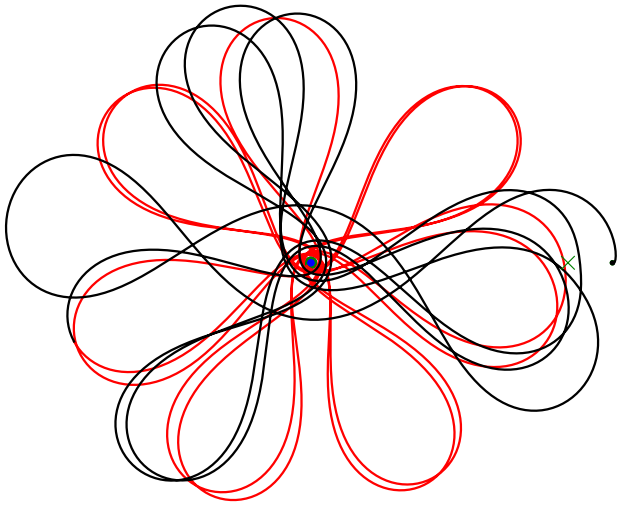
B.Sc.Eng. Thesis
Bachelor of Science in Engineering



DTU Compute
Department of Applied Mathematics and Computer Science

Low-Energy Transfer Orbits

a Theoretical and Numerical Study

Gandalf Saxe (s113093)

Kongens Lyngby 2017

DTU

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

When navigating a spacecraft between planets, moons and orbits, low-energy transfer orbits allows us to make the trade-off of a longer flight time in return for lower propellant energy requirements, suitable for unmanned space missions. Based on the restricted 3-body model and using second-order adaptive symplectic integrators inplemented parallelized in Python, we brute-force search in many directions- and velocities from low Earth orbit for transfer orbits to the Moon. The lowest-energy orbit we found to the moon has $\Delta v = 3795\,\mathrm{km/s}$ and takes 194 days, which is even better than typical values in the literrature. Our model and implementation is validated by simulating the Apollo mission's 3-day Hohmann transfer-orbit to the Moon, found to agree within 2.5%.

Preface

This bachelor's project was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a bachelor's degree in Physics and Nanotechnology.

Kongens Lyngby, April 17, 2017

Gandalf Saxe (s113093)

Acknowledgements

People I would like to thank:

- My advisors Poul G. Hjorth and Hans Henrik Brandenborg Sørensen for excellent guidance throughout the project. This project would not have been possible without either of them.
- Laursen for the XeLaTeX thesis project template for DTU Compute [Lau].
- Kjell Magne Fauske for PGF/TikZ example of how to color equations [Fau].
- Good friend Tom Parton for feedback and suggestions.

Software acknowledgements:

1. Python - for implementing numerical algorithms to solve equations of motion.
2. Numbapro - a commerical, free-for-academics Python compiler that sped up my simulations 10-100 times.
3. Inkscape - for drawing figures in vector graphics.
4. Mathematica - for miscellaneous calculations.

Contents

CHAPTER 1

Introduction

"The Moon is the first milestone on the road to the stars."

— Arthur C. Clarke

During the height of the cold war in December 1968 the NASA Apollo 8 spacecraft made history in being the first manned spacecraft to leave low Earth orbit (LEO) and also the first to orbit the moon. Less than a year later on July 20th 1969 the Apollo 11 spacecraft orbited and landed people on the moon, and was followed up by five more successful moon landing missions the next 3.5 years, Apollo 12, 14, 15-17) [Wikc]. It was a remarkable feat of science and engineering, but manned missions into deep space (beyond LEO) have not been attempted since. Former US President G.W. Bush laid out a plan for NASA in 2004 to conduct manned missions to the moon within 16 years; however this was subsequently cancelled by current US President Barack Obama in 2010 [CNN][BBC]. Nevertheless, NASA recently tested its Orion spacecraft, designed to carry a crew of up to 4 people to asteroids or Mars, and the Space Launch System (SLS), the first rocket boosters designed for human exploration in deep space since the powerful Saturn V rockets of the 1960-70's [Ram15].

With the rise of private enterprise in the space industry, not least SpaceX, the future prospects of space exploration and colonization is promising. SpaceX was "founded with the goal of reducing space transportation costs and enabling the colonization of Mars" [Wike]. SpaceX is even more ambitious about the scale of Mars colonization than any national- or intergovernmental organisations. SpaceX CEO Elon Musk wants to put 1 million people on Mars within a century. This will only be possible by making space travel two orders of magnitudes cheaper, driving down the price to tens of dollars per pound of weight. Elon Musk's opinion: the key to achieving this is reusable rockets. Musk in an interview: "'Excluding organic growth, if you could take 100 people at a time, you would need 10,000 trips to get to a million people,'. 'But you would also need a lot of cargo to support those people. In fact, your cargo to person ratio is going to be quite high. It would probably be 10 cargo trips for every human trip, so more like 100,000 trips. And we're talking 100,000 trips of a giant spaceship."[And].

Today it costs anywhere between $10,000-$20,000 per kilogram to launch the payload into LEO [Coo], although SpaceX is working these years on lowering the cost down to just below $1000 per kilogram. However going into deep space, requires much more energy, and therefore much more fuel and money. The transportation cost for going to Moon orbit is on the order of $100,000 per kilogram and to land on the moon is on the order of $1,000,000 per kilogram [Ast][Oka15].

Whenever the destination is beyond LEO, any spacecraft is usually placed in a so-called parking orbit, which is usually a roughly circular orbit around the earth. From this position the propulsion system is fired to initiate a *transfer-orbit* towards the desired location (and often several more thrusts are required along the way to correct or change orbits). Finding the optimal transfer-orbit is therefore of great interest. In general, transfer orbits will always be a tradeoff between the time travelled and the fuel expended. For unmanned missions and supply shipments for a spacestation, base or colony, we want to use low-fuel routes at expense of flight time because they are cheaper. For the remainder of this project we will focus on low-energy transfer orbits to the moon, however all the principles and qualitative conclusions will apply equally well in general and thus for going to Mars [TB14].

We have introduced what transfer orbits are and why they are interesting. Let's take a look at the roadmap:

Chapter 2 transfer orbits Introduction to the term delta-v and look at the two most basic types of transfer orbits: Hohmann and bi-elliptic. We will show that they are practically identical in delta-v and flight time for realistic Earth-Moon transfer orbits. We will then model a Hohmann transfer-orbit to the moon and derive delta-v prediction for it. Finally we will discuss Lagrange points and their connection to the concept of low-energy transfer orbits.

Chapter 3 Analytical Mechanics A quick overview of the three major formulations of classical mechanics: Newtonian, Lagrangian and Hamiltonian. We will see how Hamiltonian mechanics is the natural choice for orbital mechanics, how to use it in practice and finally derive the equations of motion for our model problem: the restricted 3-body problem.

Chapter 4 Numerical Analysis Going through several numerical techniques for solving non-linear equations of motion and what the various limitations of these methods are. We will see the first-order explicit Euler and symplectic Euler and the second-order symplectic Störmer–Verlet method. An adaptive version of the Verlet method that takes a variable step-size will be explained, implemented and compared to the non-adaptive Verlet.

Chapter 5 Simulations of Transfer Orbits Here we will simulate transfer orbits from the Earth to the moon. First we will attempt to find a Hohmann transfer-orbit and compare the results with the results from Chapter 2 and to the Apollo mission. Finally we will attempt to find low-energy transfer orbits that has a lower delta-v than Hohmann by brute force, i.e. shooting in many different directions from a circular Earth orbit with many different angles and velocities.

All basic physical constants and characteristics such as the gravitational constant G, and planetary properties are extracted from Wolfram Mathematica [Wol] and listed in appendix F.

CHAPTER 2

Transfer Orbits

"It's a strange, eerie sensation to fly a lunar landing trajectory—not difficult, but somewhat complex and unforgiving."

— Neil Armstrong

Transfer orbits is any trajectory of a spacecraft between two different closed orbits, either around the same celestial body or two different bodies. In general, transfer orbits will always be a trade-off between flight time (the total time required for the transfer orbit) and fuel consumption for a specific craft - or equivalently, a trade-off between flight time vs. Δv for any craft.

In spacecraft flight dynamics, the change in velocity[1] (Δv or delta-v) is fundamental property of a transfer orbit. The energy- and fuel consumption on the other hand, does depend on the spacecraft. Energy consumption is a practical concern, but not a key part of the underlying physical problem. This is due to the fact that a spacecraft of any mass need the same change in velocity to perform identical orbital maneuvers in a given system of celestial bodies. The basic reason for this is the cancellation of the gravitational mass in Newton's law of universal gravitation and the inertial mass Newton's 2nd law of motion [KH02], which are the same by the equivalence principles. Hence

$$\mathbf{F} = G\frac{Mm}{r^2} = m\mathbf{a} \tag{2.1}$$

$$\Leftrightarrow \mathbf{a} = \frac{GM}{r^2}\hat{\mathbf{r}}, \tag{2.2}$$

where G is the gravitational constant, M is the larger mass, m is the smaller mass, r is the separation distance between the two masses, $\mathbf{F}$ is the gravitational force vector from M on m pointing to M along the line connecting their center of masses with unit vector $\hat{\mathbf{r}}$, and $\mathbf{a}$ is the acceleration vector of m. Thus for any spacecraft of mass m, the equation of motion (and hence its trajectory) will be the same independent of the mass m, given same initial conditions and that $m \ll M$ so m does not affect the motion of M. We can easily generalize to an arbitrary number of large masses $(M, m \dots)$. This is all assuming point masses and that the small mass m is sufficiently small as to not affecting the motion of the big masses $(M, m \dots)$ significantly. This is the case for spacecrafts in

[1] We're really talking about speed here. Velocity and speed are often used interchangeably in orbital mechanics, but fortunately this rarely leads to confusion in practice.

the vicinity of the Earth, whose motion is dominated by the gravitational fields of the Sun, Earth and Moon.

So in planning- and optimizing orbital trajectories, Δv is the quantity of interest, a quantity that we may want to optimize along with flight time. Conversely, knowing the mass and fuel capacity of a spacecraft beforehand, determines the so-called Δv budget that the spacecraft have at its disposal for various maneuvers.
Before we delve into the different kinds of transfer orbits, we need to introduce some terminology. We imagine a large mass M placed at focus point F and a body of small mass m orbiting it, figure 2.1 [MD99]. Kepler's first law states that in a 2-body system with a large mass M and a small negligible mass m, the orbit of the small body is an ellipse with the large body at one of the foci [KH02]

F, F' = **Focus** Ellipses have two foci.

r = Position vector of smaller orbiting body

A = **Apoapsis** The point on the orbit of greatest distance to the central body[2].

P = **Periapsis** The point on the orbit of least distance to the central body[3].

r_A = **Distance from central body to apoapsis**

r_P = **Distance from central body to periapsis**

a = **Semi-major axis** Half the greatest diameter of the ellipse.

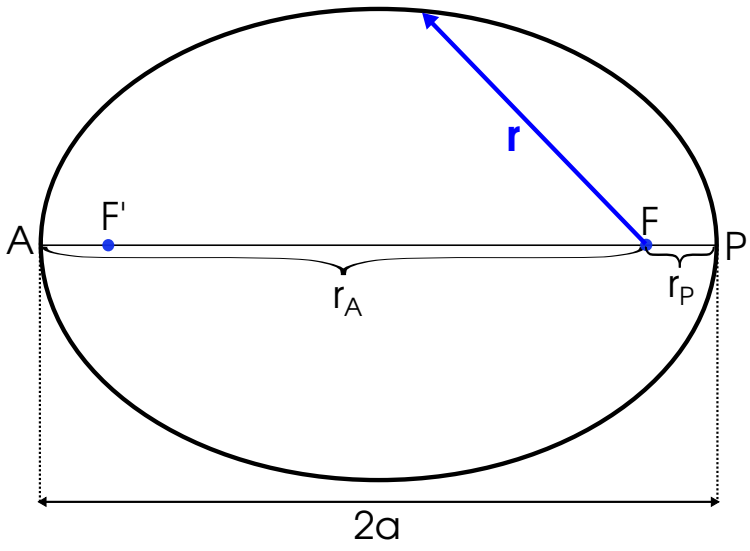


Figure 2.1: The elliptical orbit. See definitions just above figure. Source: [Use] (modified).

[2]Also known as the apocentre. If Sun/Earth is central body, also known as aphelion/apogee.
[3]Also known as the pericentre. If Sun/Earth is central body, also known as perihelion/perigee.

2.1 Hohmann Transfer-Orbit

The Hohmann transfer orbit is an elliptical orbit used to transfer between two circular orbits in the same plane. This is done by doing two engine impulses known as *burns* along the trajectory, see figure 2.2.

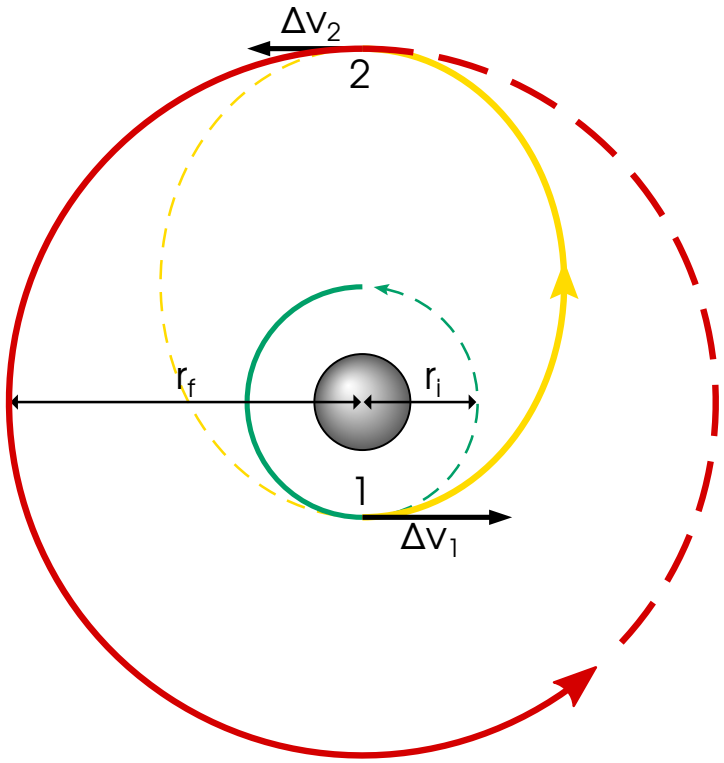


Figure 2.2: The Hohmann transfer orbit. We start in circular orbit around central body at distance r_i, 1.) Burn applied prograde (same direction as velocity vector) to change velocity by Δv_1, now in elliptical orbit around central body, 2.) Another burn is applied prograde at apoapsis to change velocity by Δv_2, now in circular orbit at distance r_f. Source: [Bar] (modified).

All the Apollo missions launched the spacecraft from Earth into a parking orbit around Earth [Whe][4], then a Hohmann-like transfer orbit to the moon, where they achieved circular lunar orbit 99 km above the moon surface. We will assume an initial parking orbit of 160 km above the earth's surface and a capture orbit 100 km above the moon surface.

It has been proven that the Hohmann transfer-orbit is the most Δv-efficient transfer-orbit from circle-to-circle (or ellipse-to-ellipse) in a 1-body system when r_f/r_i is less than ≈ 11.94 [Pru92] [Pee] [BJ00].

[4] An intermediate low circular orbit that a spacecraft is placed into temporarily after launch to ensure that all systems are nominal and calculate the precise transfer orbit to the destination

2.2 Bi-elliptic Transfer-Orbit

The bi-elliptic transfer is two elliptical orbits used to transfer between two circular orbits. Thrust is applied at three points instead of Hohmann's two, see figure 2.3. The basic idea is that the first prograde burn will bring the spacecraft close to escape velocity, then at the apoapsis another prograde burn will increase the periapsis distance (see fig 2.1) to the desired distance. Finally at periapsis a retrograde (opposite velocity vector) burn will make the orbit circular.

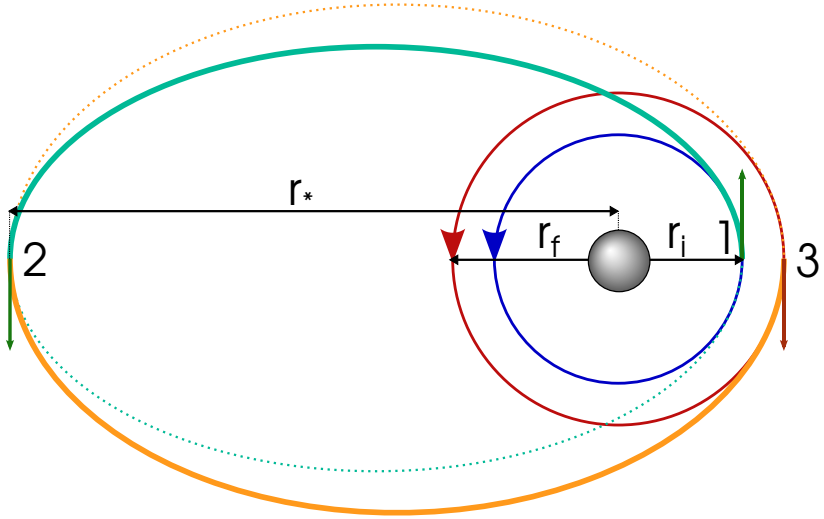


Figure 2.3: The bi-elliptic transfer-orbit. We start in a circular orbit around the central body at distance r_1, 1.) Big burn applied prograde to change velocity close to escape velocity, now in arbitrarily big elliptical orbit with r_* as apoapsis distance, 2.) Small burn applied prograde at apoapsis to increase periapsis distance from r_i to desired r_f, 3.) Small burn applied retrograde at periapsis to decrease apoapsis distance from r_* to r_f, now in circular orbit at distance r_f. Source: [Buc] (modified).

It has been proven that the bi-elliptic transfer-orbit is the most Δv-efficient transfer-orbit from circle-to-circle (or ellipse-to-ellipse) in a 1-body system when r_f/r_i is greater than ≈ 11.94 [Pru92] [Pee]. We will not consider the bi-elliptic to the Moon. The reason for this is outlined below.

