\chi^3 \\ - (m_2+3m_3)\chi^2 - (2m_2+3m_3)\chi - (m_2+m_3) = 0\end{aligned} \tag{2.17}$$

to find the unknown mass. It is possible to solve the quintic equation analytically for each of the masses as shown below.

$$m_1 = \frac{(3\chi^2+3\chi+1)m_3 - (\chi^5+2\chi^4+\chi^3-\chi^2-2\chi-1)m_2}{\chi^5+3\chi^4+3\chi^3} \tag{2.18a}$$

$$m_2 = \frac{(3\chi^2+3\chi+1)m_3 - (\chi^5+3\chi^4+3\chi^3)m_1}{\chi^5+2\chi^4+\chi^3-\chi^2-2\chi-1} \tag{2.18b}$$

$$m_3 = \frac{(\chi^5+3\chi^4+3\chi^3)m_1 + (\chi^5+2\chi^4+\chi^3-\chi^2-2\chi-1)m_2}{3\chi^2+3\chi+1} \tag{2.18c}$$

or

$$m_1 = \frac{c_3 m_3 - c_2 m_2}{c_3} \tag{2.19a}$$

$$m_2 = \frac{c_3 m_3 - c_1 m_1}{c_2} \tag{2.19b}$$

$$m_3 = \frac{c_1 m_1 + c_2 m_2}{c_3} \tag{2.19c}$$

where

$$c_1 = \chi^5 + 3\chi^4 + 3\chi^3 \tag{2.20a}$$

$$c_2 = \chi^5 + 2\chi^4 + \chi^3 - \chi^2 - 2\chi - 1 \tag{2.20b}$$

$$c_3 = 3\chi^2 + 3\chi + 1 \tag{2.20c}$$

Now the final mass may be determined so that all three positions and masses are known. Once this has been accomplished, the next step is to find the initial velocities

2.2 Finding the initial velocities

The locations and sizes of each of the bodies make up half of the initial conditions. The velocity of each of the bodies must be found in order to begin simulating the motion. As shown later in the analysis

$$\dot{\boldsymbol{r}}_i(0) = r_{i_0}\dot{f}_0\hat{\boldsymbol{\imath}}_{r_i} + r_{i_0}\omega\hat{\boldsymbol{\imath}}_{\theta_i}$$

so if we can find $\dot{f}$ and ω we can find the initial velocity vectors for each of the bodies

2.2.1 Finding $\dot{f}$ and ω

The method for finding $\dot{f}$ and ω is different when dealing with circular versus non-circular orbits. For non-circular orbits, a velocity vector of one of the bodies must be given in order to find the others, but in the case of circular orbits, none of the velocities are required initially.

2.2.1.1 Non-Circular Orbits

The book explains and proves that each of the bodies in the system must be rotating at the same speed as each other around the center of mass in order to remain in the same shape. This means that the omega vector for each of the bodies must be the same, so the key to finding the initial velocities is

$$\boldsymbol{\omega}_1 = \boldsymbol{\omega}_2 = \boldsymbol{\omega}_3 = \omega\hat{\boldsymbol{\imath}}_3 \tag{2.21}$$

and as the book points out in Equations (10.12-14) the position or any of the bodies can be related to its initial position through a common scalar factor f.

$$r_1(t) = r_{1_0} f(t) \tag{2.22a}$$

$$r_2(t) = r_{2_0} f(t) \tag{2.22b}$$

$$r_3(t) = r_{3_0} f(t) \tag{2.22c}$$

Because the initial position of each body is a constant, and f varies with time, taking the inertial derivative yields

$$\dot{r}_i(t) = r_{i_0} \dot{f}(t) \tag{2.23}$$

so $\dot{f}$ must also be the same for all the bodies. Using Equations (10.17) and (10.18) shown here:

$$\boldsymbol{r}_i = r_i \hat{\boldsymbol{\imath}}_{r_i} \tag{2.24}$$

$$\dot{\boldsymbol{r}}_i = \dot{r}_i \hat{\boldsymbol{\imath}}_{r_i} + r_i \omega \hat{\boldsymbol{\imath}}_{\theta_i} \tag{2.25}$$

Define m_i to be the mass for which you know the velocity of in the general case and using Eq 2.23

$$\dot{\boldsymbol{r}}_i = r_{i_0} \dot{f} \hat{\boldsymbol{\imath}}_{r_i} + r_{i_0} f \omega \hat{\boldsymbol{\imath}}_{\theta_i} \tag{2.26}$$

At $t = 0$, $f = 1$ since the initial conditions are the conditions at time $t = 0$. We would like to find a simple expression for the initial $\dot{f}$ so that we can plug it back into the velocity equation for each of the other bodies. Since $\dot{f}$ is the same for all the bodies, it can be found using the known velocity.

$$\dot{\boldsymbol{r}}_i(0) = r_{i_0} \dot{f}_0 \hat{\boldsymbol{\imath}}_{r_i} + r_{i_0} \omega \hat{\boldsymbol{\imath}}_{\theta_i} \tag{2.27}$$

dividing through by r_{i_0} will help to further isolate $\dot{f}$ and give a simplified equation.

$$\frac{\dot{\boldsymbol{r}}_{i_0}}{r_{i_0}} = \dot{f}_0 \hat{\boldsymbol{\imath}}_{r_i} + \omega \hat{\boldsymbol{\imath}}_{\theta_i} \tag{2.28}$$

$\dot{f}$ is now the only term in the $\hat{\boldsymbol{\imath}}_{r_i}$ direction, and if we find a way to both remove the $\hat{\boldsymbol{\imath}}_{\theta_i}$ components and scalarize the equation, we can find $\dot{f}$. This is best accomplished by dotting both sides of the equation by $\hat{\boldsymbol{\imath}}_{r_i}$

$$\left(\frac{\dot{\boldsymbol{r}}_{i_0}}{r_{i_0}} \right) \bullet \hat{\boldsymbol{\imath}}_{r_{i_0}} = (\dot{f} \hat{\boldsymbol{\imath}}_{r_i} + \omega \hat{\boldsymbol{\imath}}_{\theta_i}) \bullet \hat{\boldsymbol{\imath}}_{r_{i_0}} \tag{2.29}$$

since $\hat{\boldsymbol{\imath}}_{r_i} \bullet \hat{\boldsymbol{\imath}}_{r_i} = 1$ and $\hat{\boldsymbol{\imath}}_{\theta_i} \bullet \hat{\boldsymbol{\imath}}_{r_i} = 0$

$$\left(\frac{\dot{\boldsymbol{r}}_{i_0}}{r_{i_0}}\right) \bullet \hat{\boldsymbol{\imath}}_{r_{i_0}} = \dot{f}_0 \tag{2.30}$$

so just flipping the sides of the equation we get a nice expression for $\dot{f}$

$$\dot{f}_0 = \left(\frac{\dot{\boldsymbol{r}}_{i_0}}{r_{i_0}}\right) \bullet \hat{\boldsymbol{\imath}}_{r_{i_0}} \tag{2.31}$$

where $\hat{\boldsymbol{\imath}}_{r_{i_0}}$ can be found by dividing the position vector by its magnitude

$$\hat{\boldsymbol{\imath}}_{r_{i_0}} = \frac{\boldsymbol{r}_{i_0}}{r_{i_0}} \tag{2.32}$$

To find ω we want to start by finding the angular momentum of the body for which we know the velocity. Begin with equation (9.51) from the book, shown here

$$\boldsymbol{h} = \boldsymbol{r} \times \dot{\boldsymbol{r}} = h\hat{\boldsymbol{\imath}}_h \tag{2.33}$$

The $\hat{\boldsymbol{\imath}}_h$ unit vector can be found using the position and velocity of the particle, so for later use

$$\hat{\boldsymbol{\imath}}_h = \frac{\boldsymbol{r} \times \dot{\boldsymbol{r}}}{|\boldsymbol{r} \times \dot{\boldsymbol{r}}|} \tag{2.34}$$

The magnitude of the angular momentum h can be defined using ω which will be used reversely to find ω

$$h = \omega r_{i_0}^2 \tag{2.35}$$

Combining these equations and rearranging for ω gives an equation for ω which only uses known quantities

$$\omega = \frac{|\ \dot{\boldsymbol{r}}_{i_0} \times \boldsymbol{r}_{i_0}\ |}{r_{i_0}^2} \tag{2.36}$$

Another way to determine omega can be found by beginning with Eq. 2.28 and rearranging to get the vector component with ω on the left side by itself

$$\omega\hat{\boldsymbol{\imath}}_\theta = \left(\frac{\dot{\boldsymbol{r}}_{i_0}}{r_{i_0}} - \dot{f}\hat{\boldsymbol{\imath}}_{r_{i_0}}\right) \tag{2.37}$$

The magnitude of this vector is ω

$$\omega = \left| \frac{\dot{\boldsymbol{r}}_{i_0} - \dot{f}\boldsymbol{r}_{i_0}}{r_{i_0}} \right| \tag{2.38}$$

Use whichever definition of ω seems easier at the time

The special case is circular orbits. For circular orbits none of velocities are necessarily given. For any other case, at least 1 velocity should be given to allow the others to be determined.

