Lagrange Points

Ian Wixom* and Andy Shapiro†

(Dated: December 12, 2022)

Abstract: This paper will focus on finding the L1, L2, L3 Lagrange points of the Circular Restricted 3 Body Problem. The numerical analysis of these Lagrange points is not theoretically intensive, so we consider two Regula Falsi based root finding methods, comparing and analysing how each behaves with the given problem setup. Namely: And IRF method described by Soumen Shaw and Basudeb Mukhopadhyay [SM15], and Ridders' Method [Rid79]. We found that Ridders' Method converged slightly faster than IRF, and both performed better with improved bounds.

I. INTRODUCTION

The Three Body Problem is a simple problem statement with no known analytical solution. It can be stated as: Given three celestial bodies in deep space, describe their motion under the effect of each others' gravity. By adding certain conditions to the problem it can be simplified to a restricted case which becomes more easily solvable. One such case is known as the Circular Restricted Three Body Problem (CR3BP). The CR3BP is a special case of the Three Body Problem with the added conditions that:

a) m_1 and m_2 are two bodies in space orbiting circular around their center of mass C with constant angular velocity ω. Within (ξ, η, ζ) space, they orbit on the ξ-η plane.

b) $m_3 \ll m_2 \leq m_1$. Hence m_3 has no gravitational force but experiences the force of the other two bodies.

The Lagrange Points are the equilibrium points of the system, where the net force acting on the point is zero. The aim of this paper is to solve for the L1, L2, L3 Lagrange points using two different root finding numerical methods and comparing convergence results.

II. THEORETICAL MODEL

A detailed description of the derivation of the theoretical model will be left to Appendix A. Here is a brief summary of that process:

Using Newton's Inverse Square Law for gravity, and working off the set up for the 2 Body Problem (Appendix A), the motion of m_3 can be described by:

$$\frac{d^2\mathbf{r}}{dt^2} = -\frac{\mu_1}{\rho_1^3}(\mathbf{r} - \mathbf{r}_1) - \frac{\mu_2}{\rho_2^3}(\mathbf{r} - \mathbf{r}_2)$$

Since the angular velocity is constant, the system can be simplified further by considering it on a rotating frame, resulting in the equation:

$$\mathbf{r}'' + 2\omega \times \mathbf{r}' = -\frac{\mu_1}{\rho_1^3}(\mathbf{r} - \mathbf{r}_1) - \frac{\mu_2}{\rho_2^3}(\mathbf{r} - \mathbf{r}_2) - \omega \times (\omega \times \mathbf{r})$$

See FIG. 1. The Lagrange Points are defined as the rest points of this system, namely where $\mathbf{r}'' = \mathbf{r}' = \mathbf{0}$. Equivalently, they correspond to the points of the system where the gradient of

* iwixom@asuu.utah.edu
† u1166903@umail.utah.edu

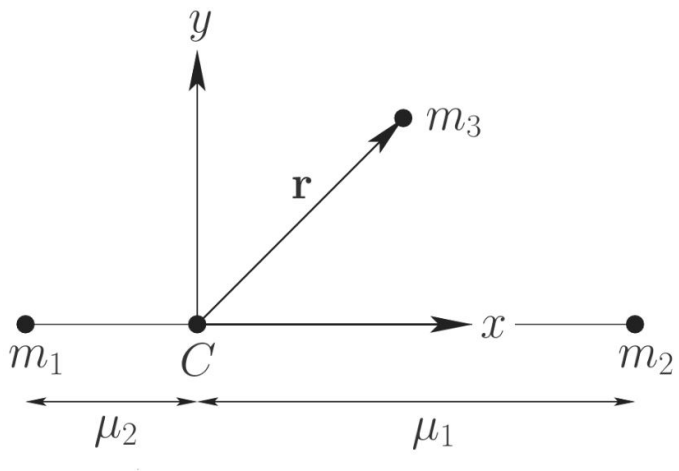


FIG. 1. Rotating Frame for CR3BP [Fit12, pp.154]

the potential energy function for m_3, U, is equal to zero (Appendix A). By setting each RHS component function of the above equation equal to zero we get the desired system of equations:

$$\begin{aligned} x'' - 2\omega y' = 0 &= -\mu_1 \frac{x+\mu_2}{\rho_1^3} - \mu_2 \frac{x-\mu_1}{\rho_2^3} + \omega^2 x \\ y'' + 2\omega x' = 0 &= -\mu_1 \frac{y}{\rho_1^3} - \mu_2 \frac{y}{\rho_2^3} + \omega^2 y \\ z'' = 0 &= -\mu_1 \frac{z}{\rho_1^3} - \mu_2 \frac{z}{\rho_2^3} \end{aligned}$$

Finally, this paper focuses on just the L1, L2, L3 Lagrange points that all lie along the x-axis, meaning $y = z = 0$. The equation is further simplified without loss of generality, by scaling the system s.t. $\omega = 1$ and $\mu_1 + \mu_2 = 1$ (Appendix A). This leaves us with just the one equation left to solve:

$$0 = -\mu_1 \frac{x+\mu_2}{|x+\mu_2|^3} - \mu_2 \frac{x-\mu_1}{|x-\mu_1|^3} + x \tag{2.1}$$

This is what we use to solve for the L1, L2, L3 Lagrange points.

