To Boldly Go: A Python Framework for Automatic Low-Energy Trajectory Design

by

Joshua Fitzgerald

A Senior Honors Thesis
Submitted to the George Center for Honors Studies
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science with Honors

Advisor: Dr. Stuart Davidson

Greensboro College
2019

ABSTRACT

Traditional space trajectory design methods use two-body models, which only simulate one spacecraft and one planet or star at a time. They consequently generate fast but fuel-inefficient and imprecise pathways through space. Space trajectory design methods that use three-body models, which simulate one spacecraft and two masses at a time, can produce "low-energy" trajectories. Compared to traditionally derived trajectories, low-energy trajectories are slower but are more accurate and are much more fuel-efficient. The goal of this project is to produce a streamlined Python program, tentatively named To Boldly Go, capable of generating low-energy trajectories automatically. To Boldly Go automatically uses the tube manifold approach to low-energy trajectory design to find initial position-velocity conditions that, when integrated, will correspond to an itinerary provided by the user. It operates within the context of a Planar Circular Restricted Three-Body Problem, a straightforward special case of the general three-body problem, with user-defined parameters. Development has followed an iterative design methodology and an approximately functional programming paradigm.

TABLE OF CONTENTS

INTRODUCTION

Low-energy trajectories are paths that can transport an object such as a spacecraft throughout space. Accessing and using them requires relatively little or no fuel. The complex gravitational effects that groups of large masses, such as moons, planets, and stars, exert upon other objects create low-energy trajectories. This thesis describes the creation and properties of *To Boldly Go*, an automatic Python framework for low-energy trajectory derivation. *To Boldly Go* incorporates existing *N*-body dynamical techniques within an easy-to-use computer program.

I programmed *To Boldly Go* following an iterative process in which I coded and tested lower-level components before higher-level components. The framework uses the basic algorithms and procedures that Koon et al. discussed in their work *Dynamical Systems, the Three-Body Problem and Space Mission Design*.

BACKGROUND

Introduction

Low-energy trajectories result from the peculiarities of *N*-body physics (Lo and Ross 4). The most accurate low-energy trajectories account for all the major masses that could perturb a travelling object (Koon et al. 262; Lo and Ross 4). In practice, however, scholars consider them using a special case of the *N*-body problem, the three-body problem (Koon et al. 262; Lo and Ross 4). The two-body problem, another special case, is also relevant under certain conditions (Topputo et al. 356). Many authors consider low-energy trajectories within the context of an Interplanetary Transport Network, a set of low-energy trajectories that enables travel around the solar system with little or no fuel (Lo and Ross 1, 4).

Utilization by Spacecraft and Natural Phenomena

Spacecraft have used low-energy trajectories for a few decades. The first spacecraft to orbit a Lagrange point was the *International Sun-Earth Explorer-3*, or *ISEE-3*, which launched in 1978 (Farquhar et al. 549; "ISEE-3/ICE | Missions"). *ISEE-3* initially gathered data on the solar wind from a halo orbit around the Sun-Earth L_1 point ("ISEE-3/ICE | Missions"). After its mission ended, researchers sent the spacecraft to the Earth's magnetic tail and to a comet and renamed it the *International Cometary Explorer* ("ISEE-3/ICE | Missions").

Unexpected failures forced *MUSES-A/Hiten* to utilize the Interplanetary Transport Network (Belbruno 19-20). In 1990, Japan, attempting to become the third country to send a probe to the Moon, launched two spacecraft named *MUSES-A* and *MUSES-B* (Belbruno 19). *MUSES-A* was supposed to orbit the Earth and to act as a communications relay for *MUSES-B,* which was supposed to reach the Moon (Belbruno 19). The Japanese lost contact with *MUSES-B* before the probe reached a stable lunar orbit (Belbruno 19). Although *MUSES-A*'s designers did not build *MUSES-A* to leave Earth orbit, the Japanese decided to try to send *MUSES-A,* which they renamed *Hiten*, to the moon instead (Belbruno 19). James Miller and Edward Belbruno ascertained a low-energy trajectory that would allow *Hiten* to enter lunar orbit (Belbruno 20, 57-9). The trajectory utilized the gravitational influences of the Sun, the Earth, and the Moon (Belbruno 20, 57-9).

Another spacecraft that utilized low-energy three-body trajectories was the *Genesis* probe (Lo et al. 1, 8-9). *Genesis*, a NASA mission which launched in 2001, gathered solar wind samples from a halo orbit about the Sun-Earth L_1 point ("Genesis : Search for Origins"; Lo et al. 9). *Genesis's* trajectory only utilized a single predetermined fuel burn to reach the

halo orbit (Lo et al. 9-10). *Genesis* was able to expend so little fuel because its trajectory followed the Interplanetary Transport Network (Lo 4).

Chang'e 2 also used low-energy trajectories ("Chang'e 2"). *Chang'e 2,* which the Chinese National Space Administration launched in 2010, travelled to the Moon find possible destinations for China's upcoming *Chang'e 3* lander ("Chang'e 2"). *Chang'e 2* then went to orbit the Sun-Earth L_2 point ("Chang'e 2"). After orbiting the L_2 point for more than seven months, the probe left to encounter the asteroid 4179 Toutatis ("Chang'e 2").

Low-energy trajectory theory explains astrophysical phenomena. Shoemaker Levy 9, a fragmented comet whose pieces collided with Jupiter in July 1994, passed through one of the Sun-Jupiter Lagrange points (Williams; Ross, "Statistical theory of interior-exterior transition" 3; Lo 2). The comet Oterma utilized a low-energy trajectory to travel to and from the vicinity of Jupiter during the 20th century (Koon et al. 30, 119). Three-body trajectories also control ring systems, the Asteroid Belt, and other noteworthy characteristics of our solar system (Lo and Ross 4).

The *N*-Body Problem: Introduction

The *N*-body problem, a scenario in which *N* masses move based upon their gravitational interactions, underlies low-energy trajectory theory (Koon et al. 1-2; Lo and Ross 1-2, 4). The different possible values of *N* determine the basic type of *N*-body problem (Koon et al. 2). Problems with larger *N* more accurately describe real-world situations (Koon et al. 2). Problems with smaller *N* are easier to calculate and can provide useful qualitative information (Koon et al. 2). One often models physical phenomena by starting with a low *N* such as 2 or 3 then gradually increasing *N* (Koon et al. 2). To model the motion of the Earth-Moon system, one might consider the 2-body problem. To obtain more accuracy, one might

utilize the corresponding 3-body problem that also accounts for the Earth, moon, and sun. The 6-body problem that accounts for all major bodies in the inner solar system except for Mars' moons would yield even more correct results. When one introduces an additional body with negligible mass, such as a spacecraft, into an *N*-body problem, one obtains a restricted *N* + 1 body problem (Koon et al. 1-2). Adding the *Apollo 11* spacecraft to the Earth-Moon-Sun three-body problem would create a restricted four-body problem.

The *N*-Body Problem: History

The two-body problem is arguably the earliest expression of the *N*-body problem. Johannes Kepler's planetary laws, which mirrored the later two-body problem, described the relationship between individual planets and the Sun in terms of planetary motion (Riebeek). Kepler's planetary laws, which were published in the early Seventeenth Century, inspired Isaac Newton to develop the mathematical law of gravity (Holton and Brush 45; Riebeek). Newton introduced the law of gravity in his book *Philosophiae Naturalis Principia Mathematica*, which he released in 1687 (Riebeek). Newton solved the two-body problem with his derivation of gravity (Kline 491-2; Riebeek). His research formed the theoretical basis for further work in both the classical two-body problem and *N*-body problems with greater *N* (Kline 491-2; Riebeek). Newton was not the only mathematician to deliver breakthroughs in the two-body problem. Leonhard Euler fully published the analytical properties of the two-body problem in his 1744 treatise *Theoria Motuum Planetarum et Cometarum* (Kline 492).

Although Newton could solve the two-body problem, he completely failed to understand the three-body problem; he alleged that humans were incapable of solving it (Snyder). Later researchers made more progress than Newton would have expected. In the

Eighteenth Century, mathematicians such as Euler and Joseph-Louis Lagrange found specific solutions of the three-body problem (Frank 1; Koon et al. 9). Euler tried to solve the general three-body problem exactly, but, like Newton, failed (Kline 496). Euler did discover the Lagrange points L_1, L_2, and L_3 and did attempt to ascertain the motion of the Moon using one of the first applications of the restricted three-body problem (Koon et al. 9; Frank 1). The restricted three-body problem, or R3BP, assumes that one body has negligible mass (Koon et al. 18; Frank 1).

Mathematicians such as Carl Jacobi, Charles-Eugène Delaunay, and George Hill further developed the R3BP in the Nineteenth Century (Koon et al. 18; Frank 1). Euler and other researchers also began to approximate the three-body problem (Klein 796-7). Lagrange found the remaining Lagrange points L_4 and L_5 (Koon et al. 9). Although Euler discovered some of the Lagrange points first, they are named after Lagrange (Koon et al. 9). Alexis Clairaut also tried and failed to solve the three-body problem exactly (Klein 496-7). He did, however, use series expansions in 1758 to predict that Halley's Comet would make its closest approach to the Sun on April 13, 1759 (Klein 496-7). His prediction was incorrect by only one month (Klein 496-7).

In the late Nineteenth Century, the journal *Acta Mathematica* and King Oscar II, the ruler of Sweden and Norway, promised to award a prize for solving one of four selected mathematical problems (Diacu 67). One problem asked mathematicians to find a general power-series expansion that approximated the solution for the general *N*-body problem for any *N* (Diacu 67). Although no one successfully managed to find the required expansion, Henri Poincaré won the award (Diacu 67). Poincaré introduced qualitative approaches to the study of the three-body problem (Diacu 67-8). He also proved that the first-integrals method

with which mathematicians had attempted to solve the *N*-body problem could not generate a general solution (Diacu 67-8). Poincaré did not prove that the entire *N*-body problem was unsolvable (Diacu 68). He showed that the general *N*-body problem was intractable using a particular method (Diacu 68). Though Poincaré was unable to find the general series expansion for the general *N*-body problem, Karl Sundman successfully derived an expansion that approximated most, but not all, possible cases of the three-body problem in the early 1900's (Diacu 69).

In the 1960's, Charles C. Conley pioneered the modern theory of low-energy trajectories (Conley 722-3; Koon et al. 9-10, 18). Conley found and categorized trajectories in the equilibrium regions about the Lagrange points, introducing the Lagrange point linearization technique (723-3). Conley's work anticipated invariant manifold theory, which postulates that trajectories asymptotically approaching Lagrange point orbits separate different types of low-energy trajectories (Koon et al. 9-10). During the late 1900's and early 2000's, Koon, Ross, and other mathematicians added to Conley's work (Koon et al. 293). These mathematicians developed invariant manifold theory, which considers bundles, or "tubes," of low-energy trajectories approaching and leaving the equilibrium regions about the Lagrange points (Koon et al. 293). Belbruno simultaneously created another framework using Weak Stability Boundaries, or WSB's (47-8). A WSB maps the positions where a mass controls a spacecraft with an arbitrary speed (Belbruno 45, 47-8). Weak Stability Boundary theory often overlaps with invariant manifold theory (Belbruno et al. 12).

The Two-Body Problem

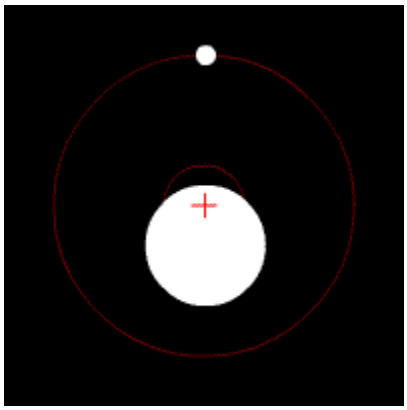

Fig. 1. An example of the two-body problem. Two masses influence each other, orbiting their center of gravity (the red cross) from: Zhatt, *English: Two Bodies with an Extreme Difference in Mass Orbiting around a Common Barycenter (Red Cross) with Circular Orbits*, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:Orbit4.gif, accessed 14 Mar. 2018.