2.2.1.2 Circular Orbits

For the circular orbit $\dot{f} = 0$ because $\dot{f}$ is the component of the velocity towards or away from the center of mass, and Bodies travelling in circles are not moving toward or away from the center of mass. Using $F = ma$ where a is the centripetal acceleration ($r_i\omega^2$) of the body around the center of mass

$$F_i = m_i r_i \omega^2 = \left| \sum_{j=1}^{3} \boldsymbol{F}_{ij} \right| \qquad j \neq i \tag{2.39}$$

Since there are only 2 forces acting on each body, the sum of the forces can be written out using

$$\boldsymbol{F}_{ij} = \frac{G m_j}{r_{ij}^3} \boldsymbol{r}_{ij} \tag{2.40}$$

so plugging in for $\boldsymbol{F}_{ij}$ and $\boldsymbol{F}_{ik}$ and rearranging for ω yields

$$\omega = \sqrt{\frac{G}{r_i} \left| \frac{m_j}{r_{ij}^3} \boldsymbol{r}_{ij} + \frac{m_k}{r_{ik}^3} \boldsymbol{r}_{ik} \right|} \tag{2.41}$$

We now have both $\dot{f}$ and ω for both the circular or non-circular orbit. To find the other velocities we have to use these values as shown below.

2.2.2 Using $\dot{f}$ and ω

To find the other velocities we must first find $\hat{\boldsymbol{\imath}}_{\theta_i}$ for each body using Eq. 2.34. This is possible since we are using a right handed coordinate system.

$$\hat{\boldsymbol{\imath}}_{\theta_i} = \hat{\boldsymbol{\imath}}_h \times \hat{\boldsymbol{\imath}}_{r_i} \tag{2.42}$$

where $\hat{\boldsymbol{\imath}}_h$ was found earlier, and is the same for all of the bodies and

$$\hat{\boldsymbol{\imath}}_{r_i} = \frac{\boldsymbol{r}_i}{r_i} \tag{2.43}$$

for each body. Then we simply need to plug the unit vectors, $\dot{f}$, and ω into equation 2.27 reshown here

$$\dot{\boldsymbol{r}}_i(0) = r_{i_0}\dot{f}_0\hat{\boldsymbol{\imath}}_{r_i} + r_{i_0}\omega\hat{\boldsymbol{\imath}}_{\theta_i} \tag{2.44}$$

Chapter 3

Jacobi Integral & Zero-Relative-Velocity Surfaces

The Jacobi Integral is a way to look at the circular restricted three-body problem in the rotating frame. The book presents the Jacobi Integral in both dimensional and non-dimensional forms. For the dimensional case the Jacobi Integral is:

$$v^2 = \omega^2(r_x^2 + r_y^2) + 2\frac{Gm_1}{\xi_1} + 2\frac{Gm_2}{\xi_2} - C \tag{3.1}$$

where

$$\xi_i = \sqrt{(r_x - r_i)^2 + r_y^2 + r_z^2} \tag{3.2}$$

To make the Jacobi Integral non-dimensional, a new time variable τ is introduced. I find it easiest to view this new time variable as analogous to an angle in radians through which the system has rotated since $t = 0$. τ is related to t through

$$\tau = \omega t \tag{3.3}$$

Time derivatives with respect to τ are designated with a "bubble" over the variable and are related to the old time derivatives through

$$\mathring{x}\,\omega = \dot{x} \tag{3.4}$$

τ has "units" of radians, and this new $\mathring{x}$ has units of 1/rad.

Lengths are non-dimensionalized by dividing the length by the distance between body 1 and body 2 (r_{12}). Masses are also non-dimensionalized by

using a new constant scalar parameter μ which is not the same as the μ used for gravitational coefficient. The mass of bodies 1 and 2 can also be expressed in terms of this new constant.

$$\mu = \frac{m_2}{m_1 + m_2} = \frac{1}{\frac{m_1}{m_2} + 1} \tag{3.5}$$

$$m_1 = 1 - \mu \tag{3.6}$$
$$m_2 = \mu \tag{3.7}$$

Similarly as shown in the book, the locations of the bodies are:

$$x_1 = -\mu \tag{3.8}$$
$$x_2 = 1 - \mu \tag{3.9}$$

This leads, as shown in the book, to the non-dimensional Jacobi Integral:

$$v^2 = (\mathring{x}^2 + \mathring{y}^2) = (x^2 + y^2) + 2\frac{1-\mu}{\rho_1} + 2\frac{\mu}{\rho_2} - C \tag{3.10}$$

where

$$\rho_i = \sqrt{(x - x_i)^2 + y^2 + z^2} \tag{3.11}$$

Also from the non-dimensionalizing analysis, the non-dimensional equations of motion can be found

3.1 Non-Dimensional Equations of Motion

The Non-Dimensional Equations of Motion are shown in the book in equations 10.86 re-shown here:

$$\mathring{\mathring{x}} - 2\,\mathring{y} = x - (1-\mu)\frac{x - x_1}{\rho_1^3} - \mu\frac{x - x_2}{\rho_2^3} \tag{3.12}$$

$$\mathring{\mathring{y}} + 2\,\mathring{x} = \left(1 - \frac{1-\mu}{\rho_1^3} - \frac{\mu}{\rho_2^3}\right) y \tag{3.13}$$

$$\mathring{\mathring{z}} = -\left(\frac{1-\mu}{\rho_1^3} - \frac{\mu}{\rho_2^3}\right) z \tag{3.14}$$

These equations of motion allow the motion of a satellite to be plotted in the rotating frame. The function `rotating()` as described in Section 6.2 plots such motions. An example plot is shown below in Figure 3.1

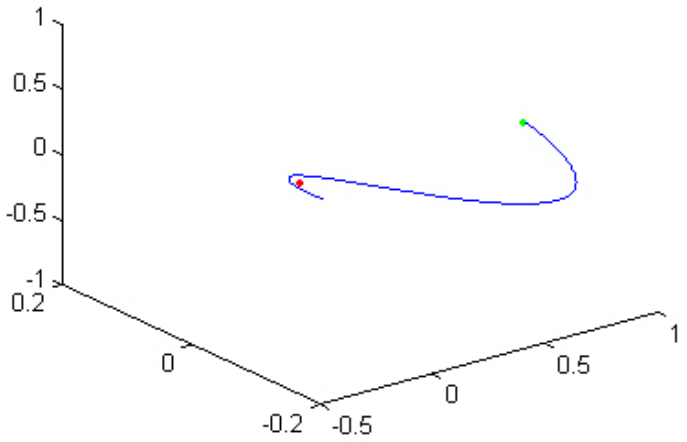


Figure 3.1: `rotating()` output graph

3.2 Zero-Relative Velocity Surfaces

Analyzing the non-dimensional Jacobi integral it is possible to set the velocity equal to zero and find a three dimensional surface for which there is a constant value for C. It is beneficial to do visualize these surfaces because for any satellite with that value of C in the Jacobi integral, it cannot pass through the zero-relative-velocity surface. The value of C for a satellite is found by plugging the velocity and position into the Jacobi integral and solving for C. If the value of C for a given satellite is greater than the value of C for a satellite positioned at L2 and not moving in the rotating frame, then the satellite can never transfer from the Earth to the Moon. similarly, satellites with a greater value of C than the C for a satellite at L3 could never leave the Earth-Moon system. Plots of the three dimensional surfaces, or 2-D contours can be done using `zerorel2b()` and `zerorel2d()` respectively as shown in Chapter 6, one example is shown below cut at $y = 0$, more can be found in the book. The equation for the surface is:

$$(x^2 + y^2) + 2\frac{1-\mu}{\rho_1} + 2\frac{\mu}{\rho_2} = C \tag{3.15}$$

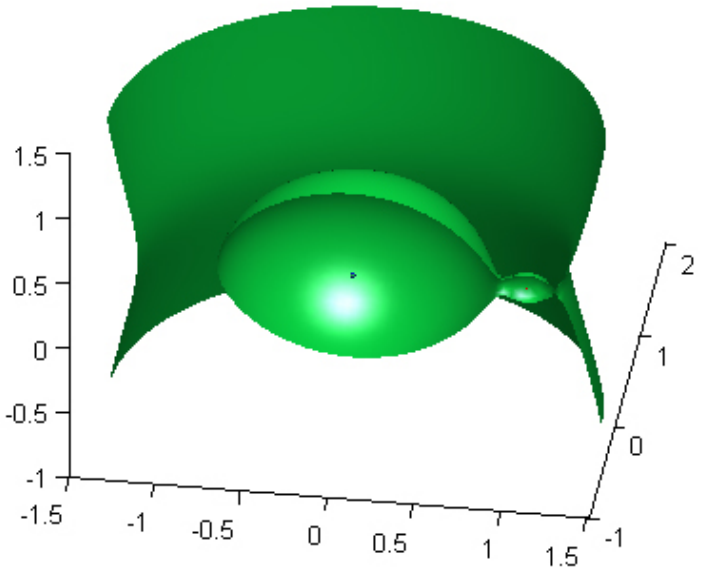


Figure 3.2: `zerorel2b()` output graph

Chapter 4

Lagrange Point Stability

As discussed in the book the L1, L2, and L3 Lagrange points are not stable due to the saddle point nature of the zero-relative-velocity surfaces. However, for some systems L4 and L5 are stable. The condition for stability is a mass ratio of

$$\frac{m_1}{m_2} \geq 24.9599 \tag{4.1}$$

The mass ratio requirement allows for stable L4 and L5 points for the Earth-Moon system with a ratio of 81.3, and Sun-Jupiter system with a ratio of approximately 1047. Theoretically since the mass ratio of every system involving the Sun and a planet has at least the mass ratio of the Sun-Jupiter system, each of these systems would have stable L4 and L5 points. However, other factors would probably play a role in making these locations unstable such as 4th body perturbations, or just simply being too close to the sun which is not a point mass. Below are figures representing the path of a body placed near each of the lagrange points for the Earth-Moon system, and near L4 for the Sun-Jupiter system. In each figure the red dot represents the first body and the green the second. Zero-relative-velocity surfaces are plotted for the Earth-Moon system

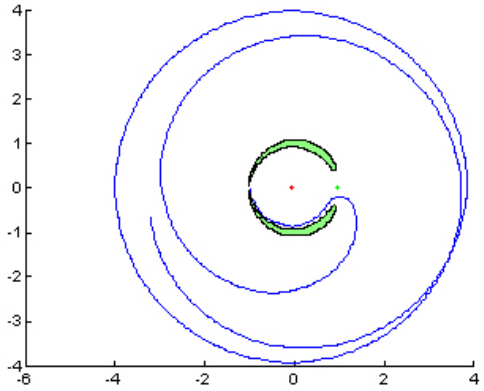


Figure 4.1: Earth-Moon L1 instability

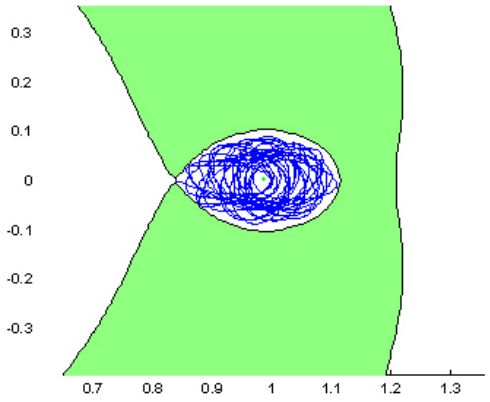


Figure 4.2: Earth-Moon L2 instability

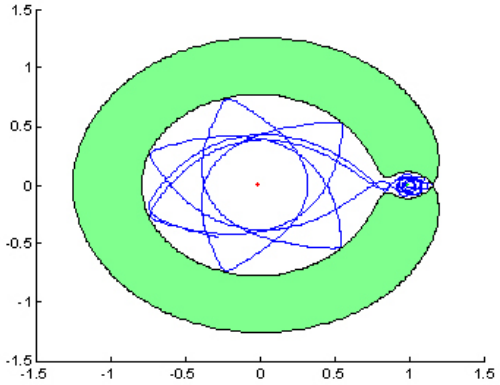


Figure 4.3: Earth-Moon L3 instability

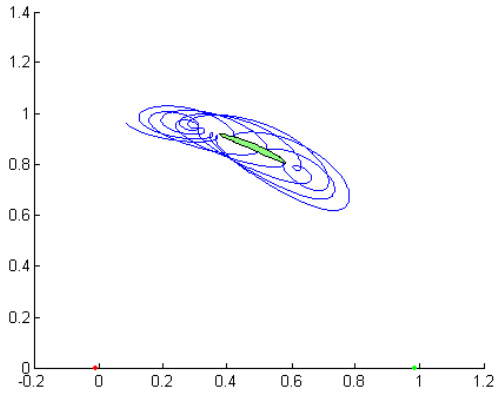


Figure 4.4: Earth-Moon L4 stability

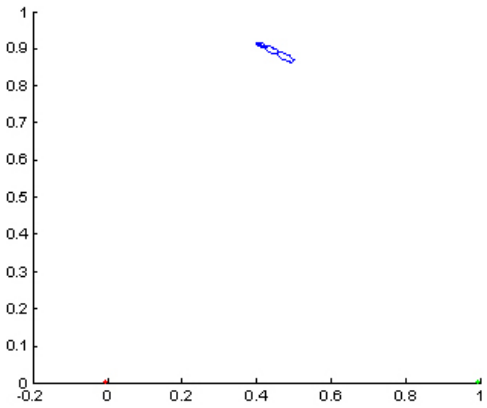


Figure 4.5: Sun-Jupiter L4 stability

Chapter 5

Personal Example: LEO-Moon Transfer

Modifying the rotating program, I developed code to simulate a moon transfer that would never have enough energy to leave the earth moon system completely. To do this, the energy would always have to be less than the L3 zero-relative-velocity energy. Working backwards from a lunar altitude of 150km, just like example 10.6 from the book, except with a different velocity. The velocity used here was calculated by:

$$C3 = x_3^2 + \frac{2(1-\mu)}{|x3+\mu|} + \frac{2\mu}{|x_3 - 1 + \mu|} \tag{5.1}$$

$$V_{3y} = \sqrt{P_{3x}^2 + \frac{2(1-\mu)}{|P_{3x}+\mu|} + \frac{2\mu}{|P_{3x} - 1 + \mu|} - C_3} = 1.8277 \tag{5.2}$$

The motion of the spacecraft was extrapolated backwards in time to find the point of closest approach with the Earth. This occurred at a non-dimensionalized time of -2.4611.

Using a trial and error method, a Δv ratio burn of .1795 working backwards was selected. This corresponds to a Δv ratio burn forwards of 5.571 times the post-burn velocity, and a Δv burn of 1.4996. In dimensional units the Δv is 1.536 km/s.