III. NUMERICAL ANALYSIS

A. Improved Regula Falsi Method (IRF)

The classical Regula Falsi method (RF) is the simple iteration scheme: given a real valued, continuous function f defined on an interval $[a, b]$. Let α be a root for f in this interval. RF says to approximate the root using a linear interpolating polynomial between the points $(a, f(a))$ and $(b, f(b))$ and calculating the root of this polynomial, which can be written as:

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{3.1}$$

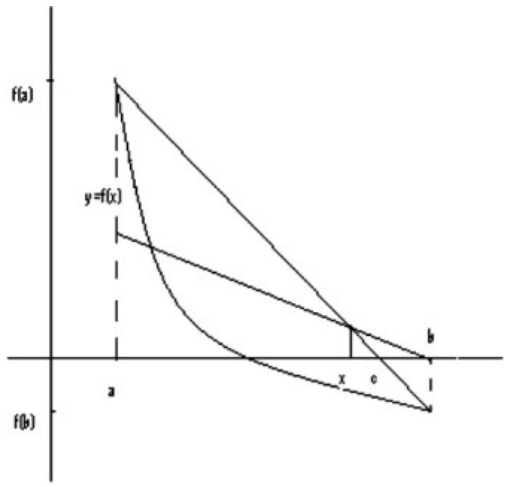


FIG. 2. $f(a)f(c) < 0$, $f(a)f(x) < 0$, so new bounds are updated by setting $b = x$ [SM15]

The iteration continues by repeating the process for the new interval determined as: $[a, c]$ if $f(a)f(c) < 0$ and $[c, b]$ if $f(a)f(c) > 0$.

The Improved Regula Falsi method (IRF) described by Soumen Shaw and Basudeb Mukhopadhyay in [SM15], adds an additional step to improve each approximation. The set up is the same for c as (3.1). If $f(a)f(c) < 0$ (resp. $f(a)f(c) > 0$), then α is between a and c (resp. between c and b). IRF improves the approximation for this root by considering another line through the points $(b, 0)$ and $(a, k * f(a))$ (resp. a line through $(a, 0)$ and $(b, k * f(b))$, where $k \in [0, 1)$. Then a better approximation is given by solving for the intersection between this line and the linear interpolating polynomial mentioned earlier, resulting in the following expressions for an approximation to the root for the $f(a)f(c) < 0$ and $f(a)f(c) > 0$ cases respectively:

$$x_i = \frac{(k-1)bf(a) + af(b)}{(k-1)f(a) + f(b)} \tag{3.2}$$

$$x_i = \frac{(k-1)af(b) + bf(a)}{(k-1)f(b) + f(a)} \tag{3.3}$$

Lastly, to update the bounds and repeat the iteration for the $f(a)f(c) < 0$ case: If $f(x_i)f(a) < 0$, update $b = x_i$. Otherwise if $f(x_i)f(a) > 0$, update $a = x_i$ and $b = c$. For the $f(a)f(c) > 0$ case: If $f(x_i)f(a) < 0$, update $a = c$ and $b = x_i$. Otherwise if $f(x_i)f(a) > 0$, update $a = x_i$.

Assume henceforth that we are in the $f(a)f(c) < 0$ case. Any results that follow can be repeated for the opposite case simply by reversing the roles of $(a, f(a))$ and $(b, f(b))$.

See FIG. 2. Note that by fixing $(a, f(a))$ and $(b, f(b))$, then for $k = 0$, we get $x = c$, and that as $k \to 1$, we get $x \to a$.

Now consider the case that $(a, f(a))$ is fixed, then as $|f(b)| \to \infty$, $\forall$ fixed $k \in [0, 1)$, we get that $x \to a$ (as well as $c \to a$).

Similarly if we instead fix $(b, f(b))$, then as $|f(a)| \to \infty$, $\forall$ fixed $k \in [0, 1)$, we get that $x \to b$ (as well as $c \to b$).