The two-body problem assumes that only two masses gravitationally interact (Koon et al. 2; see, for example, fig. 1). One might use the two-body problem to model the Earth-Moon or Pluto-Charon systems, for example. The two-body problem is completely solvable (Meyer 1). One must recast it in terms of Kepler problems, which assume that a fixed point imposes gravitational forces upon a particle (Meyer 1). Solutions to the Kepler problem, and therefore to the two-body problem, are conic sections (Meyer 1). Because solutions to the two-body problem are conic sections, the masses in a two-body problem move within ellipses, parabolas, and hyperbolas over time (Meyer 1). The two-body problem is solvable because one can produce conic sections that completely describe the motion of the problem's masses (Meyer 1). The two-body problem is unique in this regard (Meyer 1). Dynamicists

believe that one cannot derive equations that solve the *N*-body problem for any initial mass configuration for $N \geq 3$ (Meyer 1).

Mission planners often use the two-body problem to model interplanetary trajectories (Koon et al. 4-5). When using the two-body problem, mission planners assume that the two bodies within the problem are a spacecraft and the mass that currently exerts primary gravitational influence upon a spacecraft (Koon et al. 4-5). If they chain multiple two-body models together, they use an approach called the patched conic approximation (Koon et al. 4). They may then convert this patched conic approximation into a full *N*-body solution (Koon et al. 4-5). The patched conic approximation works well for spacecraft missions using high-energy, relatively robust trajectories, such as those used by the *Voyager* program (Koon et al. 4-5). It is inadequate for lower-energy, more delicate trajectory design (Koon et al. 4-5). Because high-energy trajectories tend to be rather inefficient, using the patched conic approximation can prohibitively increase the amount of fuel needed to reach a destination (Koon et al. 4-5). One needs other models to explain certain natural dynamical behavior—such as Shoemaker Levy 9's final path (Ross 3; Lo 2). The low-energy, fuel efficient trajectories that *To Boldly Go* calculates depend on three-body problems (Topputo et al. 354-5). Two-body problems can theoretically patch together low-energy trajectories, however (Topputo et al. 354-5).

The Three-Body Problem: Introduction and Cases

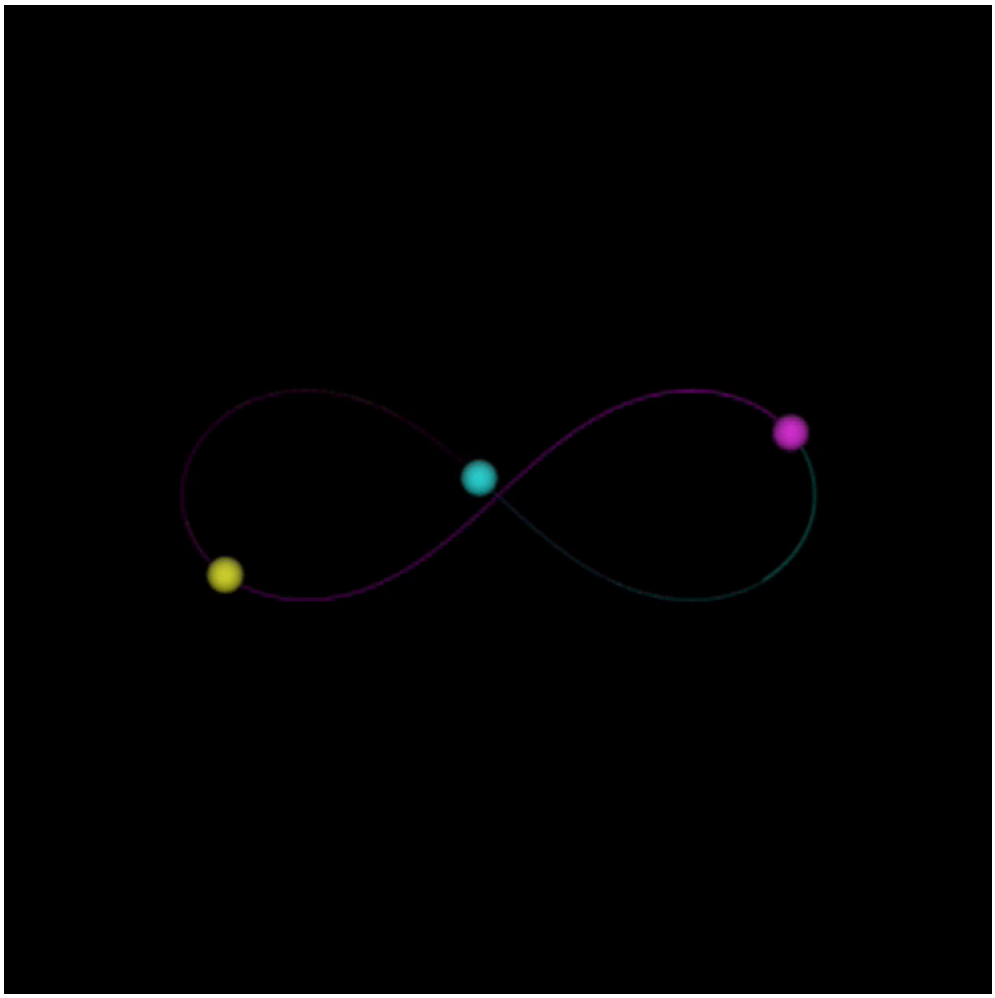

Fig. 2. An example of the three-body problem. Three masses influence each other, moving in a stable figure eight pattern from: Vanderbei, Robert, *Figure8.gif*, Palus, Shannon, "5 Gifs of N-Body Orbits [Animation]," *Scientific American Blog Network*, blogs.scientificamerican.com/guest-blog/5-gifs-of-n-body-orbits-animation/, accessed 14 Mar. 2018.

The three-body problem assumes that three masses gravitationally interact in some manner (Koon et al. 2; see, for example, fig. 2). One might use the three-body problem to model the effects of the Earth and the moon upon a spacecraft. Astrodynamicists often use certain subtypes of the three-body problem (Koon et al. 2, 18). A special case of the R3BP in which the orbits of the two massive bodies around their common center of mass are circular is the circular restricted three-body problem, or CR3BP (Koon et al. 2). A special case of the CR3BP in which all bodies, including the spacecraft, move within the same plane is the planar circular restricted three-body problem, or PCR3BP (Koon et al. 18). This case, the PCR3BP, yields much information about the qualitative behavior of a three-body system. This thesis will primarily describe and use Koon et al.'s tube-based trajectory design framework for the PCR3BP rather than Belbruno's WSB approach.

The Three-Body Problem: Parameters of the PCR3BP

Koon et al. express the PCR3BP using a rotating frame of reference that features normalized units (24-5, 28-9). In a rotating frame of reference, the two masses appear stationary because the frame rotates with them ("Rotating Frame"). For a spacecraft *P* of negligible mass and two massive bodies m_1 and m_2, where $m_1 > m_2$, the unit for distance is the distance between the centers of m_1 and m_2 (Koon et al. 24). The unit for velocity the velocity of m_1, and the unit for time the orbital period divided by 2π (Koon et al. 24-5). These assumptions ensure that the only remaining variable parameter is a mass parameter called μ where

$$\mu = \frac{m_2}{m_1 + m_2}$$ (Koon et al. 25).

In the PCR3BP, because all motion occurs within the same plane, calculations occur in a four-dimensional phase space where two dimensions represent position and two represent velocity (Koon et al. 35).

The Three-Body Problem: Lagrange Points

The normalized rotating coordinate system allows dynamicists to consider the Lagrange points, which are also known as libration points or Lagrangian points (Graps). Lagrange points are special points created by the forces acting upon a spacecraft in the CR3BP (Graps). A spacecraft positioned exactly at a Lagrange point can remain stationary in a rotating reference frame (Graps). A spacecraft close to a Lagrange point can orbit it (Graps).

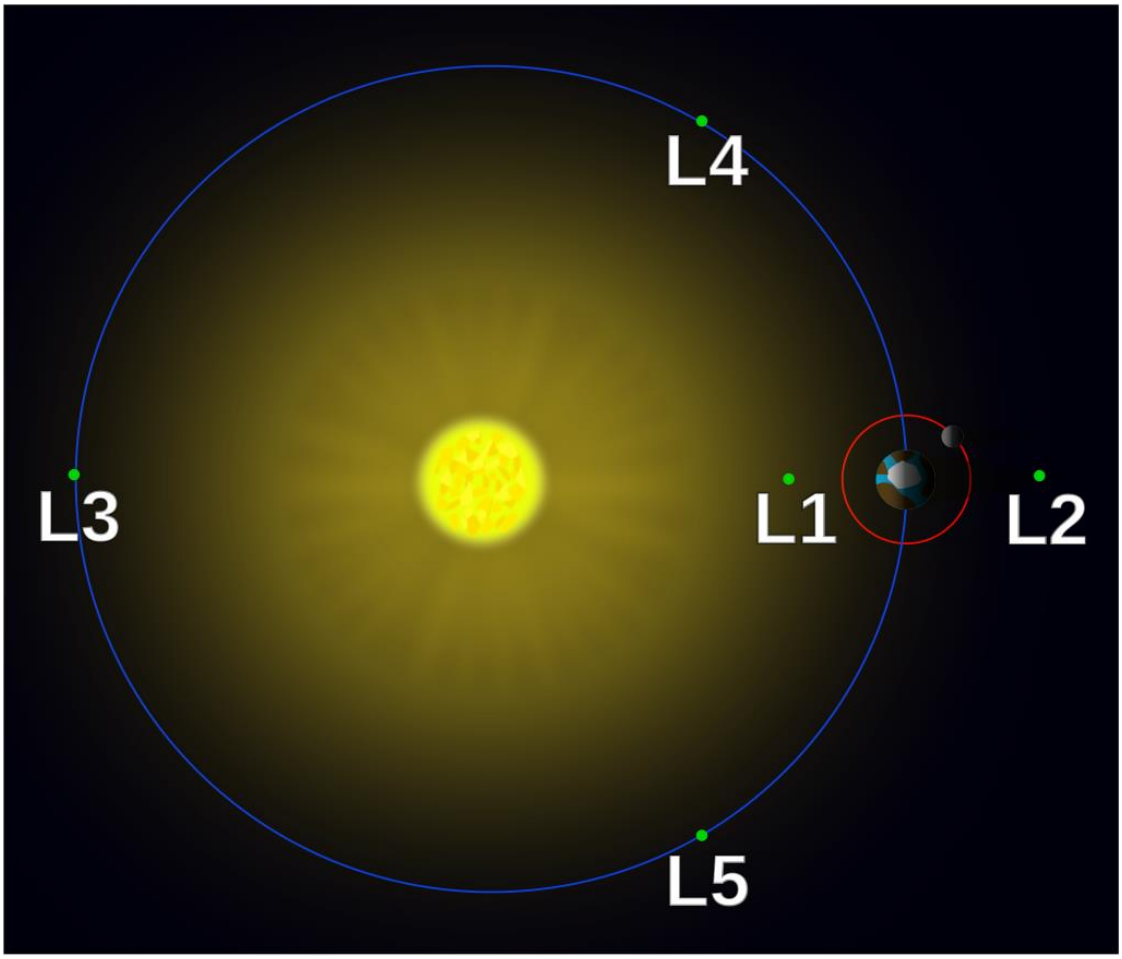

Fig. 3. The Lagrange points of the Sun-Earth three-body problem from: Xander89, *English: Simplified Version of Langrange Contour Plot (More Suitable for Intro IMO)*, 9 Nov. 2014, File:Lagrange_points2.svg, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:Lagrange_points_simple.svg, accessed 14 Mar. 2018.

