Resonant Forcing of L-Point Orbits

Geoffrey Brown

Master of Science in
Space Technology and Planetary Exploration
from the
University of Surrey

Department of Electronic Engineering
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK

August 2009
Supervised by: Dr Phil Palmer

©Geoffrey Brown 2009
DECLARATION OF ORIGINALITY

I confirm that the project dissertation I am submitting is entirely my own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of my report to the JISC anti-plagiarism software resource, I confirm that my work does not contravene the university regulations on plagiarism as described in the Student Handbook. In so doing I also acknowledge that I may be held to account for any particular instances of uncited work detected by the JISC anti-plagiarism software, or as may be found by the project examiner or project organiser. I also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then I may forfeit any credit for this module or a more severe penalty may be agreed.

Resonant Forcing Of L-Point Orbits

Geoffrey Brown

Author Signature       Date: 12/08/2009

Supervisor’s name: Dr Phil Palmer
WORD COUNT

Number of Pages: 100
Number of Words: 13733
ABSTRACT

This project will investigate low energy transfer orbits for the exploration of the Jupiter and Saturn moon systems. These transfers exploit the instability of periodic orbits about the Lagrange points. The moon systems, however, have resonant coupling between the orbits of the moons which causes a resonant forcing on periodic L point orbits. The project will consider the Circular Restricted Three Body Problem and produce a C# program to model this system. The project will also assess methods for modelling the impact of resonant coupling on periodic L point orbits.
# TABLE OF CONTENTS

Declaration of originality.................................................................ii
Word Count.........................................................................................iii
Abstract............................................................................................iv
Table of Contents .............................................................................v
List of Figures .................................................................................viii

1 INTRODUCTION ...........................................................................9
  1.1 Background and Context.................................................................9
  1.2 Scope and Objectives...................................................................10
    1.2.1 Objectives .............................................................................10
  1.3 Achievements...............................................................................10
  1.4 Overview of Dissertation..............................................................11

2 STATE OF THE ART .................................................................12
  2.1 Introduction.................................................................................12
  2.2 Resonance in the solar system.......................................................12
    2.2.1 Jupiter and Saturn Moon Systems.............................................13
  2.3 Orbital Mechanics and the Three-Body Problem............................13
  2.4 Circular Restricted Three Body Problem.........................................14
    2.4.1 State space & Surface of Section............................................15
    2.4.2 Location of the equilibrium points in the CR3BP.......................16
  2.5 Periodic Orbits around the L-points...............................................19
    2.5.1 Halo Orbit ............................................................................20
    2.5.2 Stable and Unstable Manifolds...............................................20
    2.5.3 Linearization around the equilibrium points..........................21
  2.6 State Of The Art Summary...........................................................23

3 CR3BP MATLAB MODELLING ....................................................24
  3.1 Introduction..................................................................................24
    3.1.1 MATLAB Code....................................................................24
      3.1.1.1 CR3BP Playground & Poincaré Section...............................25
      3.1.1.2 Computation of Halo orbits.............................................26
      3.1.1.3 Stable Manifolds...............................................................27
  3.2 MATLAB Modelling Summary....................................................28

4 GDB CR3BP PROGRAM ...............................................................29
  4.1 Introduction..................................................................................29
  4.2 Requirements..............................................................................29
4.2.1 User Interface & Output ................................................................. 30
4.2.2 Functionality .................................................................................. 30
4.2.3 Documentation .............................................................................. 31
4.3 Design Philosophy ........................................................................... 32
4.3.1 Introduction .................................................................................... 32
4.3.2 Software development cycle .......................................................... 32
4.4 Design & Implementation ............................................................... 33
4.4.1 User Interface (UI) ........................................................................ 33
  4.4.1.1 Main Form .................................................................................. 33
  4.4.1.2 Graphical Display Options ....................................................... 34
4.4.2 Variables, Structures & Functions .................................................. 35
4.4.3 Native Class .................................................................................... 35
  4.4.3.1 Matrix options for C# programming .......................................... 36
4.4.4 Native Class Functions ................................................................. 37
  4.4.4.1 Libration points ......................................................................... 37
  4.4.4.2 Jacobi Energy ............................................................................ 37
  4.4.4.3 Halo Orbit Calculation ............................................................... 37
  4.4.4.4 Halo Orbit Linearised Solution ................................................ 38
  4.4.4.5 Resonant Forcing modelling .................................................... 38
4.4.5 MATLAB Engine Class .............................................................. 38
  4.4.5.1 Linking C# with MATLAB ....................................................... 38
  4.4.5.2 MATLAB Engine Class Implementation ................................ 39
4.5 GDB CR3BP Program Summary ..................................................... 39
5 DISCUSSION OF RESONANT FORCING ........................................ 40
  5.1 Introduction ...................................................................................... 40
  5.2 Forced Vibration .............................................................................. 40
    5.2.1 Forced Pendulum ........................................................................ 40
    5.2.2 The Perturbation of a Halo Orbit around L1 ................................... 41
  5.3 Resonant Satellite Orbits ................................................................. 43
  5.4 The Four Body Problem ................................................................. 43
  5.5 Resonant Forcing Discussion Summary .......................................... 44
6 RESULTS & DISCUSSION ................................................................. 45
  6.1 Introduction ...................................................................................... 45
  6.2 C# program Validation ................................................................... 45
    6.2.1 Validation ................................................................................... 45
      6.2.1.1 Libration Points .................................................................... 45
      6.2.1.2 Jacobi Energy ...................................................................... 47
LIST OF FIGURES

Figure 1: Illustration of Io-Europa-Ganymede resonance. From the centre outwards: Io (yellow), Europa (gray) and Ganymede (dark) [7] ................................................... 13

Figure 2: The rotating coordinate system [5] ........................................................................ 16

Figure 3: Jacobi Integral [4] ................................................................................................. 17

Figure 4: Realms of possible motion [5] ................................................................................ 18

Figure 5: Equilibrium points in the CR3BP [5] ....................................................................... 19

Figure 6: The low-Energy Exploration Concept for transfer between L1 and L2 .................. 21

Figure 7: Linearized Solutions for a Halo orbit around L1 [5] ................................................ 21

Figure 8: ISEE3 Mission orbit around L1 [5] ......................................................................... 22

Figure 9: All the periodic and quasi-periodic orbits around L2 shown on a Poincaré section of the ecliptic plane [9] .................................................................................. 23

Figure 10: Surface of Section for the Jupiter-Io system ....................................................... 26

Figure 11: Halo orbits around L2 in the Jupiter-Io System [Please note that the Earth (M1) is not shown on this figure] .................................................................................. 27

Figure 12: Stable manifold from L1 in the Earth-Moon System [Please note that the Earth (M1) is not shown on this figure] ........................................................................... 28

Figure 13: Waterfall Software Development Method ............................................................... 32

Figure 14: Iterative Software Development Method ............................................................... 33

Figure 15: Main Form ............................................................................................................ 34

Figure 16: Simple diagram of impact of fourth body depending on the Sun-Earth system .... 42

Figure 17: Concentric Circular Model [5] .............................................................................. 44

Figure 18: L-Points output Screen .......................................................................................... 47

Figure 19: Halo Orbit plot (X-Y axes) .................................................................................... 48

Figure 20: Halo Orbit plot (X-Z axes) .................................................................................... 48

Figure 21: Halo Orbit plot (Y-Z axes) .................................................................................... 49

Figure 22: Linearised solutions output, Quasi-halo orbit ....................................................... 49

Figure 23: Linearised solution output, Halo orbit .................................................................. 50

Figure 24: The GDB CR3BP Program ................................................................................... 60

Figure 25: Choose Mass Parameter, Mu ................................................................................ 61

Figure 26: Choose Output ....................................................................................................... 61

Figure 27: Choose Engine ..................................................................................................... 62

Figure 28: Output Tabs .......................................................................................................... 62
1 INTRODUCTION

This document is the final report for the MSc project titled Resonant Forcing of L-point orbits, it serves to document the progress made and report any findings and conclusions that have been developed. A Compact Disc (CD) can be found inside the back cover of this report, it includes the program designed and implemented as part of this project. For more information about this CD, please see Appendix 6.

1.1 Background and Context

Since Newton first formulated his theory of gravity in 1687 physicists and engineers have attempted to understand how the theory can be modelled in more and more complex systems. Kepler’s laws of motion provide one example of this, where the interaction of two bodies leads to three well defined laws for the motion of one mass around the other. Solutions to the so called two body problem are easily derivable usually by treating both masses as one-body systems. However, as we increase the complexity to a three body system the problem becomes unsolvable except in a few special cases, for which the Circular Restricted Three Body Problem is one such solution, this is the key concept used for this project and further discussion is provided in section 2.4.

Interplanetary missions require a great deal of planning to understand the orbital parameters of missions to, for example, Jupiter or Saturn. In general a patched conic method is used that allows the problem to be split into more manageable parts. Up to the present, most interplanetary missions have used large velocities and fly-by techniques to be able to reach a planet-moon system and then use more thrust and fly-by techniques to modify the orbit within the system. However, in complex moon systems there is also an impact of the gravitational attraction of the moons and this effect can be used to modify orbits or assist in transferring from one orbit to another.

The concept of low-energy space exploration will be considered in this project, where the use of L-points and stable and unstable orbits will be considered as an alternative to hitherto used fly-by and gravity assist manoeuvres.

Future planned missions to Jupiter and Saturn, namely Laplace [1] and TanDEM [2] respectively, will use the tried and tested methods as before, this project will attempt to quantify whether a low-energy exploration has any advantages for this type of mission and what disadvantages this brings.

Finally, the concept of resonant forcing will be introduced and a number of approaches discussed to attempt to formulate a suitable method to model this area of interest.
1.2 Scope and Objectives

The purpose of this project is to understand the dynamics of a spacecraft within the three body problem and to assess the impact of a resonant fourth mass within this problem. All relevant theory is presented in Section 2 and the objectives of the project are detailed in section 1.2.1.

1.2.1 Objectives

The key objectives of this project are listed in bullet point below:

- Understand theory of 3 body problem and orbital mechanics
- Understand & Model CR3BP, Lagrange Points, Periodic orbits
- Model spacecraft dynamics and energy to reach L-points
- Model periodic orbits around L-points
- Assess methods for modelling resonant forcing of a satellite
- Implementation of a C# program to model the CR3BP, Lagrange Points, Periodic Orbits and Resonant Forcing

1.3 Achievements

A full list of achievements is found in this section, each referenced to the relevant section of this report:

- Literature review completed as documented in Section 2
- Initial explorative modelling using MATLAB completed, see Section 3
- Planning of the project has been an on-going process, for the project plan for the summer semester, see Appendix 1
- March Submission [3] & Viva completed
- Production of an A3 sized poster has been completed & presented, see Appendix 2
- Re-implementation of MATLAB CR3BP code to C#, see Section 4
- Consideration of a fourth mass at resonance and methods for modelling of resonant forcing, Section 5
- Final report documentation
- CD produced including setup program and relevant files for C# GDB CR3BP pro-
1.4 Overview of Dissertation

The following sections are as follows; firstly, in Section 2, a discussion of the current theory relating to this project is introduced, following that, in Section 3, a discussion of some of the initial modelling using MATLAB is included, including some of the initial results that were produced. In Section 4, the design and implementation of the C# program is considered, and in Section 5, resonant forcing is discussed. Results are presented and discussed in Section 6, and finally, conclusions are found at Section 7.
2 STATE OF THE ART

2.1 Introduction

In this section the key theory required to understand the three body problem and its solution are documented, this is based on the literature survey undertaken by the Author. The theory of the three body problem is introduced in Section 2.4, before that, in Section 2.2 the Author presents a section considering resonance in the solar system and if there are any systems where we might see good examples of resonant forcing.

There is a wealth of possible reference material for this area of interest and a number of different sources have been researched in the course of this project. However, there are two references in particular that have been used as the main source of information for this project, namely the books Orbital Motion by A. Roy [4] and Dynamical Systems by Koon et al [5].

2.2 Resonance in the solar system

The question of stability in our solar system is one of mathematics great problems. The two-body approximation gives stable orbits but does not include the influence of the many other bodies in the solar system. These other effects may be very small but over time may change orbital parameters so that the stable two body motion becomes unstable! Laplace discovered the resonance of the Jovian moons (see Section 2.2.1) but there is still a great deal of work to be done to fully understand how masses affect each other in our solar system. In addition to the moon systems of Jupiter and Saturn there are other examples of the impact of resonance in the solar system.

In general, orbital resonance may involve any of the orbit parameters (as defined by Kepler's Laws) and may act on any time scale, be it on the scale of the orbital period up to much longer periods of time (e.g. up to hundreds of years!). The impact of orbital resonance may lead to long term stabilization or cause an orbit to destabilise. There is evidence that the resonant orbits of the outer planets is the reason for a time in the Earth’s early history when the planet was hit by many thousands of asteroids that were pushed out of there stable orbits by the resonant force caused by the gas giants [6]. The Kirkwood gaps provide evidence for this theory, showing that with respect to the orbit of Jupiter there are gaps in the distribution of asteroids at the 3:1, 5:2, 7:3 and 2:1 resonances, however, this is not true in all cases. The mechanism thought to cause these gaps will be discussed in more detail in Section 5.
2.2.1 Jupiter and Saturn Moon Systems

Before moving on to the dynamics involved in the three body problem, a brief discussion of examples of the application of this project is presented. For this project we are interested in locations where we might see resonant forcing and for this reason we are particularly interested in the gas giants.

The Jovian moons are the most obvious example of where resonant forcing could have a major impact on a spacecraft in orbit near a libration point. This is due to the so-called Laplace resonance which in simple terms indicates that the moons have a simple integer ratio between their orbital periods. The Jovian moons Io, Europa and Ganymede are an excellent example of this kind of relation, having a 1:2:4 resonance respectively and Figure 1 illustrates the positions of the moons after 1, 2 and 3 Io periods.

![Figure 1: Illustration of Io-Europa-Ganymede resonance. From the centre outwards: Io (yellow), Europa (gray) and Ganymede (dark) [7]](image)

This system will form the basis for the experiments to be modelled in the programming part of this project where a spacecraft within this system will be perturbed by the movement of the moons. The question is how big is this effect caused by the moons and can the spacecraft take advantage of these perturbations?

The moons of Saturn are another example of where resonance could have a major impact on the orbit of a spacecraft. However, there are many more moons in the system than the Jovian example and no Laplace resonance exists, this system will be assessed if time permits.

2.3 Orbital Mechanics and the Three-Body Problem

The three body problem was originally formulated by Newton; it can be stated as follows:

\[\text{Given at any time the positions and velocities of three or more massive particles moving under their mutual gravitational forces, the masses also being known, calculate their positions and velocities for any other time.} \]
By using the theory of gravitation a number of useful statements can be made regarding the systems, which are known as the ten integrals of motion. These mathematical solutions assess the system by conserving Linear Momentum and Angular Momentum which provide nine of these integrals, with the final integral given by the conservation of energy. For more detail of the derivation of these Integrals, it is recommended that the reader consults [4] pp 102-105.

The general three-body problem is impossible to solve fully, and the complexity that the problem involves has perplexed many of the finest mathematical minds for many years! However, the problem can be simplified by making a number of key assumptions that mimic the real systems we observe in the solar system; the next section will explain one such system where we restrict the motion of two of the bodies to circular orbits around their common centre of mass.

2.4 Circular Restricted Three Body Problem

The general three body problem can be simplified by assuming that we limit the movement of two of the bodies to circular orbits around their common centre of mass and that we assume the third body (e.g. the spacecraft) is too small to affect the motion of the two massive bodies. The massive bodies are termed the primaries of the system, or primary and secondary is more suitable for this study (e.g. Jupiter as the primary, one of the Jovian moons as the secondary). The problem is then to solve the motion of the spacecraft, and this is the Circular Restricted Three Body Problem, or CR3BP. We can now re-write the problem as:

Consider the motion of a particle P of negligible mass moving under the gravitational influence of two masses $m_1$ and $m_2$. Assume that $m_1$ and $m_2$ have circular orbits around their common centre of mass. The particle P is free to move in the plane defined by the circular orbits of the primaries, but cannot affect their motion. [5]

The advantage of reducing the problem to that of the CR3BP is that the number of differential equations, or in other words the number of integrals of motion, that need to be solved is reduced from eighteen second order differential equations in the general three body problem to just six second order differential equations in the CR3BP. It is also possible to reduce this problem even further by limiting the movement of the small mass to the plane of the orbits of the two primaries, and this problem is known as the Planar Circular Restricted Three Body problem, or PCR3BP.

Let us now take a moment to consider the CR3BP and some of the useful assumptions that are made to solve the problem of motion of the small spacecraft. Firstly, we make the system non-dimensional by carefully choosing the units of measurement, for simplicity this is pre-
presented in bullet form:

- Unit of mass is taken as \( m_1 + m_2 \)
- Unit of length is the common separation of the two primaries
- Unit of time is set so that the orbital period of the two primaries around their common centre of mass is \( 2\pi \)
- The gravitational constant, \( G \) then becomes \( G = 1 \)

With these assumptions made, the only parameter of the system is the mass parameter, \( \mu \):

\[
\mu = \frac{m_2}{m_1 + m_2}
\]

We assume that \( m_1 > m_2 \) and thus the masses of the two primaries in terms of the non-dimensional units become \( \mu_1 = 1 - \mu \) and \( \mu_2 = \mu \). Table 1 gives values for some examples systems within the solar system, of particular interest are the values for the Jovian system, as presented in section 2.2. Also included in Table 1 are the separation distance, \( L \), the orbital velocity of the secondary body, \( V \), and the period of the orbit, \( T \).