Within the context of (2.1), due to the gravity wells at μ_1 and $-\mu_2$, situations with $|f(a)| >> |f(b)|$ arise naturally for some choices for initial bounds for the L2 and L3 points. In order to avoid having the iteration fall into an inefficient loop of guessing values very close to b, we can pick a value for k to counteract this behavior. If the choice of c shows very little improvement in the

error, we want the choice of x_i to be closer to a, which will require a choice of k close to 1. If the choice of c shows a lot of improvement in the error, we want a choice of x_i to still be close to c, suggesting a choice of k close to zero. Bearing all this in mind, we defined our k as:

$$k_i = \frac{|f(c)| \ (\text{mod} \ |f(b)|)}{|f(b)|} \tag{3.4}$$

Note the mod is used to avoid the case where $|f(c)| > |f(b)|$ leads to $k_i > 1$ breaking the code. Using our own code we recreated the tests listed in [SM15], with the a tolerance of $\epsilon = 1E-10$ for $|f(x_i)| < \epsilon$. The $k = 0$ column represents the classical RF case, the $k = 0.5$ column represents the IRF method described in their paper, the k_i column represents our choice for k described above. We get the following comparison between methods (TABLE I).

TABLE I. IRF Comparison Tests

Equation	**Initial interval**	**Iteration count**			$\alpha \approx$
		$k = 0$	$k = 0.5$	k_i	
$xe^x - 1 = 0$	[-1,1]	22	7	5	0.5671432904
$11x^{11} - 1 = 0$	[0.1,0.9]	36	9	7	0.8041330975
$e^{x^2+7x-30} - 1 = 0$	[2.8,3.1]	38	10	5	3.0000000000
$\frac{1}{x} - sinx + 1 = 0$	[-1.3,-0.5]	14	6	5	-0.6294464841
$x^3 - 2x - 5 = 0$	[2,3]	24	6	3	2.0945514815
$\frac{1}{x} - 1 = 0$	[0.5,1.5]	33	5	4	1.0000000000
$lnx = 0$	[0.5,1.5]	18	6	4	1.0000000000

B. Ridders' Method

Ridders' Method is an altered RF method that adds an exponential component to it. Devised in 1979, the author deems its utility particularly when the function is not strictly monotonic or when other three-point methods fail [Rid79]. Let us consider a function, $f(x)$, such that $H(x) = f(x)e^{\alpha x}$ with three equidistant points that have different ending sign values: $x_0, x_1, x_2 \iff f(x_0)f(x_2) < 0$. Then we can solve for α such that the following equation provides an analytical solution:

$$e^{\alpha(x_1 - x_0)} = \frac{f(x_1) - sign|f(x_0)|\sqrt{f^2(x_1) - f(x_0)f(x_2)}}{f(x_2)}$$

Then, we find the false position approximation for $H(x)$, which can be rewritten for $f(x)$:

$$x_3 = \frac{x_1 H(x_2) - x_2 H(x_1)}{H(x_2) - H(x_1)} = x_2 + sign(x_0 - x_1) * \frac{(x_2 - x_0) * f(x_2)}{\sqrt{f^2(x_2) - f(x_0)f(x_1)}}$$

After solving for x_3, we must update the points for the next iteration. If $f(x_2), f(x_3)$ as well as $f(x_0), f(x_3)$ have the same sign, then we set the $x_0 = x_3$. If the latter has the opposite sign, then $x_1 = x_3$. And if $f(x_2), f(x_3)$ does not have the same sign, then $x_0 = x_2, x_1 = x_3$.

Intuitively, the advantage that Ridders' Method is that since it relies upon linearizing the original function with sign adjustments, the function can handle non-monotonic functions a lot better than just RF. Its rate of convergence is quadratic or better, which means that it could outperform Newton's method which has a rate of convergence at least quadratic [Rid79].

C. Bounds

In order to see how the methods perform searching for the Lagrangian Points, we assessed two types of bounds: one calculated by the gravitational mass ratios, and the other an approximation of the region.

Let us consider the Earth-Moon system. If we are to consider the x-y plane, we should be able to visualize two dots on the same parallel line. If we were to imagine a line running through the dots, we can see the potential spots for where the Lagrangian Points may be located. We predicted that there would be a point between the doubled scaled value of Earth opposite of the moon, $-2U_1$, and the distance opposite of the moon subject to tolerance, $-U_2 - t$. Note that without the tolerance, the function discussed in 2.1 would have a divide by zero error. For the other two, we estimated $(-U_2 + t, U_1 - t), (U_1 + t, 2U_1)$. With the values of $U_1 = 0.9875, U_2 = 1 - U_1, t = 1e - 5$, we ended up with the following boundaries:

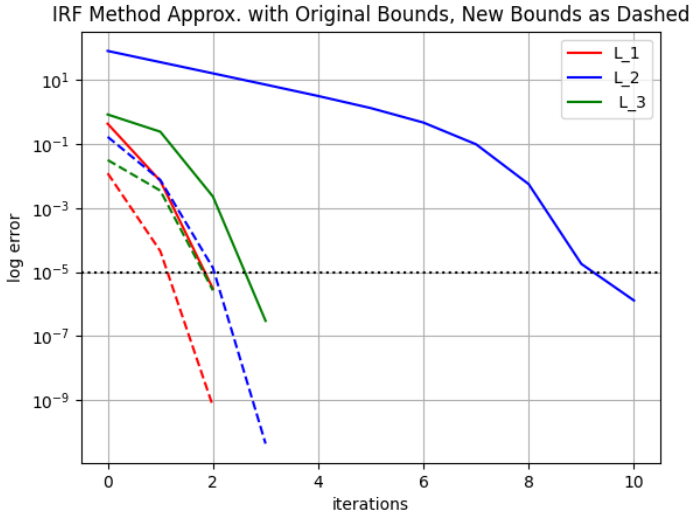
$$[-1.975, -0.01251], [-0.01249, 0.98749], [0.98751, 1.975]$$

As for the second bounds, we chose values that we knew would place the valued point somewhere in the middle of the bounds. We ended up with the following new boundaries:

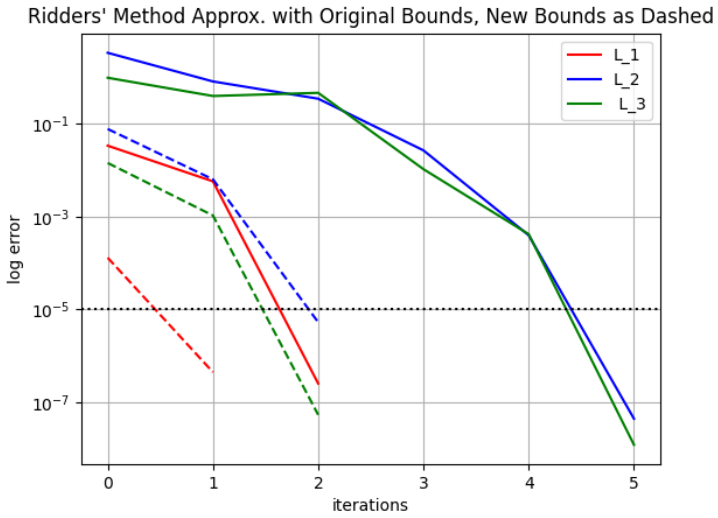
$$[-1.1, -0.9], [0.75, 0.95], [1.05, 1.25]$$

IV. NUMERICAL RESULTS

For the code and more explicit outcomes, please visit the GitHub repository. The first figure discussed will be concerning the IRF's convergence for Lagrangian Points:



We added a dashed line at 10^{-5} to show the threshold of the tolerance. We can see that the method converges for all Lagrangian Points with both bounds. Here is the following results for Ridders':



As for the output for both methods from our code, we will use an example of the first bound with the original bounds for both IRF and Ridders:

Iterations	Error	IRF Root
1	0.4343624190774859	-1.1716872496655637
2	0.007113823074387277	-1.0074225029231545
3	3.035352130465168e-06	-1.0050634064454325

Iterations	Error	Ridders Root
1	0.0340183317412378	-0.9939302563676082
2	0.005790807140870625	-1.0069827255526045
3	2.544135598993129e-07	-1.005062317616125

Any more output can be retrieved from Appendix B or the GitHub after running the script in command-line.

V. CONCLUSION/DISCUSSION

What we found was that while both methods perform similarly, The L_2 for IRF under the original bounds took longer to converge. As was discussed before under Section III Subsection B, there are conditions that can cause an oscillatory behavior between the bounds. This leads to the method requiring more iterations. We also see a fairly rigorous improvement in both methods when changing the bounds to be closer to where the effective potential equals 0. While for all Lagrangian Points the Ridders' converged more quickly than IRF, there is still discussion to be had of the amount of computations required for each method. In regards to IRF, the value k can be a topic of research as there is no conclusive way to define the variable as of yet.