In our case $r_i = 160\,\text{km}$ (Earth parking orbit) and $r_f = 3.85 \times 10^5\,\text{km}$ so

$$r_f/r_i = \frac{3.85 \times 10^5\,\text{km}}{6367\,\text{km} + 160\,\text{km}} \approx 59.0 \tag{2.3}$$

The closer the velocity is to escape velocity right after the first burn at point in fig 2.3, the greater the apoapsis distance r_* will be, going to infinity at escape velocity. As r_* increases, the flight time tends to infinity and the bi-elliptic orbit gets more efficient. Note that the Δv of the Hohmann- and the bi-elliptic transfers converges to the same

value in the limit as $r_f/r_i \to \infty$ see figure 2.4. Also note the local minimum of the ratio $\Delta v_{\text{Hohmann}}/\Delta v_{\text{bi-elliptic}}$ at $r_f/r_i = 57.19$, which is very close to the moon trip of 59.0 (caveat: r_* is set to ∞). This means that the bi-elliptic transfer should theoretically even be most beneficial compared to Hohmann at the moon distance. Even so, we will not consider the bi-elliptic to the Moon. The reason for this is outlined below. Figure

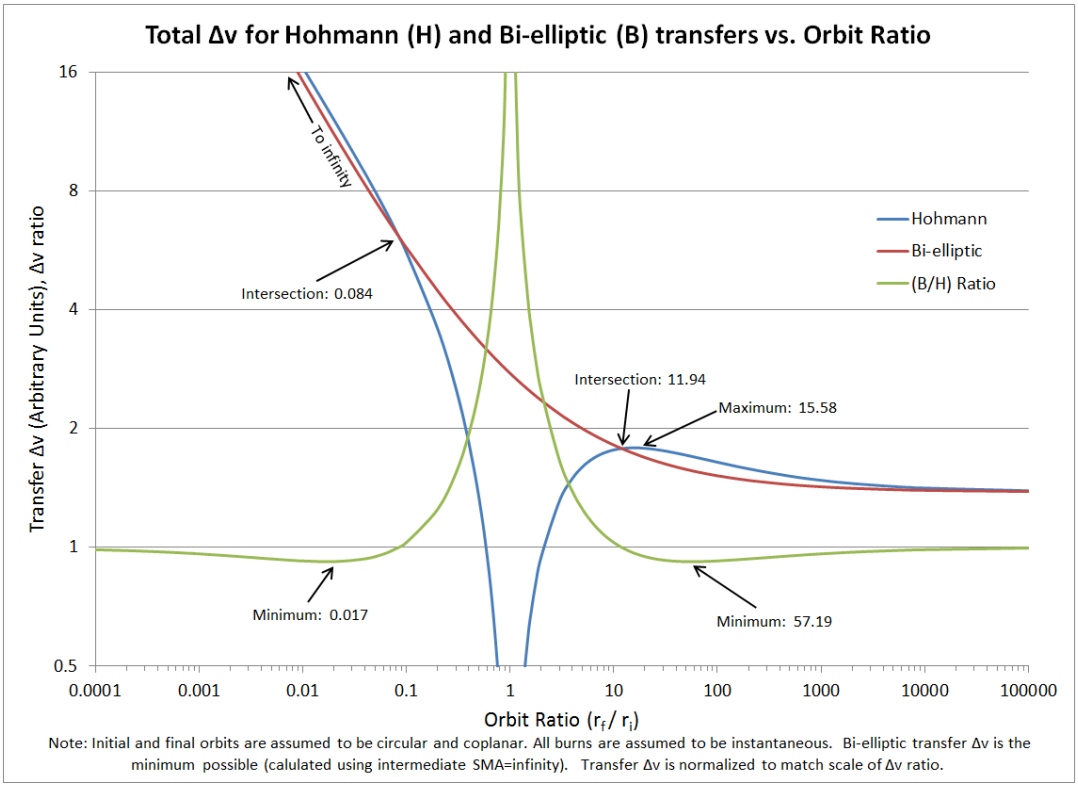


Figure 2.4: Δv as a function of the orbit ratio r_f/r_i for Hohmann and bi-elliptical transfer orbits. r_i is initial orbit distance, r_f is final orbit distance. They intersect at $r_f/r_i \approx 11.94$, where bi-elliptical becomes more efficient than Hohmann, but they converge as $r_f/r_i \to \infty$. This is best case for bi-elliptic since we have set $r_* = \infty$. Source:[Cop13].

.

2.5 shows effiencies for various intermediate orbital radii r_*. To take advantage of the bi-elliptic transfer, we must make r_* as large as possible. This is not achievable in practice due to both very long flight times and perturbations from other bodies. As an example, going from LEO at 6700 km above the earth to 1.3 times the distance to the moon, requires 99.0 % of the Hohmann Δv [Wika]. So it quickly becomes a big trade-off in flight time to save just a little Δv. While bi-elliptic transfers are considered more complicated, risky and not worth the flight time trade-off for bigger transfers, and therefore are rarely used for that [Tay09, p. 78], they are often used when performing small, $0 < r_f/r_i < 1$ maneuvers[5].

[5]Soyuz capsules heading for ISS [ESA14], launched from Earth into parking orbit use bi-elliptic

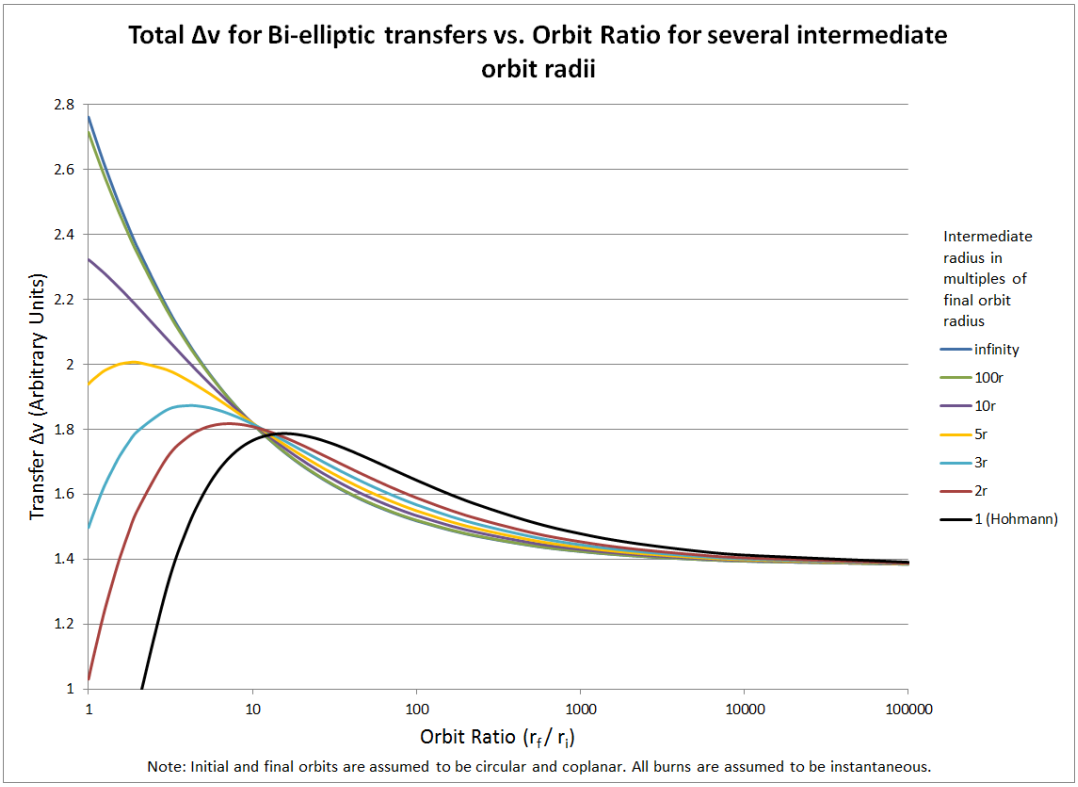


Figure 2.5: Δv for various intermediate orbit apoapsis radii $r*$ as function of the ratio r_f/r_i, where r_i is the initial orbit distance and r_f is the final orbit distance. Once gain we see the intersect at $r_f/r_i = 11.94$, where bi-elliptical becomes more efficient than Hohmann, the two of them converge as $r_f/r_i \to \infty$. Source: [Cop13].

2.3 Hohmann-Approximated Transfer-Orbit to the Moon

We will now apply the Hohmann transfer orbit, a model for a 1-body system, as an approximation to find a transfer orbit to the moon, with the Earth-Moon system being a 2-body system. We work on the assumptions:

1. **Sun is neglected.** The gravitational pull from the sun is neglected. It is not insignificant compared to the earth and the moon, but it's a first-order effect. As long as we do not get too far away from the Earth-Moon system, it should not be a problem.

2. **Moon orbit assumed circular and co-planar with the earth** We assume orbital

transfers to dock with ISS. The mid-burn gives opportunity to correct the trajectory. And safety is of great concern for a $100 billion space station [Min].

eccentricity $e = 0$ (perfect circle) and inclination angle (angle between orbital planes of the moon and Earth) $\phi = 0$. These assumptions are valid, since the true values are $e = 0.0549$ and $\phi = 5.16°$ [Wol]. This assumption reduces the problem from 3 to 2 dimensions, greatly simplifies all equations and does not affect the qualitative conclusions.

3. **Spacecraft does not affect Earth or Moon** The mass of the spacecraft itself is so small that it does not affect the motions of the earth and the moon. With this assumption the 3-body problem is called *restricted*, and can be solved analytically.

4. **Hohmann transfer can approximate a trans-lunar injection + lunar orbit insertion** A trans-lunar injection (TLI) is a propulsive maneuver used to set a spacecraft on a trajectory that will cause it to arrive at the moon [NAS66]. We assume that a Hohmann transfer, modified to end with a velocity that matches the centripetal velocity required for circular motion around the moon, is an adequate model. Hence we neglect the gravitational attraction from the moon on the spacecraft while it is in transit, up until the point when the spacecraft is in the circular orbit around the moon.

Our Hohmann transfer orbit to the moon follows the same steps as the standard Hohmann transfer described in chapter 2.1, except we end with a slightly different velocity to match a circular orbit centered around the moon, see figure 2.6. In fact we will see that we need to break (retrograde burn) instead of accelerating to achieve desired speed for orbit around the moon rather than around the former central body, the Earth.

First we need the *vis viva* equation [Pis05], which is basically energy conservation applied to elliptical orbits. It serves to relate the distance between two bodies and orbital speed in any Keplerian orbit (elliptic, parabolic, hyperbolic, or radial). We will be using it in the elliptical case:

$$v^2 = \mu \left(\frac{2}{r} - \frac{1}{a} \right), \tag{2.4}$$

v is the velocity of the smaller body relative to the central body, $\mu = GM$ is the standard gravitational parameter, M is the central body mass, r is distance from Earth (at focus) to spacecraft and a is the semi-major axis, see figure 2.6.

Using figure 2.6 as reference, we will calculate the total Δv to perform a Hohmann transfer orbit using figure 2.6 as reference, from a 160 km circular orbit around the earth

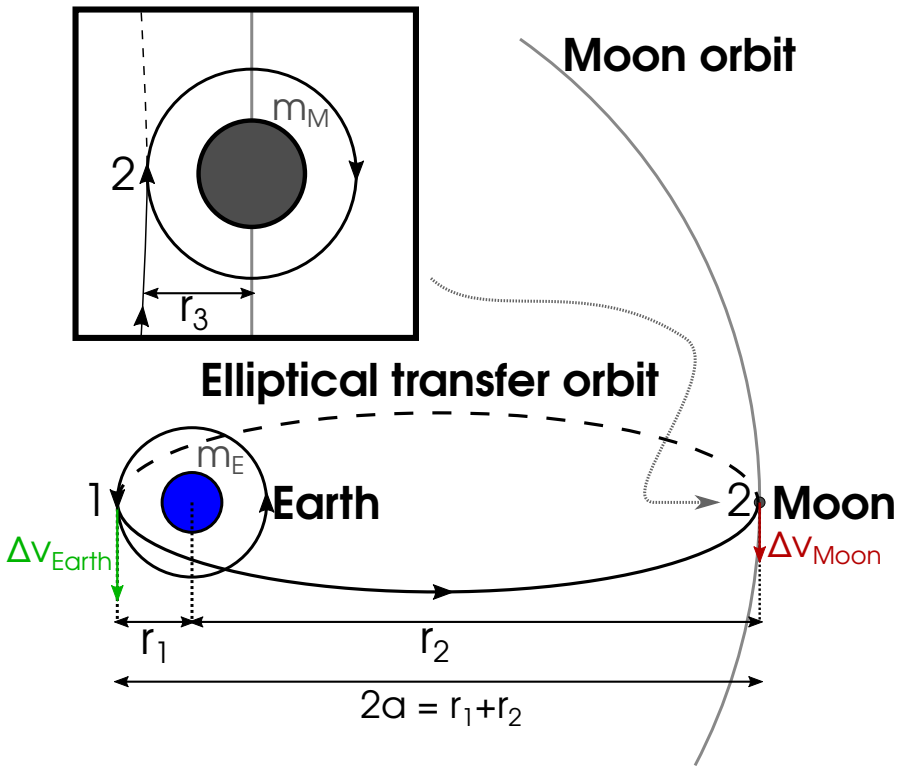


Figure 2.6: A direct transfer orbit to the moon approximated by Hohmann transfer-orbit. m_E is the mass of the earth, m_M is the mass of the moon. We start in circular orbit around the earth at distance r_1 with velocity v_{earth} 1.) Burn applied prograde to change velocity by Δv_{earth}, now in elliptical orbit around the earth with r_2 as apogee distance, 2.) Smaller burn applied retrograde at apogee to match a circular orbit around the moon at distance r_3 to Moon and velocity v_{moon} in Moon-system..

to a 100 km circular orbit around the moon. We have the quantities [Wol]

$$\mu_E = Gm_E = 6.67 \times 10^{-11} \tfrac{\text{m}^3}{\text{kg s}^2} \cdot 5.97 \times 10^{24}\,\text{kg} = 3.99 \times 10^5\,\text{km}^3/\text{s}^2 \tag{2.5}$$

$$\mu_M = Gm_M = 6.67 \times 10^{-11} \tfrac{\text{m}^3}{\text{kg s}^2} \cdot 7.35 \times 10^{22}\,\text{kg} = 4.90 \times 10^3\,\text{km}^3/\text{s}^2 \tag{2.6}$$

$$r_1 = \text{Earth radius} + 160\,\text{km} = 6367\,\text{km} + 160\,\text{km} = 6527\,\text{km} \tag{2.7}$$

$$r_2 = \text{average Earth-Moon distance} = 3.85 \times 10^5\,\text{km} \tag{2.8}$$

$$r_3 = \text{Moon radius} + 100\,\text{km} = 1738\,\text{km} + 160\,\text{km} = 1838\,\text{km}. \tag{2.9}$$

$$\tag{2.10}$$

All the following Δv impulse calculations use the vis-viva equation (2.4) with various values for r and a and velocities are in the geocentric system unless otherwise specified.

2.3.1 Impulse at Earth

Velocity in circular Earth orbit at distance r_1 ($r = r_1,\ a = r_1$):

$$v_{\text{earth}} = \sqrt{\mu_E \left(\frac{2}{r_1} - \frac{1}{r_1}\right)} = \sqrt{\frac{\mu_E}{r_1}} = 7.814\,\text{km/s} \tag{2.11}$$

Velocity in elliptical Earth orbit at perigee distance r_1 ($r = r_1,\ a = (r_1 + r_2)/2$):

$$v_p = \sqrt{\mu_E \left(\frac{2}{r_1} - \frac{1}{\frac{r_1+r_2}{2}}\right)} = \sqrt{\frac{\mu_E}{r_1}\frac{2r_2}{r_1 + r_2}} = 10.96\,\text{km/s} \tag{2.12}$$

So the neccesary delta-v at Earth:

$$\Delta v_{\text{earth}} = v_p - v_{\text{earth}} = 3.144\,\text{km/s} \tag{2.13}$$

2.3.2 Impulse at Moon orbit - ignoring the moon

Velocity in elliptical Earth orbit at apogee distance r_2 ($r = r_2,\ a = (r_1 + r_2)/2$):

$$v_a = \sqrt{\mu_E \left(\frac{2}{r_2} - \frac{1}{\frac{r_1+r_2}{2}}\right)} = \sqrt{\frac{\mu_E}{r_2}\frac{2r_1}{r_1 + r_2}} = 0.1858\,\text{km/s} \tag{2.14}$$

Velocity in circular Earth orbit at distance r_2 ($r = r_2,\ a = r_2$):

$$v_{\text{earth2}} = \sqrt{\frac{\mu_E}{r_2}} = 1.017\,\text{km/s} \tag{2.15}$$

So the necessary Δv at Moon orbit to enter circular Earth orbit (if the moon was not there) is an increase in velocity of

$$\Delta v_{\text{no-moon}} = v_{\text{earth2}} - v_a = 0.832\,\text{km/s} \tag{2.16}$$

2.3.3 Impulse at Moon orbit - including the moon

However we can't follow the moon stationary along its orbit, we need to go into a circular orbit around the moon, so we'll make a correction to our $\Delta v_{\text{no-moon}}$.

Velocity in circular orbit at distance 100 km above the moon surface *in Moon system*:

$$v_{\text{moon}} = \sqrt{\frac{\mu_M}{r_3}} = 1.633\,\text{km/s} \tag{2.17}$$

Now we can choose to add or subtract this value to $\Delta v_{\text{no-moon}}$ since we can go into orbit both ways around the moon (clockwise and counterclockwise). We will subtract v_{moon} from $\Delta v_{\text{no-moon}}$ since this minimizes the Δv.

So the necessary change in velocity needed to go from geocentric elliptical orbit to circular Moon orbit is a decrease in velocity:

$$\Delta v_{\text{moon}} = \Delta v_{\text{no-moon}} - v_{\text{moon}} = -0.8017\,\text{km/s} \tag{2.18}$$

So the total Δv for the whole trip is

$$\Delta v_{\text{total}} = |\Delta v_{\text{earth}}| + |\Delta v_{\text{moon}}| = 3.946\,\text{km/s}. \tag{2.19}$$

The flight time is approximately the time spend on the elliptical orbit, which is half the elliptical orbital period, so by Kepler's third law [MD99]:

$$t_{\text{Hohmann}} = \frac{1}{2}\sqrt{\frac{4\pi^2(\frac{r1+r2}{2})^3}{\mu_E}} = 5.0\,\text{days} \tag{2.20}$$

A more detailed analysis of a Hohmann-like LTI with identical initial and final orbits, turns out a very similar $\Delta v_{\text{total}} = 3.959\,\text{km}$ [Swe91] [Juu08]. This seems to somewhat justify our fourth and perhaps seemingly most questionable assumption at the beginning of the section.

Coincidentally our Δv_{total} in 2.19 is actually pretty close to the moon-less scenario:

$$\Delta v_{\text{total-no-moon}} = |\Delta v_{\text{earth}}| + |\Delta v_{\text{no-moon}}| = 3.976\,\text{km/s} \tag{2.21}$$

This makes it relevant to pose the question: could we possibly minimize Δv_{total} by choosing a moon orbit altitude such that the necessary moon orbting velocity in the moon system is equal to the incoming velocity on the ellipse at the intersection in the Earth system? In effect, meaning no velocity change necessary at point B in fig 2.6, i.e. $\Delta v_{\text{moon}} = 0$? We set $r_3 = \text{Moon radius} + x$ and solve $\Delta v_B = 0$ for x and find $x = 5350\,\text{km}$ and $\Delta v_{\text{total-min}} = 3.144\,\text{km/s}$ (see Appendix A for calculations and Δv_{total} vs. Moon orbital distance plot). This is a valid orbital altitude since its well within the moon's Hill sphere of radius of roughly 62.000 km. The Hill sphere radius marks the point between Earth and Moon at which a spacecraft switches from orbiting one to orbiting the other. In more precise terms the Hill sphere is the approximate equipotential surface around a smaller body m orbiting a larger body M at distance $r_{m,M}$, indicating the boundary of the region where a smaller spacecraft m_s would orbit m rather than M. In other words its the distance from m where the forces from M and m add up to the centripetal force towards M. The radius of the Hill sphere is given by [MD99]

$$r_{\text{hill}} = r_{m,M}\sqrt[3]{\frac{m}{3M}} \tag{2.22}$$

The Hill sphere of the moon in our Earth-Moon system is (appendix A):

$$r_{\text{hill-moon}} = r_2\sqrt[3]{\frac{m_M}{3m_E}} = 0.16r_2 = 62.000\,\text{km} \tag{2.23}$$

So the Hohmann transfer orbit is the most effective way of getting into a high lunar orbit. It is also the fastest way to the moon. The Apollo missions used 4.115 km/s and took 3.05 days. However ultimately we're not just interested in getting in *any* circular orbit around the moon, but in low lunar orbit (LLO) (lunar orbits less than 100 km altitude [NAS66]) at low Δv for the purpose of unmanned orbital missions- or landings.

Now that we fully understand the Hohmann-like TLI, its performance and limitations, we're ready to look at the low-energy transfer orbits.