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
<th>( \mu ) (dimensionless)</th>
<th>( L ) (km)</th>
<th>( V ) (km/s)</th>
<th>( T ) (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Earth</td>
<td>3.036 x 10^{-6}</td>
<td>1.496 x 10^{8}</td>
<td>29.784</td>
<td>3.147 x 10^{7}</td>
</tr>
<tr>
<td>Earth</td>
<td>Moon</td>
<td>1.215 x 10^{-2}</td>
<td>3.850 x 10^{7}</td>
<td>1.025</td>
<td>2.361 x 10^{7}</td>
</tr>
<tr>
<td>Jupiter</td>
<td>Io</td>
<td>4.704 x 10^{-3}</td>
<td>4.218 x 10^{7}</td>
<td>17.390</td>
<td>1.524 x 10^{7}</td>
</tr>
<tr>
<td>Jupiter</td>
<td>Europa</td>
<td>2.528 x 10^{-3}</td>
<td>6.711 x 10^{7}</td>
<td>13.780</td>
<td>3.060 x 10^{7}</td>
</tr>
<tr>
<td>Jupiter</td>
<td>Ganymede</td>
<td>7.804 x 10^{-3}</td>
<td>1.070 x 10^{8}</td>
<td>10.909</td>
<td>6.165 x 10^{7}</td>
</tr>
<tr>
<td>Jupiter</td>
<td>Callisto</td>
<td>5.667 x 10^{-3}</td>
<td>1.883 x 10^{8}</td>
<td>8.226</td>
<td>1.438 x 10^{7}</td>
</tr>
<tr>
<td>Saturn</td>
<td>Mimas</td>
<td>6.723 x 10^{-3}</td>
<td>1.856 x 10^{8}</td>
<td>14.367</td>
<td>8.117 x 10^{7}</td>
</tr>
<tr>
<td>Saturn</td>
<td>Titan</td>
<td>2.366 x 10^{-3}</td>
<td>1.222 x 10^{8}</td>
<td>5.588</td>
<td>1.374 x 10^{7}</td>
</tr>
</tbody>
</table>

*Table 1: Selected Three Body Systems with Various Parameters [5]*

### 2.4.1 State space & Surface of Section

To be able to fully understand how we can find periodic (or non-periodic) orbits we first need to consider the dynamics of the spacecraft in state space. Each mass of interest (e.g. the two primaries and the spacecraft) has six quantities that describe the position and motion of the object, or in other words, we have three components of position and three components of ve-
locity. For any unique combination of these six components at a given time, \( t \), there will be one point in state space and, as time passes, this point can move in state space.

The famous mathematician Poincaré spent a great deal of time studying the CR3BP, and in particular phase space, and one of his key advancements was to use the surface of section that allows the investigation of the variation of two of the variables within the state space, in most cases, position along the x-axis in the rotating coordinate system and the velocity in the x-direction. An example of a surface of section is given in section 3.1.1.1. This is an incredibly useful tool, and allows for the discovery of periodic orbits as these types of orbits will trace back to the same point on the surface of section after a given time.

2.4.2 Location of the equilibrium points in the CR3BP

In this sub-section a brief discussion is included on how the CR3BP is solved. However, the author recommends any reader to consult the literature as referenced, and in particular [4] and [5], for a more complete treatment of the problem and for the explicit mathematics involved.

One of the key concepts to introduce is the choice of reference frame, and as with the choice of units, the problem can be greatly simplified by this means alone. Figure 2 shows the chosen frame in pictorial format, notice that the x-axis lies along the line joining the two primaries. The whole frame rotates with unit angular velocity relative to the inertial frame with coordinates X and Y. The z-axis (which coincides with the Z-axis) is pointing out of the plane and is not shown here.

![Figure 2: The rotating coordinate system [5]](image)
By using this rotating coordinate system and solving the equations of motion, an important integral, known as the Jacobi Integral can be derived, as shown in Figure 3.

\[
\begin{align*}
V^2 &= 2U - C \\
\dot{x}^2 + \dot{y}^2 + \dot{z}^2 &= x^2 + y^2 + \frac{2(1 - \mu)}{r_1} + \frac{2\mu}{r_2} - C.
\end{align*}
\]

*Figure 3: Jacobi Integral [4]*

This integral is extremely interesting as it allows the consideration of what happens to the spacecraft when the velocity, \(V\), is set to zero. The resulting equation is known as the Hill’s limiting surface and allows us to explore the regions around the two primaries, as \(C\) relates to the initial conditions and \(U\) relates to the gravitational attraction of the spacecraft in relation to the primaries. In other words for a given value of \(C\) we can define the boundaries of the regions in which the spacecraft is permitted, as if \(2U<C\) then we would have imaginary values for \(V\).

Figure 4 shows the different regions that the spacecraft can occupy for different values of \(C\), labelled E in the figures, the grey area is forbidden and the spacecraft can only move within the clear white area. There are five different cases that can be considered:

- **Case 1** – in this case the spacecraft can only move within small areas in close proximity to the primaries and cannot transit from primary to secondary
- **Case 2** – in this case the spacecraft can now pass from the proximity of the primary through the “neck” and into the proximity of the secondary. An intermediate energy between case 1 and 2 shows an intersection between the expanding circles around the primaries that indicates the position of the L1 point.
- **Case 3** – as the energy rises we now have a situation where a spacecraft can traverse not only the area in proximity to the two primaries but can now escape out of, or more likely, enter the system.
- **Case 4** – now the spacecraft is free to move in a much larger area around the primaries.
- **Case 5** – finally, we can now move anywhere and the forbidden zones have disappeared. Where these zones vanish are the L4 and L5 points.
Where the regions intersect are the Libration or Lagrange points, and Figure 5 shows these points plotted in the rotating reference frame, with each L-point indicated.

There are also other methods that can be used for locating L-points which may be implemented in the C# program; these methods, as noted in [4] and [18], include solving a fifth order polynomial equation for the position of the L-points or using a simplified equation. The polynomial method is used by the MATLAB code, discussed in Section 3.1.1.

**Figure 4: Realms of possible motion [5]**
Figure 5: Equilibrium points in the CR3BP [5]

2.5 Periodic Orbits around the L-points

Now that we have knowledge of the L-points we can begin to investigate what can be achieved at these points in terms of the impact on a spacecraft. Whilst the Jacobi Integral showed where libration points were located by assuming a particle with zero velocity, in reality the spacecraft will always have some velocity, so what happens when a particle at an L-point is given a push in one or another direction? Will it return to its initial point after some period of time or will it depart the vicinity? By analysing this problem we can find that for some conditions the spacecraft will oscillate about the L-point, and as such be in a stable or periodic orbit, on the other hand, if the spacecraft rapidly departs the vicinity of the L-point then the orbit is unstable and non-periodic. Section 5.11.4 of [4] provides a mathematical treatment of this problem and this next section will only introduce some of the key concepts required.
2.5.1 Halo Orbit

In this project the initial state of the spacecraft is assumed to be a periodic halo orbit which is out of plane near L1. As stated by [8], from the point of view of an observer on earth (or on the secondary body in the CR3BP) a spacecraft in one of these orbits looks as though it is orbiting the primary, or tracing a halo about the primary. Section 3.1.1.2 returns to this type of orbit with a visual representation, for now, it is only worth mentioning that this kind of orbit is very useful for understanding how we may be able to transfer from one L-point to another using stable and unstable manifolds which are discussed in the next section.

2.5.2 Stable and Unstable Manifolds

Another theoretical concept that is of great importance for this project is the concept of stable and unstable manifolds that exist around the L-points. The low energy exploration method attempts to use these manifolds to transfer a satellite in a halo orbit around one L-point into another halo orbit around a second L-point. Section 3.1.1.2 gives a visual example of this concept and the full mathematical solution is not given here, [8] gives an excellent explanation for any reader interested to know more. In general, the stable manifolds associated with periodic orbits are formed by the set of trajectories converging to the periodic orbits in forward time, whereas the unstable manifolds are formed by the set of trajectories converging to the periodic orbit in backward time.

The key concept is that these manifolds give a framework for understanding transport phenomena from a geometric viewpoint. In particular, the stable and unstable invariant manifold tubes associated to libration point orbits are the phase space conduits transporting material between primary bodies for separate three-body systems [5]. This gives us a method to allow a spacecraft to spiral down a “tube” that will then intersect with another “tube” providing transport between two L-points, with no manoeuvre required. Figure 6 shows a pictorial example of this transfer for the Earth-Moon system, with a spacecraft starting in a halo orbit around L1 before transiting via the stable and unstable manifolds to end in a halo orbit around L2.
2.5.3 Linearization around the equilibrium points

To allow further study of the L-points it is useful to first consider what happens to the dynamics of a spacecraft when we are near these L-points. This can be done by first translating the co-ordinates so that the new frame of reference origin is placed at the L-point. By then linearizing the equations of motion for the three body problem we can find a bounded solution as shown in Figure 7.

\[
x = -A_x \cos(\lambda t + \phi)
\]
\[
y = kA_x \sin(\lambda t + \phi)
\]
\[
z = A_z \sin(\nu t + \psi)
\]

Figure 7: Linearized Solutions for a Halo orbit around L1 [5]

From inspection we can see that the z-axis solution is simply harmonic and does not depend on x or y. However, motion in the xy-plane is coupled. This is often termed a centre x centre x saddle configuration [8].

The equations presented in Figure 7 allow the orbit of the spacecraft around an L-point to
be modelled and the following Figure 8 show the orbit for the ISEE3 space mission [5] with the following variables:

- \( k = 3.229 \)
- \( \text{Lambda} = 2.086 \)
- \( \text{Nu} = 2.015 \)
- \( \text{Ax} = 206,000 \text{ km} \)
- \( \text{Ay} = k\text{Ax} = 665,000 \text{ km} \)
- \( \text{Az} = 110,000 \text{ km} \)

**Figure 8: ISEE3 Mission orbit around L1 [5]**

In addition to the above solution for the dynamics around an L-point there are a number of different types of orbit possible. The main objects found are planar and vertical families of Lyapunov periodic orbits; three-dimensional quasi-periodic Lissajous orbits; periodic halo orbits; and quasi-halo orbits [9]. Figure 9 shows these families of orbits on a Poincaré Section (see section 2.4.1).
Figure 9: All the periodic and quasi-periodic orbits around L2 shown on a Poincaré section of the ecliptic plane [9]

2.6 State of the Art Summary

This section has introduced a number of theoretical concepts that are required to be able to understand the project subject. Of particular importance is the CR3BP which forms the basis for the initial modelling as well as the system to be considered when discussion moves on to the problem of resonant forcing. It is worth noting that other theoretical concepts may be used in the remainder of this document, and these concepts will be introduced and referenced as required. The next chapter will discuss the initial modelling undertaken by the Author using MATLAB.
3 CR3BP MATLAB MODELLING

3.1 Introduction

This section will discuss the modelling of the problems, discussed in section 1, using MATLAB. As a basis for the modelling of the CR3BP a literature search was undertaken and led to the discovery of the excellent celestial mechanics page by Jay James [10] which includes a number of MATLAB source files that have been written to solve the CR3BP. These programs will form the basis of the modelling for this project, and a number of outputs of some of these are shown in this section to help to describe the problem area in a visual manner. The application of some of these MATLAB programs will be discussed in detail.

3.1.1 MATLAB Code

The MATLAB code referenced [10] provides a number of tools that allow for the dynamics of a spacecraft in the CR3BP to be modelled. Table 2 provides a summary of the key MATLAB routines.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>librationPoints.m</td>
<td>Given the mass parameter, Mu, this function calculates the position of the five Lagrange points for the given system.</td>
</tr>
<tr>
<td>CRTBPpoincareGrid.m</td>
<td>This function plots a Poincare grid for random initial conditions and energy. This program calls 'CRTBP.m', 'BackwardCRTBP.m', 'magnitudeVelocity.m', 'librationPoints.m', 'jacobiConst.m', 'interpCrossingData.m', and 'CRTBPpoincareNewton.m'.</td>
</tr>
<tr>
<td>HaloOrbit_NewtonMethod.m</td>
<td>Calculates and plots a halo orbit around the L1 point given a good estimate of the initial conditions (this is set up for the Sun-Earth system). The program calls 'CRTBP.m', 'librationPoints.m', 'stateTransCRTBP.m' and 'JacobiConst.m'.</td>
</tr>
<tr>
<td>Jacobiconst.m</td>
<td>Calculates the Jacobi energy for a given position, velocity, and mass parameter.</td>
</tr>
<tr>
<td>CRTBP.m</td>
<td>Updates state vector according to CR3BP model.</td>
</tr>
<tr>
<td>BackwardCRTBP.m</td>
<td>Updates state vector according to CR3BP model.</td>
</tr>
<tr>
<td>Module Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stateTransCRTBP.m</td>
<td>Computes the state transition matrix. Calls 'sysSolveCRTBP', which calls 'G_CRTBP.m'.</td>
</tr>
<tr>
<td>sysSolveCRTBP.m</td>
<td>Used to define state Transition Matrix</td>
</tr>
<tr>
<td>G_CRTBP.m</td>
<td>Function returns the matrix ‘G’ for the CRTBP</td>
</tr>
<tr>
<td>interpCrossingData.m</td>
<td>This is a special interpolator for a specific problem that comes up when computing Poincare sections of the CRTBP.</td>
</tr>
<tr>
<td>magnitudeVelocity.m</td>
<td>Computes magnitude of velocity for a given state.</td>
</tr>
<tr>
<td>vectorField_CRTBP.m</td>
<td>Computes Vector field for the CR3BP.</td>
</tr>
</tbody>
</table>

*Table 2: MATLAB modules for CR3BP [10]*

In the following sub-sections, more detail is provided on the method and output from relevant MATLAB modules. It is these functions that the program written by the Author will be required to implement for the project to be completed satisfactorily.

3.1.1.1 CR3BP Playground & Poincaré Section

One of the most useful programs taken from [10] is the CR3BP playground programme which allows a user to model the impact on a spacecraft of various initial conditions for a given value of $\mu$. A modification of this programme is CRTBPpoincareGrid.m [10] that produces a surface of section for a given system, an output plotted from this program is shown in Figure 10; it shows the surface of section for the Jupiter-Io system for a given energy. The output is extremely complex and the detail of the figure is somewhat lost due to the large number of points plotted, one of the issues to resolve during this project will be attempting to speed up the process of producing these types of plots by either improving the MATLAB code or porting the code to a more sophisticated programming language; to produce the data for Figure 10 took 17 hours of CPU time (AMD 64 X2 Dual Core Processor)! As alluded to in Section 2.5.3 the Poincaré grid can be used to identify conditions for different types of periodic orbits, and is a very useful tool for identifying halo orbits.
3.1.1.2 Computation of Halo orbits

As previously discussed in section 2.5.1, a halo orbit can be very useful and practical for
some types of satellite tasks, for example, SOHO\(^1\) was placed in an halo orbit around the L1 point of the Sun-Earth system so that it could take continuous scientific readings of the Sun whilst maintaining a constant line-of-sight communication link with Earth. There are in fact a family of halo orbits around each L-point and the programme \texttt{haloFamilyL2.m} \cite{10} plots these orbits for the L2 point. Figure 11 presents the family of halo orbits around the L2 point of the Jupiter-Io system. One of the key tasks for the remainder of this project will be plotting these families of halo orbits for other systems.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{halo_orbits.png}
\caption{Halo orbits around L2 in the Jupiter-Io System [Please note that the Earth (M1) is not shown on this figure]}
\end{figure}

As a side note it is worth mentioning that recently launched European Space Agency (ESA) spacecraft, Herschel and Planck, are at the time of writing, approaching orbit around the L2 point of the Sun-Earth system to begin their respective scientific experiments.

3.1.1.3 Stable Manifolds

As discussed in section 2.5.2, for this project we are interested in stable and unstable manifolds that emanate from the L-points as these can allow the transit of a spacecraft between two different L-points. Figure 12, produced using \texttt{stableEarthMoonLyap.m} \cite{10}, shows a stable manifold emanating from a Lyapunov\(^2\) orbit around L1 for a specific energy value (the red

\footnotesize
\begin{itemize}
\item \(^1\) Solar and Heliospheric Observatory
\item \(^2\) A Lyapunov orbit is similar to a halo orbit but rather than being out of plane is within the plane of the two primaries
\end{itemize}

- 27 -
line indicates the realm of possible motion) for the Earth/Moon System.

![Stable manifold from L1 in the Earth-Moon System](image)

*Figure 12: Stable manifold from L1 in the Earth-Moon System [Please note that the Earth (M1) is not shown on this figure]*

By increasing the energy of the spacecraft the red line indicating the realm of possible motion will increase, allowing the spacecraft to transit along this stable manifold to an unstable manifold in the vicinity of the L2 point, as shown in Figure 6. It is this kind of transfer that the low energy exploration method looks to exploit.

### 3.2 MATLAB Modelling Summary

This section has presented the results of initial MATLAB modelling for this project to assist in the understanding of the theory presented in Section 1, particularly the theory behind of the CR3BP and periodic orbits that are vital for the remainder of this project. The Author wishes to express his acknowledgement of the work [10] by Jay James for being of great assistance!
4 GDB CR3BP PROGRAM

4.1 Introduction

This section of this report provides an overview of the effort and work undertaken in produc-
ing a computer program from scratch, in order to solve the CR3BP and the problem of resonant forcing. A CD including source code, installation file, installation guide and user
guide can be found at the back of this report.

This section is split into sub sections; firstly, in Section 4.2, the requirements for the pro-
gram are detailed; in the next section (Section 4.3) the general design philosophy is presented
and finally in Section 4.4, a description of the design and coding process is included, includ-
ing information regarding any problems that were discovered and how these problems were
solved.

4.2 Requirements

In this section, the requirements for the program to be developed are documented in tabular
form. The requirements are split into three key areas, as defined in Table 3.

<table>
<thead>
<tr>
<th>Area</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface &amp; Ouptut</td>
<td>4.2.1</td>
<td>These requirements relate to the design and use of the Program by a User and the options that are available when running the Program. These requirements include how data is outputted to a User as well as the key functions that must be available to be called by a User to produce such output.</td>
</tr>
<tr>
<td>Functional</td>
<td>4.2.2</td>
<td>These requirements relate to the explicit functions that are required for the Program to be able to meet the requirements placed on it by the User Interface.</td>
</tr>
<tr>
<td>Documentation</td>
<td>4.2.3</td>
<td>These requirements relate to how the Program should be documented.</td>
</tr>
</tbody>
</table>

*Table 3: Requirements Table*

It should be noted that a number of terms are repeatedly used and the following definitions
are required:

- **User** – any person who may use the program.
• *Program* – the GDB CR3BP program

• *Native Code* – any code written from scratch by the Author

There now follows a sub-section for each area in Table 3, these tables will be returned to later in the document to assess the success (or otherwise) of the programming that has been implemented. A priority number is provided for each requirement indicating if the requirement is essential (1) or less important.