AUTHORS' CONTRIBUTIONS

Andy Shapiro: I, II, III.A, Appendix A

Ian Wixom: III.B-C, IV, V, Appendix B

SOFTWARE AVAILABILITY

The code can be accessed at the following link: https://github.com/1anw/5160-Lagrangian-NumericalAnalysis

Appendix A: Celestial Mechanics Background

A.1 Two Body Problem

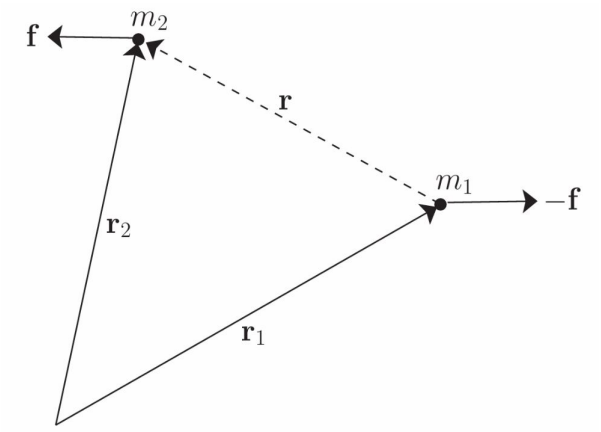


Figure A.1 [Fit12, pp.18]

Newton's Second Law:

$$m_1 \frac{d^2\mathbf{r}_1}{dt^2} = \mathbf{f}_{2\to1} = -\mathbf{f} \tag{A1}$$

$$m_2 \frac{d^2\mathbf{r}_2}{dt^2} = \mathbf{f}_{1\to2} = \mathbf{f}$$

$$\mathbf{r}_{cm} = \frac{m_1\mathbf{r}_1 + m_2\mathbf{r}_2}{m_1 + m_2}$$

$$\Rightarrow \mathbf{r}_1 = \mathbf{r}_{cm} - \frac{m_2(\mathbf{r}_2 - \mathbf{r}_1)}{m_1 + m_2} = \mathbf{r}_{cm} - \frac{m_2}{m_1 + m_2}\mathbf{r}$$

Since in an isolated system the center of mass does not accelerate, $\frac{d^2\mathbf{r}_{cm}}{dt^2} = 0$. Hence,

$$\frac{d^2\mathbf{r}_1}{dt^2} = \frac{d^2\mathbf{r}_{cm}}{dt^2} - \frac{m_2}{m_1 + m_2}\frac{d^2\mathbf{r}}{dt^2} = -\frac{m_2}{m_1 + m_2}\frac{d^2\mathbf{r}}{dt^2} \tag{A2}$$

Combining (A2) and (A1) we get:

$$\mathbf{f} = -m_1(-\frac{m_2}{m_1+m_2}\frac{d^2\mathbf{r}}{dt^2}) = \frac{m_1 m_2}{m_1+m_2}\frac{d^2\mathbf{r}}{dt^2} \tag{A3}$$

[Fit12, pp. 17-18]

A.2 Binary Star System

Inverse square law applied to gravity yeilds:

$$\mathbf{f}_{1\to 2} = -\frac{Gm_1m_2}{||\mathbf{r}||^3}\mathbf{r} \tag{A4}$$

Setting (A3) and (A4) equal to each other we get:

$$\frac{m_1 m_2}{m_1+m_2}\frac{d^2\mathbf{r}}{dt^2} = -\frac{Gm_1m_2}{||\mathbf{r}||^3}\mathbf{r}$$

$$\Rightarrow \frac{d^2\mathbf{r}}{dt^2} = -\frac{G(m_1+m_2)}{||\mathbf{r}||^3}\mathbf{r} \tag{A5}$$

Thus, for constant angular velocity ω and position vector $\mathbf{r} = (rcos\omega t, rsin\omega t, 0)$, by taking the double derivative we get:

$$\frac{d^2\mathbf{r}}{dt^2} = -\omega^2(rcos\omega t, rsin\omega t, 0) = -\omega^2\mathbf{r} \tag{A6}$$

Combining (A6) with (A5) we get:

$$-\omega^2\mathbf{r} = -\frac{G(m_1+m_2)}{||\mathbf{r}||^3}\mathbf{r}$$

$$\Rightarrow \omega^2 = \frac{G(m_1+m_2)}{||\mathbf{r}||^3} \tag{A7}$$

If we set the center of mass to be at the origin, i.e. $\mathbf{r}_{cm} = \mathbf{0}$, then we get:

$$\mathbf{0} = \frac{m_1\mathbf{r}_1 + m_2\mathbf{r}_2}{m_1+m_2}$$

$$\Rightarrow 0 = -m_1||\mathbf{r}_1|| + m_2||\mathbf{r}_2||$$

$$\Rightarrow \frac{||\mathbf{r}_1||}{||\mathbf{r}_2||} = \frac{m_2}{m_1} \tag{A8}$$

[Fit12, pp. 56]

A.3 CR3BP

Refer to Figure A.1. To simplify the process, assume the distance between m_1 and m_2, shown as a in the figure, is given in units s.t. $a = 1$. Similarly, assume the units for m_1 and m_2 are given s.t. $G(m_1 + m_2) = 1$. Applying this to (A7) we get that $\omega = 1$ as well.