Five Lagrange points exist in the circular restricted three-body problem (see fig. 3); they are called L_i for $i = 1, 2, \dots, 5$ (Koon et al. 9, 39). L_1, L_2, and L_3 are also called the collinear equilibrium points as they lie within the straight line formed by m_1 and m_2 (Koon et al. 29, 39). L_1 is between m_1 and m_2 but is typically closer to m_2 ("The Lagrange Points"). L_2 lies on the other side of m_2, and L_3 resides on the other side of m_1 ("The Lagrange Points").

L_4 and L_5 are the third vertices of the two equilateral triangles that lie in the orbital plane and whose other two vertices are m_1 and m_2 (Koon et al. 9). Orbits around the collinear Lagrange points are not stable over time; a spacecraft in an orbit around a collinear Lagrange point will eventually leave its orbit ("The Lagrange Points"). The other two Lagrange points are stable when

$$m_1 > 24.96m_2$$ ("The Lagrange Points").

The Three-Body Problem: Solvability

Unlike as in the two-body problem, one cannot derive an exact solution for the general three-body problem (Frank 1). One can, however, use numerical methods or series expansions to approximate solutions (Ryden 14; Kline 497-8). Exact, explicit solutions do exist under specific conditions (Frank 1). One periodic, explicit solution exists such that the masses move on a curve that resembles an infinity symbol (Chenciner and Montgomery 881-884; see fig. 1). Another exists such that the masses move in a circle but form an equilateral triangle (Chenciner and Montgomery 881-884). The three-body problem is extremely useful for designing low-energy trajectories. One can construct low-energy trajectories between the two major masses within the three-body problem, basing them upon the linearized orbits (Koon et al. 107-18). Multiple three-body problems can simulate a more complex problem using a technique called the patched three-body approximation (Koon et al. 4-8). One uses differential correction to convert the patched three-body approximation into a full N-body solution for $N > 3$ (Koon et al. 6). The patched three-body approximation is more suitable for low-energy, delicate trajectories than the patched conic method (Koon et al. 5).

Programming Concepts

Because To Boldly Go is a computer program, this thesis uses some programming terms and concepts. A function encapsulates computer code that performs a specific purpose ("BBC Bitesize"). A call to a function executes the code inside the function ("BBC Bitesize"). Functions can accept inputs, or parameters, and can return values ("BBC Bitesize"). In the Python language, a module is a file containing code that other programs can import ("Modules"). A package is a collection of modules arranged in a directory ("Modules"). The set of modules that standard Python installations provide is called the standard library ("Modules").

A programming paradigm is a way to write and to structure computer programs (Toal). Programs written using a functional paradigm, also known as an applicative paradigm, heavily use functions and avoid storing information outside of them (Toal). Programs written using an object-oriented paradigm are comprised of objects; objects are entities with specifiable behaviors and properties that can pass information to one another (Toal). Programs written as blackboard systems have a centralized repository that contains most of the program's crucial state information (Corkill 1-5).

Python: Introduction

```
1.      def pocorrector(mu, lagrangex, timestep, amplitude, vynew):
2.          """Finds a correct Lagrange point periodic orbit for a mu-
value, a Lagrange point x-coordinate, a timestep, an amplitude,
3.      and a guess y velocity."""
4.
5.          vx1req = False
6.          iterationcount = 1
7.
8.          while True:
9.              print "----
Beginning stabilization iteration " + str(iterationcount) + "----"
10.
11.             vynew, vx1, period = integratecore(mu, timestep, lagrangex - amplitu
de, 0, 0, vynew, crossingtest, [vynew])
12.
13.             if abs(vx1) < 1e-8:
14.                 print "----
Orbit stabilized for acceptable vx1. Use parameters from iteration " + str(iterationcount)
 + "----"
15.                 break
16.
17.             iterationcount += 1
18.
19.         return lagrangex - amplitude, vynew, period #We return the x-
coordinate, y-velocity, and period
```

Fig. 4. A Python function from To Boldly Go.

Python is a high-level, object-oriented programming language first released in 1991 (“General Python FAQ”). Python attempts to balance power and versatility with usability and syntactic clarity (“General Python FAQ”). It is portable, which means that Python programs can run on many operating systems (“General Python FAQ”). It also possesses module and package systems that allow programmers to extend Python by writing Python, C, or C++ code (“General Python FAQ”). Two major versions of the language exist; they are called Python 2 and Python 3 (“General Python FAQ”). To Boldly Go uses Python 2.

Python: REBOUND

The REBOUND package is a framework for integrating the motions of bodies within the *N*-body problem (Rein et al., “Welcome to REBOUND”). Although Rein et al. wrote

REBOUND in the C programming language, the package can also interface with Python (Rein et al., “Welcome to REBOUND”). REBOUND is modular (Rein et al., “Welcome to REBOUND”). REBOUND does not support Windows natively; one must install the Windows Subsystem for Linux to use it on Windows (Rein et al., “Welcome to REBOUND”).

One of the numerical integration components provided with REBOUND is IAS15, a 15^{th}-order, non-symplectic, adaptive timestep algorithm (Rein et al., “Available Modules”). IAS15 is extremely accurate; it usually achieves machine precision during integration (Rein et al., “Available Modules”). To Boldly Go uses the IAS15 integrator.

Python: Shapely

The Shapely package is a computational geometry framework (Gillies et al.). Gillies et al. developed Shapely for GIS systems; consequently, the package primarily analyzes two-dimensional geometric objects. Although Shapely is a Python package, it requires the GEOS C++ framework (Gilles et al.; “GEOS”). To Boldly Go no longer uses Shapely, but this section retains information about the package for historical purposes.

Linux and Operating Systems Terminology

The term Linux refers both to an open-source operating system and to the kernel, or core, of the Linux operating system (IBM Corporation 8). Many versions of Linux exist; each version of Linux is a different software package, or distribution, that builds upon the Linux kernel (IBM Corporation 5). Examples of Linux distributions include Debian, RedHat, TurboLinux, and Ubuntu (“Debian is the rock”; IBM Corporation 5). The Windows Subsystem for Linux is a software framework that allows Linux programs, such as REBOUND, and distributions to run within Windows (Rein et al., “Welcome to

REBOUND"). A command line interface is a text-only interface in which users type commands and their parameters (IBM Corporation 4). A process, also called a thread, is an independently executing subcomponent of a program (IBM Corporation 14). Multiple processes that belong to the same program can run at the same time (IBM Corporation 14).

The TRAPPIST-1 System

TRAPPIST-1A, or simply TRAPPIST-1, is a small star about twelve parsecs away from the Earth's solar system (Gillon et al. 1-2; "About TRAPPIST-1"). TRAPPIST-1 hosts seven known planets, TRAPPIST-1b-h (see fig. 6; Gillon et al. 3). Its solar system is compact; the outermost planet, TRAPPIST-1h, completes a year in 12.35 days (Gillon et al. 2). The planets have Earth-like or smaller than Earth-like radii (Gillon et al. 3).

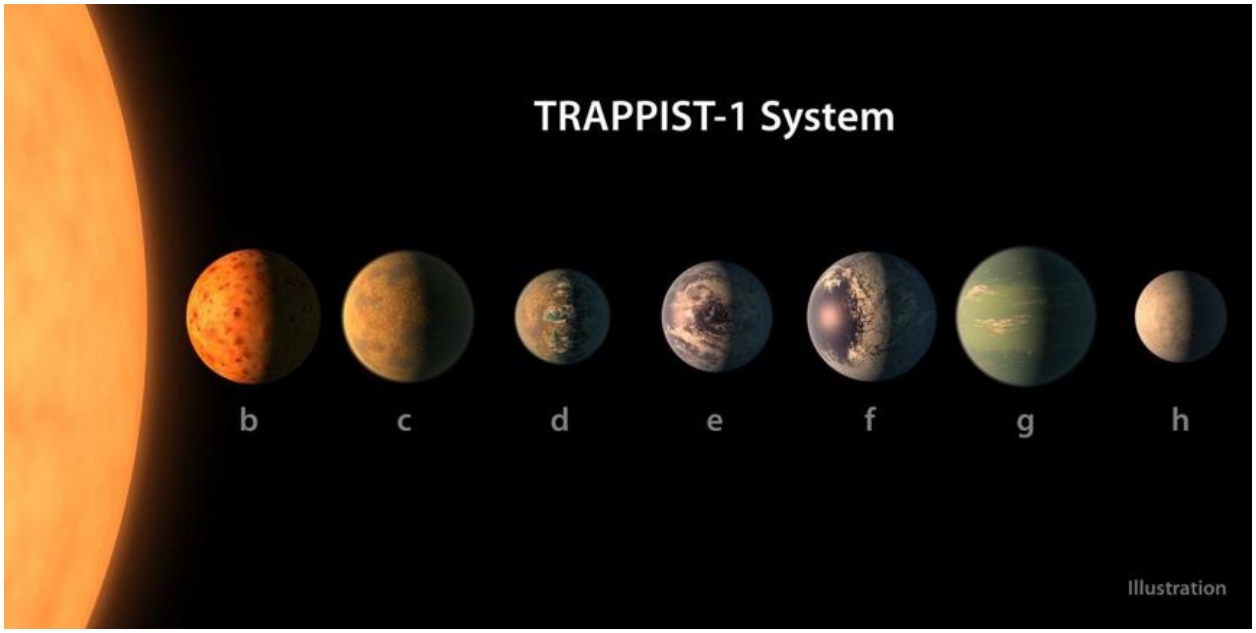

Fig. 6. An artist's depiction of the TRAPPIST-1 system from: "TRAPPIST-1 System," *TRAPPIST-1*, www.trappist.one/images/t1-sys.jpg, accessed 2 Nov. 2017.

The orbits of TRAPPIST-1's planets lie nearly within the same plane and have very low eccentricity (Gillon et al. 3). The planets orbit in a complex resonance (Gillon et al. 3-4). Three of the planets, TRAPPIST-1e-g, are positioned so that their climates could allow them to host oceans (Gillon et al. 4). Although researchers find that conditions on TRAPPIST-1b-d and h are not as favorable as on e-g, they suggest that those planets could still harbor liquid water (Gillon et al. 4).

This thesis research no longer describes the TRAPPIST-1 Interplanetary Transport Network. Information about the solar system remains for historical purposes.

LITERATURE REVIEW

Invariant Manifold Theory: Key Concepts

To Boldly Go utilizes a trajectory derivation algorithm that Koon et al. outline (95-118). The procedure depends upon invariant manifold theory, a mathematical framework in which "tubes" separate different types of trajectories within the Circular Restricted Three Body Problem (Koon et al. 9-10). Koon et al. provide their algorithm in a high-level, narrative manner; To Boldly Go adapts it as a Python program (95-118).

The concept of energy, denoted by *E*, underlies phenomena such as the realms of motion (Koon et al. 34-8). The equation yielding *E* is

$$E(x, y, \dot{x}, \dot{y}) = \frac{1}{2}(\dot{x}^2 + \dot{y}^2) + \bar{U}(x, y)$$

where $\bar{U}(x, y)$ is a value called the augmented potential (Koon et al. 32). The Jacobi integral, *C*, is a similar parameter such that

$C = -2E$ (Koon et al. 33-4).

C and *E* refer to the conserved quantity of the differential equations that determine the motion of *P* in the three-body problem (Ryden 16-7). This conserved quantity is not the kinetic energy plus the potential energy of *P* alone (Ryden 16-7). By using realms, one may grasp the concept of *E* more intuitively (Ryden 16-7).