### 4.2.1 User Interface & Output

This section of requirements relates to how a *User* will control the Program and what options are available to the *User*. Table 4 details these requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI_001</td>
<td>Allow <em>User</em> to enter Mass Parameter</td>
<td>1</td>
</tr>
<tr>
<td>UI_002</td>
<td>Allow <em>User</em> to choose Mass Parameter from relevant systems</td>
<td>2</td>
</tr>
<tr>
<td>UI_003</td>
<td>Allow <em>User</em> to choose calculation method</td>
<td>2</td>
</tr>
<tr>
<td>UI_004</td>
<td>Allow <em>User</em> to choose Output options</td>
<td>3</td>
</tr>
<tr>
<td>UI_005</td>
<td>Allow <em>User</em> to enter Initial State</td>
<td>4</td>
</tr>
<tr>
<td>UI_006</td>
<td>Allow <em>User</em> to graphically display the position of the L-points for the system defined by UI_001; UI_002.</td>
<td>1</td>
</tr>
<tr>
<td>UI_007</td>
<td>Allow <em>User</em> to display Jacobi energies for each L-point</td>
<td>2</td>
</tr>
<tr>
<td>UI_008</td>
<td>Allow <em>User</em> to graphically display Halo Orbits around the L-points for the system defined by UI_001; UI_002.</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 4: User Interface and Output Requirements*

### 4.2.2 Functionality

This section of requirements relates to the Program itself and the methods that will be required for the Program to output data to meet the requirements for the UI. To reduce the risk involved with implementing the software a two option approach was developed, where the User could choose to either use the MATLAB functions (Section 3) or use Native Code. Table 5 details these requirements.
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_001</td>
<td>Ability to use functions to solve CR3BP, integrated into the main Program</td>
<td>1</td>
</tr>
<tr>
<td>FN_002</td>
<td>Ability to use MATLAB as a secondary source for computation</td>
<td>2</td>
</tr>
<tr>
<td>FN_003</td>
<td>Calculate Mu for a given system</td>
<td>1</td>
</tr>
<tr>
<td>FN_004</td>
<td>Calculate Libration Point positions</td>
<td>1</td>
</tr>
<tr>
<td>FN_005</td>
<td>Calculate Halo Orbits for L1</td>
<td>1</td>
</tr>
<tr>
<td>FN_006</td>
<td>Calculate Jacobi Energy from given position, velocity, and mass parameter</td>
<td>2</td>
</tr>
<tr>
<td>FN_007</td>
<td>Calculate Halo Orbits for L2</td>
<td>2</td>
</tr>
<tr>
<td>FN_008</td>
<td>Calculate Poincaré Section for a given system</td>
<td>2</td>
</tr>
<tr>
<td>FN_009</td>
<td>Calculate impact of Resonant Forcing on a given system</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 5: Functional Requirements*

**4.2.3 Documentation**

Finally, these requirements indicate the level of documentation that will be required for the Program. Table 6 details these requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_001</td>
<td>Use tagged comments to provide source code understanding</td>
<td>1</td>
</tr>
<tr>
<td>DC_002</td>
<td>User Guide</td>
<td>1</td>
</tr>
<tr>
<td>DC_003</td>
<td>Design/Functional Document</td>
<td>1</td>
</tr>
<tr>
<td>DC_003</td>
<td>Installation Package</td>
<td>1</td>
</tr>
<tr>
<td>DC_004</td>
<td>Help File incorporated into C# program</td>
<td>3</td>
</tr>
</tbody>
</table>

*Table 6: Documentation Requirements*

To meet these requirements, a user guide is included at Appendix 3 and Section 4.3 and Section 4.4 provides the design document. All of the code written that is not available from open source is included in Appendix 5 to show that DC_001 has been met.
4.3 Design Philosophy

4.3.1 Introduction

The program has been written using Microsoft C# visual studio 2008, which was chosen as a suitable programming language by the Author for a number of reasons. Firstly, it allows for the use of classes and object-oriented programming methods which the Author has experience of and secondly the Author has recently been using C# as part of a consultancy job!

In general, where possible, open-source code that provided useful functionality has been used with the Author only programming those areas vital for completing the requirements stated in Section 4.2. The licence under which open source code has been released can be found at [11].

4.3.2 Software development cycle

The development of software for this project requires some thought of the process required for such activities. For this reason, the Author has spent some time researching different development cycle methods to assess which method will work most satisfactorily for this project. A waterfall method, as shown in Figure 13, was originally chosen to form the basis for the development method.

![Waterfall Software Development Method](image)

*Figure 13: Waterfall Software Development Method*

However, a much more iterative approach was undertaken, similar to an iterative development procedure shown in Figure 14. This method is much more adaptive and as the design section (4.4) will discuss, allows for any issues and problems within the programming to be
dealt with in a more natural way.

![Iterative Software Development Method](image)

*Figure 14: Iterative Software Development Method*

### 4.4 Design & Implementation

The design of the program can be considered with the same structure as the requirements and this section will approach each area in turn and discuss the key design processes that were required and what was implemented. There are some areas that have not been implemented at the time of writing and these areas are noted in the text and discussed in Section 1.

#### 4.4.1 User Interface (UI)

For any UI there is an aspiration to make the program as intuitive and easy to use as possible. This requires a certain amount of thought when placing controls to ensure that a User with no prior knowledge of the program may be able to use it without reading a User Guide (though obviously this is not recommended!). Another key consideration is ensuring that a User cannot break the program (this is often the first thing to attempt when debugging!).

In general terms the UI provides the framework from which a User may call various functions from the Program and be able to view the output in a suitable format. Extra functionality, such as the ability to zoom into graphs and print output has not been implemented at this time, however, this ability may be of some value in the future, please see Section 4.4.1.2 for further discussion of this area.

**4.4.1.1 Main Form**

The Main Form is designed to provide handling for all of the UI controls and provides the backbone of the Program from which the CR3BP functions can be called. This form is the “Face” of the Program that a user will see. Figure 15 shows a screenshot of the form as it is...
seen by the User when the program starts. The full code for the Main Form can be found at Appendix 5.

Figure 15: Main Form

4.4.1.2 Graphical Display Options

To provide a User the means to visualise the functions that are called, a graphical display is required. One of the great advantages of MATLAB is that this is a very simple process, for coding from scratch it is more of a problem! For this reason the original concept for the Program assumed that MATLAB would continue to be used for plotting output data. However, this provides some problems regarding how to output data to allow MATLAB to plot it! Also, one of the key aims of the program was to provide an alternative to MATLAB, and for this reason, a number of possible solutions where researched to provide output to the User through the Main Form, Table 7 below lists some of these options with comments regarding their suitability.

Whilst the Author did consider writing Native Code, it was obvious that the MSChart option would be most suitable as an initial output method, due in part to the Author’s experience of using MS Excel and the simplicity of the method. At the time of writing, no zoom or print function had been implemented. Example output using MSChart is shown in section 6.2.
<table>
<thead>
<tr>
<th>Option</th>
<th>Comments</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB</td>
<td>Use MATLAB to plot output data.</td>
<td>Many graph options, already implemented, including zoom and print.</td>
<td>Outputting data can be complicated and time-consuming.</td>
</tr>
<tr>
<td>Microsoft Chart</td>
<td>The simplest option. This type of object is the same as that used by MS Excel to plot data so are simple to use.</td>
<td>Simple, Easy to use.</td>
<td>Zoom and Print options not available in standard toolbox.</td>
</tr>
<tr>
<td>GraphLib</td>
<td>An open source code that provides excellent graphical functionality, including the ability to print, zoom, or play through time a dataset.</td>
<td>Sophisticated graphical tool, providing much functionality.</td>
<td>Much more complicated to implement.</td>
</tr>
<tr>
<td>Native Code</td>
<td>This option is to write from scratch a graphing class.</td>
<td>Tailored to requirements.</td>
<td>Time and Effort consuming.</td>
</tr>
</tbody>
</table>

Table 7: Output options

### 4.4.2 Variables, Structures & Functions

Good programming techniques were aspired to during the implementation of the software, and therefore, the Main Form had very few functions defined within it apart from those that control User Input. The approach taken was to include all of the variables, structures and functions within separate classes which the Main Form would initialise and call when required. The one exception to this was the function `CalculateMu` which was included in the Main Form code as it is in general just a look-up table. As the option has been included to allow a User to select the computational method (or Engine), the following sub-sections document how each Engine is implemented and comments on any problems that were encountered during the programming.

### 4.4.3 Native Class

This class was an attempt to modify the MATLAB code [10] to work within the C# environment. The full code for the Native Class can be found at Appendix 5. To begin with the MATLAB code was studied to analyse what data structures and functions would be required by any Native code. It became clear that one of the key data structures that would be required
for any Native code would be Matrices. This area is analysed in more detail in section 4.4.3.1. Another area of importance is a method for solving ordinary differential equations (ODE) and this area is discussed in section 4.4.4.3.

4.4.3.1 Matrix options for C# programming

Given that Matrices were of vital importance to the implementation of the code, a number of options were assessed. Table 8 gives a brief overview of the different options available for C#, please note that only open source software was investigated, and the Author is aware that commercial software is available for this area for which one example is provided in Table 8.

<table>
<thead>
<tr>
<th>Option</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>C# System Arrays</td>
<td>Simple</td>
<td>Simple! Does not provide MATRIX operators</td>
</tr>
<tr>
<td>CSML [14]</td>
<td>Majority of MATRIX operators supported</td>
<td>Poor help files Not all MATRIX operators supported!</td>
</tr>
<tr>
<td>DotNetMatrix [15]</td>
<td>Majority of MATRIX operators supported</td>
<td>Poor help files Not all MATRIX operators supported!</td>
</tr>
<tr>
<td>Numerical Methods [16]</td>
<td>Majority of MATRIX operators supported</td>
<td>Complicated to implement, poor help files.</td>
</tr>
<tr>
<td>Commercial Off The Shelf (COTS) (e.g. [17])</td>
<td>Commercial software so high degree of complexity</td>
<td>Expensive!</td>
</tr>
<tr>
<td>Native Code</td>
<td>Tailored to needs</td>
<td>Complex and time-consuming</td>
</tr>
</tbody>
</table>

Table 8: Matrix Mathematics Options

Many “rabbit-holes” were caused by the choice of Matrices, with problems generally encountered only some time after initial implementation of the various Matrix options! The final choice was to use the CSML matrix type, which offers most of the required functions and was the option that the Author found most intuitive to work with.
4.4.4 Native Class Functions

4.4.4.1 Libration points
This function is an adapted version of the MATLAB code, `librationPoints.m` which solves a polynomial equation to derive the L-points. This code appears to work well in C#, discussion of the output of this code, and validation against the MATLAB code is provided in section 6.2. Another method was also implemented to offer the Author choice, which was based on much simpler mathematics, as found in [18], and only used for L1, L2 and L3 with the MATLAB methods used for L4 and L5.

4.4.4.2 Jacobi Energy
This function is adapted from the MATLAB code, `jacobConst.m`, a simple piece of code that calculates the Jacobi energy given a vector position, vector velocity and mass parameter.

4.4.4.3 Halo Orbit Calculation
To be able to calculate halo orbits an attempt to convert the `HaloOrbit_NewtonMethod.m` MATLAB code was undertaken. This led to the discovery of a major problem with the program, in that the excellent MATLAB ODE solver was not available within the C# code. This led to a number of options, which are listed in Table 9 being considered for how this problem may be solved.

<table>
<thead>
<tr>
<th>Option</th>
<th>Comments</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB ODE</td>
<td>Used by MATLAB engine</td>
<td>Good ODE solver functions</td>
<td>Not native</td>
</tr>
<tr>
<td>Numerical Methods [16]</td>
<td>Open Source</td>
<td>Does include some ODE functionality</td>
<td>No help files</td>
</tr>
<tr>
<td>Native Code</td>
<td>A very large undertaking!</td>
<td>Tailored to requirements</td>
<td>Very difficult!</td>
</tr>
<tr>
<td>Other [17]</td>
<td>COTS software</td>
<td>Expensive</td>
<td>Good functionality</td>
</tr>
</tbody>
</table>

*Table 9: ODE Options*

After much effort was expended attempting to get a suitable ODE function within the C# code, at the time of writing this functionality was not implemented. It is hoped that the Author will have overcome this issue in time for the final Viva examination but does not have huge confidence in this statement!
4.4.4 Halo Orbit Linearised Solution

Given the failure to produce halo orbits, the decision was made to include a method for plotting orbits using the linearised equations, as given in section 2.5.3. These are simple functions and did not provide any programming issues and allow a User to ‘play’ with the linearised equations of motion to simulate various types of orbit. Example output for this functionality is shown in Section 6.2.

4.4.4.5 Resonant Forcing modelling

Given the problems that occurred during the implementation of the native code, a model for Resonant Forcing was not implemented due to time constraints. However, Section 5 discusses Resonant Forcing and various methods for how it could be modelled.

4.4.5 MATLAB Engine Class

Given the problems documented above with the Native class, a MATLAB class was also developed. There were a number of different methods for the implantation of this Engine, which are discussed in the following sections. The full code for the MATLAB Engine can be found at Appendix 5.

4.4.5.1 Linking C# with MATLAB

To be able to take advantage of the functionality provided by MATLAB there are a number of options regarding linking of the C# program to MATLAB. The key differences between the methods are the availability of functions and the time taken to complete commands. For more information please refer to [19].

<table>
<thead>
<tr>
<th>Option</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB Engine (via COM automation)</td>
<td>Simplest and easiest to use. Allows Debug of MATLAB at the same time as debug of Program.</td>
<td>No faster than normal MATLAB.</td>
</tr>
<tr>
<td>.NET Assembly (using MATLAB Builder for .NET)</td>
<td>Can be deployed royalty-free to machines that do not have MATLAB. Automatic data marshalling and garbage collection.</td>
<td>Builder takes time to implement and help files are not very good</td>
</tr>
<tr>
<td>C Shared library (using MATLAB Compiler)</td>
<td>Can be deployed royalty-free to machines that do not have MATLAB</td>
<td>Not stable!</td>
</tr>
</tbody>
</table>

*Table 10: Linking MATLAB to C# Options*
Initially, the option to use a .NET Assembly was taken and a class created using .NET Builder in MATLAB (type “deploytool” in MATLAB command line). This method does have advantages in terms of speed of computation but the Author could not get this method to work in a satisfactory manner for the functions required for this Program, in particular there were problems using the ODE functions within MATLAB. For this reason, the simpler option of the MATLAB Engine, via COM automation, was implemented and whilst this does not have any advantages in terms of speed of processing, it does provide a stable environment and the ability to debug in both C# and MATLAB environments, which is incredibly useful!

4.4.5.2 MATLAB Engine Class Implementation

Once the method for communicating with MATLAB was implemented, the required functions could be added to the class. These functions simply set up the required variables in MATLAB and run the specified MATLAB function in a command line fashion. The original MATLAB functions [10] were modified so that input for each function was handled by the GDB CR3BP program.

4.5 GDB CR3BP Program Summary

This section of the report has presented the progress made in designing and implementing a C# program to model the CR3BP, and in particular, periodic orbits around the L-Points. Unfortunately, there has only been limited success in producing a program that works to specification (discussed more in Section 6.5), however, the Author sees the progress made to date as being satisfactory and providing at the very least a alternative coding environment to MATLAB. This report does not mean that this program will no longer be developed and the Author intends to continue to improve the GDB CR3BP program even after the conclusion of his MSc.
5 DISCUSSION OF RESONANT FORCING

5.1 Introduction

This section of the report will document the progress made in finding a solution to the problem of resonant forcing. As previously mentioned in section 4.4.4.5 no explicit modelling of resonant forcing was implemented in the C# program but this section analyses how a method might be devised to solve the problem by discussing a number of different theoretical concepts that could be adapted to solve the resonant forcing problem.

5.2 Forced Vibration

The paper *Gravitational Theory and Resonance* [20] provided some useful information in regard to resonant forcing and this section will discuss some of the outcomes of this paper and apply these outcomes to the resonant forcing of a satellite in a halo orbit around an L1 point.

In simple terms, any circular or elliptical motion can be treated as a vector sum of two sinusoidal vibrations at right angles to each other and 90 degrees out of phase with each other. The linearised equations of motion (section 2.5.3) for a periodic orbit around an L-point are a good example of this, where the where x and y are defined by Cosine and Sine respectively. Each component can be considered to be a vibration and by choosing the axes carefully, as is the case for the linearised equations, the resulting amplitude determines the major and minor axes of the orbit.

Any vibrating object will also have certain resonant frequencies, for example, the Earth orbits the Sun with a period of exactly one year, so the principal frequency is the inverse of this period. In addition, there is the concept of harmonic resonances at different integer values of the natural resonant frequency (e.g. double, triple, quadruple, etc). Analysis shows that these resonant considerations can be applied to the orbital motion of planets, moons and ring particles [20]. Considering the natural orbital period as a “free” vibration, the action of another gravitationally attractive body (the fourth body) can then be treated as a harmonic disturbing force, or a forced vibration.

5.2.1 Forced Pendulum

The simplest case for understanding resonant forcing is that of a forced pendulum, the concept of which is a fairly common physics problem. If we have a forced damped pendulum which is given some initial conditions (an initial push) we will see linear behaviour and a damping effect that will slowly return the pendulum to its rest position. In the forced case,
consider the case where a periodic external force perturbs the pendulum, this will lead to either chaotic motion which is unpredictable or it will lead to more predictable or quasi-periodic behaviour, dependent on the initial conditions and the force perturbing the pendulum.

If the perturbing force has a frequency that is resonant with the frequency of the pendulum then even small perturbing forces can cause a large change in amplitude of the pendulum. This is best exemplified by a parent pushing a child on a swing, as long as the perturbing force is timed correctly then the swing will go higher and higher. It is conceivable that such a simple concept could be applied to the linearised equations of motion for a halo orbit around an L-point, as discussed in Section 2.5.3 though at this time no further development of this method has been implemented.

5.2.2 The Perturbation of a Halo Orbit around L1

Now consider a satellite in a halo orbit around the L1 point in, as a simple example, the Sun-Earth (& Moon) three body problem. The system has a natural vibration that has a frequency equal to the inverse of the period of the system (e.g. one year) and it also has a frequency relating to the period of the halo orbit, this means that the resonance we are interested in could affect either of these orbital frequencies. If we now add a fourth body to the system, say Jupiter, consideration can be made of the impact of this body on the halo orbit.

This problem can initially be simplified by assuming that Jupiter orbits with the same period as the Sun-Earth period, or in other words, with a 1:1 resonance. In this case, we can imagine the impact of Jupiter on the halo orbit as constantly providing a force along the x-axis of the system. Whilst this force is likely to be of small order due to the large distances involved, over a long enough period of time we would expect the halo orbit to be perturbed by this constant force and perhaps become unstable.