2.4 Low-Energy Transfer Orbits - Slow But Cost-Effective

In 1990 the Japanese spacecraft "Hiten" was launched and failed part of its mission, ending in a high elliptical around the Earth, however with insufficient fuel to perform the Hohmann transfer-orbit maneuver to reach the Moon. Edward Belbruno proposed an non-traditional trajectory that could get the spacecraft to the Moon anyway [Wil]. Originally invented by Belbruno in 1987 [See02], this marked the first practical use of a *low-energy transfer orbit* [NASc], that is an orbit that uses little fuel compared to traditional direct-route transfer-orbit such as Hohmann. Figure 2.7 shows the mission trajectory profile. It was a success and marked the beginning of an active research area in low-energy transfer orbits, also known as weak stability boundary trajectories or ballistic capture trajectories.

In the last section we found that Hohmann transfer orbits to the moon need to brake at the moon to shed 0.832 km/s. The idea of a low-energy transfer-orbit is to have the spacecraft captured ballistically by the moon, spending much less fuel on firing retrorockets when the spacecraft reaches the moon. We can expect savings in Δv up to 150 m/s at the expense of the flight time going from the order of 5 days to 60-100 days [GG07]. A saving of 150 m/s is a small proportion of the total delta-v, but since the Δv and fuel are being saved from a late point in the trip (braking at the moon), it takes a lot of fuel to transport the fuel itself to the moon and the effect is quite significant: upwards of 25 % can be saved off the Δv after leaving LEO. As a result under some circumstances, double the payload mass that can be placed into LLO (Low Lunar Orbit) [BC00].

Another advantage is more flexible launch windows. A launch window is the time intervals during which the launch of a particular mission is possible from a specific location. Launch windows vary in both size and frequency. During the Apollo missions, the monthly launch window consisted of a few days during a given month or lunar cycle. The daily window had a duration of a few hours during a given 24 hour period [NASb]. Since a low-energy transfer-orbit can approach it's target from many more directions due to it's chaotic nature, they typically provide more flexibility concerning launch time windows compared to the more traditional Hohmann transfers. It has been demonstrated that it is possible to achieve lunar orbit for a 20 minute launch window on a given day

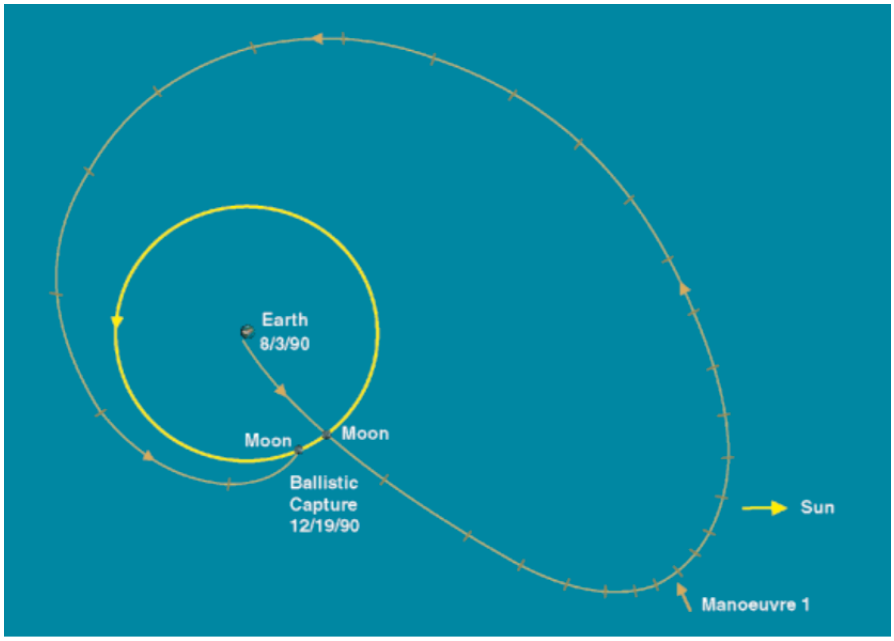


Figure 2.7: Hiten mission profile showing the Belbruno trajectory, which interacts with the moon twice in order to be captured ballistically by the moon. Tick marks are at 5 day intervals. Source: [BJ00].

for only up to about 50 meters/second Δv extra. Furthermore, for an additional 50 meters/second a launch period of about 5 days per lunar cycle can be achieved [BC00].

The flexibility advantage is even more pronounced for Mars, where three advantages can be highlighted: the Δv performance, the much more flexible launch windows (Hohmann to Mars is just a few times a year), and finally its safer than Hohmann in the sense that the capture Δv is applied far from Mars and in a gradual manner. By comparison, the capture process for a Hohmann transfer needs to be done very quickly and be precisely timed or the spacecraft is lost. [TB14].

In any optimization scheme is it very useful to know the theoretical limit. For a trajectory that starts in a circular orbit 167 km above the Earth orbit and ends at a 100 km altitude circular polar moon orbit, [Swe91] found a lower bound of Δv at 3.721 km/s. The associated trajectory does not exist since it requires, theoretically, an infinite time to approach L_1 and to depart from it [TVB05].

Even in the simplest model, the planar circular restricted 3-body problem that we have chosen, the system is chaotic. Therefore the low-energy transfer orbits cannot be found analytically and we must apply some trial-and-error to find them. However instead of relying completely on brute force, randomly searching through all possible trajectories, we can be guided by theory some of the way by studying the energy contours of the system. To this end it is useful to introduce Lagrange points.

2.4.1 Lagrange Points and Low-Energy Transfer Orbits

For two large bodies of masses M and m there exists points in space where the sum of the gravitational forces from the two bodies equals the centripetal force required to orbit with them around the center of mass [MD99]. There exists five such points denoted by $L_1 - L_5$, see figure 2.8.

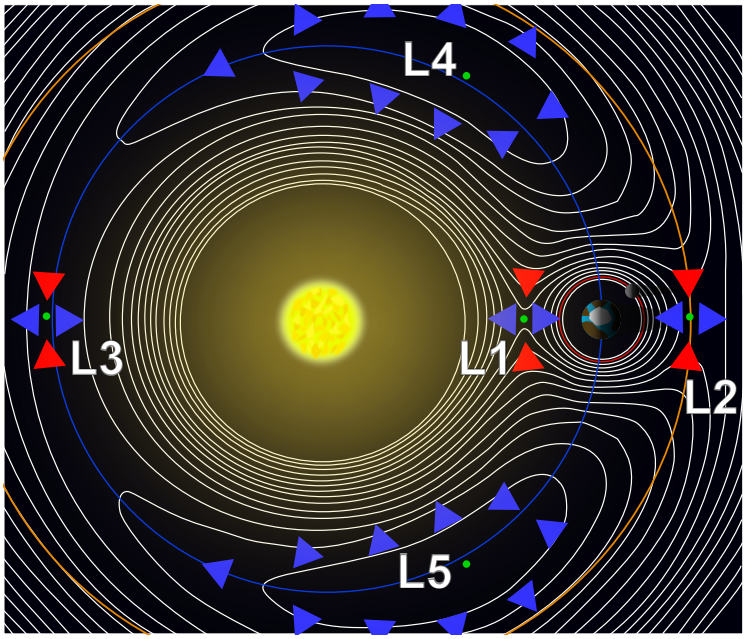
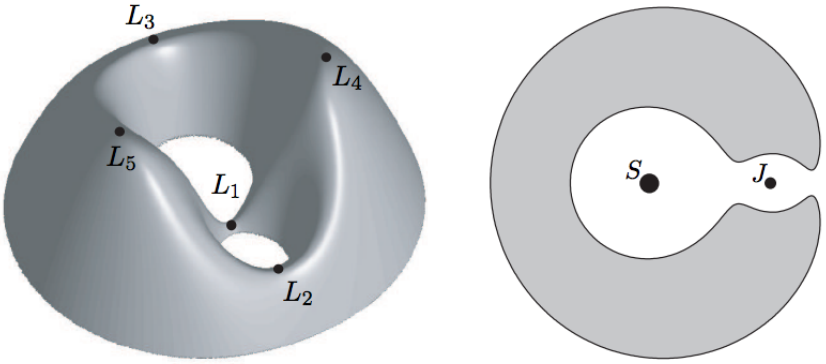


Figure 2.8: Contour plot of the effective potential due to gravity of a two-body system in a rotating frame of reference. The red and blue arrows indicate positive and negative potential gradients, respectively. Also shows all the Lagrange Points $L_1 - L_5$ are indicated by green dots. They are points of equilibria, or stationary points where the all the gravitational forces and centrifugal force cancel out. Note that this picture depicts the potential and Lagrange points of Sun-Earth (the moon is only there for show) but it applies equally well the Earth-Moon or any other approximate 2-body system.

If $M \gg m$, then the distance to L_1 and L_2 from m are exactly equal to the Hill sphere (2.22).

Looking at figure 2.9, we see that the lowest energy orbits must go through the saddle point or "neck" with the Lagrange point L_1 in the middle. However if we go straight to the Moon through the neck, we've seen that we arrive at the Moon with an excess velocity and must spend energy on breaking. Using a trajectory-search program to find trajectories through a region around L_1, we expect to find some Moon capture trajectories with lower delta-v than Hohmann.



(a) Effective potential

(b) Equipotential slice, allowed and forbidden regions

Figure 2.9: Visualizations of the potential. For some energy, 2.9(b) shows the allowed regions in white, forbidden in gray..

CHAPTER 3

Analytical Mechanics

> "*If Nature has defined the mechanics problem of the thrown ball in so elegant a fashion, might She have defined other problems similarly. So it seems now. Indeed, at the present time it appears that we can describe all the fundamental forces in terms of a Lagrangian. The search for Nature's One Equation, which rules all of the universe, has been largely a search for an adequate Lagrangian.*"
>
> — Robert Adair, The Great Design: Particles, Fields, and Creation

3.1 Newtonian Mechanics

Since the early beginnings of classical mechanics, there have existed fundamentally two different approaches to the study of motion: vectorial ("classical") and variational ("analytical") treatment of mechanics. The first, a differential approach, the second, an integral approach.

The vectorial treatment was pioneered by Isaac Newton and is based on two fundamental vectors: momentum and force. Analysis of the forces acting on a body yield second-order differential equations of motion by Newton's 2nd Law, equation (3.1). These differential equations can be solved to yield position and velocity at every point in time.

$$\textbf{Newton's 2nd Law} \qquad \sum \mathbf{F} = \frac{d\mathbf{p}}{dt} \tag{3.1}$$

3.2 Lagrangian Mechanics

The first seeds of the variational approach was sown by G.W. Leibniz, a contemporary of Newton. Leibniz advocated another quantity "vis viva " (living force), which only differs from what we now call "kinetic energy" by a factor of 2. Thus Leibniz replaced Newton's "momentum" with "kinetic energy" and "force" with "work of the force". [Lan70, p. xxi]. The power of the variational approach is that these two scalar quantities can completely contain all the dynamics of any system. Around 1743 Euler discovered what later became known as the Euler-Lagrange equations (equations (3.3) [Lan70, p. 61]) using geometrical methods [Gol80, p. 67] [Sta]. In 1755 J.L. Lagrange re-derived Euler's

results using just algebraic methods, a method that Euler dubbed "calculus of variations" [Gol80, p. xiv] [Sta]. Thoughout the 18th century new developments analytical mechanics was made, culminating in Lagrange's publication "Mécanique Analytique" in 1788 [Fra83, p. 197]. The central quantity in Lagragian mechanics is called the Lagrangian [Hjo15, p. 3]

$$\textbf{Lagrangian} \qquad L = T - V \tag{3.2}$$

, where T is the total kinetic energy of the body and V is the total potential energy of the system. The central equation is called the Euler-Lagrange Equation [Hjo15]

$$\textbf{Euler-Lagrange equation} \qquad \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = 0 \quad \text{, (k=1,2,...,n)} \ , \tag{3.3}$$

where q are the generalized coordinates (to be introduced in the next chapter) and the dot meaning the time derivative.

3.3 Hamiltonian Mechanics

Classical mechanics underwent one last important reformulation. In 1834 W.R. Hamilton reformulated classical mechanics based on Lagrange's powerful formalism [NF02, p. 161]. Hamilton's formalism is a systematic, automatic way to recast the 2nd order equations of Newton and Lagrange, and make it a first-order order system. It builds on the Lagrangian but relies on a new quantity called the generalized momentum

$$\textbf{Generalized momenta} \qquad p_i(\mathbf{q}, \dot{\mathbf{q}}) = \frac{\partial L}{\partial \dot{q}_i}, \qquad i = 1, \cdots, n \tag{3.4}$$

$$\textbf{Hamiltonian} \qquad H(\mathbf{q}, \mathbf{p}, t) = \sum_{i=1}^{n} p_i \dot{q}_i - L \tag{3.5}$$

$$\textbf{Hamilton's equations} \qquad \begin{aligned} \dot{q}_i &= +\frac{\partial H}{\partial p_i}, \\ \dot{p}_i &= -\frac{\partial H}{\partial q_i} \end{aligned} \tag{3.6}$$

[Hjo15] Thus Lagrangian mechanics and Hamiltonian mechanics is energy-centric rather than force-centric. One of the great strengths of the variational approach is that the system can be more or less solved simply by appropriate choice of coordinates. This is due to the fact that appropriate coordinates automatically accounts for the constraints of movements (e.g. a bead on a string, a ball rolling on a surface or electrical charge moving on a surface) so we don't have to account for these constraints by complicated

force analysis. These coordinates q_i we call *generalized coordinates*. They are "*any quantitative attributes of the system (for example, strength of the magnetic field at a particular location; angle of a pulley; position of a particle in space; or degree of excitation of a particular eigenmode in a complex system) which are functions of the independent variable(s)*" [Wikb]. The number of ways the system can move subject to the constrains is called the degrees of freedom. To fully describe the system, the number of generalized coordinates chosen for the system must equal the system's degrees of freedom.

Thus instead of analyzing forces, including troublesome forces of contraints, we instead define and use coordinates that only describe states of the system that satisfies the constraints. Then, using the Euler-Lagrange's equations (3.3) or Hamilton's equations (3.6), we immediately obtain the equations of motion. Another advantage of the Hamiltonian method is that the underlying symmetries in the system are often more evident in first-order differential equations. On a theoretical level this is important due to Noether's Theorem: "Each symmetry of a system leads to a physically conserved quantity" [Wei]. It's also convenient on a practical level with regards to solving 1st order equations equations numerically. In other words, we prefer $2n$ first-order ordinary differential equations (ODE) to n second-order ODEs.

3.4 Using Hamiltonian Mechanics

We will now look at the procedure for using Hamiltonian mechanics in practice. In the next chapter "Numerical Methods" we will then solve the equations of motion, in part to demonstrate the validity of Hamilton's equations of motion, and in part to analyze how various numerical techniques perform.

Step 0 Lagrangian L

a) Define general coordinates q_i, $i = 1, 2 \dots n$, where n is the degrees of freedom for the system.

b) Determine kinetic energy $T(\mathbf{q}, \dot{\mathbf{q}}, t) = \frac{1}{2}mv^2$.

c) Determine potential energy $V(\mathbf{q}, \dot{\mathbf{q}}, t)$.

d) Lagrangian:

$$L = T - V \tag{3.7}$$

.

Step 1 Generalized momenta p_i

$$p_i(\mathbf{q}, \dot{\mathbf{q}}, t) = \frac{\partial L}{\partial \dot{q}_i} \tag{3.8}$$

, for all $i = 1 \dots n$.

Step 2 Transform all the $\dot{q}_i$

Technically called a Legendre transform, this is the step that takes us from Lagrangian mechanics to Hamiltonian mechanics. We basically isolate the $\dot{q}_i$ in the p_i-equations from step 1, and eliminate all $\dot{q}_i$ in the equations in favor of p_i. Thus we go from independent variables $(\mathbf{q}, \dot{\mathbf{q}})$ to $(\mathbf{q}, \mathbf{p})$.

$$\dot{q}_i = \dot{q}_i(\mathbf{q}, \mathbf{p}, t) \tag{3.9}$$

Step 3 The Hamiltonian H

$$H(\mathbf{q}, \mathbf{p}, t) = \sum_{i=1}^{n} p_i \dot{q}_i - L \tag{3.10}$$

, for all $n = 1 \cdots n$, where it is understood that all the q_i are substituted with expressions found in step 2.

Step 4 Hamilton's Equations of Motion

$$\begin{aligned} \dot{q}_i &= +\frac{\partial H}{\partial p_i}, \\ \dot{p}_i &= -\frac{\partial H}{\partial q_i} \end{aligned} \tag{3.11}$$

, for $i = 1 \dots n$, which gives us $2n$ 1st order coupled PDEs of $2n$ variables $(q_1, q_2, \dots, q_n, p_1, p_2, \dots, p_n)$.

In general T and V can be time-dependent (and therefore L and H can too). However in many applications, including the our model problem, they are not time-dependent.

An important property of the Hamiltonian is that if it is not explicitely time dependent then it is conserved $\mathrm{d}H/\mathrm{d}t = 0$ along the $(\mathbf{p}(t), \mathbf{q}(\mathbf{t}))$ flow [Hjo15].

3.4.1 H vs. E

An important characteristic of a closed physical system is it's energy E. By a clever choice of coordinate system, it is possible to have systems where Hamiltonian H is conserved, but the total mechanical energy E is not. In that sense in can be argued that the Hamiltonian is a more general concept than energy. It can be shown that $H = E$ if and only if the following three conditions are met: [GPS02, p. 60-64] [Bra12] [HT10]

1. Equations of constraints, T and V have no explicit time-dependency.
2. V is independent of $\dot{\mathbf{q}}$.
3. T is a homogeneous quadratic in the $\dot{q}$'s, in particular if $T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top M(q)\dot{\mathbf{q}}$, where $M(q)$ is some symmetric and positive definite matrix ,

Meeting these conditions also implies that the generalized impulses p_i will be equal to well known conserved quantities such as linear momentum, angular momentum etc. We will later see that the equations for restricted 3-body system satisfy the conditions. Another way to see if T is a quadratic form is if it's a homogenous polynomial, i.e. all terms have same degree in a number of variables. For example $P(x, y) = 4x^2 + 2xy + 3y^2$ [Wikd] is quadratic.

3.5 The Restricted 3-body Problem

3.5.1 Assumptions and Setup

We make the same first three assumptions as in chapter 2.3: The gravitational effects of the sun can be neglected, the moon's orbit around the Earth is assumed to be circular and spacecraft mass has neglegible influence on the Earth and Moon. Unlike chapter 2.3, we will now include the additional gravitational influence of the moon, without which it would be impossible to find low-energy transfers to the Moon.

We define two cartesian coordinate systems, both with their origin in the Moon-Earth center of mass (CM). The $(\mathcal{X}, \mathcal{Y})$ system is a stationary inertial frame, and the (x, y) system is co-rotating with the Earth and the Moon in their orbits, see figure 3.1.

We have two choices of coordinate systems to use for Lagrangian:

1. If we choose the inertial frame of reference, T is straight forward, but distances in the V becomes bothersome, albeit we can use the straightforward gravitational potential function.

2. If we choose the rotating (x, y) system, distances for V is straight forward but then we either have to use polar coordinates, use an effective potential function instead of the standard gravitational potential, or use cartesian coordinates but find a coordinate transformation $X(x, y)$ and $Y(x, y)$.

I arbitrarily choose the latter option: use (x, y) coordinates and find $X(x, y)$ and $Y(x, y)$ transformation.