Realms of motion are distinct areas to which *P* may move for a constant energy (Koon et al. 38; see fig. 4). In a typical three-body problem, three realms of motion and a "forbidden realm" exist (Koon et al. 38). Two realms of motion correspond to the areas of space surrounding m_1 and m_2 (Koon et al. 38). The third realm, known as the exterior realm, is the infinite zone that surrounds the other two realms of motion and the forbidden realm

(Koon et al. 38). The forbidden realm is the area in which *P* cannot move; at its border, *P* has no kinetic energy or velocity (Koon et al. 35-8).

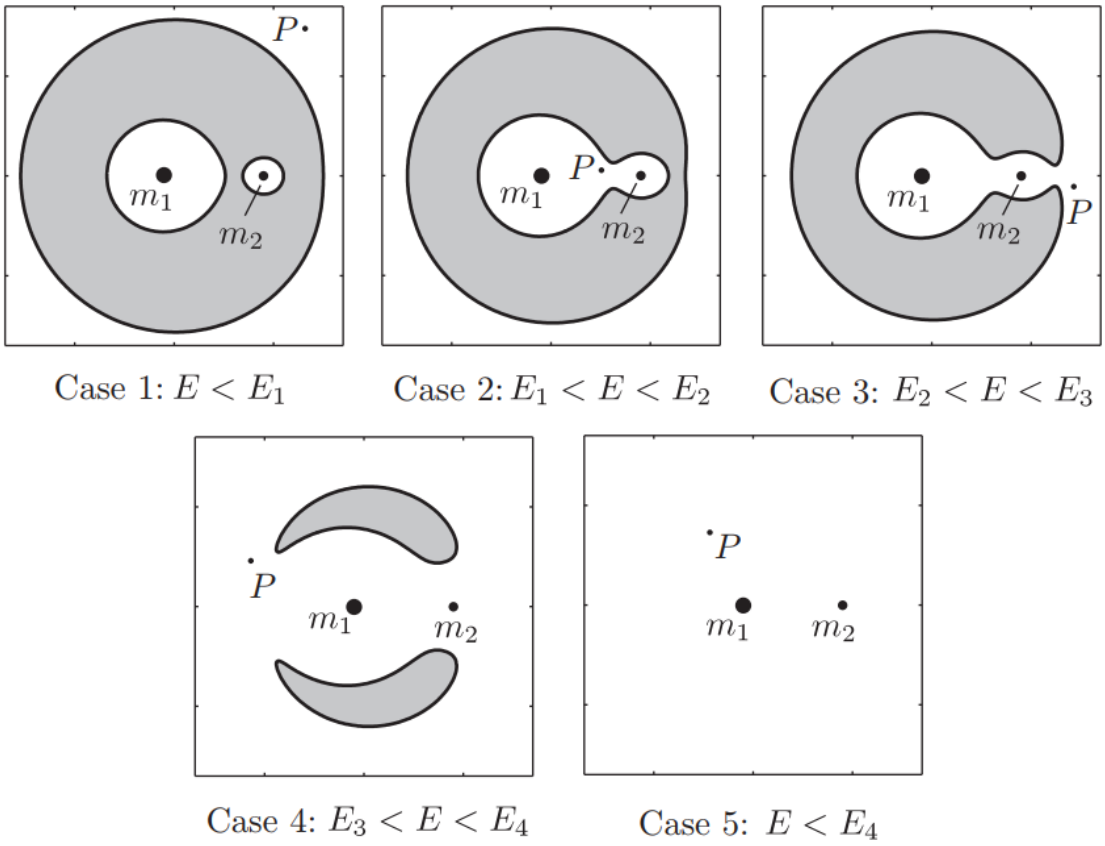

Fig. 4. The five realm scenarios. The gray area denotes the forbidden realm from: Koon, Wang Sang, et al., *Dynamical Systems, the Three-Body Problem and Space Mission Design*, Koon et al., 2011, p. 37.

If *P* has a kinetic energy *E*, then *E* determines five qualitatively different realm scenarios (Koon 36-8). First, for all *E* below some threshold, E_1, *P* must remain in the realm in which it started; the forbidden realm separates all three realms of motion (Koon et al. 37). For example, if $E < E_1$ and the m_1 realm contains *P*, then *P* cannot move outside of m_1's realm if it maintains a constant energy (Koon et al. 37). Second, for all *E* between E_1 and the next threshold, E_2, *P* may move between the m_1 and m_2 realms by travelling along an equilibrium region that forms around L_1 and that connects the two realms (Koon et al. 37). At

these values of *E*, *P* may not move between the m_1-m_2 area and the exterior realm as the forbidden realm separates these two zones (Koon et al. 37). Third, for all *E* between E_2 and the next threshold, E_3, *P* may move between the m_2 and exterior realms by utilizing an equilibrium region around L_2 (Koon et al. 37-8). Although the forbidden realm still exists, the spacecraft may commute between all three realms of motion as the zone connecting the m_1 and m_2 realms still exists (Koon et al. 37-8). Fourth, for all *E* between E_3 and the next threshold, E_4, *P* can directly move from the m_1 realm to the exterior realm, and vice versa, by utilizing an equilibrium region around L_3 (Koon et al. 38). In the previous scenario, *P* must travel through m_2 to reach the exterior region from m_1; in this case, it can commute directly (Koon et al. 38). The forbidden realm still prohibits travel to certain coordinates, but *P* may now access all realms easily (Koon et al. 37-8). Fifth, for all *E* greater than E_4, *P* can travel anywhere as the forbidden zone ceases to exist (Koon et al. 38).

An itinerary is an ordered list of realms through which *P* should travel while using a trajectory (Koon et al. 107, 116). The notation for itineraries is, in general, $(r_1, r_2, \dots, [r_k], \dots, r_n)$ for realms r_i (Koon et al. 107, 116). Each r_i belongs to a set of three symbols representing the m_1, m_2, and exterior realms; for simplicity, To Boldly Go always uses the symbols 1, 2, and X, respectively (Koon et al. 107). The ordering of the realms reflects the order in which *P* visits each realm (Koon et al. 107, 116). *P* is currently within the bracketed realm; it has already visited the realms listed before the brackets and will visit the realms listed after the brackets (Koon et al. 107, 116). The brackets are optional; without them, one does not consider the current location of the particle (Koon et al. 107).

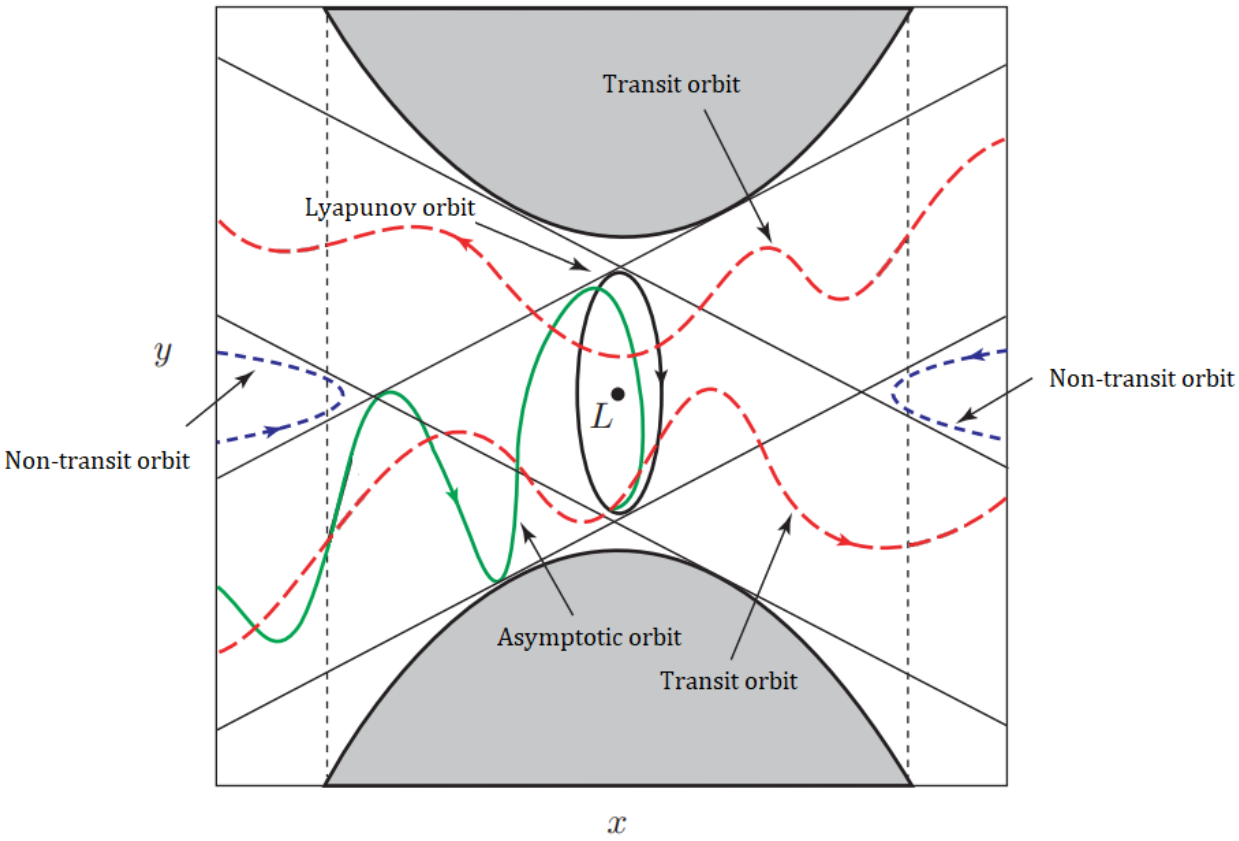

Fig. 5. The types of trajectories in the neighborhood of a linearized Lagrange point *L* modified from: Koon, Wang Sang, et al., *Dynamical Systems, the Three-Body Problem and Space Mission Design*, Koon et al., 2011, p. 55, accessed 14 Mar. 2018.

Linearizing the equilibrium regions sheds light upon the nature of inter-realm travel. For an equilibrium region $\mathcal{R}$, *P* can experience one of four types of trajectories over time *t* (Koon et al. 48-50; see fig. 5). In the first type of trajectory, known as a Lyapunov orbit, *P* enjoys a periodic orbit in $\mathcal{R}$ around $\mathcal{R}$'s Lagrange point (Koon et al. 48-9). Each Lagrange point has only one Lyapunov orbit (D. L. Smith 9). In the second type of trajectory, called an asymptotic orbit, *P* approaches the Lyapunov orbit as $t \rightarrow \infty$ or as $t \rightarrow -\infty$ (Koon et al. 49). The third type of trajectory, known as a transit orbit, allows *P* to travel between the two realms that $\mathcal{R}$ connects (Koon et al. 49). The fourth type of trajectory, a non-transit orbit,

brings P into $\mathcal{R}$ and returns it to the realm from which it came (Koon et al. 49-50). P never reaches a new realm (Koon et al. 49-50).

These trajectories have an important structural relationship: the asymptotic orbits, which approach the Lyapunov orbit, separate the transit orbits and the non-transit orbits from one another (Koon et al. 51). The set of asymptotic orbits approaching the Lyapunov orbit as $t \rightarrow \infty$ is called the stable manifold, whereas the set of asymptotic orbits approaching the Lyapunov orbit as $t \rightarrow \infty$ is called the unstable manifold (Koon et al. 50-1). The transit orbits are within a structure called a Conley-McGehee tube comprised of the asymptotic orbits, whereas the non-transit orbits lie outside of the tube (Koon et al. 9, 51).