Considering the more realistic case, the problem becomes more complicated and is dependent on the position of the fourth body, Jupiter, as the Sun-Earth frame rotates. In this case, a different force is imparted on the halo orbit as Jupiter rotates around the Sun-Earth system, for example, considering the simple case introduced above, this only applies for a small portion of Jupiter’s orbit. When Jupiter is on the opposite side of the Sun to the Earth and at opposition, then a force will be applied to the halo orbit in the negative x-direction. Figure 16 gives a pictorial example of the system with each side of the dodecahedron indicating one period of the Sun-Earth CR3BP; after each period of the Sun-Earth system, the perturbing force caused by Jupiter will act in a different direction.
In reality the orbit of Jupiter is close to, but not exactly twelve years and whilst the assumption could be made that the impact that Jupiter has on a satellite in L1 halo orbit could be assumed constant over one period of the Sun-Earth system, this system will never provide exact integer resonance, it would appear that this system is unlikely to be of any great use for a spacecraft!

The example above has adapted much from [20] in which the impact of Jupiter on the dwarf planet, Ceres, is considered. Interestingly it appears that the impact of Jupiter on the orbit of Ceres has pushed Ceres towards an orbit that is almost, but not exactly, resonant. In fact, there appears to be a general rule that resonance is an unstable state [20], which would help to explain the Kirkwood Gaps, as discussed in section 1.1.

Returning to our resonant forcing problem and considering a different system, the Jovian moons we can imagine seeing a similar situation, though this time with the outer moons orbit-
ing the inner three body system much more frequently, or in other words, at a much lower resonance (2:1) meaning that the forces imparted on an L1 halo orbit would vary more quickly and the frequency of the perturbing force has a much lower period than the case considered above with Jupiter orbiting the Sun-Earth system. This makes the case for resonant forcing much stronger in this system as the period of the outer moon will cause a 2:1 resonance on the rotating frame of the Jupiter-Io system. This suggests that there is a 2:1 resonance at the L-points as well, if we assume that the model introduced above is valid. Of course, the Jovian moons are of much lower mass than Jupiter itself, and it may require a long-time scale for any impact of this resonance to become obvious.

To model this would require analysing the impact of the resonant body as an additional term in the non-linear dynamics of the CR3BP. At this time, however, this method has not been developed further.

5.3 Resonant Satellite Orbits

Another area of interest that could be applied to the resonant forcing problem is the concept of resonance effects on a satellite if its path repeats itself relative to a rotating primary [21]. The orbit of a satellite is in resonance with the gravitational field of the Earth when the track of the satellite over the Earth repeats after an integral number of revolutions [22]. The work by Allan [21] shows that there is a significant effect of this resonance, so much so that a satellite may become locked in a particular resonance. The analysis provided by this concept [22] has helped to build an excellent model of the Earths geo-potential, as a satellites Keplerian orbital elements are affected by the Earth’s gravitational field, particularly the inclination of the satellites orbit. However, for a system such as Jupiter there are difficulties due to the estimation of the rotation period of the planet, which plays a vital part in these calculations. There is also the issue for the work presented in [21] that for the Jovian moons system the spacecraft will not be orbiting around the primary, and therefore this method may not be applicable.

This is a promising area in terms of providing an example of resonant forcing on a satellite but the Author has not been able to apply these theories to the problem where the CR3BP is concerned. The Author is unconvinced that this concept will be of particular use for modelling resonant forcing.

5.4 The Four Body Problem

Another approach that could prove useful is modelling of the four body problem. As with the general three body problem, the four body problem cannot be solved; therefore assumptions
similar to the CR3BP are made by limiting the problem to circular orbits and the Concentric Circular Method, as shown in Figure 17, appears to provide a good model, particularly for the problem of resonant forcing within the Jovian moon system.

![Figure 17: Concentric Circular Model [5]](image)

The four-body problem is understandably, even more complex than the three body problem, and much of the analysis undertaken in this area uses the bi-circular restricted four body problem which considers a system that can be assumed to be similar to the Sun-Earth-Moon system, with the Moon rotating around the Earth and both rotating around the Sun.

However, the paper, [25], does analyse periodic orbits in the restricted four-body problem with a system similar to that shown in Figure 17, which indicates that if a halo orbit (from the CR3BP) is used as an initial trajectory then the impact of the fourth mass will cause this halo orbit to destabilise. This appears to be a useful method, and the Author feels that this area is worth further investigation; however, at the time of writing, there was insufficient time to develop this method further.

### 5.5 Resonant Forcing Discussion Summary

This section of the report has considered a number of methods that could be adapted to allow for the modelling of a resonant body, both within the CR3BP and in the more complex CR4BP. Whilst progress has been limited in terms of applying these methods, the Author feels that given more effort there is potential in each of the methods.
6 RESULTS & DISCUSSION

6.1 Introduction

This section of the report documents the output from the previous sections and discusses the results produced. The section is split into sub-sections with Section 6.2 presenting and discussing the output from the C# program, providing validation of the output, section 6.3 discusses the modelling and method development for the resonant forcing problem, and section 6.5 assesses if the requirements given in section 4.2 have been met by the C# program written.

6.2 C# program Validation

This section discussed the output provided by the C# program written and provides validation of any output against the original MATLAB code [10].

6.2.1 Validation

In this section, the results output from each of the functions that have been implemented in C# are compared against the results produced by the original MATLAB code [10]. This process was completed by running both MATLAB and GDB CR3BP routines and comparing the results.

6.2.1.1 Libration Points

Table 11 shows results from the MATLAB function compared to the C# program for the calculation of L-point positions for the two methods implemented in C# for a number of test systems; the final two columns of the table are colour coded to indicate the difference between these values and the MATLAB function, librationPoints.m. In Table 11, traffic light colour coding is used where green indicates that the calculated figure is within 1% or the MATLAB figure, amber indicates within 5% and Red indicates >5%.
<table>
<thead>
<tr>
<th>System</th>
<th>L-Point</th>
<th>X (MATLAB)</th>
<th>Polynomial Method</th>
<th>Simple Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun-Earth</td>
<td>1</td>
<td>0.9900</td>
<td>0.9949</td>
<td>0.9899</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.0101</td>
<td>1.0049</td>
<td>1.0100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.0000</td>
<td>-0.9999</td>
<td>-0.9999</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.5000</td>
<td>0.4999</td>
<td>0.4999</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.5000</td>
<td>0.4999</td>
<td>0.4999</td>
</tr>
<tr>
<td>Earth-Moon</td>
<td>1</td>
<td>0.8369</td>
<td>0.9225</td>
<td>0.8406</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.1557</td>
<td>1.0842</td>
<td>1.1593</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.0051</td>
<td>-0.9949</td>
<td>-0.9949</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.4879</td>
<td>0.4875</td>
<td>0.4878</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.4879</td>
<td>0.4875</td>
<td>0.4878</td>
</tr>
<tr>
<td>Jupiter-Io</td>
<td>1</td>
<td>0.9751</td>
<td>0.9873</td>
<td>0.9749</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.0252</td>
<td>1.0123</td>
<td>1.0250</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-1.0000</td>
<td>-0.9999</td>
<td>-0.9999</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.5000</td>
<td>0.4999</td>
<td>0.4999</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.5000</td>
<td>0.4999</td>
<td>0.4999</td>
</tr>
</tbody>
</table>

*Table 11: L-Point Validation*

These results suggest that the C# function librationPoints provides good results in comparison with the MATLAB routine. However, there are some areas of concern, notably the L1 and L2 results for the Earth-Moon system. By using the alternative method as given in [18] better results for the Earth-Moon system were produced and it is this simpler method that will be included in the final program. The librationPoints are output graphically to the program, with the output as shown in Figure 18.
6.2.1.2 Jacobi Energy

This simple function has been validated by calculating the Jacobi energies for each of the L-points in the Sun-Earth system and comparing the values output by the MATLAB routine with that written for the C# program. Given the differences in the calculated positions of the L-points, a slight discrepancy is expected in the output results for the first three L-points, as shown in Table 12. Note that the Jacobi energy for L5 is not included as this is identical to that for L4.

<table>
<thead>
<tr>
<th>System</th>
<th>L-Point</th>
<th>Jacobi Energy (MATLAB)</th>
<th>Jacobi Energy (Native)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun-Earth</td>
<td>1</td>
<td>3.000897</td>
<td>3.000897</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.000897</td>
<td>3.000897</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.000003</td>
<td>3.000003</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.999996</td>
<td>2.999996</td>
</tr>
</tbody>
</table>

*Table 12: Jacobi Energy Validation*

6.2.1.3 Halo Orbits

Given the problems encountered during the programming of the C# programing (see section 4.4.4.3), no results were produced for the Halo Orbit calculation. It is hoped that results will
be produced in this are in time for the final examination Viva. MATLAB output can be plotted using the C# program, examples of which are shown in Figure 19, Figure 20, and Figure 21.

**Figure 19: Halo Orbit plot (X-Y axes)**

**Figure 20: Halo Orbit plot (X-Z axes)**
6.2.1.4 Halo Orbit Linearised Solution

This output can be compared to the data shown in [4] as previously shown in Figure 8. The output from the C# program is shown in Figure 22 for similar input data.

If Lambda and Nu are set equal we get a halo orbit, as shown in Figure 23.
6.3 Resonant Forcing

Though no method was developed that was suitable for the implementation of any model for resonant forcing, the discussion presented in Section 5 does raise some interesting issues and introduces possible methods that could be utilized for the modelling of this problem.

6.4 Low-Energy Exploration

Due to the lack of a solution to the resonant forcing problem and the issues involved with computing halo orbits, the comparison to standard interplanetary transfer of spacecraft has not been completed. However, research was undertaken in this area, and of particular interest is [24] that discusses low energy interplanetary transfers using halo orbits for Sun-Planet systems around L2 as a target point. The paper reports that there using these methods is 35% more fuel efficient than the conventional gravity assisted trajectory method but approximately 5 times slower [24].

6.5 Requirements Assessment

This next section will document the success if the C# program to meet the requirements set out in section 4.2. A success metric is included for each requirement with a score between zero and five being awarded, with zero indicating no success and five indicating complete success. Given that there are twenty metrics this will allow a score out of one hundred to be
presented to indicate the success of the program to meet the requirements.

### 6.5.1 User Interface & Output

Table 13 provides comments and an indication of success for each of the user interface & output requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Comments</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI_001 &amp;</td>
<td>User may enter Mass Parameter manually or use a list box to choose the</td>
<td>5</td>
</tr>
<tr>
<td>UI_002</td>
<td>Mass Parameter for a particular system</td>
<td></td>
</tr>
<tr>
<td>UI_003</td>
<td>User can choose to use MATLAB or Native Codes Engine</td>
<td>5</td>
</tr>
<tr>
<td>UI_004</td>
<td>Output options can be Libration Points, Halo Orbits, Poincaré Section.</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>However, not all output options are available when a user selects the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>native engine</td>
<td></td>
</tr>
<tr>
<td>UI_005</td>
<td>Initial State has not been included at this time</td>
<td>0</td>
</tr>
<tr>
<td>UI_006</td>
<td>L-point data presented to user in graphical and numerical formats</td>
<td>5</td>
</tr>
<tr>
<td>UI_007</td>
<td>Only MATLAB output is plotted, native methods use linearised equations</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>of motion only</td>
<td></td>
</tr>
</tbody>
</table>

*Table 13: UI & Output Requirements Assessment*

### 6.5.2 Functionality

Table 14 provides comments and an indication of success for each of the functionality requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Comments</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_001</td>
<td>Problems encountered solving ODEs, MATLAB engine does work</td>
<td>0</td>
</tr>
<tr>
<td>FN_002</td>
<td>MATLAB engine works well</td>
<td>5</td>
</tr>
<tr>
<td>FN_003</td>
<td>Mu can be calculated either from Masses in kilograms or from a given</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td></td>
</tr>
<tr>
<td>FN_004</td>
<td>Some success in native code though issues when output was validated</td>
<td>3</td>
</tr>
</tbody>
</table>
6.5.3 Documentation

Table 15 provides comments and an indication of success for each of the documentation requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Comments</th>
<th>Success?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_001</td>
<td>See Appendix 5 for code with comments</td>
<td>5</td>
</tr>
<tr>
<td>DC_002</td>
<td>See Appendix 3 for a simple user guide, Appendix 4 for an installation guide.</td>
<td>5</td>
</tr>
<tr>
<td>DC_003</td>
<td>Section 4.4 of this report</td>
<td>5</td>
</tr>
<tr>
<td>DC_003</td>
<td>Installation file. This has not been fully tested!</td>
<td>1</td>
</tr>
<tr>
<td>DC_004</td>
<td>This has not been implemented, it was a low priority requirement and the DC_002 provides the required help for using the program</td>
<td>0</td>
</tr>
</tbody>
</table>

6.5.4 Requirements Summary

Given the success metrics given in each of the tables above, the ability of the program to meet the requirements scores 60/100. This is a disappointing outcome and one that the Author will endeavour to improve before the final Viva examination.

6.6 Results Summary

This section has presented the results and output produced by the Author for this project. As indicated previously there is some disappointment regarding the output of the GDB CR3BP program due to the failure to implement a method for calculating halo orbits in particular.
7 CONCLUSION

7.1 Summary

This report has documented the progress made of the project Resonant Forcing of L-point orbits. The document has provided background information that has been researched via literature review and highlights the key areas of theory that were required to analyse the project problem area. MATLAB code, sourced from [10] was used initially and an attempt to convert this code to a C# program was undertaken with limited success. This report has also given consideration to the issue of resonant forcing attempting to apply research from various papers [20], [22], [21] and [25] to provide an understanding of how resonant forcing might be modelled. At the time of writing, progress has been disappointing in this area and no method had reached a suitable level of maturity to be modelled in the C# program.

The Author is understandably frustrated and disappointed with the outcome of this project and will attempt to make some more substantial progress before the final Viva examination. In wrapping up this report though, it is worth including sections on what lessons have been learnt from this project, Section 7.2, and what work can be done in the future to produce a more satisfactory output for this project problem area, Section 7.3.

7.2 Lessons Learnt

A number of lessons have been identified (and will in due course be learnt) from the work for this project in terms of the Author’s personal achievements, they are summarised in bullet point below:

- Care needs to be taken not to be overconfident in terms of programming ability; the assumption that any programming issues could be solved was not true!
- The Author did not use his supervisor in the best possible way, should have spoken to Dr Phil Palmer more often!
- “Rabbit-Holes” are very easy to lose yourself in!
- It is very easy to get distracted! The Author admits that turning 30 during the summer did not help with project work!

7.3 Future Work

Given the lack of success in modelling this problem and the issues raised whilst programming in C#, there is still a number of areas of future work that could be completed. The Author
feels it is extremely likely that programming work will continue beyond the end of this project, and a page has been set up on the Author’s Blog [23] so that continued work can be presented to the wider world! A simple list of areas that require continued study is included below:

- Continued modelling work using C# to solve Ordinary Differential Equations – this is vital for any success of the C# program.
- C# program to calculate Halo Orbits – Once ODEs can be solved this should be possible.
- Resonant Forcing Analysis – Further analysis of the literature surveyed in Section 5 is required to produce a suitable method for modelling resonant forcing.
- Resonant Forcing Modelling – Once a suitable method has been produced, modelling will be possible, with a comparison to current spacecraft fly-by techniques possible.
REFERENCES

Problem and Space Mission Design, 2006,
http://www.cds.caltech.edu/~koon/papers/KoLoMaRo_DMissionBk.pdf
[8] Celestial Mechanics Notes Set 5: Symmetric Periodic Orbits of the Circular Restricted
ThreeBody Problem and their Stable and Unstable Manifolds, J.D. Mireles James, De-
Problem Made Easy, http://www.princeton.edu/~ekolenen/publications/kolemen-kasdin-
gurfil_-_quasi_periodic_orbits_of_RTBP_made_easy.pdf
[11] The Code Project Open License (CPOL) 1.02,
[12] Microsoft Chart Controls for Microsoft .NET Framework 3.5,
http://www.microsoft.com/downloads/details.aspx?FamilyId=130F7986-BF49-4FE5-
9CA8-910AE6EA442C&displaylang=en
[13] Grafic Display Charting Tool,
http://www.codeproject.com/KB/miscctrl/GraphPlotting.aspx
[18] Cornish, Neil, J, Lagrange Point Notes,
http://wmap.gsfc.nasa.gov/media/ContentMedia/lagrange.pdf
soft.com/public/gravitat.html
[21] Allan, R.R., Resonance Effects Due to the Longitude Dependence of the Gravitational
field of a rotating Primary, RAE, 9 August 1966


APPENDIX 1 - WORK PLAN

The project is split into the following key tasks:

1. Background - Including literature review and initial understanding of theory
2. Jan Submission - Preparation of this submission
3. Method Development - Initial coding of software to solve 3-body problem
4. March Submission - Preparation and Report writing
5. Poster - Preparation of poster for presentation
6. Summer Semester Planning - Time set aside for ensuring project plan is still valid for summer semester
7. Final Coding - Final coding of software, porting of code to C#, Waterfall development method
8. Validation & Verification - Checking that software is suitable and meets requirements
10. Viva – Preparation
<table>
<thead>
<tr>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Agreement</td>
<td>03/11/08</td>
<td>03/11/08</td>
<td></td>
</tr>
<tr>
<td>Background</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background Reading</td>
<td>03/11/08</td>
<td>08/05/09</td>
<td></td>
</tr>
<tr>
<td>Literature Survey</td>
<td>09/11/08</td>
<td>24/11/08</td>
<td></td>
</tr>
<tr>
<td>Code Survey</td>
<td>24/11/08</td>
<td>15/11/08</td>
<td></td>
</tr>
<tr>
<td>Initial reading</td>
<td>24/11/08</td>
<td>15/12/08</td>
<td></td>
</tr>
<tr>
<td>Advance Survey</td>
<td>27/12/08</td>
<td>09/01/09</td>
<td></td>
</tr>
<tr>
<td>Data Submission</td>
<td>28/12/09</td>
<td>19/01/09</td>
<td></td>
</tr>
<tr>
<td>Deadline</td>
<td>28/12/09</td>
<td>19/01/09</td>
<td></td>
</tr>
<tr>
<td>Method Development</td>
<td>06/05/09</td>
<td>22/05/09</td>
<td></td>
</tr>
<tr>
<td>Coding Ready Capture</td>
<td>06/05/09</td>
<td>26/05/09</td>
<td></td>
</tr>
<tr>
<td>Initial Coding</td>
<td>19/05/09</td>
<td>06/05/09</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>22/05/09</td>
<td>22/05/09</td>
<td></td>
</tr>
<tr>
<td>March Submission</td>
<td>23/05/09</td>
<td>06/04/09</td>
<td></td>
</tr>
<tr>
<td>Document</td>
<td>23/05/09</td>
<td>06/04/09</td>
<td></td>
</tr>
<tr>
<td>Feedback</td>
<td>30/03/09</td>
<td>06/04/09</td>
<td></td>
</tr>
<tr>
<td>Poster</td>
<td>06/03/09</td>
<td>22/03/09</td>
<td></td>
</tr>
<tr>
<td>Deadline</td>
<td>06/03/09</td>
<td>22/03/09</td>
<td></td>
</tr>
<tr>
<td>Summer Semester Planning</td>
<td>09/03/09</td>
<td>18/03/09</td>
<td></td>
</tr>
<tr>
<td>Method Refinement</td>
<td>13/04/09</td>
<td>20/04/09</td>
<td></td>
</tr>
<tr>
<td>Final Coding</td>
<td>01/05/09</td>
<td>20/04/09</td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>01/05/09</td>
<td>20/04/09</td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>22/04/09</td>
<td>06/04/09</td>
<td></td>
</tr>
<tr>
<td>Validation &amp; Verification</td>
<td>23/05/09</td>
<td>23/05/09</td>
<td></td>
</tr>
<tr>
<td>Main Report</td>
<td>13/06/09</td>
<td>24/06/09</td>
<td></td>
</tr>
<tr>
<td>Document</td>
<td>13/06/09</td>
<td>17/06/09</td>
<td></td>
</tr>
<tr>
<td>Printing</td>
<td>17/06/09</td>
<td>24/06/09</td>
<td></td>
</tr>
<tr>
<td>Thesis</td>
<td>19/06/09</td>
<td>14/06/09</td>
<td></td>
</tr>
<tr>
<td>Preparation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX 2 – POSTER

Final version of this poster will be in A3 format
APPENDIX 3 – USER GUIDE

GDB CR3BP USER GUIDE

This document is part of the MSc project *Resonant Forcing of L-Point Orbits* and provides a guide to how to use the GDB CR3BP software (from here on referred to as the *Program*). Before reading this guide, you should read the Installation Instructions [APPENDIX 4].