The distances between spacecraft and Earth/Moon are delightfully easy to work out:

$$r_{S,E} = \sqrt{(x + R)^2 + y^2} \tag{3.12}$$

$$r_{S,M} = \sqrt{(x - r)^2 + y^2} \tag{3.13}$$

In anticipation for calculating the kinetic energy of the system, we need the quantity

$$\dot{\mathcal{X}}^2 + \dot{\mathcal{Y}}^2 \tag{3.14}$$

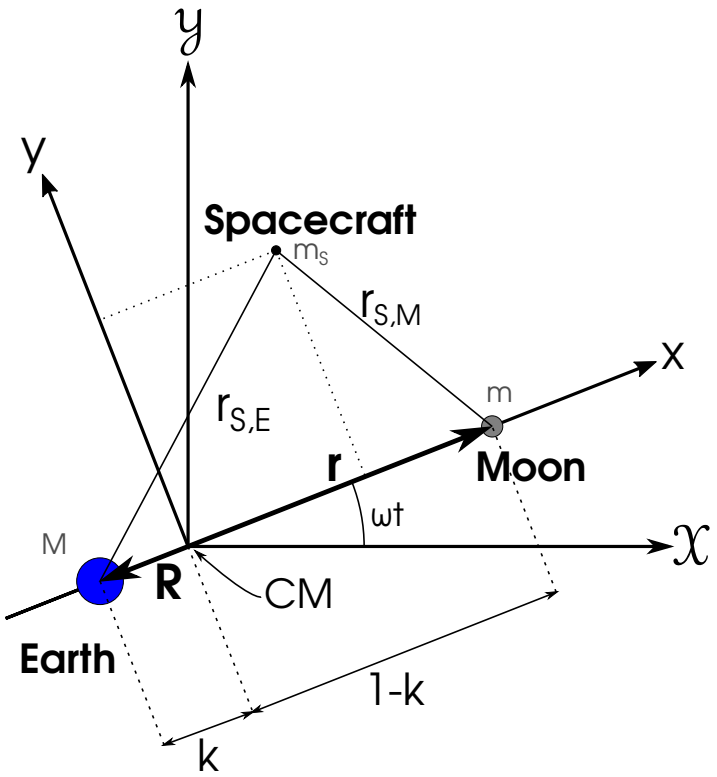


Figure 3.1: Restricted 3-body problem. Two coordinate systems both with center of mass as origin: $(\mathcal{X}, \mathcal{Y})$ is a stationary interial frame, (x, y) is a co-rotating non-interial frame that rotates with the moon at angular frequency ω. M = mass of Earth, m = mass of Moon, $\mathbf{R}$ = vector from CM to earth, $\mathbf{r}$ = vector from CM to Moon, m_s = mass of spacecraft, $r_{S,E}$ = distance from spacecraft to Earth, $r_{S,M}$ = distance from spacecraft to Moon. In the dimensionless variables we introduce later, unit distance is $R + r$, which makes the dimensionsless constant $k = \dfrac{R}{R+r}$ = the CM-Earth distance and $1 - k = \dfrac{r}{R+r}$ = the CM-Moon distance.

In matrix notation the coordinate transform between $(\mathcal{X}, \mathcal{Y})$ and (x, y) is obtained by the multiplication of the rotation matrix on the inertial frame coordinates

$$\begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \end{bmatrix} = \begin{bmatrix} \cos\omega t & -\sin\omega t \\ \sin\omega t & \cos\omega t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\Rightarrow \begin{cases} \mathcal{X} = x\cos\omega t - y\sin\omega t \\ \mathcal{Y} = x\sin\omega t + y\cos\omega t \end{cases} \tag{3.15}$$

By using the pythagorean trigonometric identity[1] a number of times (see appendix B

[1] $\cos^2\theta + \sin^2\theta = 1$

for detailed derivation) we obtain

$$\dot{\mathcal{X}}^2 + \dot{\mathcal{Y}}^2 = (\dot{x} - \omega y)^2 + (\dot{y} + \omega x)^2 \tag{3.16}$$

Now we can really see another convenience of the (x, y) coordinate system: it eliminates explicite time-dependence of L that would have been there in V in the $(\mathcal{X}, \mathcal{Y})$ system.

We are now ready to derive the equations of motion using the procedure from chapter 3.4.

3.5.2 Equations of Motion

The kinetic energy is

$$\begin{aligned} T &= \frac{1}{2} m_s (\dot{\mathcal{X}}^2 + \dot{\mathcal{Y}}^2) \\ &= \frac{1}{2} m_s \left[(\dot{x} - \omega y)^2 + (\dot{y} + \omega x)^2 \right] \end{aligned} \tag{3.17}$$

The potential energy is

$$\begin{aligned} V &= -G m_s \left(\frac{M}{r_{S,E}} + \frac{m}{r_{S,M}} \right) \\ &= -G m_s \left(\frac{M}{\sqrt{(x+R)^2 + y^2}} + \frac{m}{\sqrt{(x-r)^2 + y^2}} \right) \end{aligned} \tag{3.18}$$

The Lagrangian is

$$\begin{aligned} L &= T - V \\ &= \frac{1}{2} m_s \left[(\dot{x} - \omega y)^2 + (\dot{y} + \omega x)^2 \right] \\ &+ G m_s \left(\frac{M}{\sqrt{(x+R)^2 + y^2}} + \frac{m}{\sqrt{(x-r)^2 + y^2}} \right) \end{aligned} \tag{3.19}$$

The generalized momenta are

$$p_x = \frac{\partial L}{\partial \dot{x}} = m_s (\dot{x} - \omega y) \tag{3.20}$$

$$p_y = \frac{\partial L}{\partial \dot{y}} = m_s (\dot{y} + \omega x) \tag{3.21}$$

From which we get the $\dot{x}(x, p_x)$ and $\dot{y}(y, p_y)$

$$\dot{x} = \frac{p_x}{m_s} + \omega y \tag{3.22}$$

$$\dot{y} = \frac{p_y}{m_s} - \omega x \tag{3.23}$$

The Hamiltonian is

$$H = \sum_{i=1}^{n} p_i \dot{q}_i - L \tag{3.24}$$

$$= \frac{p_x^2 + p_y^2}{m_s} + p_x \omega y - p_y \omega x - \frac{1}{2} m_s \left[\left(\frac{p_x}{m_s} \right)^2 + \left(\frac{p_y}{m_s} \right)^2 \right] \tag{3.25}$$

$$- G m_s \left(\frac{M}{\sqrt{(x+R)^2 + y^2}} + \frac{m}{\sqrt{(x-r)^2 + y^2}} \right) \tag{3.26}$$

$$= \frac{p_x^2 + p_y^2}{2m_s} + p_x \omega y - p_y \omega x - G m_s \left(\frac{M}{\sqrt{(x+R)^2 + y^2}} + \frac{m}{\sqrt{(x-r)^2 + y^2}} \right) \tag{3.27}$$

so Hamilton's equations are

$$\dot{x} = +\frac{\partial H}{\partial p_x} = \frac{p_x}{m_s} + \omega y \tag{3.28}$$

$$\dot{y} = +\frac{\partial H}{\partial p_y} = \frac{p_y}{m_s} - \omega x \tag{3.29}$$

$$\dot{p}_x = -\frac{\partial H}{\partial x} = \omega p_y - G m_s \left[-\frac{M(x+R)}{((x+R)^2 + y^2)^{3/2}} + \frac{m(x-r)}{((x-r)^2 + y^2)^{3/2}} \right] \tag{3.30}$$

$$\dot{p}_y = -\frac{\partial H}{\partial y} = -\omega p_x - G m_s \left[-\frac{My}{((x+R)^2 + y^2)^{3/2}} - \frac{my}{((x-r)^2 + y^2)^{3/2}} \right] \tag{3.31}$$

By appropriate choice of units we can render the equations dimensionless.

A characteristic distance must be the Earth-Moon distance so:

$$\text{One unit distance} = R + r. \tag{3.32}$$

A natural choice for the characteristic time is the moon's orbital period. Because the time dimension is present in the equations as ω we use Kepler's third law to express: [MD99]

$$\text{One unit time} = \frac{1}{\omega} = \sqrt{\frac{(R+r)^3}{G(M+m)}} \tag{3.33}$$

The momentums p_x p_y of the spacecraft are going to be proportional with the spacecraft's mass m_s, so let's choose:

$$\text{One unit mass} = m_s \tag{3.34}$$

With these we can calculate:

$$\begin{aligned}\text{One unit momentum} = \text{One unit velocity} &= (\text{unit mass})\frac{\text{unit distance}}{\text{unit time}} \\ &= m_s(R+r)\sqrt{\frac{G(M+m)}{(R+r)^3}} \\ &= m_s\sqrt{\frac{G(M+m)}{(R+r)}} \end{aligned} \tag{3.35}$$

We can now non-dimensionalize all the equations and get (see appendix C)

$$\boxed{\dot{X} = P_x + Y} \tag{3.36}$$

$$\boxed{\dot{Y} = P_Y - X} \tag{3.37}$$

$$\boxed{\dot{P}_x = P_y - \frac{(1-k)(X+k)}{((X+k)^2+Y^2)^{3/2}} - \frac{k(X-1-k)}{((X-1-k)^2+Y^2)^{3/2}}} \tag{3.38}$$

$$\boxed{\dot{P}_y = -P_x - \frac{(1-k)Y}{((X+k)^2+Y^2)^{3/2}} - \frac{kY}{((X-1-k)^2+Y^2)^{3/2}},} \tag{3.39}$$

where T, X, Y, P_x and P_y are our dimensionless variables, see appendix C for details.

For the rest of the time we will denote the non-dimensionalized units simply as x, y, p_x, p_y instead of X, Y, P_x, P_y for readability.

To reiterate, we have chosen units such that

$$\text{unit length} = 3.85 \times 10^5\,\text{km} \tag{3.40}$$

$$\text{unit time} = 4.3484\,\text{days} \tag{3.41}$$

$$\text{unit velocity} = 1.025\,\text{km/s} \tag{3.42}$$

Since all of the conditions in subsection 3.4.1 are fulfilled, we have that $H = E$ and since H does not depend explicitely on time, the Hamiltonian, and thus the energy, is conserved.

CHAPTER 4

Numerical Analysis

"A computer lets you make more mistakes faster than any invention in human history - with the possible exceptions of handguns and tequila."

— Mitch Ratliffe, Technology Review, April 1992

Some systems can be solved analytically in terms of time-dependent coordinates. Other systems are non-linear and the vast majority cannot be solved analytically. This is the case for the restricted 3-body problem, (3.36) to (3.39). To solve these equations of motion, we must use numerical methods.

The simplest numerical integration is the first order linear approximation known as the Euler method. There are three variants of the Euler method: explicit, implicit and sympletic. As we will see, the explicit Euler will typically increase the energy, implicit Euler will decrease the energy and the symplectic Euler will oscillate the energy slightly but on average conserve it to a high degree.

We will now solve the restricted 3-body problem equations of motion, (3.36) to (3.39). First we discretize the equations

$$\Delta x = +\frac{\partial H}{\partial p_x}\Delta t = (p_x + y)\Delta t \tag{4.1}$$

$$\Delta y = +\frac{\partial H}{\partial p_y}\Delta t = (p_y - x)\Delta t \tag{4.2}$$

$$\Delta p_x = -\frac{\partial H}{\partial x}\Delta t = \left(p_y - \frac{(1-k)(k+x)}{((k+x)^2+y^2)^{3/2}} + \frac{k(1-k-x)}{((1-k-x)^2+y^2)^{3/2}}\right)\Delta t \tag{4.3}$$

$$\Delta p_y = -\frac{\partial H}{\partial y}\Delta t = \left(-p_x - \frac{(1-k)y}{((k+x)^2+y^2)^{3/2}} - \frac{ky}{((1-k-x)^2+y^2)^{3/2}}\right)\Delta t \tag{4.4}$$

From now on we will use h instead of Δt for the time step size.

4.1 Explicit Euler algorithm

All new values $(x, y, p_x, p_y)_{i+1}$ refer to known values from the previous timestep $(x, y, p_x, p_y)_i$.

$$x_{i+1} = (p_{x,i} + y_i)h + x_i \tag{4.5}$$

$$y_{i+1} = (p_{y,i} - x_i)h + y_i \tag{4.6}$$

$$p_{x,i+1} = \left(p_{y,i} - \frac{(1-k)(k+x_i)}{((k+x_i)^2 + y_i^2)^{3/2}} + \frac{k(1-k-x_i)}{((1-k-x_i)^2 + y_i^2)^{3/2}}\right) h + p_{x,i} \tag{4.7}$$

$$p_{y,i+1} = \left(-p_{x,i} - \frac{(1-k)y_i}{((k+x_i)^2 + y_i^2)^{3/2}} - \frac{ky_i}{((1-k-x_i)^2 + y_i^2)^{3/2}}\right) h + p_{y,i} \tag{4.8}$$

4.2 Implicit Euler algorithm

Same as explicite Euler, except that all variables in the right-hand side refers to variables in the current timestep, $i + 1$. This algorithm would involve finding four roots simultanously and numerically. Since we know the Symplectic to be more correct, we skip the implicit Euler for the restricted 3-body problem (algorithm included in appendix D.2).

4.3 Symplectic Euler algorithm

New values of $(x, y)_{i+1}$ refer to unknown values from the same timestep $(x, y, p_x, p_y)_{i+1}$ but all new values of $(p_x, p_y)_{i+1}$ refer only to known values from the previous timestep $(x, y, p_x, p_y)_i$.

$$x_{i+1} = (p_{x,i+1} + y_{i+1})h + x_i \tag{4.9}$$

$$y_{i+1} = (p_{y,i+1} - x_{i+1})h + y_i \tag{4.10}$$

$$p_{x,i+1} = \left(p_{y,i} - \frac{(1-k)(k+x_i)}{((k+x_i)^2 + y_i^2)^{3/2}} + \frac{k(1-k-x_i)}{((1-k-x_i)^2 + y_i^2)^{3/2}}\right) h + p_{x,i} \tag{4.11}$$

$$p_{y,i+1} = \left(-p_{x,i} - \frac{(1-k)y_i}{((k+x_i)^2 + y_i^2)^{3/2}} - \frac{ky_i}{((1-k-x_i)^2 + y_i^2)^{3/2}}\right) h + p_{y,i} \tag{4.12}$$

As opposed to implicit Euler, this can easily be solved by solving (4.9) and (4.10) for x_{i+1} and y_{y+1}

$$x_{i+1} = \frac{x_i + h(hp_{y,i+1} + p_{x,i+1} + y_i)}{1 + h^2} \tag{4.13}$$

$$y_{i+1} = \frac{y_i - h(hp_{x,i+1} - p_{y,i} + x_i)}{1 + h^2}, \tag{4.14}$$

and then we can run the algorithm in the order (4.11), (4.12), (4.9) and (4.10).

4.4 First Test Runs with First-Order Euler Methods

The explicit and symplectic Euler algorithm was implemented in Python. The first two things we want to investigate are:

1. Does the trajectories look like we expect them to?
2. Is the Hamiltonian conserved?

For symplectic integrators the trajectories look like we expect them to (see 4.2, but not for the non-symplectic explicit Euler (see 4.1. The Hamiltonian is not conserved for either first order (Euler) Methods, but H for the symplectic Euler was conserved on average over an orbit in a closed orbit. The simulations in figure 4.1 4.2 was run with the following parameters:

```
# Duration and stepsize
DURATION = 1.5*(2*np.pi)
H = 0.0001

# Initial Conditions
X0=0.5
Y0=0.2
PX0=0
PY0=0.5
```

Listing 4.1: Initial conditions, step size and duration for initial test run.

Having H conserved on average over an orbit is fine for a closed orbit, but for chaotic orbits around both Earth and Moon, it's not nearly good enough. We want a higher order method to conserve H better, to better gauge the numerical errors, and to allow an adaptive numerical method.

Making the algorithm select a stepsize adaptively decreases the runtime of a given simulation (though not necessarily the correctness of the solution). This is a big advantage because it allows us to search for many more orbits, which statistically will give us better orbits.

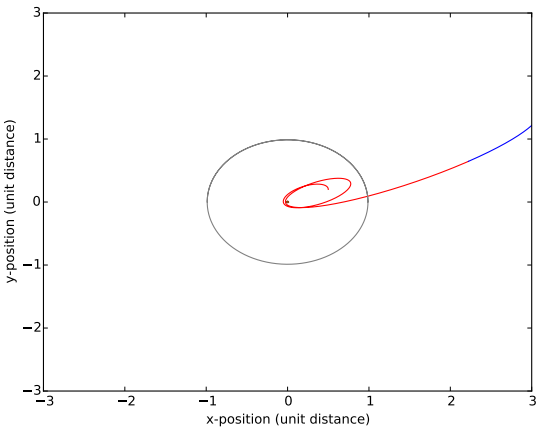
4.5 Symplectic Störmer-Verlet

We will now describe the second-order Störmer-Verlet method [Hoc08]:

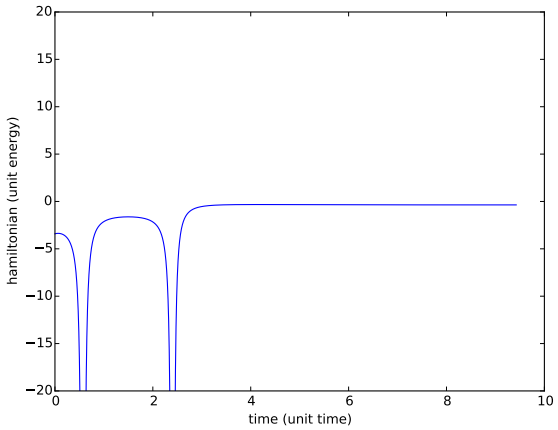
$$q_{i+1/2} = q_i + \frac{h}{2} H_p(q_{i+1/2}, p_i) \tag{4.15}$$

$$p_{i+1} = p_i + \frac{h}{2}(H_q(q_{i+1/2}, p_i) + H_q(q_{i+1/2}, p_{i+1})) \tag{4.16}$$

$$q_{i+1} = q_{i+1/2} + \frac{h}{2} H_p(q_{n+1/2}, p_{i+1/2}) \tag{4.17}$$

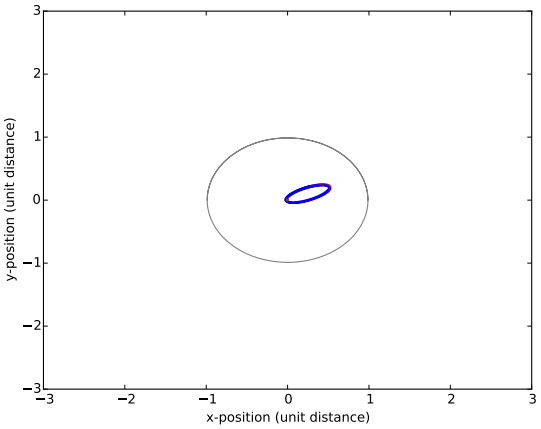


(a) Explicit Euler $(\mathcal{X}, \mathcal{Y})$ trajectory. Earth in origin, moon orbit in grey, first half of spacecraft trajectory is red, last half is blue.

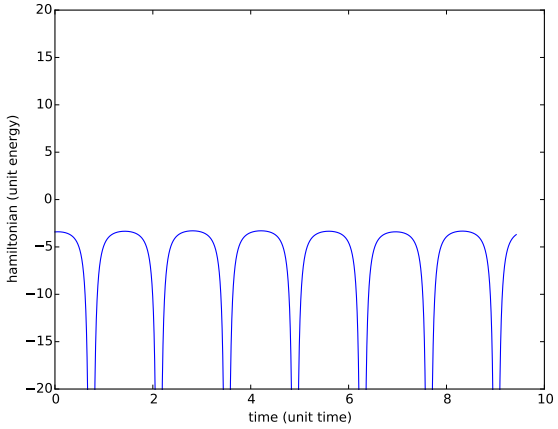


(b) Explicit Euler $H(t)$.

Figure 4.1: Explicit Euler trajectory and Hamiltonian. Ideally the $H(t)$ should be constant, but instead we see temporary dips for every Earth pass, and worse, the spacecraft picks up energy after every pass of Earth. Finally the spacecraft escapes the Earth with no apparent energy input. This is clearly wrong and not a valid trajectory.