In the three-dimensional case of the restricted circular three-body problem, the CR3BP, an additional kind of trajectory exists known as a halo orbit (Koon et al. 142). A halo orbit is a three-dimensional, unstable path around a Lagrange point (Koon et al. 142). One uses differential correction to calculate halo orbits (Koon et al. 142).

Invariant Manifold Theory: Koon et al.'s Algorithm

To Boldly Go uses Koon et al.'s trajectory design algorithm based upon invariant manifold theory instead of a different approach because Koon et al. give their algorithm very explicitly (95-118). Koon et al. provide a detailed, step-by-step set of instructions for deriving trajectories within the Planar Circular Restricted Three-Body Problem; they also give detailed information about the PCR3BP itself and other crucial concepts (23-53, 95-118). Their work is a centralized source of information about invariant manifold theory; a similarly comprehensive work about Belbruno's Weak Stability Boundary theory may not exist.

Koon et al.'s trajectory design algorithm has several steps. Consider a trajectory itinerary such as $(X, 2,1)$ or $(X, 2,1,2, X)$ (Koon et al. 107-8, 118-9). First, one must select the energy that the trajectory will use (Koon et al. 107-8). Low-energy trajectories typically use energy within (E_2, E_3), so one must derive and select a value within this interval by finding the energies of the L_2 and L_3 points (Koon et al. 38, 107-8).

Second, one must find the coordinates of L_1 and L_2, the relevant Lagrange points (Koon et al. 108-9). The collinear Lagrange points have phase-space coordinates of the form $(x, 0,0,0)$ where the x-coordinate for each Lagrange point is a critical point of the function

$$\bar{U}(x, 0) = -\frac{1}{2}x^2 - \frac{1-\mu}{|x+\mu|} - \frac{\mu}{|x-1+\mu|}$$ (Koon et al. 40-1, 108-9).

Third, one must compute the periodic orbits around the L_1 and L_2 Lagrange points that correspond to the chosen energy using differential correction and numerical continuation (Koon et al. 109-112). In differential correction, one computes a guess initial condition for a periodic orbit of the form $\left(x_e - A_x, 0,0, v_{y0}\right)$ where x_e is the x-coordinate of the relevant Lagrange point, A_x is the amplitude of the periodic orbit, and v_{y0} is an initial guess for v_{y0} (Koon et al. 108-119). One then repeatedly integrates the periodic orbit initial condition to the x-axis while simultaneously integrating Φ, the state transition matrix (Koon et al. 109-112). One repeatedly lets

$$v_{y0} \mapsto v_{y0} - \left(\Phi_{34} - \frac{v_{x1}}{v_{y1}}\Phi_{24}\right)^{-1} v_{x1}$$

where $\mapsto$ is the assignment operator and v_{x1} and v_{y1} are the x- and y-velocities at the x-axis crossing (Koon et al. 109-112). This iterative process soon yields an initial condition for a stable periodic orbit (Koon et al. 109-112).

In numerical continuation, one initially uses differential correction to stabilize "seed" orbits with small amplitudes; one uses them to create guess initial conditions for larger orbits that one must then stabilize (Koon et al. 112). One iteratively extrapolates from known periodic orbits to find new periodic orbits until one finds a periodic orbit with the desired energy (Koon et al. 112).

Fourth, one must compute the stable and unstable invariant manifolds corresponding to the periodic orbits (Koon et al. 112-3). One integrates the state transition matrix and the periodic orbit for one period of the periodic orbit; one obtains a special case of the state transition matrix called the monodromy matrix (Koon et al. 112-3). One then calculates the eigenvalues and eigenvectors of the monodromy matrix (Koon et al. 112-3). The eigenvector corresponding to the largest eigenvalue corresponds to the unstable manifold; the eigenvector corresponding to the smallest eigenvalue corresponds to the stable manifold (Koon et al. 112-3). One obtains initial conditions for the stable and unstable manifolds by repeatedly perturbing the periodic orbit initial conditions along the proper eigenvectors with various positive and negative displacements (Koon et al. 112-3).

Fifth, one must use Poincaré sections to capture a two-dimensional picture of the trajectories comprising existing tubes and manifolds (Koon et al. 114-6). Koon et al. identify four Poincaré sections, U_1, U_2, U_3, and U_4, that collectively account for all valid itineraries (Koon et al. 114-6). One integrates two manifolds or existing tubes that match the itinerary backwards or forwards until they both intersect the relevant section; one then considers the points lying on the section (Koon et al. 114-6). One is sometimes forced to allow the manifolds to intersect the section multiple times to obtain useful results (Koon et al. 118-21).

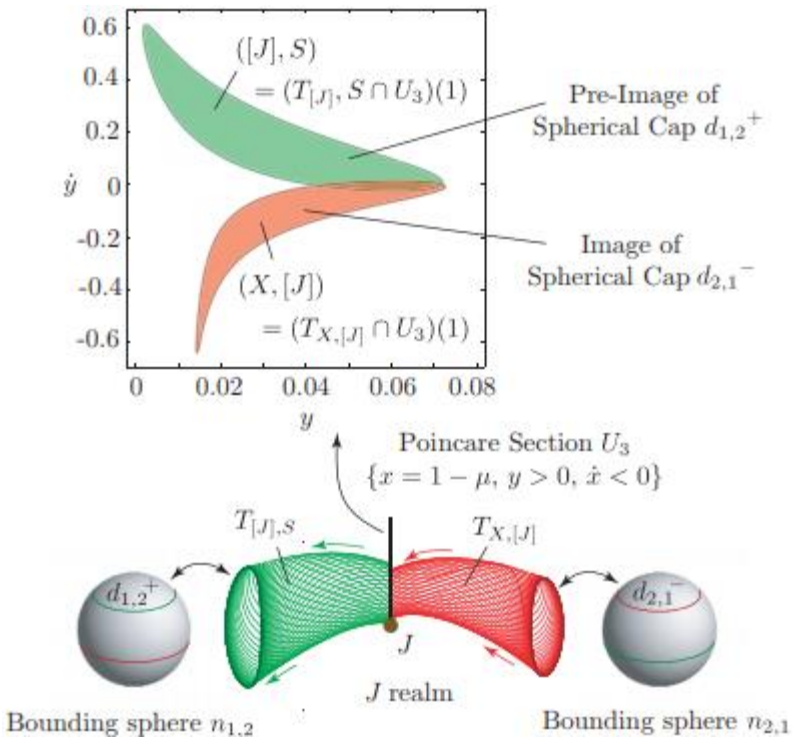

Fig. 6. A visualization of overlapping manifold tubes from: Koon, Wang Sang, et al., *Dynamical Systems, the Three-Body Problem and Space Mission Design*, Koon et al., 2011, p. 115, accessed 5 Dec. 2018.

Sixth, one must intersect the tubes using the Poincaré sections to obtain a new tube matching more of the itinerary (Koon et al. 116-21). Suppose that one has used Poincaré sections to capture a two-dimensional portrayal of the flow of tubes matching the itineraries $(X, [2])$ and $([2], 1)$ (Koon et al. 116-21). Each shape on the section represents a tube boundary; the points inside each shape correspond to initial conditions that match the tube's itinerary (Koon et al. 116-21). For example, the area inside of the $(X, [2])$ tube's shape contains initial conditions that came from the exterior realm and are now within the m_2 realm. The intersection of the shapes therefore represents initial conditions that match the itinerary $(X, [2], 1)$ (Koon et al. 116-21).

Seventh, one must integrate initial conditions within the intersection region (Koon et al. 117-21). One may perform this step for a couple of reasons. If one has just found initial conditions within a tube that only matches part of the itinerary, one may integrate them forward to another Poincaré section to intersect them with another manifold or tube (Koon et al. 117-21). This process will create a new tube that matches the itinerary more completely (Koon et al. 117-21). If one has found an initial condition in a tube corresponding to the entire itinerary, one may integrate it forwards and backwards to obtain a full trajectory for the full itinerary (Koon et al. 117-21).

Programming and Computer Science

To Boldly Go uses Python instead of a different programming language for a couple of reasons. First, Python is a high-level language in which one can quickly and painlessly write succinct code ("Comparing Python"). Python programs can be as much as five times shorter than equivalent Java programs and as much as ten times shorter than equivalent C++ programs ("Comparing Python"). One can write Python programs in a fraction of the time required to write equivalent C++ programs ("Comparing Python").

Second, Python has an extensive ecosystem for which programmers have created countless packages with countless functions ("PyPi"). The Python Package Index, the official repository for Python packages, contains over 150,000 projects as of November 2018 ("PyPi"). CRAN, a similar repository for the R language, contained only 10,000 projects as of January 2017, even though R was the most popular data science programming language at the time (D. Smith).

To Boldly Go uses the REBOUND numerical integration framework because Rein et al. wrote REBOUND in C but also provided an optional Python interface (Rein et al.,

"Welcome to REBOUND"). The Python frontend allows To Boldly Go to utilize REBOUND and the C backend decreases execution time (Rein et al., "Welcome to REBOUND"). Many other numerical integrators are incompatible with Python; for example, the SWIFT integrator uses Fortran (Bottke). To Boldly Go reaps not only the advantages of REBOUND's straightforward Python interface but also the benefits of the framework's very fast C codebase (Rein et al., "Welcome to REBOUND").

METHODOLOGY

Materials

To Boldly Go uses the three-body tube-manifold formulae and methodology outlined by Koon et al. One cannot use patched conics or similar two-body approaches to find ITN trajectories. See the "Procedures" section below.

This thesis research utilizes a variety of computational resources. To solve equations, to find derivatives and integrals, and to perform similar calculations, I used my TI-nspire CX CAS calculator, the Wolfram Alpha web service, Mathematica, and the Wolfram Language. I also utilized the Desmos online graphing calculator during feature testing to plot manifold overlap areas.

To Boldly Go itself uses Python 2.7. It depends upon a variety of Python packages, including but not limited to Numpy, SciPy, Matplotlib, REBOUND, and Shapely. The Python code has been tested using my 64-bit Windows 10 machine within Ubuntu. The Windows Subsystem for Linux interfaces between Windows 10 and Ubuntu; running Ubuntu inside of Windows is necessary because REBOUND does not natively support Windows (Rein et al., "Welcome to REBOUND"). The Windows 10 machine has an Intel Core i7-7700K CPU running at 4.20 GHz and 16 gigabytes of RAM.

Procedures: General Development Methodology

Development of *To Boldly Go* followed a basic iterative methodology. After writing framework components, I tested them for accuracy using sample scenarios that Koon et al. provided. I usually verified their correctness by creating visualizations of component output. For example, I attempted to obtain initial conditions for itineraries corresponding to sample trajectories; I then attempted to show with graphs that the trajectories behaved as they did in

Koon et al.'s work. I wrote and tested higher-level components after creating and verifying lower-level components; I then wrote and tested even higher-level components in a cyclical, iterative process.

I wrote To Boldly Go using an approximately functional programming paradigm. The program is functional because it contains mainly Python functions calling other functions rather than objects interacting with one another with methods or other code structures. I experimented briefly with other paradigms such as the object-oriented approach; I found that the less elegant functional code was substantially faster.

Procedures: Detailed Development Overview

The original purpose of this thesis was to describe the Interplanetary Transport Network for the TRAPPIST-1 system; I originally began writing the program that became To Boldly Go to derive the TRAPPIST-1 ITN's trajectories. Early versions of the code that became To Boldly Go used parameters corresponding to the TRAPPIST-1-TRAPPIST-1e three body problem for testing purposes.

Because I wrote To Boldly Go using an iterative methodology, I created several versions of the program. I usually wrote each new version by altering existing code from earlier versions; the new versions usually possessed additional components or code rewrites to make the framework more general or less error-prone.