Once the *Program* has been installed correctly, the *Program* is designed to be simple to use. On loading, a user will see the form as shown in Figure 24.

![Figure 24: The GDB CR3BP Program](image)

In this document that follows, a quick-start guide is provided. For more information regarding the *Program*, users should read the main report.
U 1: Quick Start Guide

All of the main user controls are found on the left hand side of the form, and the basic premise of the Program is a 3-step method. This method is explained below using numbered bullets.

1. Firstly, select the system to be modelled using the “Choose Mu” section of the form, as shown in Figure 25.

![Choose Mu](image1)

*Figure 25: Choose Mass Parameter, Mu*

2. Now select the output options you would like, this is done by ticking the relevant areas in the “Choose Output” section, see Figure 26.

![Choose Output](image2)

*Figure 26: Choose Output*

3. Finally, the engine to be used is selected, this is a simple choice between “Matlab” and “GDB CR3BP”, using the area of the form as shown in Figure 27.
Figure 27: Choose Engine

4. Now press the “Calculate” button and the output will be displayed on the tabs (Figure 28), according to what options have been chosen. The output displayed on each tab is summarised in Table 16.

![Choose Engine](image)

**Figure 28: Output Tabs**

<table>
<thead>
<tr>
<th>Tab</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPoint Data</td>
<td>This tab provides output from either engine regarding the L-points output in numerical form.</td>
</tr>
<tr>
<td>LPoints</td>
<td>This tab includes a graphical representation of the L-point data calculated</td>
</tr>
<tr>
<td>Halo Orbit Data</td>
<td>This tab provides numerical output information about a Halo Orbit. (MATLAB Engine Only)</td>
</tr>
<tr>
<td>Halo Orbit (XY)</td>
<td>This tab provides a graphical representation of the Halo orbit calculation. It plots X and Y data of the orbit against each other. (MATLAB Engine Only)</td>
</tr>
<tr>
<td>Halo Orbit (XZ)</td>
<td>This tab provides a graphical representation of the Halo orbit calculation. It plots X and Z data of the orbit against each other. (MATLAB Engine Only)</td>
</tr>
<tr>
<td>Halo Orbit (YZ)</td>
<td>This tab provides a graphical representation of the Halo orbit calculation. It plots Y and Z data of the orbit against each other. (MATLAB Engine Only)</td>
</tr>
<tr>
<td>Halo Solution</td>
<td>This tab provides four graphs that plot the output data for the Halo orbit linearised solution equations. Clockwise from top-left, there is a plot of X Vs Y data, X Vs Z data, YZ data and a plot of X, Y and Z against time. (GDB CR3BP Engine Only)</td>
</tr>
</tbody>
</table>

Table 16: Output data information
APPENDIX 4 – INSTALLATION GUIDE

To be able to install the GDB CR3BP Program please follow the following guide:

Pre-requisites:

- This software has only been tested on Windows XP SP3 & Windows Vista Business SP1, so it is recommended that only these OS are used
- Microsoft .NET Framework 3.5 or higher
- To use the MATLAB engine, MATLAB must be installed.

To install the program requires a two-step approach, as listed below:

1. Copy the contents of the /MATLAB/ folder to a new directory at C://CRTBP/MATLAB/
2. Run the ‘Setup.exe’ program found in the /INSTALL/ folder of the attached CD.

Please note that this install procedure has only been tested on the Author’s machine and any problems encountered should be reported to the Author, who will be able to help you to get the software working. Alternatively, help and advice for installation problems can be found on the Authors website (http://godofbiscuits79.wordpress.com/MSc-project/)


**APPENDIX 5 – C# PROGRAM CODE**

In this appendix, the key C# code is presented, including the code for the MainForm, MatlabCmdLine Class and the Native Methods, GDB_CR3BP Class.

The program was coded in C#; for more information please see the Microsoft Visual C# Developer Centre Website, http://msdn.microsoft.com/en-us/vcsharp/default.aspx.

Any 3rd Party Software that has been used has been used under Licence as detailed at http://www.codeproject.com/info/cpol10.aspx.
C:\C#\CSharp_MATLAB\VSPccj\GB_CE38FForm.cs

1 /// <class name="GB_CE38FForm.cs" />
2 /// <description>Main form for the program</description>
3 /// <description>Calls calculation engine and provides form controls</description>
4 /// <version>0.1</version>
5 using System;
6 using System.Collections.Generic;
7 using System.ComponentModel;
8 using System.Text;
9 using System.Drawing;
10 using system.reflection;
11 using System.Runtime.InteropServices;
12 using System.Runtime.InteropServices.ComTypes;
14 using System.Security;
15 using System.Threading;
16 using System.Text;
17 using System.Windows.Forms;
18 using System.Drawing;
19 using System.Drawing.Charting;
21 using System.Diagnostics;
22 using matlabEngine;
23 using nativeCK3Bp;
24 namespace GB_CE38F
25 {
26    public partial class GB_CE38FForm : Form
27    {
28        // Form controls
29        public bool LockCkBox = false;
30        public bool blkSystem = false; // true when system selected
31        public bool blkOptions = false; // true when options selected
32        public bool lbEngine = false; // engine has been set
33        public string updDateLbl = "";
34        public bool ErrorOccured = false;
35        // count how long calls take
36        public DateTime TimeStart;
37        public DateTime TimeEnd;
38        public TimeSpan RunTime;
39
40        // set up global variables
41        /// <summary>
42        /// Double Setup
43        /// set up masses, all values in kg
44        /// </summary>
45        public bool useMasses = false;
46        public double Mf = 1898600.0000000000000000000.0;
47        public double Ms = 899200.0000000000000000000.0;
48        public double Ms = 48000.0000000000000000000.0;
49        public double Mq = 14860.0000000000000000000.0;
50        public double Me = 16800.0000000000000000000.0;
51        public double Me = 56486.0000000000000000000.0;
52        public double Mlt = 13500.0000000000000000000.0;
53        public double Mb = 59734.0000000000000000000.0;
54        public double Ms = 7437400.0000000000000000000.0;
55        public double Mf = 19885200.0000000000000000000.0;
56        public double MassCmz = 0;
57        public double Lf;
58        public double T;
59        public double U;
60        /// <summary>
61        /// Matlab engine
62        /// </summary>
63        public matlabEngine matlabEngine; matlabEnginelm = new matlabEngine.matlabEngine();
64    }
65    public bool blkmatlabEngine = true;
67    /// <construction>
68    public GB_CE38FForm()
69    {
70        InitializeComponent();
71        lblStatus.Text = "Welcome!";
72        lblVersion.Text = "GB_CE38F: V0.1";
73        //\$0.1.050731
C:\\CERTB\\\Sharp_MATLAB\\\ESProj\\\\2GB_CB1BFForm.cs

```csharp
71
    // auto use abel
72
73    /// <Menu Item>
74    /// Description Form Control File Exit
75    ///
76    private void exitToolStripMenuItem_Click(object sender, EventArgs e)
77    {
78    // exit program
79    Application.Exit();
80    }
81
82    /// <Menu Item>
83    /// Description Form Control About
84    ///
85    private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
86    {
87    
88    /// <Menu Item>
89    /// Description Form Control ListBox Systems Index Changed
90    ///
91    private void listBoxSystems_SelectedIndexChanged(object sender, EventArgs e)
92    {
93    int caseSwitch = listBoxSystems.SelectedIndex;
94    try
95    {
96    UpdateMs(caseSwitch);
97    binSystem = true;
98    }
99    catch (Exception ex)
100    {
101    lblStatusUpdate("Error!", ex.Message);
102    }
103    }
104
105    /// <Menu Item>
106    /// Description Form Control lblStatusUpdate
107    ///
108    private void lblStatusUpdate(string updateString, string updateString2)
109    {
110    lblStatus.Text = updateString;
111    lblStatus2.Text = updateString2;
112    }
113
114    /// <Menu Item>
115    /// Description Form Control textBox_MU
116    ///
117    private void textBox_MU_TextChanged(object sender, EventArgs e)
118    {
119    // check the input is valid
120    }
121    }
122
123    /// <Menu Item>
124    /// Description Form Control chkListBox_Output
125    ///
126    private void chkListBox_Output_SelectedIndexChanged(object sender, EventArgs e)
127    {
128    
129    if (!chkBoxLock)
130    {
131    int SelIndex = chkListBox_Output.SelectedIndex;
132    if (chkListBox_Output.GetItemCheckState(SelIndex) == System.Windows.Forms.CheckState.Checked)
133    {
134    
135    
136    
137    
138    
139    
140    
141    
142    
143    
144    
```
C:\CERFP\Sharp_MATLAB\FigProj\MGR_CERF.mpForm.cs

```csharp
else
    
    ckbListBox_Output.SetItemChecked(i, true);

binOptions = true;
```
if (blnOptions && blnSystem && blnEngine)
{
    checkBox = true;
    for (int i = 0; i <= caseOutput; i++)
    {
        ckListBox.Output.SelectedIndex = i;
        {
            switch (ckListBox.Output.SelectedIndex)
            {
                case 0: // L-points
                    if (blnMatlabEngine)
                    {
                        matlabEngine1.startMatlab();
                        matlabEngine1.LibrationPoints(MassConst);
                        matlabEngine1.GetPointData();
                        matlabEngine1.stopMatlab();
                        UpdateGraphsUsingMatlabData(true, false, false);
                    }
                    else
                    {
                        NativeMethods.LibrationPoints(MassConst);
                        UpdateGraphsUsingNativeData(true, false, false);
                    }
                    break;
                case 1: // Halo
                    if (blnMatlabEngine)
                    {
                        matlabEngine1.startMatlab();
                        matlabEngine1.HaloOrbit(MassConst, 1, T);
                        matlabEngine1.GetPointData();
                        matlabEngine1.UpdateMatlabData();
                        UpdateGraphsUsingMatlabData(true, true, false);
                    }
                    else
                    {
                        matlabEngine1.stopMatlab();
                    }
                    break;
                case 2:// Halo Solution
                    // use harmonic solution for 3BP
                    if (blnMatlabEngine)
                    {
                        lbiStatusUpdate("No MATLAB Function", "Select MATLAB engine!");
                        // nativeHalo
                        // UpdateGraphsUsingNativeData(true, true, false);
                        break;
                    }
                    break;
                case 3:// Poincare Section
                    if (blnMatlabEngine)
                    {
                        lbiStatusUpdate("Not Implemented!", ",");
                        //matlabEngine1.startMatlab();
                        //matlabEngine1.LibrationPoints(MassConst);
                        //Get L-point data so we know C below
                        //matlabEngine1.GetPointData();
                    }
                    break;
            }
        }
        else
        {
            UpdateGraphsUsingNativeData(false, false, true);
        }
    }
}

break;

case 1:// Halo Solution
if (blnMatlabEngine)
{
    lbiStatusUpdate("No MATLAB Function", "Select MATLAB engine!");
    // UpdateGraphsUsingMatlabData(false, false, true);
}
else
{
    UpdateGraphsUsingNativeData(false, false, true);
}
break;

case 3:// Poincare Section
if (blnMatlabEngine)
{
    lbiStatusUpdate("Not Implemented!", ",");
    //matlabEngine1.startMatlab();
    //matlabEngine1.LibrationPoints(MassConst);
    //Get L-point data so we know C below
    //matlabEngine1.GetPointData();
}
// C = 3.0;
//double CPG = matlabEngine1.C1 - 0.5 *

(matlabEngine1.C1 - matlabEngine1.C2);

//matlabEngine1.evalpoisacarid(MassConst, 1

, 5, CPG);

//matlabEngine1.GetPoints2RelData();

///

UpdateGraphsUsingMatlabData(false, false,

false, true);

//matlabEngine1.stepMatlab();

else

{

lblStatusUpdate("Not implemented!", "");

//UpdateGraphsUsingMatlabData(false, false,

false, true);

break;

}

if (ErrorOccurred)

{lblStatusUpdate("Error Reported!", "");

crResults.Points.Series.Clear();

dlblPoint1.Text = "";

dlblPoint2.Text = "";

dlblPoint3.Text = "";

dlblPoint4.Text = "";

dlMyPoints.Text = "";

lblIC1.Text = "";

lblIC0.Text = "";

lblIC1.Text = "";

lblIC3.Text = "";

lblIC4.Text = "";

lblIC5.Text = "";

} else

{lblStatusUpdate("Calculation complete OK", "");

}

else

{

if (!lblSystem)

{lblStatusUpdate("Check Input Data", "No system selected");

} else if (!lblOptions)

{lblStatusUpdate("Check Input Data", "No output options selected");

} else if (!lblEngine)

{lblStatusUpdate("Check Input Data", "No engine selected");

}

TimeEnd = DateTime.Now;

RunTime = TimeEnd - TimeStart;

lblTime.Text = RunTime.Hours.ToString();

lblTime.Text += "h;";

lblTime.Text += RunTime.Minutes.ToString();

lblTime.Text += "m;";

lblTime.Text += RunTime.Seconds.ToString();

lblTime.Text += "s;";

LockKCRHex = false;
```csharp
private void UpdateGraphsUsingMatlabData(bool Case0, bool Case1, bool Case2, 
bool Case3)
{
    //two alternatives
    //use graph on G3R3P Form or load multiple on.GraphicsDisplay.
    MainForm
    
    //Set series names
    chrtResults_LPoints.Series[Series1].MarkerSize = 6; // Set
    //Marker size
    chrtResults_LPoints.Series[Series1].MarkerStyle = MarkerStyle.Diamond; // Set marker shape
    
    //add data
    chrtResults_LPoints.Series[Series1].Points.Clear();
    chrtResults_LPoints.Series[Series1].Points.AddXY(-matlabEngine1.mu0, 0);
    chrtResults_LPoints.Series[Series1].Points.AddXY((1 + 
    matlabEngine1.mu0), 0);
    
    Series2 = "\Points";
    chrtResults_LPoints.Series[Series2].Points.Clear();
    chrtResults_LPoints.Series[Series2].MarkerStyle = MarkerStyle.Circle; // Set marker shape
    
    //add data
    for (int k = 1; k <= 5; k++)
    {
        chrtResults_LPoints.Series[Series3].Points.AddXY(Convert.ToDouble(matlabEngine1.cmx_LPoints[k, 1].Re), 
        Convert.ToDouble(matlabEngine1.cmx_LPoints[k, 2].Re));
    }

    //L1
    lblPoint1.Text = "x: ";
    lblPoint1.Text += matlabEngine1.cmx_LPoints[1, 1].ToString() + 
    "y: ";
    lblPoint1.Text += matlabEngine1.cmx_LPoints[1, 2].ToString() + 
    "z: ";
    lblPoint1.Text += matlabEngine1.cmx_LPoints[1, 3].ToString()
    
    //L2
    lblPoint2.Text = "x: ";
    lblPoint2.Text += matlabEngine1.cmx_LPoints[2, 1].ToString() + 
    "y: ";
    lblPoint2.Text += matlabEngine1.cmx_LPoints[2, 2].ToString() + 
    "z: ";
    lblPoint2.Text += matlabEngine1.cmx_LPoints[2, 3].ToString()
    
    //L3
    lblPoint3.Text = "x: ";
    lblPoint3.Text += matlabEngine1.cmx_LPoints[3, 1].ToString() + 
    "y: ";
    lblPoint3.Text += matlabEngine1.cmx_LPoints[3, 2].ToString() + 
    "z: ";
    lblPoint3.Text += matlabEngine1.cmx_LPoints[3, 3].ToString()
```
lblPoint3.Text += " | ";
//L4
lblPoint4.Text += "x[ ";
lblPoint4.Text += matlabEngine.cmx_LPoints[4, 1].ToString() + " |

lblPoint4.Text += " | y[ ";
lblPoint4.Text += matlabEngine.cmx_LPoints[4, 2].ToString() + " |

lblPoint4.Text += " | z[ ";
lblPoint4.Text += matlabEngine.cmx_LPoints[4, 3].ToString() + " |

lblPoint4.Text += " | ";
//L5
lblPoint5.Text += "x[ ";
lblPoint5.Text += matlabEngine.cmx_LPoints[5, 1].ToString() + " |

lblPoint5.Text += " | y[ ";
lblPoint5.Text += matlabEngine.cmx_LPoints[5, 2].ToString() + " |

lblPoint5.Text += " | z[ ";
lblPoint5.Text += matlabEngine.cmx_LPoints[5, 3].ToString() + " |

lblPoint5.Text += " | ";

///////////
//energies
lblC1.Text = Convert.ToString(matlabEngine.C11);
lblC2.Text = Convert.ToString(matlabEngine.C12);
lblC3.Text = Convert.ToString(matlabEngine.C13);
lblC4.Text = Convert.ToString(matlabEngine.C44);
lblC5.Text = Convert.ToString(matlabEngine.C55);