Let $\mu_1 = Gm_1$ and $\mu_2 = Gm_2 = 1 - \mu_1$. Then from (A8), and since $r_1 + r_2 = \mu_1 + \mu_2 = 1$, it follows that: $r_1 = \mu_2$ and $r_2 = \mu_1$. Now we can define $\mathbf{r}_1$ and $\mathbf{r}_2$ by:

$$\mathbf{r}_1 = (\xi_1, \eta_1, 0) = (-\mu_2 cos\omega t, -\mu_2 sin\omega t, 0) \tag{A9}$$

$$\mathbf{r}_2 = (\xi_2, \eta_2, 0) = (\mu_1 cos\omega t, \mu_1 sin\omega t, 0) \tag{A10}$$

m_3 has position vector $\mathbf{r} = (\xi, \eta, \zeta)$

We can use the work shown in **A.1** to model the acceleration of m_3 due to m_1 and m_2. In the case of m_1, the vector from m_1 to m_3 is given by $(\mathbf{r} - \mathbf{r}_1)$ and let $||(\mathbf{r} - \mathbf{r}_1)|| = \rho_1$, for m_2 it is $(\mathbf{r} - \mathbf{r}_2)$ with $||(\mathbf{r} - \mathbf{r}_2)|| = \rho_2$. Applying both vectors to (A5) we get:

$$\frac{d^2\mathbf{r}}{dt^2} = -\frac{\mu_1}{\rho_1^3}(\mathbf{r} - \mathbf{r}_1) - \frac{\mu_2}{\rho_2^3}(\mathbf{r} - \mathbf{r}_2) \tag{A11}$$

Hence, for each Cartesian component we get:

$$\frac{d^2\xi}{dt^2} = -\mu_1\frac{\xi - \xi_1}{\rho_1^3} - \mu_2\frac{\xi - \xi_2}{\rho_2^3} \tag{A12}$$

$$\frac{d^2\eta}{dt^2} = -\mu_1\frac{\eta - \eta_1}{\rho_1^3} - \mu_2\frac{\eta - \eta_2}{\rho_2^3} \tag{A13}$$

$$\frac{d^2\zeta}{dt^2} = -\mu_1\frac{\zeta}{\rho_1^3} - \mu_2\frac{\zeta}{\rho_2^3} \tag{A14}$$

[Fit12, pp. 147-148]

A.4 Rotating Frame

We have that the velocity of an object on a non-rotating frame is equal to the velocity of a rotating frame added with the velocity of the object on the rotating frame. Written as an equation of $\mathbf{r}$ this gives us:

$$\frac{d\mathbf{r}}{dt} = \frac{d\mathbf{r}}{dt'} + \omega \times \mathbf{r} \tag{A15}$$

Here, ω is thought of as the vector $(0, 0, \omega)$. The derivative of an object in circular orbit is equal to the position vector of that object crossed with the angular velocity multiplying a unit vector in the direction normal to the plane of circular orbit. In this case, $\mathbf{r}$ is orbiting normal to

the vector (0,0,1) so we end up with the second term on the right of the equation representing the velocity of the frame.

This gives us a way to relate the derivative of $\mathbf{r}$ on the non-rotating frame $\frac{d}{dt}$ to the derivative of $\mathbf{r}$ on the rotating frame $\frac{d}{dt'}$:

$$\frac{d}{dt} \equiv (\frac{d}{dt'} + \omega\times) \tag{A16}$$

Applying (A16) to (A15) a second time gives us:

$$\frac{d^2\mathbf{r}}{dt^2} = (\frac{d}{dt'} + \omega\times)(\frac{d\mathbf{r}}{dt'} + \omega \times \mathbf{r})$$

$$\frac{d^2\mathbf{r}}{dt^2} = \frac{d^2\mathbf{r}}{dt'^2} + 2\omega \times \frac{d\mathbf{r}}{dt'} + \omega \times (\omega \times \mathbf{r}) \tag{A17}$$

[Fit12, pp. 72-73]

A.5 Co-Rotating Frame CR3BP

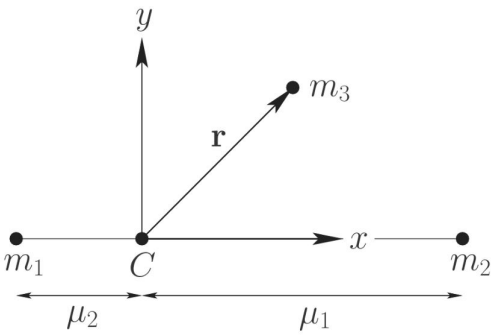


Figure A.2 [Fit12, pp.154]