Next the flight was extrapolated backwards until the LEO altitude of 298km was reached. This occurred at a non-dimensional time of .1827. At this point the spacecraft is travelling with a speed of 10.328, which is 10.581 km/s.

At the LEO altitude beginning the mission, the spacecraft would have a circular velocity of 7.7268km/s. To obtain the required velocity for the second phase (LEO being first) of the mission, the satellite would need to undergo a Δv or 2.855 km/s.

If the mission did not desire to circularize about the moon, a Δv_{total} of 4.391 km/s is necessary.

If the mission does desire to circularize about the moon, the 3rd Δv of the mission needs to be considered A circular orbit about the moon at an altitude of 150km has a velocity of 1.6105 km/s. At periselenium the satellite has a non-dimensionalized velocity of 1.8277 and a dimensional velocity of 1.873 km/s. This requires the third Δv to be -0.262 km/s

Summing up the 3 required Δv's, a Δv_{total} of 4.653 km/s is necessary

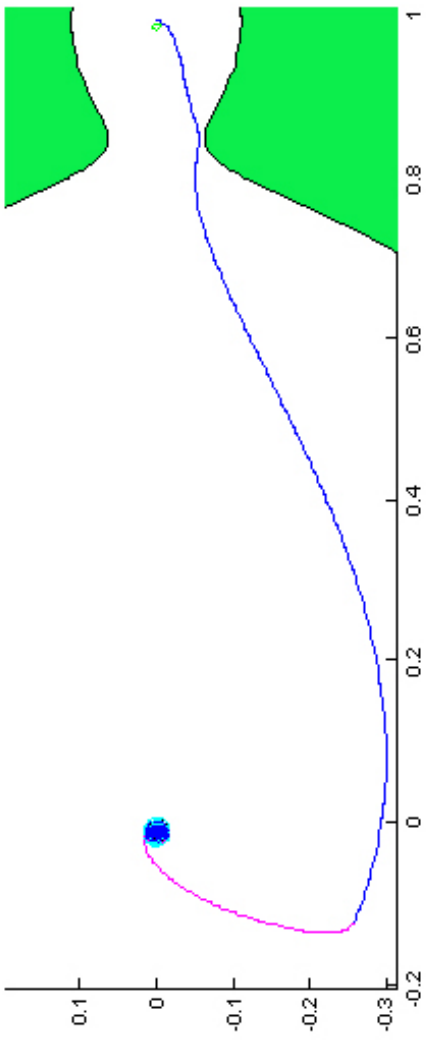


Figure 5.1: Multiple Burn LEO-Moon Transfer

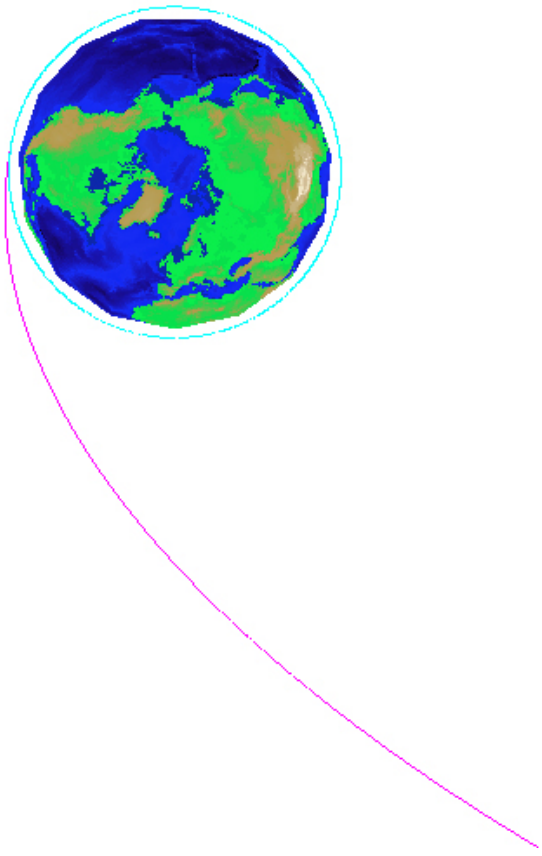


Figure 5.2: Earth, LEO and the First Transfer

Chapter 6

MATLAB Code Description/Usage

6.1 general3d.m & general3dode.m

`general3d()` determines the position, masses, and velocities for all 3 bodies in the 3-body system and plots their motion for the specified time. The variables which can be set are all located at the beginning of general3d.m, and are shown in Table 6.1. Set as many of the variables as you can but not all of the variables must be set for each case. Table 6.2 shows the variables which must be set for each run to work. This program is based off of the algorithm described in the "Cookbook" in Chapter 2. Some of the flag variables are for which body you know or don't know something about. For these variables, in the triangle arrangement the ordering does not matter, but for the collinear arrangement the bodies should be in the order 1—2—3.

All vectors should be in the form [x y z]. Position vectors only need to be relative to the inertial frame, not the center of mass, especially since in many cases the location and masses of all of the bodies are not known.

6.2 rotating.m

The `rotating()` function uses two files: rotating.m and rotatingode.m. Table 6.3 shows the variables which should be set in rotating.m and their default values.

Table 6.1: Variables to be set in general3d.m

Variable	**Default**	**Options/Description**
SHAPE	0	0 for the triangle 1 for collinier
CIRCULAR	0	1 for circular 0 for elliptical
MISSINGINFO	3	1—2—3
MASSMISSING	1	1 if the mass is unknown 0 if the position is unknown
P1	[0 0 0]	the location of the first body (Km)
P2	[1e8 0 0]	the location of the second body (Km)
P3	[.5e8 1e8*sqrt(3)/2 0]	the location of the third body (Km)
VWHICHKNOWN	1	which body you know the velocity of
Vknown	[0 1 1]*1000/sqrt(2)	the known velocity (Km/s)
m1	1e24	the mass of the first body (Kg)
m2	1.5e24	the mass of the second body (Kg)
m3	2e24	the mass of the third body (Kg)
WHATVEC	1	1 if you have a vector normal to the motion 0 if you have a vector in the plane of motion
planevec	[0 10 10]	the vector in the plane of motion
hdir	[0 1 1]	the vector in the normal direction
AdditionalT	2e5	how long the simulation should run initially (sec)

Table 6.2: Variables which require values for each case

	Equilateral		Collinear	
Variable	Circular	Elliptical	Circular	Elliptical
SHAPE	*	*	*	*
CIRCULAR	*	*	*	*
MISSINGINFO			*	*
MASSMISSING			*	*
VWHICHKNKOWN		*		*
Vknown		*		*
WHATVEC			*	
planevec *or* hdir			*	
AdditionalT	*	*	*	*
All Masses	*	*		
All Positions	*			
P1 & P2		*		
All Positions, 2 Masses or All Masses, 2 Positions			*	*

Table 6.3: Variables to set in rotating.m

Variable	**Default**	**Description**
P3x	P2x+1.74e3+150	x location of the satellite relative to Earth
P3y	0	y location of the satellite
P3z	0	z location of the satellite
V3x	0	Speed of satellite in x direction
V3y	-2.47	Speed of satellite in y direction
V3z	0	Speed of satellite in z direction
AdditionalT	0.67	Simulation run time in dimensionless units

6.3 surfacevalue.m

`surfacevalue()` computes and returns the corresponding zero relative surface value for a given location in the rotating frame and velocity. Usage:

```
C=surfacevalue(x,y,z,v)
```

All input values should be non-dimensional. The value C may be input into the function `zerorel3b()` to plot the surface

6.4 zerorela.m & zerorelb.m

zerorela&b are functions to plot zero-relative-velocity surfaces. The programs allow for default options of surfaces touching the lagrange points or a custom surface. zerorel2a should be run once per computer and computes a three dimensional array of surface values which is used by zerorel2b. The array is saved in a file called zerorelative.mat. The default step size in x,y, and z directions is 0.05. While using a smaller step size will increase the accuracy and resolution of the surface drawings, it will also rapidly increase the time the programs have to run to compute the array and surfaces.