} // if (Case1)

// Halo Orbit
// XY
Series1 = "HaloOrbit";

// Set marker shape
chart Halo1.Series[Series2].MarkerSize = 6; // Set marker size
Series2 = ";Point";

// Set marker shape
chart Halo1.Series[Series2].MarkerSize = 12; // Set marker size
chart Halo1.Series[Series2].Points.AddX(Convert.ToDouble(matlabEngine.cmx_LPoints[1, 1].Re), Convert.ToDouble(matlabEngine.cmx_LPoints[1, 2].Re));

for (int k = 1; k <= 999; k++)
{
    chart Halo2.Series[Series1].Points.AddX(Convert.ToDouble(matlabEngine.cmx_LPoints[k, 1].Re), Convert.ToDouble(matlabEngine.cmx_LPoints[k, 2].Re));
}

// XY

// Set marker shape
chart Halo2.Series[Series1].MarkerSize = 6; // Set marker size

// Set marker shape
chart Halo2.Series[Series2].MarkerSize = 12; // Set marker size
chart Halo2.Series[Series2].Points.AddX(Convert.ToDouble(matlabEngine.cmx_LPoints[1, 1].Re), Convert.ToDouble(matlabEngine.cmx_LPoints[1, 3].Re));

} //add data

// XY

// Set marker shape
chart Halo2.Series[Series2].MarkerSize = 6; // Set marker size

// Set marker shape
chart Halo2.Series[Series2].MarkerSize = 12; // Set marker size
chart Halo2.Series[Series2].Points.AddX(Convert.ToDouble(matlabEngine.cmx_LPoints[1, 1].Re), Convert.ToDouble(matlabEngine.cmx_LPoints[1, 3].Re));

} //add data
for (int k = 1; k <= 1999; t++)
{
    chr Halo3.Series[Series1].Points.XY[Convert.ToDouble(matlabEngine1.cmtr_xHalo(k, 1), Me)], Convert.ToDouble(matlabEngine1.cmtr_yHalo(k, 1), 1), Me));
}

//YZ
// Set marker shape
chr Halo3.Series[Series1].MarkerSize = 6; // Set marker size
// Set marker shape
chr Halo3.Series[Series2].Points.AddXY[Convert.ToDouble(matlabEngine1.cmtr_xHalo(k, 2), Me), Convert.ToDouble(matlabEngine1.cmtr_yHalo(k, 3), 1), Me]);

//Add data
for (int k = 1; k <= 1999; t++)
{
    chr Halo3.Series[Series1].Points.AddXY[Convert.ToDouble(matlabEngine1.cmtr_xHalo(k, 1), 2), Me], Convert.ToDouble(matlabEngine1.cmtr_yHalo(k, 1), 3), Me));
}

lblIC.Text = "x: ";
lblIC.Text += matlabEngine1.cmtr_InitialState[1, 1].ToString();
lblIC.Text += "y: ";
lblIC.Text += matlabEngine1.cmtr_InitialState[2, 1].ToString();
lblIC.Text += "z: ";
lblIC.Text += matlabEngine1.cmtr_InitialState[3, 1].ToString();
lblIC.Text += "in";
lblIC.Text += matlabEngine1.cmtr_InitialState[4, 1].ToString();
lblIC.Text += "in";
lblIC.Text += matlabEngine1.cmtr_InitialState[5, 1].ToString();
lblIC.Text += "in";
lblIC.Text += matlabEngine1.cmtr_InitialState[6, 1].ToString();

lblHalo.Text = "x: ";
lblHalo.Text += matlabEngine1.cmtr_Halo[1, 1].ToString();
lblHalo.Text += "y: ";
lblHalo.Text += matlabEngine1.cmtr_Halo[2, 1].ToString();
lblHalo.Text += "z: ";
lblHalo.Text += matlabEngine1.cmtr_Halo[3, 1].ToString();
lblHalo.Text += "n";
lblHalo.Text += matlabEngine1.cmtr_Halo[4, 1].ToString();
lblHalo.Text += "n";
lblHalo.Text += matlabEngine1.cmtr_Halo[5, 1].ToString();
lblHalo.Text += "n";
lblHalo.Text += matlabEngine1.cmtr_Halo[6, 1].ToString();

lblHaloXhi.Text = "xhi: ";
lblHaloXhi.Text += "\n";
for (int i = 1; i <= 6; i++)
{
    for (int j = 1; j <= 5; j++)
    {
        int j = j;
        String s =lblHalPhaseText + j.ToString();
        if (Case2)
        {
        }
        try
        {
            nativeCR3BP();
            double x;
            double y;
            double z;
            double Ax = Convert.ToDouble(txtAx.Text);
            double Ax = Convert.ToDouble(txtAx.Text);
            double Lambda = Convert.ToDouble(txtLambda.Text);
            double Nu = Convert.ToDouble(txtNu.Text);
            double x = Convert.ToDouble(txtX.Text);
            double Phi = 0;
            double Phi = 0;
            int limit = Convert.ToInt32(txtLimit.Text);
            if (limit > 1500)
            {
                double step = Convert.ToDouble(txtStep.Text);
                double t = 0;
                for (int j = 1; j <= limit; j++)
                {
                    Lambda, Phi, out x);
            y = Convert.ToDouble(s);
            chrtHalPhase.Series.Series[3].Points.AddXY(x, y);
            t += step;
        }
        }
        }
    }
    Lambda, Phi, out y);
    x = Convert.ToDouble(s);
    chrtHalPhase.Series.Series[3].Points.AddXY(x, y);
    t += step;
}
```csharp
C:\CTAF\Sharp_matlab\VSCO3\var\var.cs

686 t = 0;
687 for (int j = 1; j <= limit; j++)
688 {
689     y = Convert.ToDouble(sative.CalclFunction_0(t, (k * k + k));
690     x = Convert.ToDouble(sative.CalclFunction_0(t, An, Ns, k));
691     chartHeatSoYX.Series[Series1].Points.AddXY(y, x);
692     t += step;
693 }
694 t = 0;
695 for (int j = 1; j <= limit; j++)
696 {
697     y = Convert.ToDouble(sative.CalclFunction_0(t, An, Ns, k));
698     x = Convert.ToDouble(sative.CalclFunction_0(t, (k * k + k));
699     chartHeatSoYX.Series[Series1].Points.AddXY(y, x);
700     t += step;
701 }
702 }
703 }
704 catch (Exception ex)
705 {
706     lblStatus.Update("Error!", ex.Message);
707 }
708 if (cbms3)
709 {
710     //plot point data
711     //not implemented yet!
712 }
713 } //GraphicsDisplay.MainForm//send data to graph tool
714 //Form Graph = new GraphicsDisplay.MainForm(Me, lPoints);
715 //Graph.Show();
716 }
717 /// <summary>
718 /// UpdatesGraphsUsingNativeData
719 /// updates MSCharts on main form
720 /// </summary>
721 private void UpdateGraphsUsingNativeData(bool Case0, bool Case1, bool Case2, bool Case3)
722 {
723     //two alternatives
724     //graph in GMBAP form or load multiple on GraphicsDisplay.MainForm
725     //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
```
```csharp
  chrResults1Points.Series[Series2].MarkerSize = 6; // Set marker size

  // Add data
  for (int k = 1; k < 5; k++)
  {
    chrResults1Points.Series[Series3].Points.AddXY(Convert.ToDouble(  
        (NativeMethods.cmxt1Points[k, 1].Re), Convert.ToDouble(NativeMethods.cmxt1Points[k, 2].Re)));
  }

  //1
  lblPoint1.Text = "x: [ ";
  lblPoint1.Text += NativeMethods.cmxt1Points[1, 1].ToString();
  lblPoint1.Text += " y: [ ";
  lblPoint1.Text += NativeMethods.cmxt1Points[1, 2].ToString();
  lblPoint1.Text += " z: [ ";
  lblPoint1.Text += NativeMethods.cmxt1Points[1, 3].ToString();
  lblPoint1.Text += " ];";

  //2
  lblPoint2.Text = "x: [ ";
  lblPoint2.Text += NativeMethods.cmxt1Points[2, 1].ToString();
  lblPoint2.Text += " y: [ ";
  lblPoint2.Text += NativeMethods.cmxt1Points[2, 2].ToString();
  lblPoint2.Text += " z: [ ";
  lblPoint2.Text += NativeMethods.cmxt1Points[2, 3].ToString();
  lblPoint2.Text += " ];";

  //3
  lblPoint3.Text = "x: [ ";
  lblPoint3.Text += NativeMethods.cmxt1Points[3, 1].ToString();
  lblPoint3.Text += " y: [ ";
  lblPoint3.Text += NativeMethods.cmxt1Points[3, 2].ToString();
  lblPoint3.Text += " z: [ ";
  lblPoint3.Text += NativeMethods.cmxt1Points[3, 3].ToString();
  lblPoint3.Text += " ];";

  //4
  lblPoint4.Text = "x: [ ";
  lblPoint4.Text += NativeMethods.cmxt1Points[4, 1].ToString();
  lblPoint4.Text += " y: [ ";
  lblPoint4.Text += NativeMethods.cmxt1Points[4, 2].ToString();
  lblPoint4.Text += " z: [ ";
  lblPoint4.Text += NativeMethods.cmxt1Points[4, 3].ToString();
  lblPoint4.Text += " ];";

  //5
  lblPoint5.Text = "x: [ ";
  lblPoint5.Text += NativeMethods.cmxt1Points[5, 1].ToString();
  lblPoint5.Text += " y: [ ";
  lblPoint5.Text += NativeMethods.cmxt1Points[5, 2].ToString();
  lblPoint5.Text += " z: [ ";
  lblPoint5.Text += NativeMethods.cmxt1Points[5, 3].ToString();
  lblPoint5.Text += " ];";

  //energies
  lbC1.Text = Convert.ToString(NativeMethods.C1);
  lbC2.Text = Convert.ToString(NativeMethods.C2);
  lbC3.Text = Convert.ToString(NativeMethods.C3);
  lbC4.Text = Convert.ToString(NativeMethods.C4);
  lbC5.Text = Convert.ToString(NativeMethods.C5);

  if (Casel)
  {
    // Halo Orbit
    //xy
    Series1 = "HaloOrbit";
    chr_Halo1.Series[Series1].MarkerStyle = MarkerStyle.Circle; // Set marker shape
    chr_Halo1.Series[Series1].MarkerSize = 6; // Set marker size
    Series2 = "Pointss";
```

- 75 -
Geoffrey Brown, MSc dissertation

```csharp
lb Halo.Text += NativeMethods commodo[3, 1].ToString();
lb Halo.Text += "\n";
lb Halo.Text += "x dot: ";
lb Halo.Text += NativeMethods commodo[4, 1].ToString();
lb Halo.Text += "\n";
lb Halo.Text += "y dot: ";
lb Halo.Text += NativeMethods commodo[5, 1].ToString();
lb Halo.Text += "\n";
lb Halo.Text += "z dot: ";
lb Halo.Text += NativeMethods commodo[6, 1].ToString();

for (int i = 1; i <= 6; i++)
{
    for (int j = 1; j <= 6; j++)
    {
        lb Halo Phi.Text += NativeMethods commodo_StatePosition[i, j].ToString();
        lb Halo Phi.Text += " \n";
    }
    lb Halo Phi.Text += "\n";
}
}
if (Case2)
{
    // Halo orbits using solution
    Serial1 = "Haloorbit";
    // Halo orbits using solution
    chr HaloSolX, X.Series[Serial1].Points.Clarray();
    chr HaloSolX, X.Series[Serial1].Marker.style = MarkerStyle.Circle;///

    Set marker shape
    chr HaloSolX, X.Series[Serial1].MarkerSize = 1;/// Set marker size
    chr HaloSolX, X.Series[Serial1].Points.Clarray();
    chr HaloSolX, X.Series[Serial1].Marker.style = MarkerStyle.Circle;///

    Set marker shape
    chr HaloSolY, Y.Series[Serial1].MarkerSize = 1;/// Set marker size
    chr HaloSolY, Y.Series[Serial1].Points.Clarray();
    chr HaloSolY, Y.Series[Serial1].Marker.style = MarkerStyle.Circle;///

    Set marker shape
    chr HaloSolZ, Z.Series[Serial1].MarkerSize = 1;/// Set marker size

    try
    {
        // add data
        double x;
        double y;
        double z;
        double Ax = Convert.ToDouble(txt Ax.Text);
        double Az = Convert.ToDouble(txt Az.Text);
        double Lambda = Convert.ToDouble(txt Lambda.Text);
        double Nu = Convert.ToDouble(txt Nu.Text);
        double k = Convert.ToDouble(txt k.Text);
        double Phi = 0;
        double Ps1 = 0;
        int limit = Convert.ToInt32(txt Limit.Text);
        // check we dont do too many steps!
        if (limit > 1500)
        {
            limit = 1500;
        }
        double step = Convert.ToDouble(txt Step.Text);
        if (Lambda == Nu)
        {
            // assume northern halo
            Psi = (Math.PI / 2) - Phi;
        }
        double t = 0;
        for (int i = 1; i <= limit; i++)
        {
            x = Convert.ToDouble(NativeMethods Calc canution_0(t, Ax, Ps1));
        }
    }
```
c:\\\csharp_paulm\\veroj\\gene_cyberpunk.cs

```csharp
lambda, Phi, out x));
```
L = 4.218 * Math.Pow(10.0, 5.0);
V = 17.390;
T = 1.524 * Math.Pow(10.0, 5.0);
textBox M1.Text = MassConst.ToString();
listBox 4n.Items.Clear(); // Update ListBox_4n
listBox 4n.Items.Add("Europa");
listBox 4n.Items.Add("Callisto");
break;

// Jupiter-Europa
if (useMasses)
{
    MassConst = CalculateM(Mj, Mm, out MassConst);
}
else
{
    MassConst = 2.598 * Math.Pow(10.0, -5.0);
    L = 6.711 * Math.Pow(10.0, 5.0);
    V = 13.780;
    T = 3.060 * Math.Pow(10.0, 5.0);
textBox M1.Text = MassConst.ToString();
listBox 4n.Items.Clear(); // Update ListBox_4n
listBox 4n.Items.Add("Europa");
listBox 4n.Items.Add("Callisto");
break;

// Jupiter-Ganymede
if (useMasses)
{
    MassConst = CalculateM(Mj, Mm, out MassConst);
}
else
{
    MassConst = 7.004 * Math.Pow(10.0, -5.0);
    L = 1.070 * Math.Pow(10.0, 6.0);
    V = 10.909;
    T = 6.165 * Math.Pow(10.0, 5.0);
textBox M1.Text = MassConst.ToString();
listBox 4n.Items.Clear(); // Update ListBox_4n
listBox 4n.Items.Add("Ganymede");
listBox 4n.Items.Add("Callisto");
break;

// Jupiter-Callisto
if (useMasses)
{
    MassConst = CalculateM(Mj, Mm, out MassConst);
}
else
{
    MassConst = 5.667 * Math.Pow(10.0, -5.0);
    L = 1.863 * Math.Pow(10.0, 6.0);
    V = 8.276;
    T = 1.438 * Math.Pow(10.0, 5.0);
textBox M1.Text = MassConst.ToString();
listBox 4n.Items.Clear(); // Update ListBox_4n
listBox 4n.Items.Add("Callisto");
listBox 4n.Items.Add("Ganymede");
break;

// Saturn-Titan
if (useMasses)
{
    MassConst = CalculateM(Ms, Mm, out MassConst);
}
else
{
    MassConst = 2.336 * Math.Pow(10.0, -4.0);
    L = 1.522 * Math.Pow(10.0, 6.0);
    V = 5.588;
    T = 1.374 * Math.Pow(10.0, 6.0);
textBox M1.Text = MassConst.ToString();
listBox 4n.Items.Clear(); // Update ListBox_4n
break;

}
```c
C:\CTAF\CTAF_PhilR\Port\src\x86\csharp\Trojan.cs

901 return Mu;
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909
csharp
```
Geoffrey Brown, MSc dissertation

```csharp
namespace matlabEngine
{
    public class matlabEngine
    {
        public double Mm; // Mass
        public double ml; // primary mass
        public double m2; // secondary mass
        public double w; // distance
        public double r1; // r1
        public double r2; // r2
        public double C; // 
        Jacobi Energy
        public double C1; // Jacobi Energy - L1
        public double C2; // Jacobi Energy - L1
        public double C3; // Jacobi Energy - L1
        public double C4; // Jacobi Energy - L1
        public double C5; // Jacobi Energy - L1
        public double C6; // Jacobi Energy - L1
        public double C7; // Jacobi Energy - L1
        public double C8; // Jacobi Energy - L1
        public double L; // 
        public double L2; // 
        public double L3; // 
        public double L4; // 
        public double L5; // 
        public double L6; // 
        
        public double haloPeriod;
        public double Ly;
        public double Lz;
        public double Lx;
        public double Lz;
        
        // Matrix Setup
        // set up matrices
        // CSM.
        public CSM.Matrix cmx_LP3 = new CSM.Matrix(6, 4); // 
        public CSM.Matrix cmx_InitialData = new CSM.Matrix(6, 1); // 
        public CSM.Matrix cmx_State = new CSM.Matrix(6, 1); // 
        public CSM.Matrix cmx_StateTransition = new CSM.Matrix(6, 6); // 
        public CSM.Matrix cmx_x = new CSM.Matrix(2300, 6); // 
        
        // <summary>
        // <remarks>
        // <example>
        // <includeonly>
        // <excludeonly>
        // <thead>
        // <toctree>
        // <version>0.1</version>
        using System;
        using System.Collections.Generic;
        using System.Text;
        using MApp;
        using CSM;
    }
}
```
public void startMatlab()
{
    if (matlab == null)
    {
        matlab = new MApp.MAppClass();
        matlab.Visible = true;
    }

    matlab.Execute("clear");
    matlab.Execute("cd C:\\MATLAB\\\MATLAB\\\\);
}

/// Matlab Engine Stop
/// closes command window, change directory
///
public void stopMatlab()
{
    matlab.Execute("com.mathworks.ml.services.MLEditorServices.closeAll()" );
    matlab.Execute("quit all");
    matlab.Execute("clear");
    matlab.Quit();
    matlab = null;
}

// Function calls

///<summary>
/// LibrationPoints
/// calculates l-points for given system (Mu)
/// matlab engine
///</summary>

///<param name="Mu"></param>
public void LibrationPoints(Double Mu)
{
    try
    {
        string ExecuteString = "";
        ExecuteString += "MassConst = ";
        ExecuteString += Mu.ToString();
        ExecuteString += ";
        ExecuteString += "r1, r2, r3, r4, r5, r6 = 
        executeString();
        ExecuteString += "MassConst ";
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()" );
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");

        catch (Exception ex)
        {
            string matlabErrorString = ex.Message;
        }
}

/// HaloOrbit
/// calculates halo orbit state vector for given l-point & system
/// matlab engine
///</summary>

///<param name="Mu"></param>
///<param name="T"></param>
///<param name="L"></param>
public void HaloOrbit(double Mu, double T, double L) //EarthSunHaloOrbit.m
{
    try
    {
        // Initial Conditions
        ExecuteString = "MassConst = ";
        ExecuteString += Mu.ToString();
        ExecuteString += ";
        ExecuteString += "r1, r2, r3, r4, r5, r6 = 
        executeString();
        ExecuteString += "MassConst ";
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");
        ExecuteString += "; 
        ExecuteString += matlab.Execute("executeString()");

        // physical coordinates are given w.r.t. the earth, so a shift is also needed.
        ExecuteString = "Dist = ";
        ExecuteString += "Set Dist //ae 
        executeString();
        ExecuteString += "L.ToString();

```
Geoffrey Brown, MSc dissertation

```matlab
matlab.Execute(ExecuteString);

// dependent on system!!!!!
"n",

matlab.Execute(ExecuteString);

ExecuteString = "Time=
";

ExecuteString = T.ToString();

matlab.Execute(ExecuteString);

ExecuteString = "v0 = ";

v0 = (1/60) * (1.0 * 1000 / (2 * pi) * (0.0 - 0.0));

// dependent on system!!!!!

matlab.Execute("ode13t(ode,tspan,y0, g"

options,warnargin)

}

} catch (Exception ex)
{
    mlibErrorString = ex.Message;
}

}

</summary>
</C#2004poincareGrid>

// calculates poincare grid for random start position

// matlab engine

<summary>

<param name="num">

<param name="T">

public void PoincareGrid(double Nu, int loops, int iterates, double C)

{ try

    // Initial Conditions

    ExecuteString = "MassConst = ";

    ExecuteString = Nu.ToString();

    ExecuteString = ", T = ";

    ExecuteString = "K = ";

    ExecuteString = "Loops = ";

    ExecuteString = "Iterates = ";

    ExecuteString = "C = ";

    matlab.Execute(ExecuteString);

    // Convert physical units to the scaled units of the CKBF. The physical coordinates are given w.r.t. the earth, so a shift is also needed.