(a) Symplectic Euler $(\mathcal{X}, \mathcal{Y})$ trajectory. Earth in origin, moon orbit in grey, first half of spacecraft trajectory is red, last half is blue.



(b) Symplectic Euler $H(t)$.

Figure 4.2: Symplectic Euler trajectory and Hamiltonian. The first half trajectory is colored red, the last half is blue. First we note the approximate ellipctical orbit as expected. The $H(t)$ shows dips is the spacecraft passes the Earth, but on average over an orbit, $H(t)$ is approximately constant..

where H_q and H_p denotes the partial derivatives of H. Implemented for our equations we get

$$x_{i+1/2} = (p_{x,i} + y_{i+1/2})\frac{h}{2} + x_i \tag{4.18}$$

$$y_{i+1/2} = (p_{y,i} - x_{i+1/2})\frac{h}{2} + y_i \tag{4.19}$$

$$p_{x,i+1} = \left[-H_{x,i} + (p_{x,i+1} + y_{i+1/2})\right]\frac{\Delta t}{2} + p_{x,i} \tag{4.20}$$

$$p_{y,i+1} = \left[-H_{y,i} + (p_{y,i+1} - x_{i+1/2})\right]\frac{\Delta t}{2} + p_{y,i} \tag{4.21}$$

$$x_{i_1} = (p_{x,i+1} + y_{i+1/2})\frac{h}{2} \tag{4.22}$$

$$y_{i_1} = (p_{y,i+1} - x_{i+1/2})\frac{h}{2}\ , \tag{4.23}$$

where

$$-H_{x,i} = \left(p_{y,i+1} - \frac{(1-k)(k+x_{i+1})}{((k+x_{i+1})^2 + y_{i+1}^2)^{3/2}} + \frac{k(1-k-x_{i+1})}{((1-k-x_{i+1})^2 + y_{i+1}^2)^{3/2}}\right) \tag{4.24}$$

$$-H_{y,i} = \left(-p_{x,i+1} - \frac{(1-k)y_{i+1}}{((k+x_{i+1})^2 + y_{i+1}^2)^{3/2}} - \frac{ky_{i+1}}{((1-k-x_{i+1})^2 + y_{i+1}^2)^{3/2}}\right) \tag{4.25}$$

4.6 Adaptive Symplectic Störmer-Verlet

Let z denote a vector of the position variables of the variables (x, y).

$$\text{Euler step result:} \qquad z_1 = z + O(h) \tag{4.26}$$

$$\text{Verlet step result:} \qquad z_2 = z + O(h^2)\ , \tag{4.27}$$

$$\tag{4.28}$$

where O(h) denotes an error term of order h. Then we take the difference

$$\|z_1 - z_2\| = O(h) - O(h^2) \tag{4.29}$$

$$\approx O(h)\ , \tag{4.30}$$

since $O(h) \gg O(h^2)$. Thus we approximate the error difference between the Euler and Verlet method as the actual error we make at stepsize h. The idea is to make both an Euler and a Verlet step for every time-step to assess the error and adjust the stepsize accordingly as implemented in listing 4.2

```
if err < tol or h <= hmin:

    # Accept step
    x = x2
    y = y2
    px = px2
    py = py2

    # Forward time by step
    t = t+h
    h = max(hmin, h*max(0.1, 0.8*sqrt(tol/err)))

else:
    # No accept, reduce h to half
    h = max(hmin, 0.5*h)
```

Listing 4.2: Adaptive method implemented in python. We accept the Verlet step only if the error is no more than a given tolerance tol. Subsequently we estimate a new h that will yield an error of tol on the next step and use 0.8 of this value to avoid frequent rejects. If the step is rejected we reduce the step size by half.

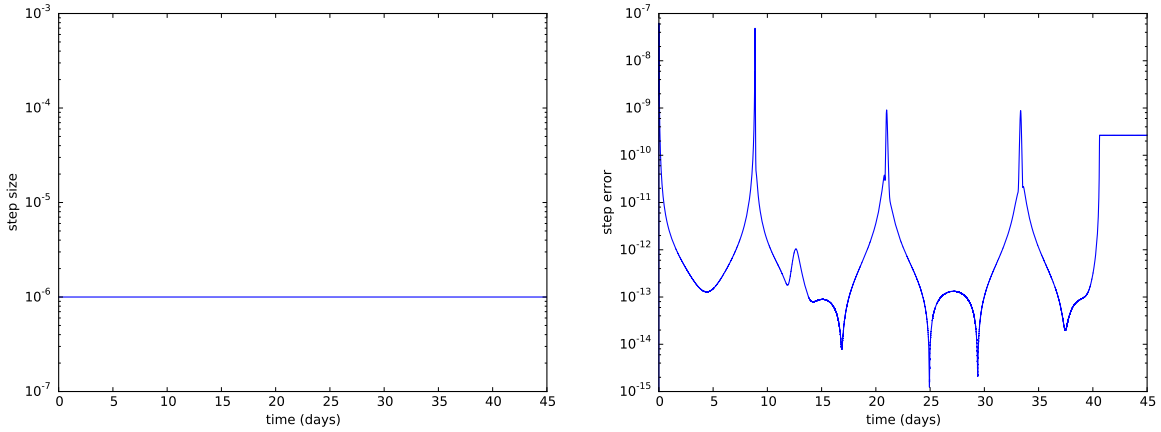
For every single step the step size is changed either up or down, depending on the errors and tolerance. As a result we always stay near the same error in every step, only taking as small steps as necessary in each iteration.

The non-adaptive algorithm is fixed in step size but varies in error per step. The adaptive algorithm varies in step size in an attempt to fix the error per step.

For all simulations we have set 10^{-9} as the maximum tolerated error per step in the adaptive algorithm. For the non-adaptive algorithm we select a fixed stepsize, 10^{-6}, to ensure that it's reasonably low most of the time. For comparison a simulation was run with same initial conditions using the adaptive and non-adaptive algorithm, see figures 4.3 - 4.4. Position plot 4.5 shown for completeness.

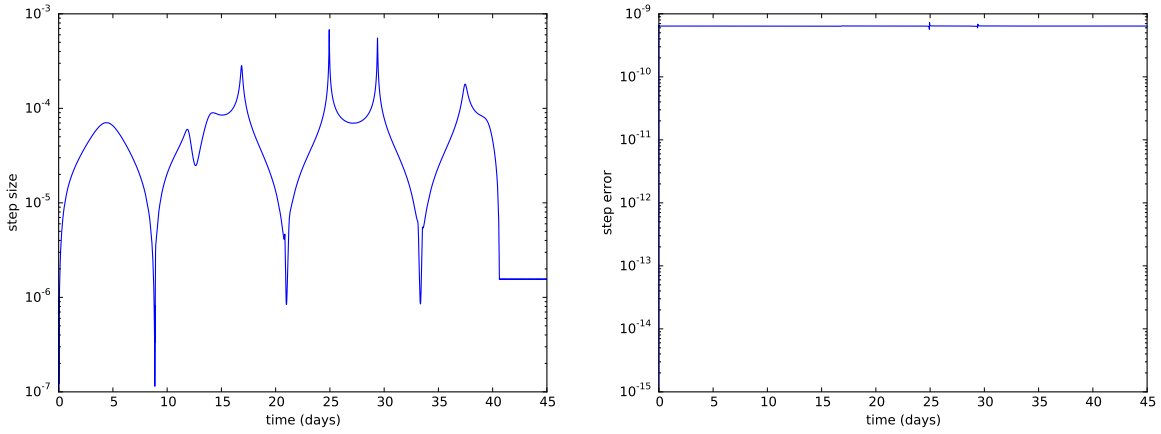
Note on the stepsize figure 4.4(a) that the adaptive ends up with constant stepsize at the end at 40 days once captured by the Moon. Thus when we are in circular orbit around a single body, we don't gain much by using adaptive methods. However as the spacecraft are in free space far away from both Earth/Moon, we gain a lot by taking longer steps (raising the stepsize). However as we make close passes to Earth or Moon, then the step size dips sharply to maintain constant error. We see this around day 9, 21 and 33 in 4.4(a). For most simulations, the adaptive method ran 10-100 times faster.

The code for the full Python implementation is in appendix E.



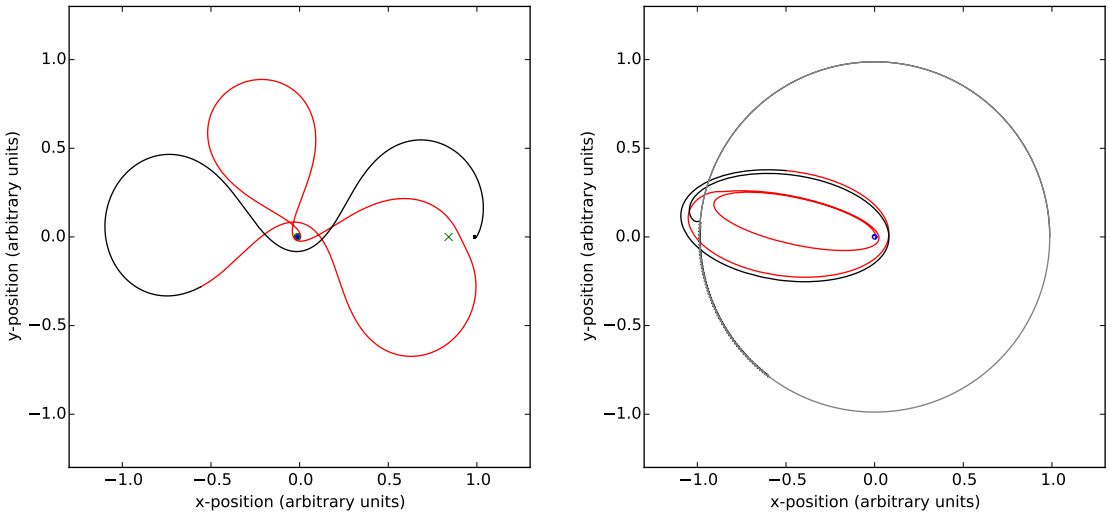
(a) Non-adaptive symplectic Verlet step size. **(b)** Non-adaptive symplectic Verlet error per step.

Figure 4.3: Non-adaptive method step size and error per step. Step size is fixed and as a result the error varies. All the work spend on calculating the many steps with lower error than 10^{-9} in 4.3(b) are wasted since the error spikes to somewhere 10^{-8} and 10^{-7} during a close fly-by, which might invalidate all decimals after the 7th decimal on the non-adaptive method. The solutions are practically identical, but the adaptive is much faster.



(a) Adaptive symplectic Verlet step size. **(b)** Adaptive symplectic Verlet error per step.

Figure 4.4: The adaptive method constantly varies the step-size to ensure a constant error around 10^{-9}. As a result it takes longer steps when there is less change in the variabels, in the space far from the bodies, and shorter steps when passing close by the Earth or Moon.



(a) Non-adaptive symplectic Verlet (x, y) trajectory.

(b) Non-adaptive symplectic Verlet $(\mathcal{X}, \mathcal{Y})$ trajectory.

Figure 4.5: The trajectories of same initial conditions for the adaptive and non-adaptive method was visually indisinguishable. Included for context to figures 4.3 - 4.4.

CHAPTER 5

Simulations of Transfer Orbits

> "*If you don't put your breaks on when you reach the moon with a spaceship with people in it, you just keep going. And you're dead. So the Hohmann transfer is risky, it's a little bit dangerous and it uses a lot of fuel because you have to slow down (…). I think Apollo used a couple hundred pounds of fuel to slow down. It's a million dollars a pound to bring anything to the Moon, including fuel. So just to slow down was a quarter of a billion dollars. I'm thinking, can you go into orbit without using your engines? Then you save all that money.*"
>
> — Edward Belbruno, Painting the Way to the Moon (2015)

5.1 Setup and Circular Orbits

Our search algorithm is basically a brute force search. All orbits are found by search method that start from circular Earth Orbit 160 km altitude to circular Moon Orbit 100 km altitude.

A search algorithm in our program then initiates many trajectory calculations with many slightly different initial conditions in the following manner:

1. Many different initial positions in Earth parking orbit denoted by θ, the angle from the positive x axis, see figure 3.1 for reference.
2. Many Δv_{earth} around some value.
3. Many small variations of $\Delta\phi$, the angle between impulse $\mathbf{\Delta v}$ and start velocity vector $\mathbf{v_0}$.

Each of the trajectories are iteratively calculated using the adaptive Verlet algorithm described in chapter 4 with the additional conditions:

1. If we hit the earth, the trajectory is discarded.
2. If we hit the moon at altitude (100 ± 10) km, the necessary Δv_{moon} to enter circular orbit is calculated and applied.

Throughout a simulation run, only the trajectory with the lowest Δv_{moon} is saved. This effectively filters out all the trajectories hitting the Moon head-on or high velocities; the best orbit will enter the Moon's orbit on an angle approximately tangent to a 100 km circular orbit.

5.2 Hohmann Transfer Orbits

If the moon was stationary in an inertial system, we would simply shoot from the opposite side of the Earth, at $\theta = \pi$. But because we must shoot not for where the moon is, but for where the moon will be. We also know that the flight time is approximately 5 days. We search for Hohmann orbits with simulation times of 6 days as follows:

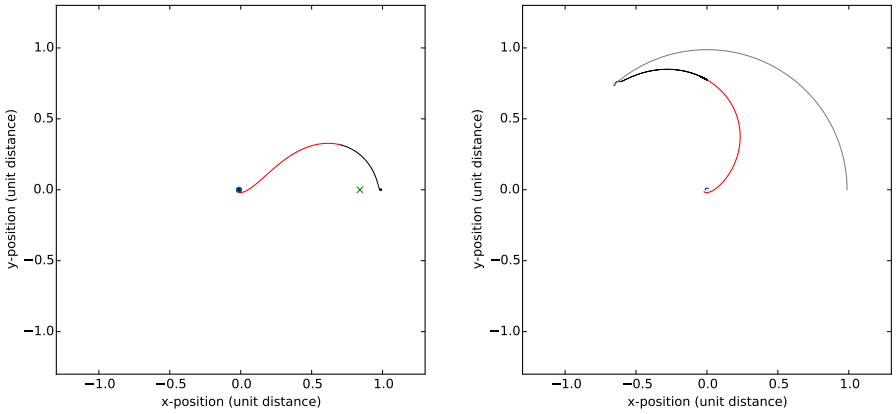
1. Angular position from Earth varied in 100 positions uniformly spaced around $\theta = -3\pi/4$ in range $\pm\ \pi/4$
2. Earth burn Δv_{earth} varied in 200 velocities uniformly around 3.11 km/s (found by trial-and-error, starting from 2.13) in range $\pm 0.1 * \texttt{unit\_velocity} = 0.102$ km/s
3. One refinement-run on the best trajectory as initial guess, where all the ranges are set to 1/10 their previous value, and number of all three parameters set to 55.

Listing 5.1 show the lowest energy Hohmann trans-lunar injection found among $100 \times 200 + 55 \times 55 \times 55 = 20000 + 3375 = 23375$ trajectories.

```
# ------------------------------------------------------------------------
duration = 5/unit_time
pos      = -2.086814820119193
ang      = -0.000122173047640
burn     = 3.111181716545691/unit_vel
x0       = -0.020532317163607
y0       = -0.014769797663479
px0      = 9.302400979050308
py0      = -5.289712560652044
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.111182 km/s
# dV(moon-capture) = 0.800682 km/s
# dV(total)        = 3.911863 km/s
# Flight-time      = 4.300078 days
# ------------------------------------------------------------------------
# Runtime = 0.51s
# Final position: 0.992505 -0.001507
# Final impulse: 0.503920 2.476928
# Final H: -2.730124
# Total runtime = 0.67s
# ------------------------------------------------------------------------
```

Listing 5.1: Best Hohmann orbit. `pos` = angular difference with start angle (here $\theta = -3\pi/4$), `ang` = angle to velocity vector in Earth parking orbit, `burn` = Δv_{earth}, `(x0,y0,px0,py0)` are the initial conditions..

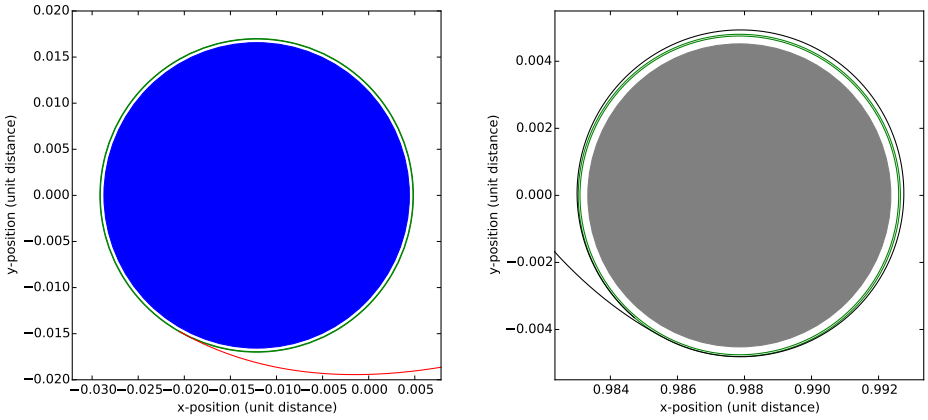
See figure 5.1 - 5.4 for trajectory position-, entry/exit- and Hamiltonian plots.



(a) Short LETO in co-rotating (x, y), Earth and Moon stationary.

(b) Hohmann in $(\mathfrak{X}, \mathfrak{Y})$, inertial system stationary on CM.

Figure 5.1: Position plots of simulated Hohmann transfer-orbit. Earth in origin, moon orbit in grey, first half of trajectory is red, last half is black, green cross is first Lagrangepoint L_1.

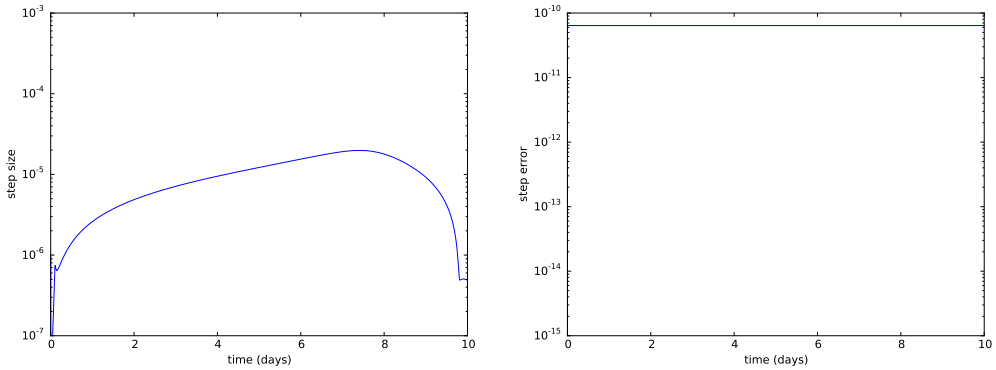


(a) Exit from circular Earth parking orbit, 100 km altitide

(b) Entry to circular Moon orbit, 100 km altitude

Figure 5.2: Exit- and entry orbits of simulated Hohmann. The green band around the moon is the altitude range of ±10 km around 100 km that triggers a capture, meaning a Δv_{moon} is calculated and applied if a discrete time step lands inside this band.

Note from listing 5.1 that $\Delta v_{\text{total}} = 3.911\,\text{km}$ is in within reasonable agreement with the prediction from the simple Hohmann model in chapter 2 of $\Delta v_{\text{total}} = 3.946\,\text{km}$. Now we just need to see if we can find a low-energy transfer-orbit significantly cheaper than this.



(a) Step size. **(b)** Error per step.