Let us now consider a rotating frame with coordinate system (x, y, z) where m_1 and m_2 are now fixed to be on the x-axis and the rotation of the frame is about the z-axis, with $\omega = (0, 0, \omega)$. Let the position vector for m_3 now be represented by $\mathbf{r} = (x, y, z)$, the position vector for m_1 be $\mathbf{r}_1 = (-\mu_2, 0, 0)$, and the position vector for m_2 be $\mathbf{r}_2 = (\mu_1, 0, 0)$

By setting (A11) equal to (A17) we get:

$$-\frac{\mu_1}{\rho_1^3}(\mathbf{r} - \mathbf{r}_1) - \frac{\mu_2}{\rho_2^3}(\mathbf{r} - \mathbf{r}_2) = \mathbf{r}'' + 2\omega \times \mathbf{r}' + \omega \times (\omega \times \mathbf{r})$$

Here, $2\omega \times \mathbf{r}'$ is called the Coriolis Acceleration and $-\omega \times (\omega \times \mathbf{r})$ is called the Centrifugal Acceleration. Re-ordering we get a more convenient form:

$$\mathbf{r}'' + 2\omega \times \mathbf{r}' = -\frac{\mu_1}{\rho_1^3}(\mathbf{r} - \mathbf{r}_1) - \frac{\mu_2}{\rho_2^3}(\mathbf{r} - \mathbf{r}_2) - \omega \times (\omega \times \mathbf{r}) \tag{A18}$$

Note $2\omega \times \mathbf{r}' = (-2\omega y', 2\omega x', 0)$ and $\omega \times (\omega \times \mathbf{r}) = (-\omega^2 x, -\omega^2 y, 0)$. Hence each component can be written as:

$$x'' - 2\omega y' = -\mu_1 \frac{x + \mu_2}{\rho_1^3} - \mu_2 \frac{x - \mu_1}{\rho_2^3} + \omega^2 x \tag{A19}$$

$$y'' + 2\omega x' = -\mu_1 \frac{y}{\rho_1^3} - \mu_2 \frac{y}{\rho_2^3} + \omega^2 y \tag{A20}$$

$$z'' = -\mu_1 \frac{z}{\rho_1^3} - \mu_2 \frac{z}{\rho_2^3} \tag{A21}$$

The potential energy of gravitational and centrifugal forces is represented by:

$$U(x, y, z) = -\frac{\mu_1}{\rho_1} - \frac{\mu_2}{\rho_2} - \frac{\omega^2}{2}(x^2 + y^2) \tag{A22}$$

It follows that:

$$x'' - 2\omega y' = -\frac{\partial U}{\partial x}$$

$$y'' + 2\omega x' = -\frac{\partial U}{\partial y}$$

$$z'' = -\frac{\partial U}{\partial z}$$

[Fit12, pp. 153-154]