Usage:

```
zerorela()

zerorelb(isosurf)
```

using the lone input isosurf, zerorel2b computes and plots the corresponding isosurface. To plot the isosurface corresponding to a Lagrange point, input 'L1' or 'L2' etc.. For any other surface, input the value C for the surface The function `surfacevalue(x,y,z,v)` can calculate C for you

Example:

```
zerorelb('L2')
```

-or-

```
zerorelb(3.1)
```

-or-

```
C= surfacevalue(x,y,z,v)
zerorelb(C)
```

Inside zerorel2b.m two variables may be set, the variables and their options are listed below.

- cutlocation

  options:

  'x' cuts at x=0 showing all values with positive x component

  'y' cuts at y=0 showing all values with positive y component

  'z' cuts at z=0 showing all values with negative z component

  'none' no cutaway

- transparency

  options:

  0 completely transparent

  0 < # < 1 translucent

  1 opaque

  default: 0.6

- color

  options:

  'red' or 'r' completely transparent

  'green' or 'g' translucent

  'blue' or 'b' opaque

  'cyan' or 'c' 0.6

  'magenta' or 'm'

  'yellow' or 'y' completely transparent

  'black' or 'k' translucent

  'white' or 'w' opaque

  [# # #] [r g b] color values, 0-1

  default: 'green'

6.5 zerorel2d.m

`zerorel2d()` plots a filled contour of the desired zero-relative-velocity surface in the x-y plane. The options and usage are below:

```
zerorel2d(isosurf)
```

Input options:

'L1' Isosurface contour touching L1

'L2' Isosurface contour touching L2

'L3' Isosurface contour touching L3

'L4' or 'L5' Isosurface contour touching L4 and L5

Isosurface contour at a C value of

Input usage and examples are identical to `zerorel2b()` as seen in Section 6.4

NOTE: The Z array is ONLY computed the first time `zerorel2d()` is run to change the resolution of the matrix you MUST delete zerorelativecontour.mat and rerun the function

6.6 solvequintic.m

`X=solvequintic()` solves Lagrange's quintic equation and returns a value X which is the χ value for the masses specified in sovequintic.m. Change the values for the masses and run the function to solve the quintic equation for χ

6.7 circle.m

`[x y]=circle(r)` returns vectors x and y with coordinates for plotting a circle with radius r, and is used by a few of the programs mentioned above.