Figure 5.3: Step size and error per step of simulated Hohmann. As expected step size is longer in weaker gravitational field and vice versa, and constant in circular orbit. Error is maintained at 10^{-9}, as ensured by the adaptive method.

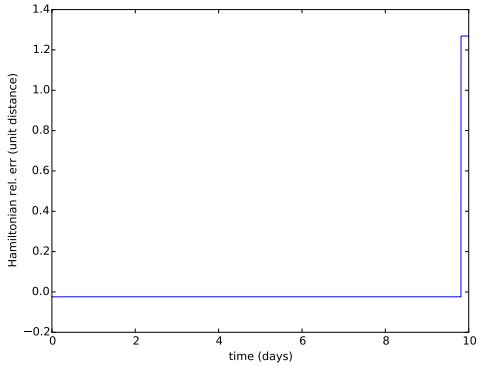


Figure 5.4: Hamiltonian H is conserved along the trajectory until Δv_{moon} is applied, as expected.

5.3 Low-Energy Transfer Orbits (LETO)

We searched for LETOs running simulations up to 200 days as follows:

1. Angular position from Earth varied in 55 positions uniformly spaced around $\theta = 0$ in range $\pm\ \pi$, i.e. in all directions.

2. Earth burn Δv_{earth} varied in 55 velocities uniformly around 3.12 km/s (found by trial-and-error) in range $\pm 0.1 * \texttt{unit\_velocity} = 0.102\,\text{km/s}$

3. Angle ϕ to velocity vector in circular orbit varied in 55 angles uniformly spaced around $\phi = 0$ in range $\pm\pi/100$

4. 7 refinement-runs around the iteratively best trajectory, where all the ranges are set to 1/10 their previous value.

We will look at two LETOs:

Short LETO Flight time of 40.6 days and $\Delta_v$total $= 3.896\,\mathrm{km/s}$, found in simulation run with maximum duration of 41 days.

Long LETO Flight time of 194.3 days and $\Delta_v$total $= 3.795\,\mathrm{km/s}$, found in a simulation run of 195 days.

Both was found with the same search parameters, except simulation duration among $55 \times 55 \times 55 \times 8 = 1{,}331{,}000$ trajectories as described above.

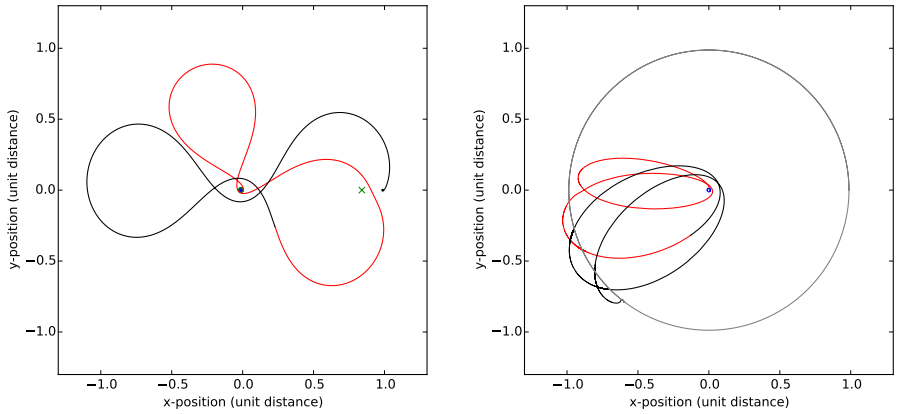
5.3.1 Short LETO

Listing 5.2 show the lowest Δv_{total} LETO trans-lunar injection

```
# ------------------------------------------------------------------------
duration = 41/unit_time
pos      = -0.138042744751570
ang      = -0.144259374836607
burn     = 3.127288444444444/unit_vel
x0       = 0.004665728429046
y0       = -0.002336647636098
px0      = 1.904735175752430
py0      = 10.504985512873279
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.127288 km/s
# dV(moon-capture) = 0.768534 km/s
# dV(total)        = 3.895822 km/s
# Flight-time      = 40.617871 days
# ------------------------------------------------------------------------
# Runtime = 0.54s
# Final position: 0.990701 0.003888
# Final impulse: 1.277242 0.047338
# Final H: -2.730117
# Total runtime = 0.71s
# ------------------------------------------------------------------------
```

Listing 5.2: Short LETO. `pos` = angular difference with start angle (here $\theta = -3\pi/4$), `ang` = angle to velocity vector in Earth parking orbit, `burn` = Δv_{earth}, `(x0,y0,px0,py0)` are the initial conditions..

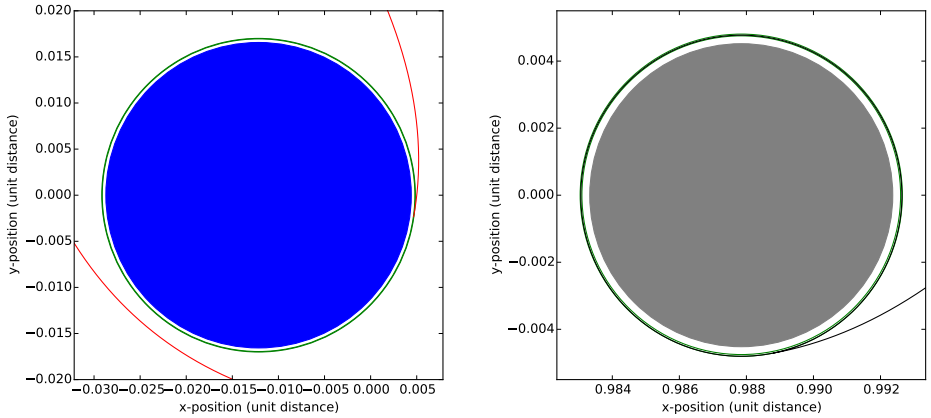
See figure 5.5 - 5.8 for trajectory position-, entry/exit- and Hamiltonian plots.



(a) Short LETO in co-rotating (x, y), Earth and Moon stationary.

(b) Short LETO in $(\mathcal{X}, \mathcal{Y})$, inertial system stationary on CM.

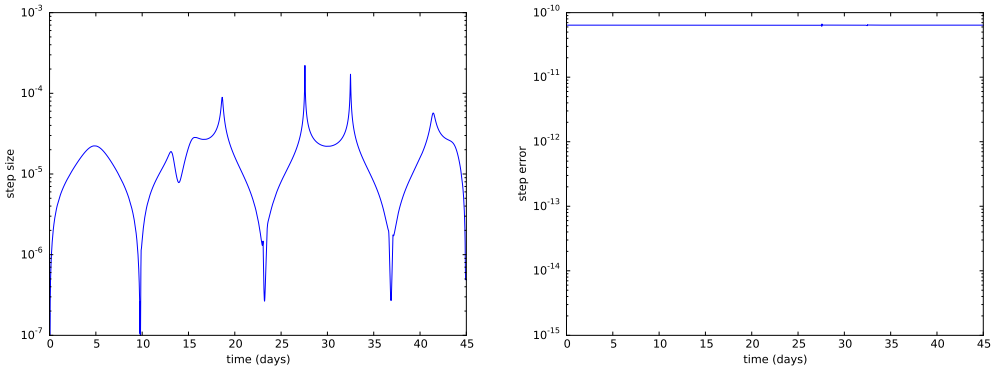
Figure 5.5: Position plots of short LETO. Earth in origin, moon orbit in grey, first half of trajectory is red, last half is black, green cross is first Lagrangepoint L_1.



(a) Exit from circular Earth parking orbit, 100 km altitide

(b) Entry to circular Moon orbit, 100 km altitude

Figure 5.6: Exit- and entry orbits of short LETO. The green band around the moon is the altitude range of ±10 km around 100 km that triggers a capture.



(a) Step size. **(b)** Error per step.

Figure 5.7: Step size and error per step of simulated short LETO.

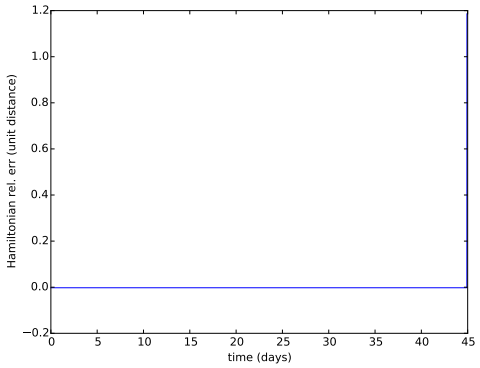


Figure 5.8: Hamiltonian H is conserved along the trajectory until Δv_{moon} is applied, as expected.

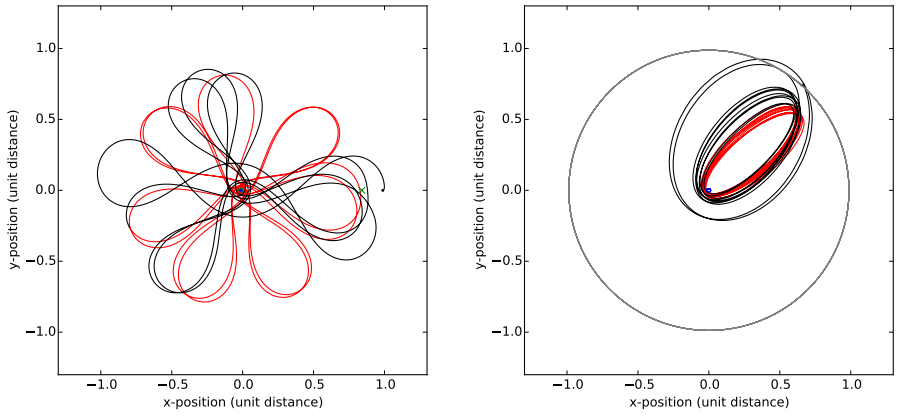
5.3.2 Long LETO

Listing 5.3 show the lowest Δv_{total} LETO trans-lunar injection

```
# ------------------------------------------------------------------------
duration = 195/unit_time
pos      = 3.794182930145708
ang      = 0.023901745288554
burn     = 3.090702702702703/unit_vel
x0       = -0.025645129237870
y0       = -0.010311570301966
px0      = 6.539303578815582
py0      = -8.449205705334165
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.090703 km/s
# dV(moon-capture) = 0.704113 km/s
# dV(total)        = 3.794815 km/s
# Flight-time      = 194.275487 days
# ------------------------------------------------------------------------
# Runtime = 0.98s
# Final position: 0.992404 0.001553
# Final impulse: 0.509302 -0.516927
# Final H: -2.730123
# Total runtime = 1.16s
# ------------------------------------------------------------------------
```

Listing 5.3: Long LETO. `pos` = angular difference with start angle (here $\theta = -3\pi/4$), `ang` = angle to velocity vector in Earth parking orbit, `burn` = Δv_{earth}, `(x0,y0,px0,py0)` are the initial conditions..

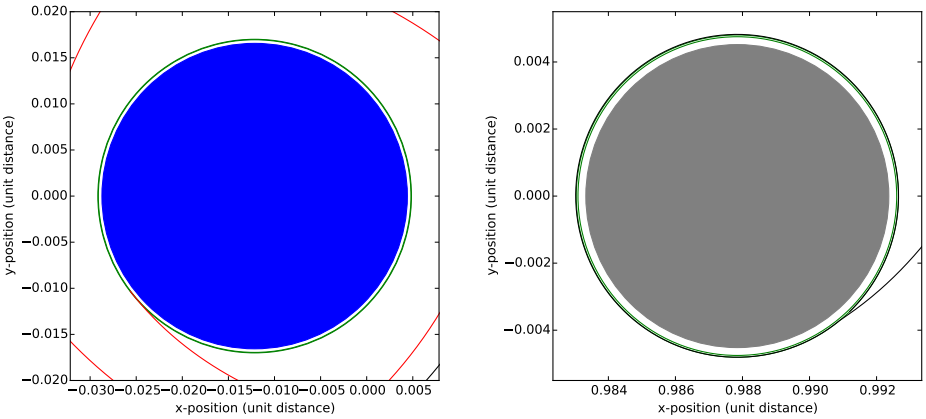
See figure 5.9 - 5.12 for trajectory position-, entry/exit- and Hamiltonian plots.



(a) Long LETO in co-rotating (x, y), Earth and Moon stationary.

(b) Long LETO in $(\mathcal{X}, \mathcal{Y})$, inertial system stationary on CM.

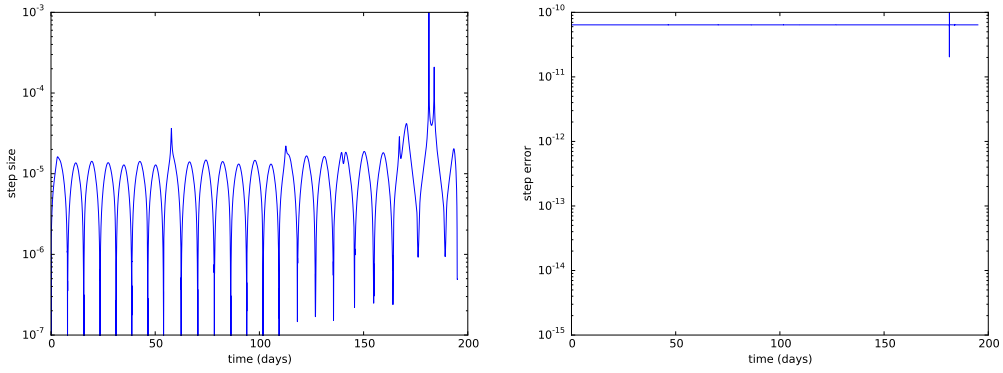
Figure 5.9: Position plots of long LETO. Earth in origin, moon orbit in grey, first half of trajectory is red, last half is black, green cross is first Lagrangepoint L_1.



(a) Exit from circular Earth parking orbit, 100 km altitide

(b) Entry to circular Moon orbit, 100 km altitude

Figure 5.10: Exit- and entry orbits of long LETO. The green band around the moon is the altitude range of ±10 km around 100 km that triggers a capture.



(a) Step size.

(b) Error per step.

Figure 5.11: Step size and error per step of simulated long LETO.

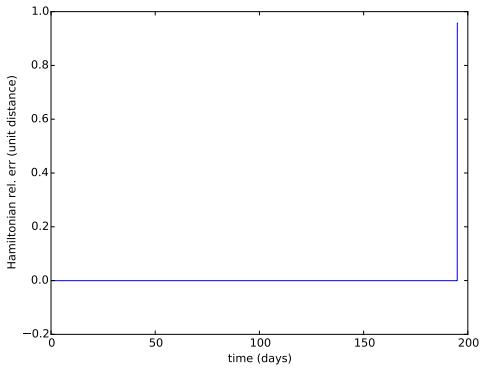


Figure 5.12: Hamiltonian H is conserved along the trajectory until Δv_{moon} is applied, as expected.

5.4 Results Summary and Discussion

Table 5.1 summerizes the Δv we found for the Hohmann, and the low-energy transfer orbts, short LETO and long LETO. In addition we've added a few more Hohmann (appendix D.3) to showcase the *Deltav* vs. flight time dynamics and to compare with the Apollo missions. In early simulations, the brute force approach was good enough to find a good Hohmann transfer, but not good enough for low-energy. After some code optimization, parallelization and running sufficiently many simulations on multi-core processors yielded good resulsts even for low-energy transfers in the end.

- The brute force method worked surpringsly well; we were able to find low-energy transfer orbits of lower Δv than any of the two cited from literature.
- Results for the 3-day Hohmann are with 2.5% of the Apollo Δv and flight time, which adds to the credibility of our problem model and implementation.
- It would possibly be advantagous to go up to a symplectic 4.-5.th order Runge-Kutta method, but was not implemented due to time-constraints. However we do see that the error can be kept small around 10^{-9} within a reasonable runtime, which can somewhat justify staying in 2nd order methods. In other words, we would not necessarily find better results, just a bit lower runtime since we could take fewer steps with a 4th order method.
- In further investigations it would be interesting to include:
    1. Patch together orbits in Lagrange points. This could be done by patching together brute forced trajectories from Earth to L_1 (integrated forwards) and L_1 to moon (integrated backwards).
    2. To from 2 to 3 dimensions.

3. Include the gravitational force from the Sun. This allows us to explot the dynamics of the Sun-Earth-Moon system. For example deliberately approach the moon from the far side (L_2) [Koo+01].
4. Exploit the dynamics of the Moon resonances as described in [TVB05].

Trajectory	Flight time	Δv_{total} (km/s)	Δv_{earth} (km/s)	Δv_{moon} (km/s)
Minimum	N/A	3.721	3.099	0.622
Long LETO	194 days	3.795	3.091	0.704
Belbruno-Miller	3 months	3.838	3.187	0.651
Topputo	8 months	3.895	3.265	0.630
Short LETO	41 days	3.896	3.127	0.769
Hohmann - long (sim)	4.3 days	3.912	3.111	0.801
Hohmann - (model)	5.0 days	3.946	3.144	0.802
Hohmann - medium (sim)	3.00 days	4.015	3.136	0.880
Apollo (Hohman)	3.05 days	4.115	3.048	1.067
Hohmann - short (sim)	1.00 days	6.823	3.809	3.014

Table 5.1: Summerized flight time and Δv for various trajectories, sorted by Δv_{total}. Belbruno and Topputo from [Juu08] are for orbits going from 167 km Earth parking to 100 km circular moon orbit. Apollo mission data form [NAS66] [NASa]..

CHAPTER 6

Conclusion

To make it clear, low-energy transfer orbits are not fit for manned missions. Those types is missions require optimization for flight time due to resource usage of maintaining the lives of the astronoughts.

We were able to find a low-energy transfer orbit to the moon with a flight time of 194 days and a Δv as low as 3795 km/s, which is better than any of the values we compared with from the literrature but still higher than the theoretical minimum of $\Delta v = 3721$ km/s, as it should be. We ensured that these results were numerically solid by implementing an adaptive Störmer–Verlet method that maintained a step-error of 10^{-9}. Furthermore the code was implemented in Python optimized for parallelization, enabling feasable brute force search for several kinds of orbits. We also found another with a more moderate flight time of 41 days and Δv of respectable 3896 km/s, which is very close to value found by Tuppoto [TVB05], but that had a flight time of 8 months. Furthermore a Hohmann transfer to the Moon was modelled and gave a prediction of $\Delta v = 9.946$ km/s with a flight time of 5.0 days. Our simulation found a Hohmann orbit of $\Delta v = 9.912$ km/s and flight time 4.3 days. We also found the 3-day Hohmann to be within 2.5% agreement with the real world 3-day trip to the moon realized by the Apollo missions, which validates our model and code implementation.