Appendix B: Codes

```
# irf.py
#   Improved Regula Falsi method by Soumen Shaw and Basudeb Mukhopadhyay, with parameter k = f
#   coded in python
# By: Andy Shapiro and Ian Wixom

import math
import matplotlib.pyplot as plt

# Approximate Earth-Moon system scaled values for CR3BP model
U_1 = 0.98785
U_2 = 1 - U_1


def f(x):
    return x - U_1 * (x + U_2) / (abs(x + U_2)) ** 3 - U_2 * (x - U_1) / (abs(x - U_1)) ** 3
```

```
def method(array, tolerance):
    lagrange = [[], [], []]

    for i in range(3):
        a = array[i][0]
        b = array[i][1]

        f_a = f(a)
        f_b = f(b)

        count = 0
        x = a
        error = abs(f_a)

        data = []
        print("Bound {}:".format(i+1))
        while error > tolerance:
            c = (a * f_b - b * f_a) / (f_b - f_a)
            f_c = f(c)

            if f_a * f_c < 0:
                k = (abs(f_c) % abs(f_b)) / (abs(f_b))

                x = ((k - 1) * b * f_a + a * f_b) / ((k - 1) * f_a + f_b)
                f_x = f(x)

                if f_a * f_x < 0:
                    b = x
                    f_b = f_x
                else:
                    a = x
                    f_a = f_x

                    b = c
                    f_b = f_c
            else:
                k = (abs(f_c) % abs(f_a)) / (abs(f_a))

                x = ((k - 1) * a * f_b + b * f_a) / ((k - 1) * f_b + f_a)
                f_x = f(x)

                if f_a * f_x < 0:
                    a = c
                    f_a = f_c

                    b = x
                    f_b = f_x
                else:
```

```
                    a = x
                    f_a = f_x

            count = count + 1
            error = abs(f_x)
            lagrange[i].append(error)

            print("Iteration {}: error = {} at x_3 = {}".format(count, error, x))

    print(lagrange)
    return lagrange


def main():
    tolerance = 1E-5

    bounds_1 = [[-2*U_1, -U_2 - tolerance], [-U_2 + tolerance, U_1 - tolerance], [U_1 + tolera
    bounds_2 = [[-1.1, -0.9], [0.75, 0.95], [1.05, 1.25]]
    print("original, guessed bounds: ")
    original_result = method(bounds_1, tolerance)
    print("approximated bounds: ")
    bounded_result = method(bounds_2, tolerance)

    plt.plot(original_result[0], color = 'r')
    plt.plot(original_result[1], color = 'b')
    plt.plot(original_result[2], color = 'g')

    plt.legend(["L_1", "L_2", " L_3"])

    plt.title('IRF Method Approx. with Original Bounds, New Bounds as Dashed')

    plt.plot(bounded_result[0], color =  'r', ls = '--')
    plt.plot(bounded_result[1], color = 'b', ls = '--')
    plt.plot(bounded_result[2], color = 'g', ls = '--')

    plt.axhline(y = 1e-5, color = 'k', linestyle = ':')
    plt.grid(True)

    plt.xlabel('iterations')
    plt.ylabel('log error')
    plt.yscale('log')

    plt.show()
main()

"""
ridder.py
    Ridder method by C.J.F Ridder, with false position method determined by f(x)
    Coded in Python
```

```
By: Andy Shapiro and Ian Wixom
"""

import math
import numpy as np
import matplotlib.pyplot as plt
from numpy import sign

# Approximate Earth-Moon system scaled values for CR3BP model
U_1 = 0.98785
U_2 = 1 - U_1


def f(x):
    return x - U_1 * (x + U_2) / (abs(x + U_2)) ** 3 - U_2 * (x - U_1) / (abs(x - U_1)) ** 3

def method(array, tolerance):
    lagrange = [[], [], []]

    for i in range(3):
        error = 1000
        x_0 = array[i][0]
        x_1 = array[i][1]
        x_2 = 0
        x_3 = 0
        t = 0
        print("Bound {}:".format(i+1))
        while(error > tolerance and t < 50):
            x_2 = (x_0 + x_1)/2

            fx_0 = f(x_0)
            fx_1 = f(x_1)
            fx_2 = f(x_2)
            d = (math.sqrt(fx_2**2 - fx_0*fx_1))
            if d == 0:
                return None
            dx = (x_2 - x_0)*fx_2/d
            if (x_0 - x_1) < 0.0: dx = -dx
            x_3 = x_2 + dx
            fx_3 = f(x_3)

            if sign(fx_2) == sign(fx_3):
                if sign(fx_0) != sign(fx_3):
                    x_1 = x_3
                else:
                    x_0 = x_3
            else:
                x_0 = x_2
                x_1 = x_3
```

```
            error = abs(f(x_3))
            print("Iteration {}: error = {} at x_3 = {}".format(t+1, error, x_3))
            t += 1
            lagrange[i].append(error)

        print("val of root: ", x_3)
    return lagrange

def main():
    tolerance = 1E-5

    bounds_1 = [[-2*U_1, -U_2 - tolerance], [-U_2 + tolerance, U_1 - tolerance], [U_1 + tolera
    bounds_2 = [[-1.1, -0.9], [0.75, 0.95], [1.05, 1.25]]
    print("original, guessed bounds: ")
    original_result = method(bounds_1, tolerance)
    print("approximated bounds: ")
    bounded_result = method(bounds_2, tolerance)

    plt.plot(original_result[0], color = 'r')
    plt.plot(original_result[1], color = 'b')
    plt.plot(original_result[2], color = 'g')

    plt.legend(["L_1", "L_2", " L_3"])

    plt.title('Ridders\' Method Approx. with Original Bounds, New Bounds as Dashed')

    plt.plot(bounded_result[0], color =  'r', ls = '--')
    plt.plot(bounded_result[1], color = 'b', ls = '--')
    plt.plot(bounded_result[2], color = 'g', ls = '--')

    plt.axhline(y = 1e-5, color = 'k', linestyle = ':')
    plt.grid(True)

    plt.xlabel('iterations')
    plt.ylabel('log error')
    plt.yscale('log')

    plt.show()

main()
```

[Fit12] Richard Fitzpatrick. *An Introduction to Celestial Mechanics*. Cambridge University Press, New York, 2012.

[Rid79] C.J.F. Ridders. A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, 26(11):979–980, 1979.

[SM15] Soumen Shaw and Basudeb Mukhopadhyay. An improved regula falsi method for finding simple roots of nonlinear equations. *Applied mathematics and computation*, 254:370–374, 2015.