Appendix A

MATLAB Code

A.1 general3d.m

```
function general3d ()
%Determines the locations and/or masses of a 3 body problem
%The following variables need to be set,
%but not all apply to each case.
%For a listing of which variables must be set for each case
%refer to the report.
SHAPE=0;%0 for the triangle, 1 for collinier
CIRCULAR=0;%1 for circular, 0 for noncicular

%With the bodies alligned in the following order,
%which is of unknown mass or position
MISSINGINFO=3;%1-----2-----3
MASSMISSING=0;%1 if the mass is unknown,
              %0 if the position is unknown
%Enter all coordinates and vectors in the form [x y z] with brackets
P1=[0 0 0];%the location of the first body (Km)
P2=[1e8 0 0];%the location of the second body (Km)
P3=[.5e8 1e8*sqrt(3)/2 0];%the location of the third body (Km)
VWHICHKNOWN=1;%which body you know the velocity of
Vknown=[0 1 1]*1000/sqrt(2);%the known velocity (Km/s)
m1=1e24;%the mass of the first body (Kg)
m2=1.5e24;%the mass of the second body (Kg)
m3=2e24;%the mass of the third body (Kg)
```

```
WHATVEC=1;%1 if you have a vector normal to the motion
          %0 if a vector in the plane of motion
planevec=[0 10 10];%the vector in the plane of motion
hdir=[0 1 1];%the vector in the normal direction
AdditionalT=2e5;%how long the simulation should run initially (sec)

%when the program asks if you would like it to draw lines connecting
%the objects, 1 means yes, 0 means no
G=6.67e-11;

P1=transpose(P1);
P2=transpose(P2);
P3=transpose(P3);
Vknown=transpose(Vknown);
planevec=transpose(planevec);
hdir=transpose(hdir);

if SHAPE==0
    if (CIRCULAR~=1)
        r12=P2-P1;
        hdir=cross(r12,Vknown);
        maghdir=sqrt(dot(hdir,hdir));
        ih=hdir./maghdir;

        rho=sqrt(dot(r12,r12));
        ir12=r12./rho;
        it12=cross(ih,ir12);

        r13=.5*rho.*ir12 + sqrt(3)/2*rho.*it12;

        P3=P1+r13;

        rc=(m1.*P1 +m2.*P2 +m3.*P3)./(m1+m2+m3);
        r1=P1-rc;
        r2=P2-rc;
        r3=P3-rc;
        magr1=sqrt(dot(r1,r1));
        magr2=sqrt(dot(r2,r2));
        magr3=sqrt(dot(r3,r3));
        ir1=r1./magr1;
        ir2=r2./magr2;
        ir3=r3./magr3;
```

```
    it1=cross(ih,ir1);
    it2=cross(ih,ir2);
    it3=cross(ih,ir3);

    switch VWHICHKNOWN
    case 1
        fdot=dot(Vknown./magr1,ir1);
        omega=dot(Vknown./magr1,it1);
    case 2
        fdot=dot(Vknown./magr2,ir2);
        omega=dot(Vknown./magr2,it2);
    case 3
        fdot=dot(Vknown./magr3,ir3);
        omega=dot(Vknown./magr3,it3);
    end
end
if (CIRCULAR)
    fdot=0;
    rc=(m1.*P1 +m2.*P2 +m3.*P3)./(m1+m2+m3);
    r1=P1-rc;
    r2=P2-rc;
    r3=P3-rc;

    magr1=sqrt(dot(r1,r1));
    magr2=sqrt(dot(r2,r2));
    magr3=sqrt(dot(r3,r3));

    ir1=r1./magr1;
    ir2=r2./magr2;
    ir3=r3./magr3;

    r12=P2-P1;
    r13=P3-P1;

    magr12=sqrt(dot(r12,r12));
    magr13=sqrt(dot(r13,r13));

    hdir=cross(r12,r13);
    maghdir=sqrt(dot(hdir,hdir));
    ih=hdir./maghdir;
```

```
        it1=cross(ih,ir1);
        it2=cross(ih,ir2);
        it3=cross(ih,ir3);

        A=m2/magr12^3.*r12+m3/magr13^3.*r13;
        magA=sqrt(dot(A,A));

        omega=sqrt(G*magA/magr1);
    end

    V1=magr1*fdot.*ir1 + omega*magr1.*it1;
    V2=magr2*fdot.*ir2 + omega*magr2.*it2;
    V3=magr3*fdot.*ir3 + omega*magr3.*it3;

else%collinear

    if MASSMISSING
        r23=P3-P2;
        r12=P2-P1;
        magr23=sqrt(dot(r23,r23));
        magr12=sqrt(dot(r12,r12));
        x=magr23/magr12;
        switch MISSINGINFO
        case 3
            m3 = (x.^3.*m1.*(x.^2+3*x+3)+...
                  m2.*(x.^5+2.*x.^4+x.^3-x.^2-2.*x-1))./(3.*x.^2+3.*x+1);
        case 2
            m2 = (m3.*(3.*x.^2+3.*x+1)-...
                  x.^3.*m1.*(x.^2+3*x+3))./(x.^5+2.*x.^4+x.^3-x.^2-2.*x-1);
        case 1
            m1 = (m3.*(3.*x.^2+3.*x+1) -...
                  m2.*(x.^5+2.*x.^4+x.^3-x.^2-2.*x-1))./(x.^3.*(x.^2+3*x+3));
        end
    else
        syms m1a m2a m3a x;
        temp=solve(subs((m1a+m2a)*x.^5 + (3*m1a+2*m2a)*x.^4+...
                        (3*m1a+m2a)*x.^3 - (m2a+3*m3a)*x.^2-...
                        (2*m2a + 3*m3a)*x - (m2a+m3a),...
                        [m1a,m2a,m3a],[m1,m2,m3]));
        temp=sym2poly(temp);
        x=temp(5);
        switch MISSINGINFO
```

```
    case 1
        r23=P3-P2;
        r12=r23./x;
        P1=P2-r12;
    case 2
        r13=P3-P1;
        r12=r13./(1+x);
        P2=P1+r12;
    otherwise
        r12=P2-P1;
        r13=r12.*(1+x);
        P3=P1+r13;
    end
end

%find velocities
if (CIRCULAR~=1)
    r12=P2-P1;
    hdir=cross(r12,V1);
    maghdir=sqrt(dot(hdir,hdir));
    ih=hdir./maghdir;

    rho=sqrt(dot(r12,r12));
    ir12=r12./rho;
    it12=cross(ih,ir12);

    r13=.5*rho.*ir12 + sqrt(3)/2*rho.*it12;

    P3=P1+r13;

    rc=(m1.*P1 +m2.*P2 +m3.*P3)./(m1+m2+m3);
    r1=P1-rc;
    r2=P2-rc;
    r3=P3-rc;
    magr1=sqrt(dot(r1,r1));
    magr2=sqrt(dot(r2,r2));
    magr3=sqrt(dot(r3,r3));
    ir1=r1./magr1;
    ir2=r2./magr2;
    ir3=r3./magr3;

    it1=cross(ih,ir1);
```

```
    it2=cross(ih,ir2);
    it3=cross(ih,ir3);

    switch VWHICHKNOWN
    case 1
        fdot=dot(Vknown./magr1,ir1);
        omega=dot(Vknown./magr1,it1);
    case 2
        fdot=dot(Vknown./magr2,ir2);
        omega=dot(Vknown./magr2,it2);
    case 3
        fdot=dot(Vknown./magr3,ir3);
        omega=dot(Vknown./magr3,it3);
    end
end
if (CIRCULAR)
    fdot=0;
    rc=(m1.*P1 +m2.*P2 +m3.*P3)./(m1+m2+m3);
    r1=P1-rc;
    r2=P2-rc;
    r3=P3-rc;

    magr1=sqrt(dot(r1,r1));
    magr2=sqrt(dot(r2,r2));
    magr3=sqrt(dot(r3,r3));

    ir1=r1./magr1;
    ir2=r2./magr2;
    ir3=r3./magr3;

    r12=P2-P1;
    r13=P3-P1;

    magr12=sqrt(dot(r12,r12));
    magr13=sqrt(dot(r13,r13));

    if WHATVEC~=1
        hdir=cross(r12,planevec);
    end
    maghdir=sqrt(dot(hdir,hdir));
    ih=hdir./maghdir;
```

```
        it1=cross(ih,ir1);
        it2=cross(ih,ir2);
        it3=cross(ih,ir3);

        A=m2/magr12^3.*r12+m3/magr13^3.*r13;
        magA=sqrt(dot(A,A));

        omega=sqrt(G*magA/magr1);
    end
    V1=magr1*fdot.*ir1 + omega*magr1.*it1;
    V2=magr2*fdot.*ir2 + omega*magr2.*it2;
    V3=magr3*fdot.*ir3 + omega*magr3.*it3;
end

P1x=P1(1);
P1y=P1(2);
P1z=P1(3);
P2x=P2(1);
P2y=P2(2);
P2z=P2(3);
P3x=P3(1);
P3y=P3(2);
P3z=P3(3);

V1x=V1(1);
V1y=V1(2);
V1z=V1(3);
V2x=V2(1);
V2y=V2(2);
V2z=V2(3);
V3x=V3(1);
V3y=V3(2);
V3z=V3(3);

oldT=0;
while AdditionalT>0
    IC=[m1;m2;m3;
        P1x;P2x;P3x;
        P1y;P2y;P3y;
        P1z;P2z;P3z;
        V1x;V2x;V3x;
        V1y;V2y;V3y;
```

```
    V1z;V2z;V3z];
tspan=linspace(oldT,AdditionalT+oldT,10);
options=odeset('AbsTol',1e-6,'RelTol',1e-6);
[t,X]=ode45('general3dode',tspan,IC,options);

figure(1);
hold on;
plot3(X(:,4),X(:,7),X(:,10),'r.-',...
      X(:,5),X(:,8),X(:,11),'g.-',...
      X(:,6),X(:,9),X(:,12),'b.-');
xlabel('x');
ylabel('y');
zlabel('z');
campos([15e7 -4e7 8e7]);

oldT=oldT+AdditionalT;
AdditionalT = input('How much longer should the simulation run  ');
draw = input('Draw lines connecting the bodies?  ');

size=size(X);
height=size(1);
P1x=X(height,4);
P2x=X(height,5);
P3x=X(height,6);

P1y=X(height,7);
P2y=X(height,8);
P3y=X(height,9);

P1z=X(height,10);
P2z=X(height,11);
P3z=X(height,12);

V1x=X(height,13);
V2x=X(height,14);
V3x=X(height,15);

V1y=X(height,16);
V2y=X(height,17);
V3y=X(height,18);

V1z=X(height,19);
```

```
    V2z=X(height,20);
    V3z=X(height,21);
    if(draw)
        figure(1);
        line1x=linspace(P1x,P2x);
        line2x=linspace(P1x,P3x);
        line3x=linspace(P2x,P3x);
        line1y=linspace(P1y,P2y);
        line2y=linspace(P1y,P3y);
        line3y=linspace(P2y,P3y);
        line1z=linspace(P1z,P2z);
        line2z=linspace(P1z,P3z);
        line3z=linspace(P2z,P3z);
        plot3(line1x,line1y,line1z,'k-')
        plot3(line2x,line2y,line2z,'k-')
        plot3(line3x,line3y,line3z,'k-')
        draw=0;
        x12=sqrt((P2x-P1x)^2 + (P2y-P1y)^2 + (P2z-P1z)^2);
        x13=sqrt((P3x-P1x)^2 + (P3y-P1y)^2 + (P3z-P1z)^2);
        x23=sqrt((P3x-P2x)^2 + (P3y-P2y)^2 + (P3z-P2z)^2);
    end
end
```

Sample Output:

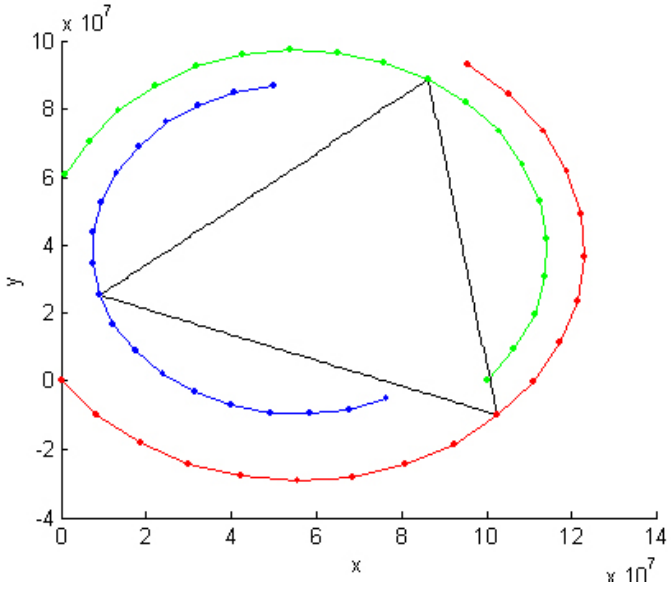


Figure A.1: `general3d()` output graph

general3dode.m

```
function dy=general3dode(t,Y,flag,param)
m1=Y(1);
m2=Y(2);
m3=Y(3);

P1x=Y(4);
P2x=Y(5);
P3x=Y(6);

P1y=Y(7);
P2y=Y(8);
P3y=Y(9);

P1z=Y(10);
P2z=Y(11);
P3z=Y(12);

V1x=Y(13);
V2x=Y(14);
V3x=Y(15);

V1y=Y(16);
V2y=Y(17);
V3y=Y(18);

V1z=Y(19);
V2z=Y(20);
V3z=Y(21);


G=6.67259*10^(-11);
        %Relative position vectors
        r12x = P2x - P1x;
        r12y = P2y - P1y;
        r12z = P2z - P1z;
        r13x = P3x - P1x;
        r13y = P3y - P1y;
        r13z = P3z - P1z;
        r23x = P3x - P2x;
        r23y = P3y - P2y;
        r23z = P3z - P2z;
```

```
r12 = sqrt(r12x^2+r12y^2+r12z^2);
r13 = sqrt(r13x^2+r13y^2+r13z^2);
r23 = sqrt(r23x^2+r23y^2+r23z^2);

%Force components on each body
F12x = G*m1*m2*r12x/r12^3;
F12y = G*m1*m2*r12y/r12^3;
F12z = G*m1*m2*r12z/r12^3;
F13x = G*m1*m3*r13x/r13^3;
F13y = G*m1*m3*r13y/r13^3;
F13z = G*m1*m3*r13z/r13^3;
F23x = G*m2*m3*r23x/r23^3;
F23y = G*m2*m3*r23y/r23^3;
F23z = G*m2*m3*r23z/r23^3;

F1x =  F12x + F13x;
F2x = -F12x + F23x;
F3x = -F13x - F23x;

F1y =  F12y + F13y;
F2y = -F12y + F23y;
F3y = -F13y - F23y;

F1z =  F12z + F13z;
F2z = -F12z + F23z;
F3z = -F13z - F23z;

%Determine new accellerations Fx=m*ax
a1x=F1x/m1;
a1y=F1y/m1;
a1z=F1z/m1;
a2x=F2x/m2;
a2y=F2y/m2;
a2z=F2z/m2;
a3x=F3x/m3;
a3y=F3y/m3;
a3z=F3z/m3;

dy=[0;0;0;
    V1x;V2x;V3x;
```

```
V1y;V2y;V3y;
V1z;V2z;V3z;
a1x;a2x;a3x;
a1y;a2y;a3y;
a1z;a2z;a3z];
```

A.2 rotating.m & rotatingode.m

The `rotating()` function uses 2 files...rotating.m and rotatingode.m(for ODE45)

rotating.m

```
function rotating()
m1=5.97e24;
m2=7.35e22;
m3=100;%satellite mass(never used but passed to ode45)
P1x=0;
P2x=3.84e5;
%VARIABLES TO CHANGE
P3x=P2x+1.74e3+150;%x location of the satellite
                   %relative to Earth
                   %default=P2x+1.74e3+150
P3y=0;%y location of the satellite default=0
P3z=0;%z location of the satellite default=0
V3x=0;%Speed of satellite in x direction
      %default=0
V3z=0;%Speed of satellite in x direction
      %default=0
V3y=-2.47;%Speed of satellite in x direction
          %default: -2.47
AdditionalT = .67;%how long in dimensionless units
                %the simulation should run
                %default=.67

r12=P2x-P1x;
P1x=P1x/r12;
P2x=P2x/r12;
P3x=P3x/r12;
mu=m2/(m1+m2);

xcm=(m1*P1x+m2+P2x)/(m1+m2);
P1x=P1x-xcm;
P2x=P2x-xcm;
P3x=P3x-xcm;


oldT=0;
IC=[m1;m2;m3;
    P1x;P2x;
    P3x;P3y;P3z;
```

```
    V3x;V3y;V3z];
tspan=linspace(oldT,AdditionalT+oldT,10000);
options=odeset('AbsTol',1e-6,'RelTol',1e-6);
[t,X]=ode45('rotatingode',tspan,IC,options);

plot3(P1x,0,0,'r.-',P2x,0,0,'g.-',X(:,6),X(:,7),X(:,8),'b-');
```

Sample Output:

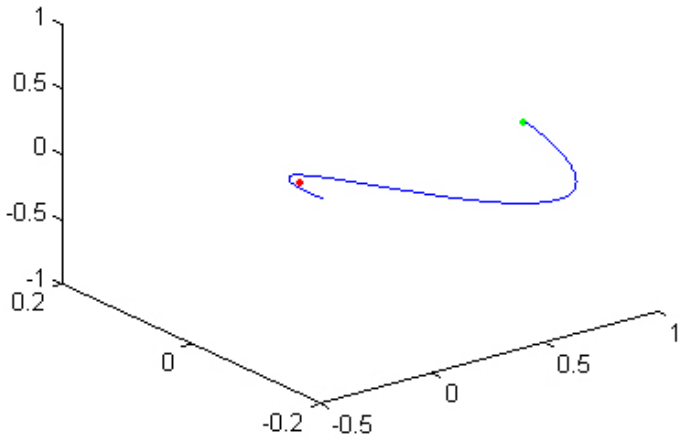


Figure A.2: `rotating()` output graph

rotatingode.m

```
function dy=rotatingode(t,Y,flag,param)
m1=Y(1);
m2=Y(2);
m3=Y(3);

P1x=Y(4);
P2x=Y(5);

P3x=Y(6);
P3y=Y(7);
P3z=Y(8);

V3x=Y(9);
V3y=Y(10);
V3z=Y(11);

mu=m2/(m1+m2);
%Relative position vectors
r13x = P3x - P1x;
r23x = P3x - P2x;

rho1 = sqrt(r13x^2+P3y^2+P3z^2);
rho2 = sqrt(r23x^2+P3y^2+P3z^2);


a3x=P3x-(1-mu)*r13x/rho1^3-mu*r23x/rho2^3+2*V3y;
a3y=(1-(1-mu)/rho1^3-mu/rho2^3)*P3y-2*V3x;
a3z=-((1-mu)/rho1^3+mu/rho2^3)*P3z;

dy=[0;0;0;%mass changes
    0;0;%changes in body 1 and 2 locations
    V3x;V3y;V3z;%change in body 3 position
    a3x;a3y;a3z];%change in body 3 velocity
```

A.3 surfacevalue.m

```
function C=surfacevalue(x,y,z,v)
%C=surfacevalue(x,y,z,v) computes the corresponding
%zero relative surface value C for a given location
%and velocity.
%x, y, and z are the nondimensional coordinates
%in the rotating frame, and v is the nondimensional
%velocity
mu=1/(81.3+1);
C=x^2+y^2+2*(1-mu)/sqrt((x+mu)^2+y^2)+2*mu/sqrt((x-1+mu)^2+y^2)-v^2;
```

A.4 zerorela.m

```
function zerorela()
%This function creates a file called zerorelative.mat
%which contains all the zero relative surface values
%in a matrix C for use in zerorel2b.m
mu=1/(81.3+1);
x=-1.5:.05:1.55;%to increase the resolution of the surfaces,
%change the middle number..default is .05
y=-1.5:.05:1.55;
z=-1:.05:1.05;
for i=1:length(x)
    for j=1:length(y)
        for k=1:length(z)
            C(j,i,k)=x(i)^2+y(j)^2+2*(1-mu)/sqrt((x(i)+mu)^2...
            +y(j)^2+z(k)^2)+2*mu/sqrt((x(i)-1+mu)^2+y(j)^2+z(k)^2);
        end
    end
end save('zerorelative.mat','x','y','z','C')
```