APPENDIX A

More details: Hohmann Transfer-Orbit to the Moon

In[12]:=
```
Quiet@Remove["`*"]
```

In[13]:=
```
SetOptions[SelectedNotebook[],
 PrintingStyleEnvironment → "Printout", ShowSyntaxStyles → True]
```

Setup

In[14]:=
```
(*Pariapsis: Point of least distance in elliptical orbit
  Apoapsis: Point of greatest distance in elliptical orbit*)
```

In[15]:=
```
μE = UnitConvert[G * PlanetData[Earth (planet), "Mass"], "km^3/s^2"]
```

Out[15]= $3.986 \times 10^5\ \mathrm{km^3/s^2}$

In[16]:=
```
μM = UnitConvert[G * PlanetaryMoonData[Moon (planetary moon), "Mass"], "km^3/s^2"]
```

Out[16]= $4.90 \times 10^3\ \mathrm{km^3/s^2}$

In[17]:=
```
(*Distance of Satellite's cicular orbit around Earth in Earth system*)
r1 = PlanetData[Earth (planet), "Radius"] + 160 km
```

Out[17]= 6527.4447 km

In[18]:=
```
(*Distance of Satellite's elliptical
 orbit around Earth at apoapsis in Earth system*)
r2 = PlanetaryMoonData[Moon (planetary moon), "AverageDistanceFromEarth"]
```

Out[18]= $3.850 \times 10^5\ \mathrm{km}$

In[19]:=
```
(*Distance of Satellite's circular orbit around Moon in Moon system*)
```

In[20]:=
```
r3 = PlanetaryMoonData[Moon (planetary moon), "Radius"] + 100 km
```

Out[20]= 1837.5 km

Hohmann transfer orbit to the Moon

Impulse A

In[21]:=
```
(*Speed in circular orbit, distance r1, at point A in Earth system*)
```

$$\text{vEarth} = \sqrt{\frac{\mu E}{r1}}$$

Out[21]= 7.814 km/s

In[22]:=
```
(*Speed in elliptical orbit, distance r1, at point A in Earth system*)
```

$$\text{vp} = \sqrt{\frac{\mu E}{r1}\,\frac{2\,r2}{r1 + r2}}$$

Out[22]= 10.96 km/s

In[23]:=
```
(*Necessary Δv in point A, a speed-up of 3.14 km/s*)
ΔvEarth = Sqrt[μE/r1] (Sqrt[2 r2/(r1 + r2)] - 1) // N(*vp-vEarth*)
```

Out[23]= 3.14423 km/s

Impulse B - neglecting circular motion around moon

In[24]:=
```
(*Speed in elliptical orbit, distance r2, at point B in Earth system*)
va = Sqrt[μE/r2 2 r1/(r1 + r2)]
```

Out[24]= 0.1858 km/s

In[25]:=
```
(*Speed in circular orbit, distance r2, at point B in Earth system
 (i.e. speed of the Moon in Earth system)*)
vNoMoon = Sqrt[μE/r2]
```

Out[25]= 1.017 km/s

In[26]:=

In[27]:=
```
(*Change of speed needed to "follow along with the moon",
a speed-up of 0.831 km/s*)
ΔvNoMoon = Sqrt[μE/r2] (1 - Sqrt[2 r1/(r1 + r2)]) // N(*v3NoMoon-v2B*)
```

Out[27]= 0.831692 km/s

Moon circular orbit correction

In[28]:=
```
(*However we can't follow the moon stationary along it's orbit
 since the Moon attracts the satellite and it would crash into the
 surface. We need to go into a circular orbit around the moon*)
```

In[29]:=
```
(*Speed v3 in circular orbit,
distance 100 km above the Moon surface, in Moon system*)
vMoon = Sqrt[(μM/r3)] // N
```

Out[29]= 1.63342 km/s

In[30]:=
```
(*We can choose to add or subtract this value to ΔvBNoMoon since
 we can go around in orbit both ways. Subtracting v3 from ΔvBNoMoon
 gives numerically the smallest Δv so that's what we'll do*)
```

In[31]:=
```
(*Change in speed needed to go from geocentric elliptical
 orbit to circular Moon orbit, a speed-down of 0.802 km/s*)
ΔvMoon = ΔvNoMoon - vMoon
```

Out[31]= −0.801723 km/s

Total

In[32]:=
```
(*Regarding fuel/energy cost,
we're interested in in how much we need to change the speed,
irregardless of directio. Total Δv for
 the whole trip: impulse at A plus impulse at B*)
ΔvTotal = Abs[ΔvEarth] + Abs[ΔvMoon] // N
```

Out[32]= 3.94595 km/s

In[33]:=
```
(*Result if we had neglected the circular motion
 around the moon at the end. Close, but not identical*)
ΔvTotalNoMoon = Abs[ΔvMoon] + Abs[ΔvNoMoon]
```

Out[33]= 1.63342 km/s

Flight time

In[34]:=
```
tHohmann = UnitConvert[1/2 Sqrt[4 π^2 * ((r1+r2)/2)^3 / μE], "Days"]
```

Out[34]= 4.99 days

Trying to minimize ΔvTotal by choosing appropriate moon oribt altitude

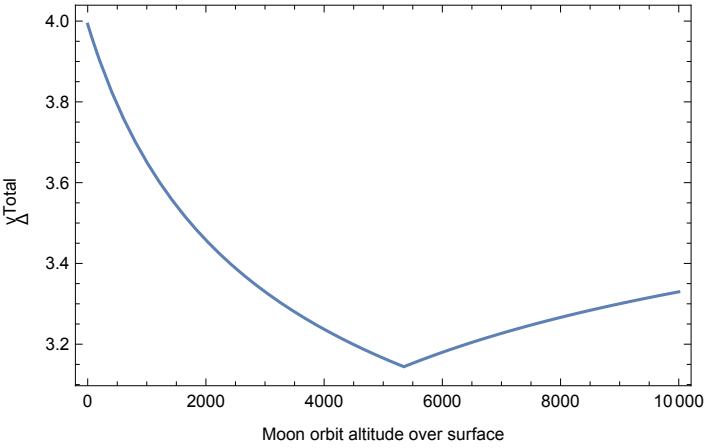
In[35]:=
```
RMoon = PlanetaryMoonData[Moon (planetary moon), "Radius"]
```

Out[35]= 1737.5 km

Plot

In[36]:=
```
Plot[Abs[Sqrt[μE/r1] (Sqrt[2 r2/(r1 + r2)] - 1)] + Abs[Sqrt[μE/r2] (1 - Sqrt[2 r1/(r1 + r2)])] - Sqrt[μM/(RMoon + altitude)] /.
  altitude → Quantity[x, "Kilometers"], {x, 0, 10^4}, Frame → True,
 FrameLabel → {"Moon orbit altitude over surface", "ΔvTotal"}]
```

Out[36]=



Moon orbit altitude that gives minimum ΔvTotal

In[37]:=
```
Solve[Sqrt[μE/r2] (1 - Sqrt[2 r1/(r1 + r2)]) - Sqrt[μM/(RMoon + altitude)] == 0, altitude] // N
```

Out[37]=
```
{{altitude → 5350.04 km}}
```

Minimum ΔvTotal

In[38]:=
```
ΔvTotalMin = Abs[Sqrt[μE/r1] (Sqrt[2 r2/(r1 + r2)] - 1)] + Abs[Sqrt[μE/r2] (1 - Sqrt[2 r1/(r1 + r2)])] - Sqrt[μM/(RMoon + x)] /.
  x → 5350.0393315174060602869`2.9098829878759 km] // N
```

Out[38]=
```
3.14423 km/s
```

In[39]:=
```
(*Yes but not practical*)
```

Moon hill sphere

In[40]:=
```
rHillMoon = N@(PlanetaryMoonData[Moon (planetary moon), "Mass"]/
   (3 * PlanetData[Earth (planet), "Mass"])) ^ (1 / 3)
```

Out[40]=
```
0.160053
```

In[41]:= `rHillMoon * PlanetaryMoonData[Moon (planetary moon), "AverageOrbitDistance"]`

Out[41]= 61 618.7 km

APPENDIX B

Detailed Derivation: Coordinate Velocity Transform

Recall $(\mathcal{X}, \mathcal{Y})$ is the inertial frame coordinates and (x, y) is the rotating frame coordinates 3.1. The goal is to go from $\mathcal{X}(x, y)$ and $\mathcal{Y}(x, y)$ to $(\dot{\mathcal{X}}^2 + \dot{\mathcal{Y}}^2)(x, y)$. This is used in (3.17) to obtain the kinetic energy $T(x, y)$ From equation (3.15) we had

$$\begin{cases} \mathcal{X} = x\cos\omega t - y\sin\omega t \\ \mathcal{Y} = x\sin\omega t + y\cos\omega t \end{cases}$$

Taking the time derivative we get

$$\begin{cases} \dot{\mathcal{X}} = \dot{x}\cos\omega t - \dot{y}\sin\omega t - \omega x\sin\omega t - \omega y\cos\omega t \\ \dot{\mathcal{Y}} = \dot{x}\sin\omega t + \dot{y}\cos\omega t + \omega x\cos\omega t - \omega y\sin\omega t \end{cases} \tag{B.1}$$

We'll take the squares, but first, to get a good overview of the terms, we note that $\dot{\mathcal{X}}^2$ matches

$$\begin{aligned} &(a-b-c-d)^2 \\ &= a^2 + b^2 + c^2 + d^2 \\ &- 2ab - 2ac - 2ad \\ &+ 2bc + 2bd + 2cd \end{aligned} \tag{B.2}$$

Now we take the squares, align the terms as in (B.2) and note what terms add and cancel in the sum $\mathcal{X}^2 + \mathcal{Y}^2$:

$$\begin{aligned} \dot{\mathcal{X}}^2 &= (\dot{x}\cos\omega t - \dot{y}\sin\omega t - \omega x\sin\omega t - \omega y\cos\omega t)(\dot{x}\cos\omega t - \dot{y}\sin\omega t - \omega x\sin\omega t - \omega y\cos\omega t) \\ &= \dot{x}^2\cos^2 t + \dot{y}^2\sin^2 t + \omega^2 x^2\sin^2 t + \omega^2 y^2\cos^2 t \\ &- 2\dot{x}\dot{y}\cos\omega t\sin\omega t - 2\omega x\dot{x}\cos\omega t\sin\omega t - 2\omega\dot{x}y\cos^2 t \\ &+ 2\omega x\dot{y}\sin^2 t + 2\omega y\dot{y}\cos\omega t\sin\omega t + 2\omega^2 xy\cos\omega t\sin\omega t \end{aligned} \tag{B.3}$$

Likewise $\dot{y}^2$ matches

$$\begin{aligned}&(a+b+c-d)^2\\&=a^2+b^2+c^2+d^2\\&+2ab+2ac-2ad\\&+2bc-2bd-2cd\end{aligned} \tag{B.4}$$

so we have

$$\begin{aligned}\dot{y}^2 &= (\dot{x}\sin\omega t+\dot{y}\cos\omega t+\omega x\cos\omega t-\omega y\sin\omega t)(\dot{x}\sin\omega t+\dot{y}\cos\omega t+\omega x\cos\omega t-\omega y\sin\omega t)\\&= \dot{x}^2\sin^2 t+\dot{y}^2\cos^2 t+\omega^2x^2\cos^2 t+\omega^2y^2\sin^2 t\\&+2\dot{x}\dot{y}\cos\omega t\sin\omega t+2\omega x\dot{x}\cos\omega t\sin\omega t-2\omega\dot{x}y\sin^2 t\\&+2\omega x\dot{y}\cos^2 t-2\omega y\dot{y}\cos\omega t\sin\omega t-2\omega^2xy\cos\omega t\sin\omega t\end{aligned} \tag{B.5}$$

Terms that add in $\mathcal{X}^2+\mathcal{Y}^2$

Terms that cancel in $\mathcal{X}^2+\mathcal{Y}^2$

(B.6)

Cancelling terms and repeated use of $\cos^2 t+\sin^2 t=1$ finally gives us

$$\begin{aligned}\dot{\mathcal{X}}^2+\dot{\mathcal{Y}}^2&=\dot{x}^2+\dot{y}^2+\omega^2x^2+\omega^2y^2-2\omega\dot{x}y+2\omega x\dot{y}\\&=(\dot{x}-\omega y)^2+(\dot{y}+\omega x)^2\end{aligned} \tag{B.7}$$

APPENDIX C

Non-Dimensionalization of Hamiltons Equations

We set:

$$t = k_t T \tag{C.1}$$
$$x = k_x X \tag{C.2}$$
$$y = k_y Y \tag{C.3}$$
$$p_x = k_{px} P_x \tag{C.4}$$
$$p_y = k_{py} P_Y, \tag{C.5}$$

with

$$k_t = \frac{1}{\omega} = \sqrt{\frac{(R+r)^3}{G(M+m)}} \tag{C.6}$$

$$k_x = k_y = R + r \tag{C.7}$$

$$k_{px} = k_{py} = m_s \sqrt{\frac{G(M+m)}{(R+r)}}. \tag{C.8}$$

So we get for $\dot{X}$

$$\frac{k_x}{k_t}\frac{\mathrm{d}X}{\mathrm{d}T} = \frac{k_{px}P_x}{m_s} + \omega k_y Y \tag{C.9}$$

$$\Leftrightarrow \qquad \frac{\mathrm{d}X}{\mathrm{d}T} = \frac{k_t k_{px}}{k_x}\frac{P_x}{m_s} + \omega k_t \frac{k_y}{k_x} Y \tag{C.10}$$

$$\Leftrightarrow \qquad \dot{X} = P_x + Y. \tag{C.11}$$

And for $\dot{Y}$ by symmetry

$$\dot{Y} = P_Y - X \ , \tag{C.12}$$

For $\dot{P}_x$ we get

$$\frac{\mathrm{d}P_x}{\mathrm{d}T} = \frac{k_{py}}{k_{px}} k_t \omega P_y - \frac{k_t}{k_{px}} G m_s \left[\frac{M(k_x X + R)}{\left((k_x X + R)^2 + k_y^2 Y^2\right)^{3/2}} + \frac{m(k_x X - r)}{\left((k_x X - r)^2 + k_y^2 Y^2\right)^{3/2}} \right] \tag{C.13}$$

$$= P_y - \left(\sqrt{\frac{(R+r)^3}{G(M+m)}} \right) \left(\frac{1}{m_s} \sqrt{\frac{(R+r)}{G(M+m)}} \right) G m_s \cdot \tag{C.14}$$

$$\left[\frac{M[(R+r)X + R]}{[((R+r)X + R)^2 + (R+r)^2 Y^2]^{3/2}} + \frac{m[(R+r)X - r]}{[((R+r)X - r)^2 + (R+r)^2 Y^2]^{3/2}} \right]$$

$$= P_y - \frac{(R+r)^2}{M+m} \cdot \tag{C.15}$$

$$\left[\frac{M(R+r)\left(X + \dfrac{R}{R+r}\right)}{(R+r)^3 \left[\left(X + \dfrac{R}{R+r}\right)^2 + Y^2\right]^{3/2}} + \frac{m(R+r)\left(X - \dfrac{r}{R+r}\right)}{(R+r)^3 \left[\left(X - \dfrac{r}{R+r}\right)^2 + Y^2\right]^{3/2}} \right]$$

$$= P_y - \left[\frac{\dfrac{M}{M+m}\left(X + \dfrac{R}{R+r}\right)}{\left(\left(X + \dfrac{R}{R+r}\right)^2 + Y^2\right)^{3/2}} + \frac{\dfrac{m}{M+m}\left(X - \dfrac{r}{R+r}\right)}{\left(\left(X - \dfrac{r}{R+r}\right)^2 + Y^2\right)^{3/2}} \right], \tag{C.16}$$

where $\dfrac{M}{M+m}$ and $\dfrac{R}{R+r}$ are dimensionless parameters. There is a simple relation between the two given by the equation for the center of mass:

$$M\mathbf{R} + m\mathbf{r} = \mathbf{0} \tag{C.17}$$

$$\Rightarrow \qquad \frac{m}{M+m} = \frac{R}{R+r} = k \tag{C.18}$$

$$\Leftrightarrow \qquad \frac{M}{M+m} = \frac{r}{R+r} = 1 - k \tag{C.19}$$

With that in mind we finally get

$$\dot{P}_x = P_y - \frac{(1-k)(X+k)}{((X+k)^2+Y^2)^{3/2}} - \frac{k(X-1-k)}{((X-1-k)^2+Y^2)^{3/2}} \tag{C.20}$$

by symmetry for P_y we get

$$\dot{P}_y = -P_x - \frac{(1-k)Y}{((X+k)^2+Y^2)^{3/2}} - \frac{kY}{((X-1-k)^2+Y^2)^{3/2}} \tag{C.21}$$

APPENDIX D

Miscellaneous

D.1 Reduced 3-body Problem Symplectic Euler x_{i+1}, y_{i+1}Mathematica)

```
In[89]:= eqn1 = x1 == (px1 + y1) h + x0;

In[90]:= eqn2 = y1 == (py1 - x1) h + y0;

In[91]:= Solve[{eqn1, eqn2}, {x1, y1}] // FullSimplify
```

$$\text{Out[91]=}\ \left\{\left\{\text{x1} \to \frac{\text{x0} + \text{h}\ (\text{px1} + \text{h py1} + \text{y0})}{1 + \text{h}^2},\ \text{y1} \to \frac{-\text{h}\ (\text{h px1} - \text{py1} + \text{x0}) + \text{y0}}{1 + \text{h}^2}\right\}\right\}$$

D.2 Implicit Euler algorithm

All new values $(x, y, p_x, p_y)_{i+1}$ refer to the same unknown values in the same timestep $(x, y, p_x, p_y)_{i+1}$.

$$x_{i+1} = (p_{x,i+1} + y_{i+1})h + x_{i+1} \tag{D.1}$$

$$y_{i+1} = (p_{y,i+1} - x_{i+1})h + y_{i+1} \tag{D.2}$$

$$p_{x,i+1} = \left(p_{y,i+1} - \frac{(1-k)(k+x_{i+1})}{((k+x_{i+1})^2 + y_{i+1}^2)^{3/2}} + \frac{k(1-k-x_{i+1})}{((1-k-x_{i+1})^2 + y_{i+1}^2)^{3/2}}\right)h + p_{x,i} \tag{D.3}$$

$$p_{y,i+1} = \left(-p_{x,i+1} - \frac{(1-k)y_{i+1}}{((k+x_{i+1})^2 + y_{i+1}^2)^{3/2}} - \frac{ky_{i+1}}{((1-k-x_{i+1})^2 + y_{i+1}^2)^{3/2}}\right)h + p_{y,i} \tag{D.4}$$

D.3 Medium and Short Hohmann orbits

```
# ------------------------------------------------------------------------
duration = 3/unit_time
pos      = -2.272183066647597
ang      = -0.075821466029764
```

```
burn      = 3.135519748743719/unit_vel
x0        = -0.023110975767437
y0        = -0.012972499765730
px0       = 8.032228991913522
py0       = -7.100537706154897
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.135520 km/s
# dV(moon-capture) = 0.879826 km/s
# dV(total)        = 4.015346 km/s
# Flight-time      = 2.999939 days
# ------------------------------------------------------------------------
```

Listing D.1: Medium duration Hohmann.

```
# ------------------------------------------------------------------------
duration = 2/unit_time
pos       = -2.277784119105456
ang       = 0.046759232345463
burn      = 3.809267777777778/unit_vel
x0        = -0.023183463163465
y0        = -0.012910923775798
px0       = 8.760501647975921
py0       = -7.267405934327472
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.809268 km/s
# dV(moon-capture) = 3.014142 km/s
# dV(total)        = 6.823410 km/s
# Flight-time      = 1.000007 days
# ------------------------------------------------------------------------
```

Listing D.2: Fast duration Hohmann.