I created integratororbits.py, one of the very first versions of the code, by modifying a REBOUND example program that integrated and visualized the trajectory of the comet 67P/Churyumov–Gerasimenko. integratororbits.py instead integrated and visualized the trajectory of a particle at the TRAPPIST-1-TRAPPIST-1e L_1 point. integratororbits.py did not automatically derive the location of the L_1 point; I hardcoded it into the program. Many

other values, such as the mass parameter, were similarly unmodifiable. The main purpose of this program was to test my ability to use REBOUND and to simulate Lagrange points and their locations.

The next program I created was integratororbits1.py. This program was like integratororbits.py; it also visualized particle trajectories. Users could now specify the initial x and vy coordinates of the particle, as well as the integration start and end times, as command line arguments. The program assumed that the initial y and vx coordinates of the particle were both zero. I used integratororbits1.py to experiment with trajectories around and near Lagrange points.

I based integratororbits1a.py upon integratororbits1.py; however, I first began to write integratororbits1a.py after beginning to write integratororbits5a.py, a much later version. I wrote integratororbits1a.py as a "helper" visualizer program. integratororbits2.py and subsequent versions of the framework until toboldlygo2.py lacked native trajectory visualization functionality, so integratororbits1a.py allowed me to visualize the output of these programs. Users could specify initial x, y, vx, and vy particle coordinates, as well as starting and ending visualization times and mass parameters.

integratororbits2.py, which I based upon integratororbits1.py, automatically integrated one-half of an approximation of a Lagrange point periodic orbit. The program integrated a hardcoded initial condition until it automatically detected the "crossing," or the point at which it intersected the x-axis. It accepted the integration stop time and the number of points to be stored during the integration as command line arguments. The program structurally resembled integratororbits1.py; however, it could not create visualizations. Instead, it saved the points that it found on the derived trajectory to a text file.

integratororbits3.py was able to derive the y-velocity of a TRAPPIST-1 - TRAPPIST-1-e L_1 point periodic orbit with amplitude 0.00001. The program accepted a timestep and a guess value for the y-velocity. It then repeatedly applied differential correction to periodic orbit initial conditions until the x-velocity of the particle at the "crossing" was satisfactory. It extensively logged its findings, including initial conditions of the stabilized orbit.

integratororbits4.py was an improved version of integratororbits3.py that also featured numerical continuation. Instead of only stabilizing the TRAPPIST-1 - TRAPPIST-1-e L_1 point periodic orbit with amplitude 0.00001, integratororbits4.py derived periodic orbits for provided energies, mass parameters, Lagrange points, timesteps, and seed orbits. I began to use the argparse Python module instead of the sys module to handle command line functionality; using the argparse library gave the program a proper help infrastructure. integratororbits4.py also contained substantial code rewrites elsewhere; it introduced an early version of the current, modular Core Integrator component.

integratororbits5.py was like integratororbits4.py; however, it used separate processes for particle integration, matrix integration, and integration state testing using the multiprocessing Python library. I discovered that using multiple processes did not significantly improve performance, so I abandoned the concept.

With integratororbits5a.py, I began to add manifold derivation functionality to the integratororbits4.py codebase. I did not completely understand how to derive manifolds, so this version made erroneous assumptions. I also made minor adjustments to the Core Integrator to accommodate new requirements and to attempt to speed up the code.

integratororbits5amap.py fixed the methodological errors in and incorrect assumptions of integratororbits5a.py. The program became able to export lists of points on

the boundaries of manifolds; however, it had limitations. Users had to visualize points that the program provided in an external graphing program such as the Desmos online graphing calculator. They also had to manually edit the source code to derive different types of manifolds.

integratororbits6.py and integratororbits7.py reflected attempts to rewrite the program using different paradigms. integratororbits6.py used an object-oriented approach, whereas integratororbits7.py used a blackboard paradigm. I abandoned both attempts to rewrite the program after noticing that the functional code was substantially faster than both the object-oriented and blackboard programs.

integratororbits5b.py, which I created after integratororbits6.py and integratororbits7.py, introduced minor improvements into the manifold integration process. It also improved the procedure by which test functions passed to the Core Integrator could modify the integration state. It was a modified version of the integratororbits5.py code.

toboldlygo1.py was my first attempt to make a program capable of following the entire trajectory derivation process. I based it on integratororbits5b.py; however, I introduced functions to analyze itineraries themselves. It used a manual command line system for finding overlapping manifolds in which users specified the coordinates of points within the overlap between the two tube shapes. It also reworked the command line syntax: most arguments became optional.

toboldlygo2.py introduced an automatic manifold intersection system incorporating the Shapely Python module and an interactive trajectory visualization generator. toboldlygo2.py also introduced changes to the Core Integrator and added code allowing To

Boldly Go to integrate additional tube-Poincaré section “hits” if shapes formed from initial “hits” failed to intersect.

toboldlygo3.py, the latest version of the program, introduced several changes to make trajectory derivation more user friendly and accurate. Visualizations, such as animations of derived trajectories, used interactive windows supporting panning and zooming. The Shapely automatic overlap finder proved rarely accurate, so I replaced it with a manual system in which users could click within a plot to specify the boundary of the overlap region. I also introduced an interactive command-line prompt that allowed the user to alter certain parameters during program execution and utility Python functions that helped the user ascertain the energy and mass parameter needed for a trajectory. I also made several other changes to the program. The version of To Boldly Go on GitHub, simply called toboldlygo.py, is currently identical to toboldlygo3.py.

RESULTS

Program Structure

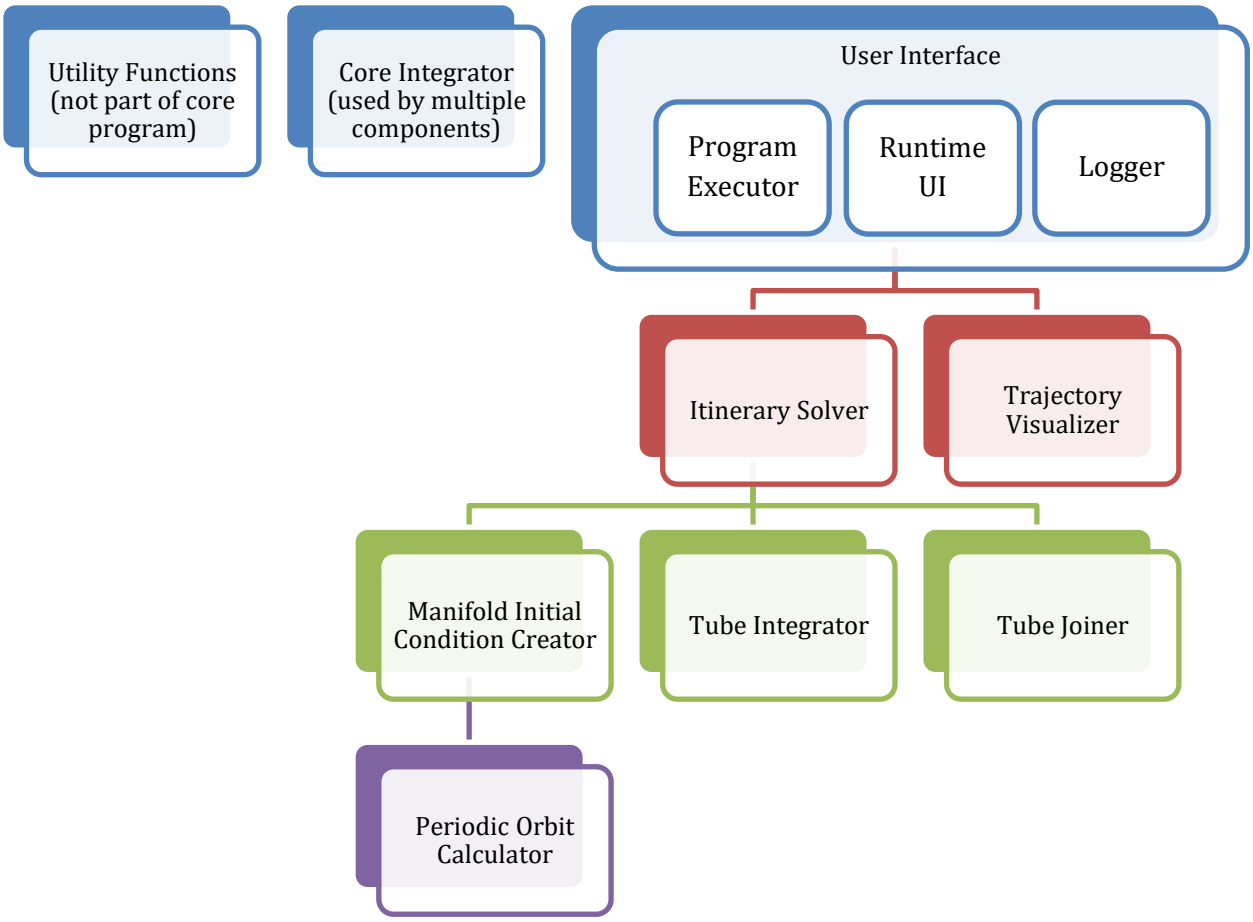

Fig 7. A high-level visualization of To Boldly Go's structure.

To Boldly Go defines dozens of functions. These functions, as well as the rest of the code, divide into roughly nine components: The Utility Functions, the Core Integrator, the User Interface, the Itinerary Solver, the Trajectory Visualizer, the Manifold Initial Condition Creator, the Tube Integrator, the Tube Joiner, the Periodic Orbit Calculator. These components are not entities in the code; they are high-level divisions drawable between parts of the code. Many components use other components.

The main program does not use the two Utility Functions; the user calls them at a Python prompt or in another program to determine the parameters he or she will use when starting To Boldly Go. The mufinder function calculates the value of the mass parameter given the two masses in the PCR3BP, and the lowenergyinterval function calculates the (E_2, E_3) energy interval for a mass parameter.

The Core Integrator is a set of functions that provides a useful interface to the REBOUND integration package. It simplifies the process of trajectory and state transition matrix integration. The Core Integrator sets up a three-body problem and integrates and stores positions while converting between rotating and inertial frames. It can simultaneously integrate a state transition matrix, but this functionality is optional. It provides a customizable, straightforward framework for ending, dynamically modifying, or returning values from the integration: other components can pass testing functions to the integratecore function, which will call the testing functions and handle their output. Several functions are part of the Core Integrator. The integratecore function contains the component's complete functionality.

The User Interface is code scattered throughout the program that handles input and output. One can divide its functionality into roughly three categories or subcomponents: The Program Executor, the Runtime UI, and the Logger. The Program Executor subcomponent allows users to supply relevant mass parameters, energy values, itineraries, and several optional parameters to a Linux command that also starts the program. The Program Executor calls the finditinvisual function, which combines the Itinerary Solver and Trajectory Visualizer components together into a coherent workflow. The Program Executor is not a function and is not usable when programs import To Boldly Go as a Python module. The

Runtime UI provides prompts and interactive plots throughout the program for selecting parameters and overlap regions and for viewing trajectories. The Logger is code scattered throughout the program that logs information about the status of the program to the console.

The Itinerary Solver "grows" a trajectory corresponding to an itinerary by iteratively deriving and intersecting trajectory components with an existing trajectory. First, it uses the itineraryisvalid function to ensure that the user-provided itinerary is valid. It repeatedly calls the Manifold Initial Condition Creator, the Tube Integrator, and the Tube Joiner, utilizing output from these components until the resulting initial condition matches the desired itinerary. It also repeatedly selects Poincaré surfaces for patching together tubes with the findsection function. The finditinerary function contains the component's complete functionality.