A.5 zerorelb.m

```
function zerorelb(isosurf)
%using the lone input isosurf,
%zerorel2b computes and plots the corrosponding isosurface.
%
%To plot the isosurface corresponding to
%a Lagrange point, input 'L1' or 'L2' etc..
%For any other surface, input the value C for the surface
%The function surfacevalue(x,y,z,v) can calculate C for you
%
%NOTE: zerorela must be run once prior to ever running zerorelb
%to create the array stored in zerorelative.mat

cutlocation='none';
%to change which plane the surface is cut at, change cutlocation
%options:
%'z' cuts at z=0 showing all values with negative z component
%'y' cuts at y=0 showing all values with positive y component
%'none' no cutaway

transparency= .6;
%sets the value for transparency for no cutaway
% 0 transparent, 1 opaque, default:.6

color='green';
%sets the color of the surface, options are MATLAB defined colors
%see either MATLAB help or report for a list


mu=1/(81.3+1);
switch upper(isosurf)
case 'L1'
    x1=-1.00506;
    Csurf= x1^2+2*(1-mu)/abs(x1+mu)+2*mu/abs(x1-1+mu);
    Csurf=Csurf+.002;
case 'L2'
    x2=.836915;
    Csurf= x2^2+2*(1-mu)/abs(x2+mu)+2*mu/abs(x2-1+mu);
case 'L3'
    x3=1.15568;
    Csurf= x3^2+2*(1-mu)/abs(x3+mu)+2*mu/abs(x3-1+mu);
case {'L4','L5'}
```

```
    x4=.5-mu;
    y4=sqrt(3)/2;
    Csurf=x4^2+y4^2+2*(1-mu)/sqrt((x4+mu)^2+y4^2)...
            +2*mu/sqrt((x4-1+mu)^2+y4^2);
otherwise
    Csurf=isosurf;
end

load('zerorelative.mat')
switch lower(cutlocation)
case 'z'
    p=patch(isosurface(x,y,z(1:length(z)/2),...
            C(:,:,(1:length(z)/2)),Csurf));
case 'y'
    p=patch(isosurface(x,y(length(y)/2:length(y)),...
            z,C((length(y)/2:length(y)),:,:),Csurf));
case 'x'
    p=patch(isosurface(x(length(x)/2:length(x)),...
            y,z,C(:,(length(x)/2:length(x)),:),Csurf));
otherwise
    p=patch(isosurface(x,y,z,C,Csurf));
end
    set(p,'FaceAlpha',transparency);
    set(p,'FaceColor',color,'EdgeColor','None');
    camlight;lighting gouraud;
 figure(1)
 load topo;
 [x y z] = sphere(15);
 xe=x.*6378/384405-mu;
 ye=y.*6378/384405;
 ze=z.*6378/384405;
 s = surface(xe,ye,ze,'facecolor','texture','cdata',topo,...
            'edgecolor','none');
 set(s,'backfacelighting','unlit');
 colormap(topomap1); alpha('direct'); alphamap([.1;1])
 campos([2 -13 10]);
 xm=x.*1.74e3/384405+1-mu;
 ym=y.*1.74e3/384405;
 zm=z.*1.74e3/384405;
 s = surface(xm,ym,zm,'facecolor','red','edgecolor','none');
```

Sample Output:

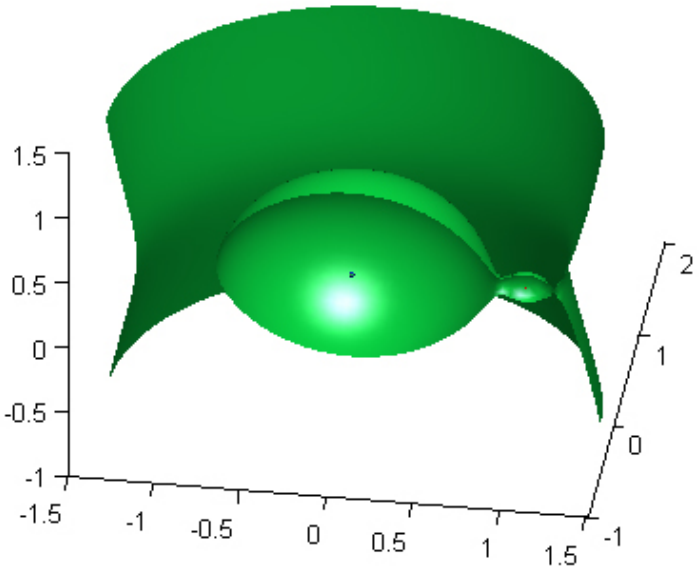


Figure A.3: `zerorel2b()` output graph

A.6 zerorel2d.m

```
function zerorel2d(isosurf)
%plots a filled contour isosurface at a value of isosurf
%options for isosurf include:
%'L1','L2' etc... or a value like 3
%NOTE: The Z array is ONLY computed the first time this
%function is run; to change the resolution of the matrix
%you MUST delete zerorelativecontour.mat and rerun the function
    mu=1/(81.3+1);
if (~exist('zerorelativecontour.mat'))
    disp('running')
    x=-1.5:.01:1.55;%to increase the resolution of the surfaces,
                    %change the middle number..default is .05
    y=-1.5:.01:1.55;
    for i=1:length(x)
        for j=1:length(y)
            Z(j,i)=x(i)^2+y(j)^2+2*(1-mu)/sqrt((x(i)+mu)^2...
                    +y(j)^2)+2*mu/sqrt((x(i)-1+mu)^2+y(j)^2);
        end
    end
    save('zerorelativecontour','x','y','Z');
else
    load('zerorelativecontour');
end switch upper(isosurf) case 'L1'
    x1=-1.00506;
    Csurf= x1^2+2*(1-mu)/abs(x1+mu)+2*mu/abs(x1-1+mu);
case 'L2'
    x2=.836915;
    Csurf= x2^2+2*(1-mu)/abs(x2+mu)+2*mu/abs(x2-1+mu);
case 'L3'
    x3=1.15568;
    Csurf= x3^2+2*(1-mu)/abs(x3+mu)+2*mu/abs(x3-1+mu);
case {'L4','L5'}
    x4=.5-mu;
    y4=sqrt(3)/2;
    Csurf=x4^2+y4^2+2*(1-mu)/sqrt((x4+mu)^2+y4^2)...
            +2*mu/sqrt((x4-1+mu)^2+y4^2)+.001;
otherwise
    Csurf=isosurf;
end

[c,h]=contourf(x,y,-Z,[-Csurf -Csurf]);
```

Sample Output:

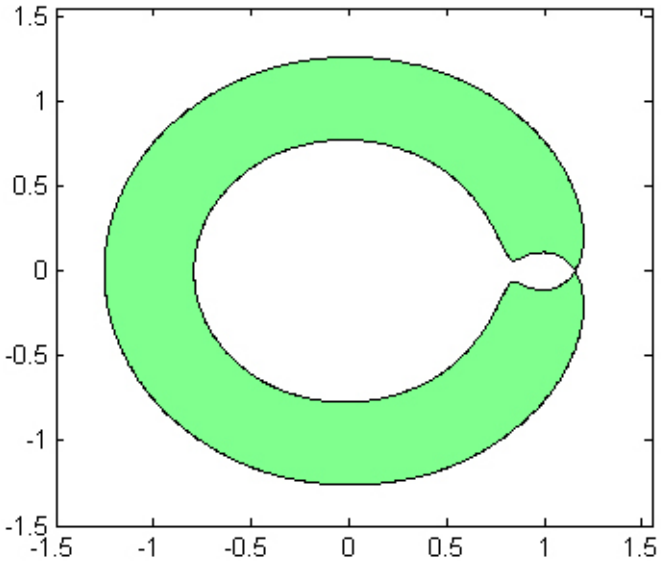


Figure A.4: `zerorel2d()` output graph

A.7 solvequintic.m

```
function X=solvequintic()
%solves lagrange's quintic equation for X
%for the masses set below:
m1=0;
m2=100;
m3=200;
syms m1a m2a m3a x;
temp=solve(subs((m1a+m2a)*x.^5 + (3*m1a+2*m2a)*x.^4...
            +(3*m1a+m2a)*x.^3 - (m2a+3*m3a)*x.^2...
             - (2*m2a + 3*m3a)*x - (m2a+m3a),...
             [m1a,m2a,m3a],[m1,m2,m3]));
temp=sym2poly(temp);
X=temp(5);
```

A.8 circle.m

```
function[x,y]=circle(xc,yc,radius)
x1=linspace(-radius+xc,radius+xc,500);
x2=linspace(radius+xc,-radius+xc,500);
y1=sqrt(radius^2-(x1-xc).^2)+yc;
y2=-sqrt(radius^2-(x1-xc).^2)+yc;
x=[x1,x2];
y=[y1,y2];
```