    ExecuteString = "n = ";

    ExecuteString = "size = ";

    ExecuteString = C.ToString();

    matlab.Execute(ExecuteString);

    // C# run MATHLAB code

    matlab.Execute("CKBFpoincareGrid(); ode113(ode,tspan,y0,options, x"

    warning)

    catch (Exception ex)
    {
        mlibErrorString = ex.Message;
    }

}  // data update

<summary>

<param name="x">

<param name="y">

public CSM.EigenMatrix GetMatlabInts(string dataname, int x, int y, out CSM.

Matrix cmat_DataOut)

```
```csharp
CSML.Matrix DataOut = new CSML.Matrix(x, y);
System.Array ar = new double[x, y];
System.Array ai = new double[x, y];
try {
    // Phi 6x6 288 double
    matlab.GetFullMatrix(datasname, "Base", ref ar, ref ai);
    for (int i = 0; i < (x-1); i++)
        for (int j = 0; j < (y-1); j++)
            DataOut(i + 1), (j + 1).Re, = Convert.ToDouble(ar.GetValue(i, j));
    catch (Exception ex)
        stibErrorString = ex.Message;
    catch (Exception)
        cmtx DataOut = DataOut;
    return DataOut;
}

public double GetMatlabData(string datasname, out double dblOutput)
{
    System.Array ar = new double[1];
    System.Array ai = new double[1];
    try {
        //mlx
        matlab.GetFullMatrix(datasname, "Base", ref ar, ref ai);
        dblOutput = Convert.ToDouble(ar.GetValue(0));
    } catch (Exception ex)
    { stibErrorString = ex.Message;
        dblOutput = 0;
    }
    return dblOutput;
}

public void UpdateMatlabData()
{
    string ErrorString = ""
    try {
        //mlx
        //x
        //Term 1
        //Term 2
        //Error Star = "CTarget" //CTargetOrbit
        //CTargetOrbit = GetMatlabData("CTargetOrbit", out CTargetOrbit);
        //msg
    }
}
Geoffrey Brown, MSc dissertation

c:\\csharp\\csharp TempData\\vertoj\\MatlabCodeCopy.cs

```csharp
264  // t_initial  1x1  8  double  
265  // tau  1x1  8  double  
266  // tau_n  1x1  8  double  
267  // testDetPhi  1x1  8  double  
268  // tf  1x1  8  double  
269  ErrorString = "haloPeriod";  // haloPeriod  1x1  8  double  
270  haloPeriod = GetMatlabData("haloPeriod", out haloPeriod);  
271  // out_i  1x1  8  double  
272  // periodicityCheck  1x1  8  double  
273  // deltaInitialEnergy  1x1  8  double  
274  // outTime  1x1  8  double  
275  // x  6x1  48  double  
276  ErrorString = "x";  // x  6x1  48  double  
277  cmx InitialState = GetMatlabData("x", 6, 1, out cmx InitialState);  
278  ErrorString = "x";  // x  6x1  48  double  
279  cmx Halo = GetMatlabData("x", 6, 1, out cmx InitialState);  
280  ErrorString = "x";  // x  6x1  48  double  
281  cmx StateTransition = GetMatlabData("x", 6, out cmx StateTransition);  
282  // x = 6x1  48  double  
283  // PhiEigenValues  6x1  96  double  
284  complex  
285  // 1151x1151  
286  // x  1151x1151  10598400  double  
287  // y  1151x1151  10598400  double  
288  // z  1151x1151  10598400  double  
289  // x2000  
290  // x2000 = x2000  16000  double  
291  // 2000x1  
292  // t  2000x1  16000  double  
293  // x_halo  
294  ErrorString = "x_halo";  // x_halo  2000x6  96000  double  
295  cmx x_halo = GetMatlabData("x_halo", 2000, 6, out cmx x_halo);  
296  ErrorString = "x";  // x  2000x6  96000  double  
297  cmx x = GetMatlabData("x", 2000, 6, out cmx x);  
298  // x_ref  2000x6  96000  double  
299  
300  }  
301  catch (Exception ex)  
302  {  
303      ErrorString = ex.Message;  
304  }  
305  }  
306  // <summary>  
307  // GetLPointData  
308  // update LPoints Matrix to plot  
309  // <param name="x">/\x\</param>  
310  // <param name="x">/\x\</param>  
311  public void GetLPointData()  
312  {  
313      string ErrorString = "";  
314  }  
315  try  
316  {  
317      ErrorString = "CI1";  // CI1  1x1  8  double  
318  }  
319  catch (Exception ex)  
320  {  
321      ErrorString = "CI2";  // CI2  1x1  8  double  
322  }
```

- 85 -
c:\ctsof\csharp_matlab\verso\matlabcodesuite.cs

```csharp
    % CI4 = GetMatlabData("CI4", out CI4);
    ErrorString = "CI4";//CI4

    % CI5 = GetMatlabData("CI5", out CI5);
    ErrorString = "CI5";//CI5

    double CI3 = GetMatlabData("CI3", out CI3);
    ErrorString = "CI3";//CI3

    double CI4 = GetMatlabData("CI4", out CI4);
    ErrorString = "CI4";//CI4

    double CI5 = CL4;

    C3DML.Matrix tmpLP = new C3DML.Matrix(3, 1);
    string Datename = "";
    for (int i = 1; i <= 5; ++i)
    {
        ErrorString = "L";
        ErrorString += i.ToString();
        Datename = "L";//outL1.2.3.4.5 3x1 24 double
        dataname += i.ToString();
        tmpLP = GetMatlabData(dataname, 3, 1, out tmpLP);
        for (int j = 1; j <= 3; j++)
            { cmtx LPoints[i, j].Re = Convert.ToDouble(tmpLP[i, j].Re);
            } //set mtx LPoints[1] to this data
    }

    catch (Exception ex)
    {
        ErrorString = ex.Message;
    }
```

- 86 -
Geoffrey Brown, MSc dissertation

```csharp
using System;
using System.Collections.Generic;
using System.Text;

namespace nativeCR3BP
{
    public class nativeCR3BP
    {
        /// <summary>
        /// double Setup
        /// set up doubles
        /// </summary>
        public double m0; //Mass
        public double m1; //Mass
        public double s0; //Secondary mass
        public double s1; //Secondary mass
distance
        public double r1; //r1
        public double r2; //r2
        public double r3; //r3
        public double C1; //C1
        public double C2; //C2
        public double C3; //C3
        public double C4; //C4
        public double C5; //C5
        public double C6; //C6
        public double r_3TargetObject;
        public double r_3System;
        public double t0;
        public double t1;
        public double t2;
        public double t3;
        public double t4;
        public double t5;
        public double t6;
        
        public CSML.Matrix cmx_VEqult = new CSML.Matrix(3, 1); //V[0]
        public CSML.Matrix cmx_LPoints = new CSML.Matrix(5, 3); //L
        public CSML.Matrix cmxInitialState = new CSML.Matrix(6, 1); //I
        public CSML.Matrix cmx_StateTrans = new CSML.Matrix(6, 1); //P
        public CSML.Matrix cmx_VectorField = new CSML.Matrix(6, 1); //R
        public CSML.Matrix cmx_IState = new CSML.Matrix(6, 1); //State Vector IC
        public CSML.Matrix cmx_Halo = new CSML.Matrix(2000, 6); //Halo Data
        public CSML.Matrix cmx_k = new CSML.Matrix(2000, 6); //Halo Data
        string nativeError = "";
        /// calculates libration points using native
    }
}
```
public void LibrationPoints(Double Nu)
{
    // for a given value of mu, this function computes the location
    // of all five libration points for the circular restricted three
    // body problem. It returns them as equilibrium points in R^3
    // up to space. Then the output is five points each with three components.
    ctx LPoints = CMSL.Matrix.Zeros(5, 3);
    // alternate method
    // compute the location of the libration points
    double l = 1 - Nu;
    // (1/l^2 - 1) * (1, 2 * (mu - 1), 1 + 2 * l * (mu - 1) + mu * l^2, 2 * mu * l * (1 - mu), mu * l^2)
    Polynomial pl = new Polynomial(l, 2 * (mu - 1), Math.Pow(l, 2) - (4 * l * Nu + Math.Pow(Nu, 2)), 2 * Nu * l * (l - Nu) + Nu - l, Math.Pow(Nu, 2));
    Polynomial lPow = (l * Math.Pow(Nu, 2) + 2 * Math.Pow(Nu, 3)) + Math.Pow(Nu, 3) - Math.Pow(l, 3);
    complex[] roots = pl.Roots();
    // initialize L for loop
    ctx LPoints[1, 1].Re = 0;
    for (int i = 1; i < 6; i++)
    {
        // if (roots[i].Re > Nu) && (roots[i].Re < l)
        {
            ctx LPoints[1, i].Re = roots[i].Re;
        }
    }
    ctx LPoints[1, 1].Re = 1 - (Math.Pow(Nu, 3) / 3.0, (1.0 / 3.0));
    ctx LPoints[1, 2].Re = 0;
    ctx LPoints[1, 3].Re = 0;
    ctx LPoints[1, 4].Re = 0;
    ctx LPoints[1, 5].Re = 0;
    // roots = p2.Roots();
    // initialize L for loop
    ctx LPoints[2, 1].Re = 0;
    for (int i = 1; i < 5; i++)
    {
        // if (roots[i].Re > Nu) && (roots[i].Re > l)
        {
            ctx LPoints[2, i].Re = roots[i].Re;
        }
    }
    ctx LPoints[2, 1].Re = 1 + (Math.Pow(Nu, 3) / 3.0, (1.0 / 3.0));
    ctx LPoints[2, 2].Re = 0;
    ctx LPoints[2, 3].Re = 0;
    Polynomial p3 = new Polynomial(l, 2 * (mu - 1), Math.Pow(l, 2) - Math.Pow(Nu, 2), 2 * Nu * l * (l - Nu) - (Nu - 1), Math.Pow(Nu, 2));
    // roots = p4.Roots();
    // initialize L for loop
    ctx LPoints[3, 1].Re = 0;
    for (int i = 1; i < 5; i++)
    {
        if (roots[i].Re < -Nu)
        {
            ctx LPoints[3, i].Re = roots[i].Re;
        }
    }
    ctx LPoints[3, 1].Re = -1 + (5.0 / 12.0) * Nu;
    ctx LPoints[3, 2].Re = 0;
    ctx LPoints[3, 3].Re = 0;
    if (5.4
    ctx LPoints[4, 1].Re = -Nu + 0.5;
    ctx LPoints[4, 2].Re = Math.Sqrt(3.0) / 2.0;
    if (5.5
    ctx LPoints[5, 1].Re = -Nu + 0.5;
    ctx LPoints[5, 2].Re = -Math.Sqrt(3.0) / 2.0;
public double obJaccobi(CSM.Matrix J1, CSM.Matrix J2, double MassConstant) {
    double ReturnValue;
    //the distances
    r1 = Math.Sqrt((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2);
    r2 = Math.Sqrt((x1-x3)^2 + (y1-y3)^2 + (z1-z3)^2);
    r3 = Math.Sqrt((x1-x4)^2 + (y1-y4)^2 + (z1-z4)^2);
    //compute Jacobi Energy
    E = (1/r1)/(Math.Pow(MassConstant + J1[1,1]*r1, 2.0)) + (1/r2)/(Math.Pow(MassConstant + J2[1,1]*r2, 2.0)) + (1/r3)/(Math.Pow(MassConstant + J3[1,1]*r3, 2.0));
    //split calculation to help with validation
    double RV = 1/((x1-x2)/r1 + (x1-x3)/r2 + (x1-x4)/r3);
    TypeReturnValue = RV;
    return ReturnValue;
}
```c
int numSteps = 2000;
CNSL_Matrix tspan = new CNSL_Matrix(numSteps, 1);
double t_initial = 1.45; // initial time guess
double tau_n = t_initial;
double tau = tau_n;
double t0 = 0;
double tF = tau_n;
double outTime;
double CTargetOrbit;
double deltaInitialEnergy;
double halfPeriod;

// ------------------------------------- DATA -------------------------------------

---

// %convert physical units to the scaled units of the CRFBF. The physical coordinates are given w.r.t. the earth, so a shift is also needed.
// % r0 = [1-MassConst; 0; (1/2) * (1200000; 0; 28300000];
// % The time units in the CRFBF are not seconds, but the physical units were given in km/s. For the velocity:
// % v0 = (1/au)^3 * 60*60*24*365.25 / (2*pi)^2 * [0; -0.350; 6]
// % x0 = [0; x0]

// IC
x0[1, 1].Re = (1 - Mu) + ((1 / L) * (-1200000)); // x
x0[2, 1].Re = 0; // y
x0[3, 1].Re = (1 - Mu) + ((1 / L) * (-28300000)); // z
x0[4, 1].Re = 0; // x dot
x0[5, 1].Re = (1 / L) * T / 2 * Math.PI * -3.56; // y dot
x0[6, 1].Re = 0; // z dot

// Compute the Jacobi constant for the given initial conditions
r0[1, 1] = x0[1, 1];
r0[2, 1] = x0[2, 1];
r0[3, 1] = x0[3, 1];
v0[1, 1] = x0[4, 1];
v0[2, 1] = x0[5, 1];
v0[3, 1] = x0[6, 1];
double C = dblJacobi(r0, v0, Mu);

---

// %--------------------------Libration Points------------------------------------
LibrationPoints(Mu); // populates cnts_1Points and Ch1,2,3,4,5

---

// %----------------------Newton Method: find orbits-----------------------------

// % initial guess
x_n = x0;
int i = 14;
for (int i = 1; i < N; i++)
{
    // We need some information from the 'reference trajectory'.
    t0 = 0;
    tf = tau_n;
    numSteps = 2000;
    // tspan = linspace(0, tf, numSteps);
    for (int j = 1; j < numSteps; j++)
    {
        t = tspan[j];
        double tt = tf / numSteps;
        tspan[i, 1].Re = (i + tf);
    }
    // Numerical integration
    y0 = x_n.transpose(); // initial condition
    // options = odeSet('Raletol', 2.5e-14, 'Absrel', 1e-23); // test tolerances
    // [t, x_ref] = ode45('crfbf', tspan, y0, options, [1, 0, MassConst]);
    // Start computing the differential of the Newton condition vector
    // Need the STM for nth guess at sth time (r, t, x, n, MassConst);
    // STM = stateTransSTM(r0, t, x, n, MassConst);
    // Some of the terms depend on the vector field
    s = -vectorfield_CRFBF(y_ref); // s = vectorfield_CRFBF(y_ref, x0, t, tf, tf, 0, MassConst);
    // Construct the needed differential of the vector field
    // constraint vector
}
```
Geoffrey Brown, MSc dissertation

```c
C:\\CSharp\\varialle\vary\native\native.cs

252 //
253 // DF=[Phi(4,3), Phi(4,5), f x(4)];
254 // Phi(6,3), Phi(6,5), f x(6);
255 // Phi(2,3), Phi(2,5), f x(2)];
256 //
257 DF(k, 2) = cmx StateTransition(k, 3);
258
259    //
260 // $\text{\texttt{\textbf{D}}}$ is the inverse of $\text{\texttt{\textbf{DF}}}$;
261 // Compute next (or final/target) initial conditions
262 // Compute next steps
263 // Term 1 = \{ x n(3); x n(5); tau n; \}
264 // Term 2 = \{ x star(3); x star(5); \}
265 // Set up x n for next iteration or output
266 // Put it all into x n (which is really x n(+) as it occurs at the end)
267 // if the loop
268 //
269 // x n=r0(1);
270 // 0;
271 // x star(1);
272 // 0;
273 // x star(2);
274 // 0;
275 // tau=n x star(3);
276
277 //
278 cuxk = x n;
279
280 cutime = tau n;
281 // don't need r0 and v0 anymore so update them and pass to Jacobi
282 r0(1,1)=x n[1,1];
283 r0(2,1)=x n[2,1];
284 r0(3,1)=x n[3,1];
285 v0(1,1)=x n[4,1];
286 v0(2,1)=x n[5,1];
287 v0(3,1)=x n[6,1];
288
289 CTargetOrbit = dbJ{Jacobi(r0, v0, Mu);}
290 deltaInitialEnergy = Maxh.Abs(C - CTargetOrbit);
291
292 ---
293 // Now compute the whole orbit using symmetry
294 r0=0;
295 haloPeriod = 2 * tau n;
296 numSteps=2000;
297 for (int i = 1; i <= numSteps; i++)
298 {
299    // tspan = linspace(0, haloPeriod, numSteps);
300    double tt = haloPeriod / numSteps;
301    tspan[1], 1, 1 = (1 * tt);
302 }
303
304 // Numerical Integration
305 // y0=x n', initial condition
306 y0 = x n.Transpose();
307 options = odeset('RelTol', 1e-2, 'AbsTol', 1e-2); % set tolerances
308 % [x, x.halo] = ode113('CRAWD', tspan, y0, options, [], G, MassCost);
309
310 % // some analysis of the orbit; NOT VITAL SO NOT CODED!
311 % // check the order of magnitude to which the orbit is periodic
312 % // periodicityCheck = norm(x Halo(1, 1:9) - x Halo(numSteps, 1:9))
313 % // Compute the monodromy matrix for the periodic orbit
314 % // Phi = StateTransition(99, haloPeriod, x n, MassCost);
315 % // eigs(Phi)
316 % // testC3(Phi)
317 % // Get eigenvalues and eigenvectors
318 % [Phi, eigvecs, Phi_eigvalOnDiag] = eigs(Phi);
```