The Trajectory Visualizer allows users to visualize trajectories that To Boldly Go found. Its Interactive Visualizer appears after To Boldly Go finds an initial condition corresponding to a requested trajectory; it gives users a prompt at which they can repeatedly request trajectory visualizations with provided start times and end times. It then produces an interactive plot which shows the motion of the particle along the derived trajectory. The trajectoryviewer function provides most of the Trajectory Visualizer's functionality. The Trajectory Visualizer also uses the Core Integrator.

The Manifold Initial Condition Creator finds initial conditions for stable and unstable manifolds that the Tube Integrator and Tube Joiner will later utilize. After determining the Lagrange point for and the correct branch of the manifold it will be deriving, the component uses the Periodic Orbit Calculator to find the relevant periodic orbit. It then calculates the monodromy matrix for the periodic orbit and its eigenvectors with the monodromy and

monodromyeigen functions. The monodromy function uses the Core Integrator. The Manifold Initial Condition Creator yields a region of manifold initial conditions perturbed from the initial condition of the periodic orbit along the eigenvectors of the monodromy matrix. The manifoldinitial function provides most of the Manifold Initial Condition Creator's functionality.

The Tube Integrator uses the Core Integrator to integrate manifold and tube initial condition regions. It selects each point in an initial condition region and integrates it until the particle intersects the relevant Poincaré surface the correct number of times or until too much time has elapsed. It outputs a region representing the intersection of the tube or manifold it has integrated with the Poincaré surface. The integrateregion function provides most of the Tube Integrator's functionality.

The Tube Joiner finds the intersection of two manifolds or tubes using the Shapely computational geometry package. The Tube Joiner reduces sets of four-dimensional points to two dimensions using the createpolygon function. It produces an interactive window that plots the boundaries of the manifolds with the getoverlap function. Users can middle-click in the window to specify points on the boundary of the overlap region. If the window is showing the last manifold overlap for the entire trajectory, the first point selected on the boundary of the overlap region becomes the "representative point," the initial condition for the trajectory that the trajectory visualizer will show.

The Periodic Orbit Calculator derives periodic orbit initial conditions for requested energies and Lagrange points. It first uses differential correction to derive initial conditions for two "seed orbits" with known amplitudes. The differential correction technique uses a state transition matrix integrated with the trajectory to progressively refine a starting initial

condition. It applies numerical continuation to the seed orbits until it finds a periodic orbit with a satisfactory energy. The Periodic Orbit Calculator uses the Core Integrator and functions such as findparams and pocorrector. The energypo function contains most of the Periodic Orbit Calculator's functionality.

Program Output Example

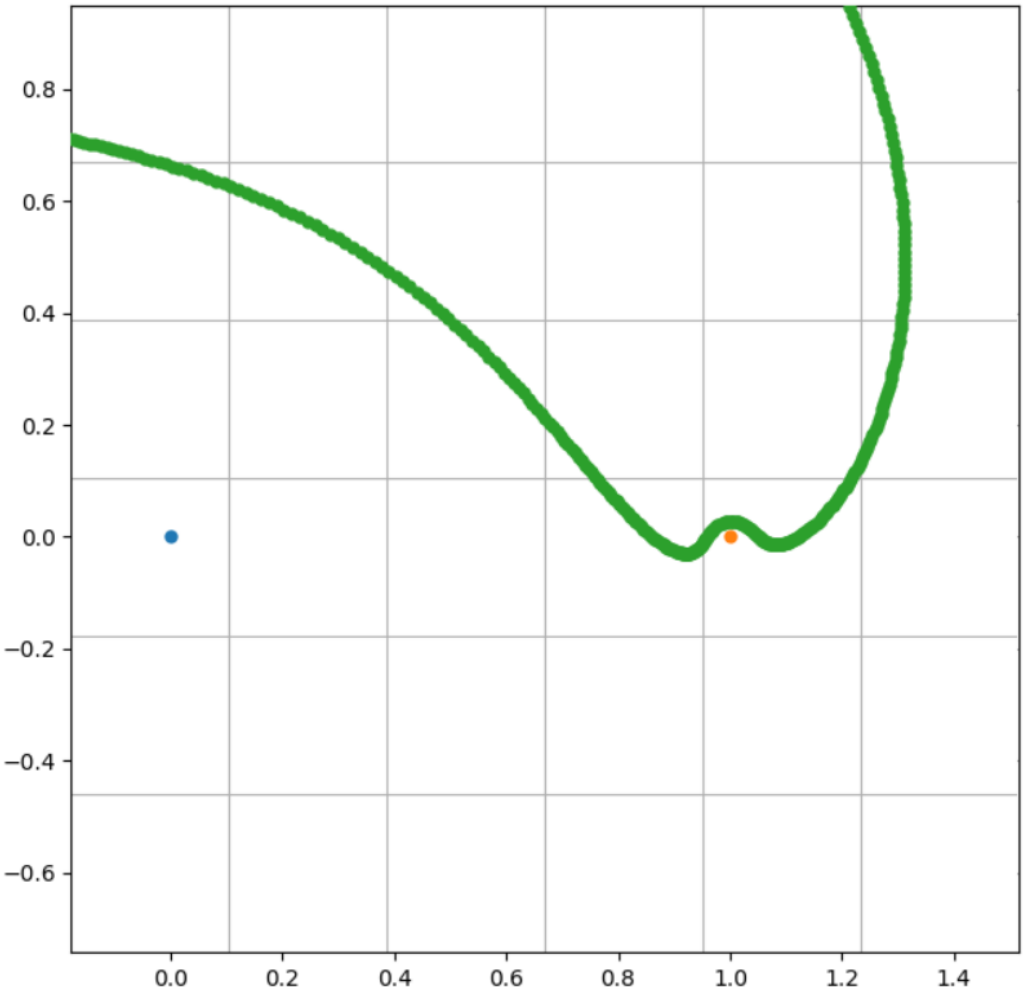

Fig. 6. A trajectory for the Sun-Jupiter PCR3BP with energy -1.515 and mass parameter 9.537e-4.

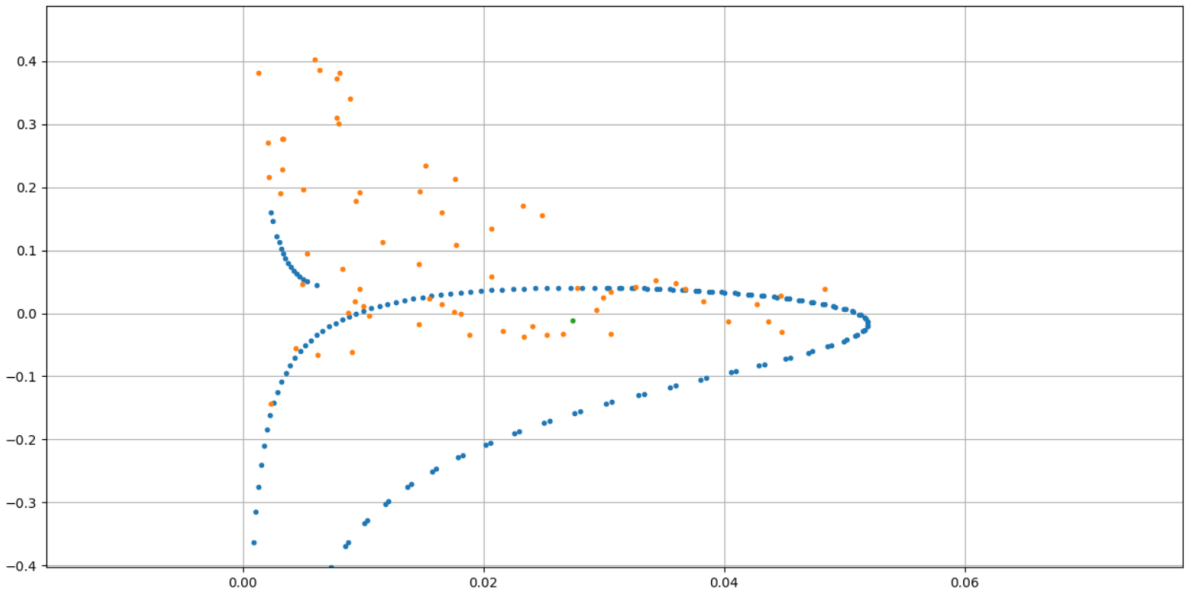

Fig. 7. The only manifold overlap region used to derive the trajectory in Fig. 6.

Using the command sudo python toboldlygo2.py -1.515 9.537e-4 x21, I requested a trajectory that matched the itinerary $(X, 2, 1)$ with energy -1.515 and mass parameter 9.537e-4. The mass parameter corresponds to the Sun-Jupiter PCR3BP (Koon et al. 107-118).

I derived a trajectory with the initial condition $(0.9990463, 0.027430483173323805, -0.19113618234711469, -0.011949048475592694)$. Fig. 6 represents the motion of the particle along the trajectory in a rotating frame of reference, and Fig. 7 represents the plot of the only manifold overlap region used to derive the trajectory. The orange dot is Jupiter whereas the blue dot is the Sun. The green dot is the initial condition that I selected for the trajectory.

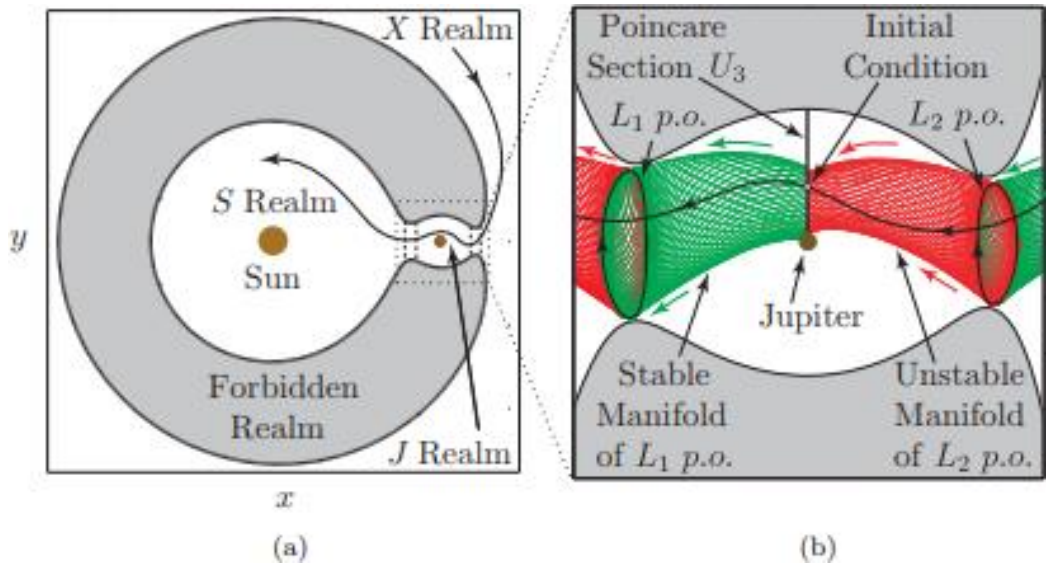

Fig. 8. Another visualization of a Sun-Jupiter PCR3PP trajectory with itinerary $(X, 2, 1)$ from: Koon, Wang Sang, et al., *Dynamical Systems, the Three-Body Problem and Space Mission Design*, Koon et al., 2011, p. 108, accessed 5 Dec. 2018.

Koon et al. derive a trajectory with the same itinerary and parameters; they provide a visualization of the trajectory, as well as of other crucial elements of the Sun-Jupiter PCR3BP, in Fig. 8 (Koon et al. 107-118). The similarities between Fig. 6 and Fig. 8 suggest To Boldly Go's trajectory is qualitatively correct.

Limitations

This thesis research has a few limitations. First, To Boldly Go uses as its model the Planar Circular Restricted Three-Body Problem, so it will incorrectly model scenarios that do not fit the PCR3BP. For example, To Boldly Go is unable to model properly situations in which three or more masses greatly influence the particle or in which the masses do not move in circular orbits. It is also incapable of finding spatial trajectories—trajectories with three position coordinates and three velocity coordinates—or trajectories in cases where the particle has nonnegligible mass.

Second, To Boldly Go does not currently work properly for every set of command-line parameters. It sometimes yields incorrect results for certain energy, mass parameter, and itinerary combinations. If I had more time to complete To Boldly Go, I might be able to fix the some of the glitches preventing the program from working or at least to handle the cases that fail.

Third, because To Boldly Go is a Python program, it may be slower than a framework written in another language, like C. Although To Boldly Go uses REBOUND, which primarily utilizes C, the program itself uses Python. If I had written To Boldly Go in C or some other language, I would have taken much longer to complete the program; however, the code might have been much faster when executed.

Future Research

This thesis presents opportunities for future research; many of these opportunities involve extending or modifying To Boldly Go. First, future research could attempt to extend To Boldly Go to a more general model, such as the spatial Circular Restricted Three-Body Problem or even an N-body problem for $N > 3$. Changing the model underlying To Boldly Go would substantially improve its applicability to real gravitational systems.

Second, future research could attempt to rewrite the Core Integrator and other crucial components in C to improve program execution speed. To Boldly Go could still use REBOUND, as REBOUND is still usable without a Python frontend. Some of the framework could remain in Python; rewriting the Core Integrator alone might dramatically improve performance.

Third, future research could attempt to identify, to assess, and to fix situations in which To Boldly Go fails to deliver usable trajectories. Some solutions might only involve

fixing bugs that prevent the framework from deriving trajectories properly. Viable trajectories may not exist for some command line parameters; implementing more checks and tests in the code might help identify these situations.

WORKS CITED

"About TRAPPIST-1." *TRAPPIST-1*, www.trappist.one/#about. Accessed 18 Oct. 2017.

"BBC Bitesize - GCSE Computer Science - Functions, Procedures and Modules - Revision 1." *BBC Bitesize*, www.bbc.com/bitesize/guides/z9hykqt/revision/1. Accessed 12 Nov. 2018.

Belbruno, Edward. *Fly Me to the Moon: An Insider's Guide to the New Science of Space Travel.* Princeton University Press, 2013.

Belbruno, Edward, et al. "Geometry of Weak Stability Boundaries." *ArXiv:1204.1502 [Math, Nlin]*, Apr. 2012. *arXiv.org*, arxiv.org/abs/1204.1502. Accessed 23 Nov. 2018.

Bottke, Bill. "Re: Numerical Integration Techniques." Received by Joshua Fitzgerald, 13 Sept. 2017.

IBM Corporation. "A Brief Linux Glossary for Windows Users," *Tech Insider*, 2000, tech-insider.org/linux/research/acrobat/010222-b.pdf. Accessed 28 Nov. 2018.

"Chang'e 2." *NASA Space Science Data Coordinated Archive*, nssdc.gsfc.nasa.gov/nmc/masterCatalog.do?sc=2010-050A. Accessed 22 Oct. 2017.

Chenciner, Alain, and Richard Montgomery. "A Remarkable Periodic Solution of the Three-Body Problem in the Case of Equal Masses." *Annals of Mathematics. Second Series*, vol. 152, no. 3, 2000, pp. 881–901. *EMIS*, www.emis.ams.org/journals/Annals/152_3/chencine.pdf. Accessed 21 Oct. 2017.

"Comparing Python to Other Languages." *Python Software Foundation*, 1997, www.python.org/doc/essays/comparisons/. Accessed 22 Nov. 2018.

Conley, C. "Low Energy Transit Orbits in the Restricted Three-Body Problems." *SIAM Journal on Applied Mathematics*, vol. 16, no. 4, July 1968, pp. 732–746. *ProQuest*, doi:10.1137/0116060. Accessed 22 Oct. 2017.

Corkill, Daniel. "Blackboard Systems." *The GBBopen Project*, gbbopen.org/papers/ai-expert.pdf. Accessed 3 Dec. 2018.

"Debian is the rock on which Ubuntu is built." *Ubuntu*, Canonical www.ubuntu.com/community/debian. Accessed 28 Nov. 2018.

Diacu, Florin. "The Solution of the n-body Problem." *The Mathematical Intelligencer*, vol. 18, no. 3, June 1996, pp. 66–70. *Springer Link*, doi:10.1007/BF03024313. Accessed 21 Oct. 2017.

Farquhar, Robert W., et al. “Trajectories and Orbital Maneuvers for the First Libration-Point Satellite.” *Journal of Guidance, Control, and Dynamics*, vol. 3, no. 6, 1980, pp. 549–54. *American Institute of Aeronautics and Astronautics*, doi:10.2514/3.56034. Accessed 22 Oct. 2017.

Frank, Juhan. “The Three-Body Problem.” *Louisiana State University*, 11 Oct. 2006, www.phys.lsu.edu/faculty/gonzalez/Teaching/Phys7221/ThreeBodyProblem.pdf. Accessed 21 Oct. 2017.

“General Python FAQ — Python 3.7.1 Documentation.” *Python*, Python Software Foundation, docs.python.org/3/faq/general.html. Accessed 12 Nov. 2018.

“Genesis : Search for Origins | Mission : History - Launch | JPL | NASA.” *NASA*, genesismission.jpl.nasa.gov/gm2/mission/launch.htm. Accessed 22 Oct. 2017.

“GEOS.” *GEOS*, 2018, trac.osgeo.org/geos/. Accessed 12 Nov. 2018.

Gillies, Sean, et al. “Shapely — Shapely 1.6 Documentation.” 2013, shapely.readthedocs.io/en/latest/project.html. Accessed 12 Nov. 2018.

Gillon, Michaël, et al. “Seven Temperate Terrestrial Planets around the Nearby Ultracool Dwarf Star TRAPPIST-1.” *Nature*, vol. 542, no. 7642, Feb. 2017, pp. 456–460. *arXiv*, doi:10.1038/nature21360. Accessed 29 Sept. 2017.

Graps, Amara. “Stanford SOLAR Center -- Ask A Solar Physicist FAQs - Answer.” *Stanford SOLAR Center*, solar-center.stanford.edu/FAQ/QL1.html. Accessed 20 Oct. 2017.

Holton, Gerald James, and Stephen G. Brush. *Physics, the Human Adventure: From Copernicus to Einstein and Beyond.* Rutgers University Press, 2001.

“ISEE-3/ICE | Missions.” *NASA*, solarsystem.nasa.gov/missions/iseeice/indepth. Accessed 22 Oct. 2017.

Koon, Wang Sang, et al. *Dynamical Systems, the Three-Body Problem and Space Mission Design.* Koon et al., 2011.

Kline, Morris. *Mathematical Thought from Ancient to Modern Times*, Vol. 2, Oxford University Press, 1972.

“The Lagrange Points.” *NASA*, July 2012, map.gsfc.nasa.gov/mission/observatory_l2.html. Accessed 20 Oct. 2017.

Lo, M.W. “The InterPlanetary Superhighway and the Origins Program.” *Proceedings of the IEEE Aerospace Conference, Big Sky, MT, 9-16 March 2002*, IEEE Aerospace and Electronic Systems Society, 2002. *NASA*, trs.jpl.nasa.gov/bitstream/handle/2014/8065/02-0006.pdf. Accessed 1 Sept. 2017.

Lo, Martin, et al. “Genesis Mission Design.” *AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Boston, MA, 10-12 Aug. 1998*, American Institute of Aeronautics and Astronautics, 1998. *American Institute of Aeronautics and Astronautics*, doi:10.2514/6.1998-4468. Accessed 1 Oct. 2017.

Lo, Martin, and Shane Ross. “The Lunar L1 Gateway: Portal to the Stars and Beyond.” *AIAA Space 2001 Conference and Exposition, Albuquerque, NM, 28-30 Aug. 2001*, American Institute of Aeronautics and Astronautics, Aug. 2001. *The NASA Jet Propulsion Laboratory*, trs.jpl.nasa.gov/handle/2014/40516. Accessed 22 Oct. 2017.

Meyer, Kenneth R. *Periodic Solutions of the N-Body Problem*. Springer, 2006.

“Modules — Python 3.7.1 Documentation.” *Python Software Foundation*, 2018, docs.python.org/3/tutorial/modules.html. Accessed 12 Nov. 2018.

“PyPI – the Python Package Index.” *PyPI*, Python Software Foundation, pypi.org. Accessed 22 Nov. 2018.

Rein, Hanno, et al. “Available Modules — REBOUND 3.6.8 Documentation.” 2015, rebound.readthedocs.io/en/latest/. Accessed 12 Nov. 2018.

---. “Welcome to REBOUND! — REBOUND 3.5.10 Documentation.” 2015, rebound.readthedocs.io/en/latest/. Accessed 7 Nov. 2017.

Riebeek, Holli. “Planetary Motion: The History of an Idea That Launched the Scientific Revolution : Feature Articles.” *NASA*, 7 July 2009, earthobservatory.nasa.gov/Features/OrbitsHistory/page2.php. Accessed 19 Oct. 2017.

Ross, Shane. “Statistical theory of interior-exterior transition and collision probabilities for minor bodies in the solar system.” *Proceedings of the International Conference on Libration Point Orbits and Applications, Parador d’Aiguablava, Spain, 10–14 June 2002*, Institut d'Estudis Espacials de Catalunya, 2002. *Virginia Tech*, www2.esm.vt.edu/~sdross/papers/ross-barcelona-2002.pdf. Accessed 2 Nov. 2017.

“Rotating Frame.” *ELSTER*, 2017, mriquestions.com/rotating-frame.html. Accessed 19 Oct. 2017.

Ryden, Barbara. *Dynamics*. The Ohio State University, 2016.

Smith, David. “CRAN now has 10,000 R packages. Here's how to find the ones you need.” *Revolutions*, 27 Jan. 2017, blog.revolutionanalytics.com/2017/01/cran-10000.html. Accessed 22 Nov. 2018.

Smith, Douglas L. “Next Exit 0.5 Million Kilometers.” *Engineering and Science*, vol. 65, no. 4, 2002, pp. 6–15. *California Institute of Technology*, calteches.library.caltech.edu/4074/1/Exit.pdf. Accessed 23 Oct. 2017.

Snyder, David. “Gravity, Part 2: Newton, Hooke, Halley and the Three Body Problem.” *University Lowbrow Astronomers*, University of Michigan, Apr. 2006, umich.edu/~lowbrows/reflections/2006/dsnyder.17.html. Accessed 21 Oct. 2017.

Toal, Ray. “Programming Paradigms.” *Loyola Marymount University*, cs.lmu.edu/~ray/notes/paradigms/. Accessed 12 Nov. 2018.

Topputo, Francesco, et al. “Low Energy Interplanetary Transfers Exploiting Invariant Manifolds of the Restricted Three-Body Problem.” *Journal of the Aeronautical Sciences*, vol. 53, no.4, pp. 353-372, 2006. *ResearchGate*, www.researchgate.net/profile/Massimiliano_Vasile2/publication/40705665_Low_Energy_Interplanetary_Transfers_Exploiting_Invariant_Manifolds_of_the_Restricted_Three-Body_Problem/links/0c960527a98426f09e000000.pdf. Accessed 30 Sept. 2017.

Williams, David. “Comet Shoemaker-Levy 9 (NSSDCA).” *NASA*, 4 Feb. 2005, nssdc.gsfc.nasa.gov/planetary/comet.html. Accessed 2 Nov. 2017.

APPENDIX A: TO BOLDLY GO SOURCE CODE

The most recent version of To Boldly Go and the final version of this thesis are available at https://github.com/knightofkessler/To-Boldly-Go.