- 91 -
Geoffrey Brown, MSc dissertation

```java
// Phi: Eigenvalues-diag(Phi_eigValOnDiag)
// 4 simulate the original conditions
tf = 2.0*π;
numsteps = 2000;
for (int i = 1; i <= numsteps; i++) {
    // tsp: linspace(0, tf, numSteps);
    double tt = tf / numSteps;
    tspan[0, 1], Re = (1 + tt);
}
// Numerical integration
y0, X0: transpose(); // Initial condition
// options: set(’Re=10’, 1e-12, ’ abstol’, 1e-22); tset tolerances
[t,x] = ode15(’CKTB’, tspan, y0, options, 1:6, MassCoast);
```

```
public CSM.Matrix vectorfield_CKTB(CSM.Matrix y)
{
    // function f=vectorfield_CKTB(y, Mu)
    // the distances
    r1 = Math.Sqrt(Math.Pow(Mu+y[1,1], 1.0)+Math.Pow(y[2,1], 1.0));
    r2 = Math.Sqrt(Math.Pow(y[1,1], 1.0)+Math.Pow(y[2,1], 1.0));
    M = Math.Pow(r1, 3.0)/r2^3;
    // m masses
    M1 = Mu;
    M2 = Mu;
    CSM.Matrix ReturnMat = new CSM.Matrix(6, 1);
    ReturnMat[1, 1] = y[4, 1]; //x = y[4];
    ReturnMat[2, 1] = y[5, 1]; // y = y[5];
    ReturnMat[3, 1] = y[6, 1]; // y = y[6];
    ReturnMat[4, 1], Re = y[1, 1], Re + 2 * y[5, 1], Re + M1 * (1 - Mu - y[1, 1], Re) / ε;
    ReturnMat[5, 1], Re = y[2, 1], Re + 2 * y[5, 1], Re + M2 * (1 - Mu - y[1, 1], Re) / ε;
    ReturnMat[6, 1], Re = y[3, 1], Re + M1 * (1 - Mu - y[1, 1], Re) / ε;
    return ReturnMat;
}
```

```
public static CSM.Matrix matrixCKTB(double t0, double tf, CSM.Matrix y, double Mu, out CSM.Matrix ydot)
{
    CSM.Matrix y dot = new Matrix(6, 1);
    // the distances
    double r1 = 0;
    double r2 = 0;
    double m1 = 0;
    double m2 = 0;
    t = Math.Sqrt((Mu + Math.Pow(ConverToDouble(y[1, 1], 5.0)) + Math.Pow((Math.Pow(ConverToDouble(y[2, 1], 2.0)), 3.0)) + Math.Pow(ConverToDouble(y[3, 1], 2.0)));
    r2 = Math.Sqrt((1 - Mu - Math.Pow(ConverToDouble(y[1, 1], 2.0])) + Math.Pow(ConverToDouble(y[2, 1], 2.0))) + (Math.Pow((Math.Pow(ConverToDouble(y[3, 1], 2.0)), 3.0)) + Math.Pow(ConverToDouble(y[3, 1], 2.0));
    // m masses
    m1 = 1 - Mu;
    m2 = Mu;
    y dot[1, 1], Re = y[4, 1], Re / ydot[1, 4];
    y dot[2, 1], Re = y[5, 1], Re / ydot[5];
```
public CSM.LMatrix symbolsolverCRTPD(double t, CSM.LMatrix y, double Nu)
{
    CSM.LMatrix OutputMat = new CSM.LMatrix(6, 1);
    //function ydot=symbolsolverCRTPD(t, y, options, flag, ma)
    //This file contains the right hand side for the system which defines the
    //state transition matrix for the spatial CRTPD.
    //======================================================================

    //The structure of the system is
    // 4f = Ax + f(x)
    //Where A,D,f, are 5x6 matrices and A' is the derivative of A.
    //4f is the derivative of the N-body vector field f and has the form
    //   | 0 1 |
    //   | 0 0 |
    //   | G 0 |
    //Where all submatrices are 3x3 and G has depends only on the position
    //Vectors of the bodies (all be it a complicated way), and x=f(x)
    //is the spatial ODE's. All of this is carried onto one
    //system of 1st order ODEs, the right hand side for the system is coded
    //in the remainder of the file.
    //======================================================================

    //FIRST
    //The matrix G is computed
    //This matrix depends on the positions of the N-body problem. These
    //positions are contained in the 6 entries y(145:151).
    //put the positions aside
    CSM.LMatrix x = new CSM.LMatrix(3, 1);
    x[0, 1] = y[37, 1];
    x[1, 1] = y[38, 1];
    x[2, 1] = y[39, 1];
    //x(1:3)=y(37:39);

    //Now compute the matrix G. Since 'G' already denotes the gravitational
    //constant call the matrix G 'GMatrix'. This is done by calling 'GCRTPD'.
    //GMatrix=GCRTPD(x, Nu);
    CSM.LMatrix G CRTPD = new CSM.LMatrix(3, 3);
    G CRTPD = GMatrix(x, Nu);

    //SECOND
    //The right hand side for the state transition system is computed
Geoffrey Brown, MSc dissertation

```c
CML_Mat C = CML_MatZero(3); // 0 = zeros(3);
CML_Mat I = CML_MatEye(3); // 1 = eye(3);
CML_Mat R = CML_MatZero(3);
E = [0, 2, 0; -2, 0, 1; 0, 1, 0];
R(1, 2) = R(2, 1) = 2;
R(2, 1) = -2;

// Now the complete Jacobian of f is assembled
// If
CML_Mat df = CML_MatBlockMatrix{0, 1, 6, ONTRP, R};
// Id
// G = matrix, R;

// Stake A
for i=1:6
    for j=1:6
        A(i,j) = y(6*(i-1)+j);
    end
end

// Then compute the 6x6 matrix of A, is named dfa.
// dfa = A;
// dfa = dfA;

// This has to be put into vector format. We temporarily place the results
// in the 36-vector 'a'. Later this will be the first 36 components of ydot.
// a = 0;
for i=1:6
    for j=1:6
        a(6*(i-1)+j) = dfA(i,j);
    end
end

// VTHIRD
// The last 6 entries are the vector field for the ONTRP,
// These are stored in 'e'.
--
// ---------The following code is essentially the code from---------
// ---------'CRTP.m' copied here and adjusted for 2 dimensions---------
--
// ---------The distances
// r1 = sqrt((mu+y(37))^2+y(38)^2+y(39)^2);
// r2 = sqrt((1-mu-y(37))^2+y(38)^2+y(39)^2);
// v1 = m1 = 1 - mu;
// v2 = mu;
// c constructs a vector whose first 3 entries are the velocities and whose
// last 3 entries are the accelerations
// e = (y(40);
// y(41);
// y(42);
// y(37)+2*y(41)+G*m1*(m1-y(37))/(r1^3) + G*m2*(1-mu-y(37))/(r2^3);
// y(38)+2*y(40)+G*m1*y(38)/(r1^3) + G*m2*y(38)/(r2^3);
--
// ---------'CRTP.m' ---------
```
Geoffrey Brown, MSc dissertation

```matlab
// Put it all together and pass back to integrator
ydot = a' * y;
OutputMat = CSML_Matrix.TimeSeries(6, 1);
return OutputMat;
```

```matlab
// function returns the matrix 'G' for the CRTPB with parameter nu.

```matlab
public CSML_Matrix GetMatrix(CSML_Matrix x, double mu) {
    // function returns the matrix 'G' for the CRTPB with parameter mu.
    CSML_Matrix Output = new CSML_Matrix(3, 3);
    // the distances
    r1 = Math.Sqrt(Math.Pow(x[1,1].Re + mu, 2) + Math.Pow(x[2,1].Re, 2) +
                    Math.Pow(x[3,1].Re, 2));
    r2 = Math.Sqrt(Math.Pow(x[1,1].Re - (1-Mu), 2) + Math.Pow(x[2,1].Re, 2) +
                    Math.Pow(x[3,1].Re, 2));
    // if U is the potential for the CRTPB then this code is computing the derivative of the gradient of U. The gradient has three components which we will call u1, u2, u3. The differential of the gradient is the matrix of partials of these functions.
    // These will be denoted by u1 x, u1 y, and so forth.
    // u1 x = 1 - (1 - nu) * X / (r1 * r1 * 3 - 3 * x[1] * x[1] + mu * 2) / (r1 * r1 * 5);
    double u1 x = 1 - (1 - nu) * (1 / (Math.Pow(r1, 2) * 3) - 3 * x[1] * x[1] + mu * 2) / (r1 * r1 * 5);
    double u1 y = 1 - (1 - nu) * (1 / (Math.Pow(r1, 2) * 3) - 3 * x[1] * x[1] + mu * 2) / (r1 * r1 * 5);
    // u2 x = 3 * (1 - nu) * X / (r2 * r2 * 3 - 3 * x[1] * x[1] + mu * 2) / (r2 * r2 * 5);
    double u2 x = 3 * (1 - nu) * X / (r2 * r2 * 3 - 3 * x[1] * x[1] + mu * 2) / (r2 * r2 * 5);
    double u2 y = 1 - (1 - nu) * (1 / (Math.Pow(r2, 2) * 3) - 3 * x[1] * x[1] + mu * 2) / (r2 * r2 * 5);
    // u3 x = (1 - nu) * X / (r3 * r3 * 3 - 3 * x[1] * x[1] + mu * 2) / (r3 * r3 * 5);
    double u3 x = (1 - nu) * X / (r3 * r3 * 3 - 3 * x[1] * x[1] + mu * 2) / (r3 * r3 * 5);
    // u3 y = 3 * (1 - nu) * X / (r3 * r3 * 3 - 3 * x[1] * x[1] + mu * 2) / (r3 * r3 * 5);
    double u3 y = 3 * (1 - nu) * X / (r3 * r3 * 3 - 3 * x[1] * x[1] + mu * 2) / (r3 * r3 * 5);
    double u3 z = u1 y;
    return OutputMat;
```
```c
//Matrix=[u1_x, u1_y, u1_z];
// u2_x, u2_y, u2_z;
// u3_x, u3_y, u3_z];
return Output;
}

//<summary>
///Phi Matrix
///calculates Phi Matrix native (THI)
//</summary>
///<param name="t0">"</param>
///<param name="tf">"</param>
///<param name="state">"</param>
public void PhiMatrix(double t0, double tf, CSML.Matrix state)
{

    //Update cmtx StateTransition
cmtx StateTransition = CSML.Matrix.Xeros(6, 6);
    // function A=StateTransCRTBP(t, tf, t, m)
    //compute the state transition matrix at time tf for the
    //interval x(t) with x(t0)=0
    //tspan=[t0,tf];
        //time span over which to run the integration

        //--------------------------------------------------

    //The state transition matrix is determined by a 6x6 matrix Cm
    //This gives rise first to a system of 36 ODEs. However the matrix
    //Cm is nonautonomous and depends on a particular solution

    //of the CRTBP. This trajectory itself the solution of a system
    //of 6 ODEs. The two systems are solved simultaneously, giving an
    //nonautonomous system of 36+6=42 ODEs.
        //--------------------------------------------------

    //Set up the initial condition, y0 for the 42 component system

    //since the initial condition for the 6x6 matrix system is the 6x6
    //identity matrix the initial condition vector is very sparse. The first
    //6x6 components are 0s and 1s, and the last 6 are the initial
    //conditions from the CRTBP.

    //initialize a 6x6 identity matrix
    //for i=1:6
    // y(6*(i-1)+1)=1(i,j);
    //end

    //the initial conditions for the particular orbit of the CRTBP have
    //been passed in as 'a'. This to the end of y.

    //Convert to a column vector and pass the initial conditions to the
    //integrator

    //y0=;'initial
    //tolerances
    //([t,Y]=ode15s('sysolveCRTBP',tspan,y0,options,[],m)); integrate
    //the system
    //y(t,Y) is a huge matrix. it is made up of a row for each time and
    //42
    //columns. Due the desired data is the state transition matrix at
    //the final
    //time. Then the first 36 entries of the last row of y must be put
```
Geoffrey Brown, MSc dissertation

```csharp
c:\csharp\csproj\nativecs.cs

int a
  // 6x6 matrix which will be passed back to the caller
  // /b-size(r); returns the number of rows and columns as a row
  vector
  // b(1,1); len 'm' is the number of rows
  // c=(m,1:36); kmo 'c' is a vector containing the first 36 entries
  // of the last row of Y

  // know c has to be made into a 12x12 matrix
  // /d-0;
  // for i=1;6
  //   d(i,3)=c(6*(i-1)+1);
  // /end
  //end

  // d is the state transition matrix at the final time and is passed
  // back to
  // these caller
  // /d-1;

  // Poincare section
  // Stable Manifold
  // Unstable Manifold
  // Halo orbits
  // CSRM

  // data get functions
  // <summary>
  // GetMatrix
  // returns CSRM.Matrix depending on id
  // </summary>
  // <param name="id"></param>
  // <param name="mtx_Output"></param>

  public CSRM.Matrix GetMatrix(string id, out CSRM.Matrix mtx_Output)
  {
    int i;
    int j;
    CSRM.Matrix mtx_Output1;
    if (id == "m")
    {
      // mtx_initialState
      // i = 6;
      // j = 1;
      mtx_Output1 = new CSRM.Matrix(1, 1);
      try
      {
        mtx_Output1 = cmtx_initialState;
      }
      catch (Exception ex)
      {
        // wrong size?
        nativeError = ex.Message;
      }
    }
    else if (id == "ip")
    {
      // cmtx_LPoints
      // i = 5;
      // j = 3;
      mtx_Output1 = new CSRM.Matrix(1, 1);
      try
      {
        mtx_Output1 = cmtx_LPoints;
      }
      catch (Exception ex)
      {
        // wrong size?
        nativeError = ex.Message;
      }
    }
    else if (id == "cs")
    {
      // cmtx_Tpoints
      // i = 3;
      // j = 1;
      mtx_Output1 = new CSRM.Matrix(1, 1);
      try
      {
        mtx_Output1 = cmtx_Tpoints;
      }
      catch (Exception ex)
      {
        // wrong size?
        nativeError = ex.Message;
      }
    }
```

- 97 -
```csharp
[  
    // ctx_state
    i = 6;
    j = 1;
    mtx_Output1 = new CSML.Matrix[i, j];
    try
    {
        mtx_Output1 = ctx_state;
    }
    catch (Exception ex)
    {
        // wrong size?
        nativeError = ex.Message;
    }
    else if (id == "sd")
    {
        // ctx_state transition
        i = 6;
        j = 6;
        mtx_Output1 = new CSML.Matrix[i, j];
        try
        {
            mtx_Output1 = ctx_stateTransition;
        }
        catch (Exception ex)
        {
            // wrong size?
            nativeError = ex.Message;
        }
        else if (id == "ve")
        {
            // ctx_vectorField
            i = 6;
            j = 6;
            mtx_Output1 = new CSML.Matrix[i, j];
            try
            {
                mtx_Output1 = ctx_vectorField;
            }
            catch (Exception ex)
            {
                // wrong size?
                nativeError = ex.Message;
            }
        }
        else if (id == "vn")
        {
            // ctx_VM
            i = 3;
            j = 1;
            mtx_Output1 = new CSML.Matrix[i, j];
            try
            {
                mtx_Output1 = ctx_VM;
            }
            catch (Exception ex)
            {
                // wrong size?
                nativeError = ex.Message;
            }
            else if (id == "rs")
            {
                // ctx_res
                i = 6;
                j = 1;
                mtx_Output1 = new CSML.Matrix[i, j];
                try
                {
                    mtx_Output1 = ctx_res;
                }
            }
    ```
```csharp
    catch (Exception ex)
    {
        //wrong size?
        nativeError = ex.Message;
    }

    else if (id == "X")
    {
        //ctx.xHalo
        i = 200;
        j = 6;
        mtx_Output1 = new CMK.Matrix(i, j);
        try
        {
            mtx_Output1 = ctx.xHalo;
        }
        catch (Exception ex)
        {
            //wrong size?
            nativeError = ex.Message;
        }

    }

    else if (id == "XX")
    {
        //ctx.x
        i = 200;
        j = 6;
        mtx_Output1 = new CMK.Matrix(i, j);
        try
        {
            mtx_Output1 = ctx.x;
        }
        catch (Exception ex)
        {
            //wrong size?
            nativeError = ex.Message;
        }

    }

    else
    {
        mtx_Output1 = new CMK.Matrix(1, 1);
        mtx_Output = mtx_Output1;
        return mtx_Output;
    }

    //<summary>
    //calcKxFunction 0
    //calculates Y according to linearized solution equation
    //</summary>
    //</param>
    //</param>
    //</param>
    //</param>
    public double CalcKxFunction_0(double idx, double Ax, double Lambda, double Phi)
    {
        return OUT = -Ax * Math.Cos((Lambda + idx) + Phi);
    }
```
APPENDIX 6 – COMPACT DISC

Please find attached in the CD Envelope the following:

- GDB CR3BP Source Code (Requires MS Visual C# studio)
- Install folder
- MATLAB code folder
- Installation & User Guides