APPENDIX E

Python Code for Restricted 3-Body Problem

E.1 Main Script

Calling all the other functions, initiating searches, and storing interesting orbits that has been found (filename: testing.py).

```
import os
import time
from math import pi,cos,sin
import numpy as np
import matplotlib.pyplot as plt
from const import *
import reduced3body as r3b

try:
    threads = int(os.environ["OMP_NUM_THREADS"])
except KeyError:
    threads = 1

runtime = time.time()

# Precalculated initial Conditions
#demo = 'earth_orbit'
#demo = 'lunar_orbit'
#demo = 'hohmann'
#demo = '3_day_hohmann'
#demo = '1_day_hohmann'
#demo = 'reverse_hohmann'
demo = 'low_energy_short'
#demo = 'low_energy_long'
#demo = 'earth_to_L1'

# or Search for trajectories
#demo = 'search_hohmann'
#demo = 'search_low_energy'
#demo = 'search_low_energy_parts8'
#demo = 'search_refine'

n = 1000000
```

```
34
35 # Set coordinates
36 if demo == 'earth_orbit':
37     duration = (2.0*pi*leo_orbit/leo_orbit_vel)/(unit_time*day)
38     r = leo_orbit/unit_len
39     v = 0.99732*leo_orbit_vel/unit_vel
40     theta = 0
41     x = r*cos(theta)
42     y = r*sin(theta)
43     vx = -v*y/r
44     vy = v*x/r
45     pos = 0
46     ang = 0
47     burn = 0
48     x0 = earth_pos_x+x
49     y0 = y
50     px0 = vx-y0
51     py0 = vy+x0
52 elif demo == 'lunar_orbit':
53     duration = (2.0*pi*lunar_orbit/lunar_orbit_vel)/(unit_time*day)
54     r = lunar_orbit/unit_len
55     v = 0.99732*lunar_orbit_vel/unit_vel
56     theta = 0
57     x = r*cos(theta)
58     y = r*sin(theta)
59     vx = -v*y/r
60     vy = v*x/r
61     pos = 0
62     ang = 0
63     burn = 0
64     x0 = moon_pos_x+x
65     y0 = y
66     px0 = vx-y0
67     py0 = vy+x0
68 elif demo == 'hohmann':
69     #demo = 'search_refine'
70 # -----------------------------------------------------------------------
71     duration = 5/unit_time
72     pos      = -2.086814820119193
73     ang      = -0.000122173047640
74     burn     = 3.111181716545691/unit_vel
75     x0       = -0.020532317163607
76     y0       = -0.014769797663479
77     px0      = 9.302400979050308
78     py0      = -5.289712560652044
79 # -----------------------------------------------------------------------
80 # dV(earth-escape) = 3.111182 km/s
81 # dV(moon-capture) = 0.800682 km/s
82 # dV(total)        = 3.911863 km/s
83 # Flight-time      = 4.300078 days
84 # -----------------------------------------------------------------------
85 elif demo == 'reverse_hohmann':
```

```
86 # ------------------------------------------------------------------------
87     duration = 4/unit_time
88     pos      = -2.282942228154665
89     ang      = 0.000000000000000
90     burn     = -3.149483130653266/unit_vel
91     x0       = -0.023249912090507
92     y0       = -0.012853859046429
93     px0      = -8.098481905534163
94     py0      = 6.978997254692934
95 # ------------------------------------------------------------------------
96 # dV(earth-escape) = 3.149483 km/s
97 # dV(moon-capture) = 0.968488 km/s
98 # dV(total)        = 4.117971 km/s
99 # Flight-time      = 3.875497 days
100 # ------------------------------------------------------------------------
101 elif demo == 'low_energy_long':
102 # ------------------------------------------------------------------------
103     duration = 195/unit_time
104     pos      = 3.794182930145708
105     ang      = 0.023901745288554
106     burn     = 3.090702702702703/unit_vel
107     x0       = -0.025645129237870
108     y0       = -0.010311570301966
109     px0      = 6.539303578815582
110     py0      = -8.449205705334165
111 # ------------------------------------------------------------------------
112 # dV(earth-escape) = 3.090703 km/s
113 # dV(moon-capture) = 0.704114 km/s
114 # dV(total)        = 3.794816 km/s
115 # Flight-time      = 194.275480 days
116 # ------------------------------------------------------------------------
117 # ------------------------------------------------------------------------
118     #demo = 'search_refine'
119 #    duration = 195/unit_time
120 #    pos      = 3.794182930145708
121 #    ang      = 0.023901745288554
122 #    burn     = 3.090702702702703/unit_vel
123 #    x0       = -0.025645129237870
124 #    y0       = -0.010311570301966
125 #    px0      = 6.539303578815583
126 #    py0      = -8.449205705334164
127 # ------------------------------------------------------------------------
128 # dV(earth-escape) = 3.090703 km/s
129 # dV(moon-capture) = 0.704114 km/s
130 # dV(total)        = 3.794817 km/s
131 # Flight-time      = 194.275480 days
132 # ------------------------------------------------------------------------
133 elif demo == 'low_energy_short':
134     #demo = 'search_refine'
135 # ------------------------------------------------------------------------
136     duration = 41/unit_time
137     pos      = -0.138042744751570
```

```
    ang       = -0.144259374836607
    burn      = 3.127288444444444/unit_vel
    x0        = 0.004665728429046
    y0        = -0.002336647636098
    px0       = 1.904735175752430
    py0       = 10.504985512873279
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.127288 km/s
# dV(moon-capture) = 0.768534 km/s
# dV(total)        = 3.895822 km/s
# Flight-time      = 40.617871 days
# ------------------------------------------------------------------------
elif demo == '3_day_hohmann':
    #demo = 'search_refine'
# ------------------------------------------------------------------------
    duration = 3/unit_time
    pos       = -2.272183066647597
    ang       = -0.075821466029764
    burn      = 3.135519748743719/unit_vel
    x0        = -0.023110975767437
    y0        = -0.012972499765730
    px0       = 8.032228991913522
    py0       = -7.100537706154897
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.135520 km/s
# dV(moon-capture) = 0.879826 km/s
# dV(total)        = 4.015346 km/s
# Flight-time      = 2.999939 days
# ------------------------------------------------------------------------
elif demo == '1_day_hohmann':
    #demo = 'search_refine'
    duration = 1/unit_time
    pos       = -2.277654673852600
    ang       = 0.047996554429844
    burn      = 3.810000000000000/unit_vel
    x0        = -0.023181791813268
    y0        = -0.012912351430812
    px0       = 8.764829132987316
    py0       = -7.263069305305378
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.810000 km/s
# dV(moon-capture) = 3.319455 km/s
# dV(total)        = 7.129455 km/s
# Flight-time      = 0.997234 days
# ------------------------------------------------------------------------
elif demo == 'earth_to_L1':
    demo = 'search_refine'
# ------------------------------------------------------------------------
    duration = 191/unit_time
    pos       = 2.843432239707429
    ang       = 0.000000000000000
    burn      = 3.091851851851852/unit_vel
```

```
    x0        = -0.028385246222264
    y0        = 0.004988337832881
    px0       = -3.136296304910217
    py0       = -10.217405925499762
# ------------------------------------------------------------------------
# dV(earth-escape) = 3.091852 km/s
# dV(at L1)        = 0.676226 km/s
# dV(total)        = 3.768078 km/s
# Flight-time      = 190.001881 days
# ------------------------------------------------------------------------

#################### FUNCTION CALLS ####################

if demo == 'search_hohmann':
    tlist,xlist,ylist,pxlist,pylist,errlist,hlist = r3b.hohmann(threads,n)
elif demo == 'search_low_energy':
    tlist,xlist,ylist,pxlist,pylist,errlist,hlist = r3b.low_energy(threads,n
        )
elif demo == 'search_low_energy_parts8':
    tlist,xlist,ylist,pxlist,pylist,errlist,hlist = r3b.low_energy_parts8(
        threads,n)
elif demo == 'search_refine':
    tlist,xlist,ylist,pxlist,pylist,errlist,hlist = r3b.refine(threads,n,
        duration,pos,ang,burn,x0,y0,px0,py0)
else:
    tlist,xlist,ylist,pxlist,pylist,errlist,hlist = r3b.trajectory(n,
        duration,pos,ang,burn,x0,y0,px0,py0)
Hlist = pxlist**2/2 + pylist**2/2 + ylist*pxlist - xlist*pylist - (1-mu)/np.
    sqrt(np.power(mu+xlist,2)+np.power(ylist,2)) - mu/np.sqrt(np.power(1-mu-
    xlist,2)+np.power(ylist,2))
print("# Final position: %f %f" %(xlist[n-1],ylist[n-1]))
print("# Final impulse: %f %f" % (pxlist[n-1],pylist[n-1]))
print("# Final H: %f" % (Hlist[n-1]))
runtime = time.time()-runtime
print("# Total runtime = %3.2fs" % (runtime))
print("#
    ------------------------------------------------------------------------
    ")
print("# --- Done with FUNCTION CALLS")
#exit()

#################### PLOTS: POSITION ####################

xlist1 = xlist[:n/2]
ylist1 = ylist[:n/2]
xlist2 = xlist[n/2:]
ylist2 = ylist[n/2:]

Xlist1 = xlist[:n/2]*np.cos(tlist[:n/2]) - ylist[:n/2]*np.sin(tlist[:n/2])
Ylist1 = xlist[:n/2]*np.sin(tlist[:n/2]) + ylist[:n/2]*np.cos(tlist[:n/2])
Xlist2 = xlist[n/2:]*np.cos(tlist[n/2:]) - ylist[n/2:]*np.sin(tlist[n/2:])
Ylist2 = xlist[n/2:]*np.sin(tlist[n/2:]) + ylist[n/2:]*np.cos(tlist[n/2:])
```

```
234
235 Xlist_earth = earth_pos_x*np.cos(tlist)
236 Ylist_earth = -earth_pos_x*np.sin(tlist)
237
238 Xlist_moon = moon_pos_x*np.cos(tlist)
239 Ylist_moon = moon_pos_x*np.sin(tlist)
240
241 # Rel. err
242 plt.figure()
243 plt.plot(tlist*unit_time, errlist)
244 plt.xlabel("time (days)")
245 plt.ylabel("step error")
246 plt.yscale('log')
247
248 # Step sizes
249 plt.figure()
250 plt.plot(tlist*unit_time, hlist)
251 plt.xlabel("time (days)")
252 plt.ylabel("step size")
253 plt.yscale('log')
254
255 # Total energy
256 havg = np.sum(Hlist)/n
257 hrelerr = (Hlist-havg)/havg
258 plt.figure()
259 plt.plot(tlist*unit_time, hrelerr)
260 plt.xlabel("time (days)")
261 plt.ylabel("Hamiltonian rel. err (arbitrary units)")
262
263 # Zoom earth
264 xlim = 0.02
265 ylim = 0.02
266 xmin = earth_pos_x-xlim
267 xmax = earth_pos_x+xlim
268 ymin = -ylim
269 ymax = ylim
270 plt.figure()
271 earth=plt.Circle((earth_pos_x,0),earth_radius/unit_len,color='blue')
272 earthorbit1=plt.Circle((earth_pos_x,0),(leo_orbit-orbit_range)/unit_len,
      color='g',fill=False)
273 earthorbit2=plt.Circle((earth_pos_x,0),(leo_orbit+orbit_range)/unit_len,
      color='g',fill=False)
274 plt.gcf().gca().add_artist(earth)
275 plt.gcf().gca().add_artist(earthorbit1)
276 plt.gcf().gca().add_artist(earthorbit2)
277 plt.plot(xlist1,ylist1,'r-')
278 plt.plot(xlist2,ylist2,'k-')
279 plt.xlim(xmin,xmax)
280 plt.ylim(ymin,ymax)
281 plt.gca().set_aspect('equal', adjustable='box')
282 plt.xlabel("x-position (arbitrary units)")
283 plt.ylabel("y-position (arbitrary units)")
```

```
284
285 # Zoom moon
286 xlim = 0.0055
287 ylim = 0.0055
288 xmin = moon_pos_x-xlim
289 xmax = moon_pos_x+xlim
290 ymin = -ylim
291 ymax = ylim
292 plt.figure()
293 moon=plt.Circle((moon_pos_x,0),moon_radius/unit_len,color='grey')
294 moonorbit1=plt.Circle((moon_pos_x,0),(lunar_orbit-orbit_range)/unit_len,
        color='g',fill=False)
295 moonorbit2=plt.Circle((moon_pos_x,0),(lunar_orbit+orbit_range)/unit_len,
        color='g',fill=False)
296 plt.gcf().gca().add_artist(moon)
297 plt.gcf().gca().add_artist(moonorbit1)
298 plt.gcf().gca().add_artist(moonorbit2)
299 plt.plot(xlist1,ylist1,'r-')
300 plt.plot(xlist2,ylist2,'k-')
301 plt.xlim(xmin,xmax)
302 plt.ylim(ymin,ymax)
303 plt.gca().set_aspect('equal', adjustable='box')
304 plt.xlabel("x-position (arbitrary units)")
305 plt.ylabel("y-position (arbitrary units)")
306
307 # View center of mass
308 xlim = 1.3
309 ylim = 1.3
310 xmin = -xlim
311 xmax = xlim
312 ymin = -ylim
313 ymax = ylim
314
315 # Position plot (X,Y)
316 plt.figure()
317 plt.plot(Xlist1,Ylist1,'r')
318 plt.plot(Xlist2,Ylist2,'k')
319 plt.plot(Xlist_earth, Ylist_earth, 'blue')
320 plt.plot(Xlist_moon, Ylist_moon, 'grey')
321 plt.xlim(xmin,xmax)
322 plt.ylim(ymin,ymax)
323 plt.gca().set_aspect('equal', adjustable='box')
324 plt.xlabel("x-position (arbitrary units)")
325 plt.ylabel("y-position (arbitrary units)")
326
327 # Position plot (x,y)
328 plt.figure()
329 plt.plot(xlist1,ylist1,'r-')
330 plt.plot(xlist2,ylist2,'k-')
331 earth=plt.Circle((earth_pos_x,0),earth_radius/unit_len,color='blue')
332 earthorbit1=plt.Circle((earth_pos_x,0),(leo_orbit-orbit_range)/unit_len,
        color='g',fill=False)
```

```
earthorbit2=plt.Circle((earth_pos_x,0),(leo_orbit+orbit_range)/unit_len,
    color='g',fill=False)
moon=plt.Circle((moon_pos_x,0),moon_radius/unit_len,color='grey')
moonorbit1=plt.Circle((moon_pos_x,0),(lunar_orbit-orbit_range)/unit_len,
    color='g',fill=False)
moonorbit2=plt.Circle((moon_pos_x,0),(lunar_orbit+orbit_range)/unit_len,
    color='g',fill=False)
plt.gcf().gca().add_artist(earth)
plt.gcf().gca().add_artist(earthorbit1)
plt.gcf().gca().add_artist(earthorbit2)
plt.gcf().gca().add_artist(moon)
plt.gcf().gca().add_artist(moonorbit1)
plt.gcf().gca().add_artist(moonorbit2)
plt.plot(L1_pos_x,0,'gx')
plt.xlim(xmin,xmax)
plt.ylim(ymin,ymax)
plt.gca().set_aspect('equal', adjustable='box')
plt.xlabel("x-position (arbitrary units)")
plt.ylabel("y-position (arbitrary units)")
#plt.savefig('fig/r3b/r3b_y(x)_euler_symplectic.pdf',bbox_inches='tight')
plt.show()
plt.close()
print("# --- Done with PLOTS")
```

.

E.2 Constants

Constants defined for the rest of the program to use (filename: const.py)

```
from math import pi,sqrt,pow

# Table constants
earth_moon_distance = 384400.0 # km
moon_orbit_duration = 27.322 # days
earth_radius = 6367.4447 # km
earth_mass = 5.9721986e24
moon_mass = 7.34767309e22
G = 6.67384e-11 # m^3 kg^-1 s^-2
moon_radius = 1737.1 # km
leo_orbit = 160.0+earth_radius # km
lunar_orbit = 100.0+moon_radius # km
orbit_range = 10.0 # km
day = 24.0*3600.0 # s

# Units
unit_len = earth_moon_distance # km
unit_time = moon_orbit_duration/(2.0*pi) # days
unit_vel = unit_len/(unit_time*day) # km/s

```

```
# Gravitational constants
mu = moon_mass/(earth_mass+moon_mass) # 1

# Calculation constants
leo_orbit_vel = sqrt(G*earth_mass/(leo_orbit*1000.0))/1000.0 # km/s
lunar_orbit_vel = sqrt(G*moon_mass/(lunar_orbit*1000.0))/1000.0 # km/s
moon_pos_x = 1-mu
earth_pos_x = -mu
L1_pos_x = (1-pow(mu/3.0,1.0/3.0))
```

.

E.3 Plotter and Trajectory Finders

trajectory plot trajectories, hohmann and low_energy finds transfer orbits (filename: reduced3body.py - most of it).

```
import time
from math import pi,sqrt
import numpy as np
from const import *
from numbapro import *
from search import search_mt, search, print_search_results
from symplectic import symplectic

# **BRUGER IKKE pos, ang, burn til noget, kun til print
def trajectory(n,duration,pos,ang,burn,x0,y0,px0,py0):
    """Integrate trajectory for the reduced 3-body problem.

    Args:
        n (int): Positions stored.
        duration (float): Time duration of simulation.
        x0 (float): Initial x-coordinate
        y0 (float): Initial y-coordinate
        px0 (float): Initial generalized x-momentum
        py0 (float): Initial generalized y-momentum

    Returns:
        Tuple of time-, x-, y-, px- and py lists.

    """
    print("# Running trajectory.")

    # Initialize arrays
    tlist = np.linspace(0,duration,n)
    xlist = np.zeros(n)
    ylist = np.zeros(n)
    pxlist = np.zeros(n)
    pylist = np.zeros(n)
    errlist = np.zeros(n)
```

```
34      hlist = np.zeros(n)
35      info = np.zeros(2)
36
37      # Find orbits
38      runtime = time.time()
39      status = symplectic(n,duration,x0,y0,px0,py0,xlist,ylist,pxlist,pylist,
            errlist,hlist,info)
40      runtime = time.time()-runtime
41
42      # Display result
43      print_search_results(status,pos,ang,burn,x0,y0,px0,py0,info[0],info[1])
44      print("# Runtime = %3.2fs" % (runtime))
45      return tlist,xlist,ylist,pxlist,pylist,errlist,hlist
46
47  def hohmann(threads,n):
48      """Finding Hohmann trajectory for the reduced 3-body problem.
49
50      Args:
51          n (int): Positions stored.
52
53      Returns:
54          Tuple of time-, x-, y-, px- and py lists.
55
56      """
57
58      print("# Running Hohmann.")
59
60      # Hohmann trajectory < 6 days
61      duration = 6/unit_time
62      best_total_dv = 1e9
63      positions = 100
64      angles = 1
65      burns = 200
66      pos = -3*pi/4
67      ang = 0
68      burn = 3.11/unit_vel # Forward Hohmann
69      #burn_low = -3.14/unit_vel # Reverse Hohmann
70
71      # Super fast Hohmann trajectory < 1 days
72      #duration = 3/unit_time
73      #best_total_dv = 1e9
74      #positions = 10
75      #angles = 10
76      #burns = 200
77      #pos = -3*pi/4
78      #ang = 0
79      #burn = 3.7/unit_vel # Forward Hohmann
80
81      pos_range = pi/4
82      ang_range = pi/8
83      burn_range = 0.1/unit_vel
84
```

```
85      # Start search
86      searches = 0
87      max_searches = 5
88      while searches < max_searches:
89          runtime = time.time()
90          searches += 1
91          print("############### Search %i ###############" % (searches))
92          print("# pos              = %f" % (pos))
93          print("# ang              = %f" % (ang))
94          print("# burn             = %f" % (burn))
95          pos_low = pos-pos_range
96          pos_high = pos+pos_range
97          ang_low = ang-ang_range
98          ang_high = ang+ang_range
99          burn_low = burn-burn_range
100         burn_high = burn+burn_range
101         stat,pos,ang,burn,x0,y0,px0,py0,dv,toa = search_mt(threads,1,
                duration,positions,angles,burns,pos_low,pos_high,ang_low,ang_high
                ,burn_low,burn_high)
102
103         if stat < 0:
104             total_dv = abs(burn)+dv
105             if best_total_dv > total_dv:
106                 best_total_dv = total_dv
107                 best_stat = stat
108                 best_pos = pos
109                 best_ang = ang
110                 best_burn = burn
111                 best_x0 = x0
112                 best_y0 = y0
113                 best_px0 = px0
114                 best_py0 = py0
115                 best_dv = dv
116                 best_toa = toa
117             else:
118                 break
119
120         pos_range *= 0.1
121         ang_range *= 0.1
122         burn_range *= 0.1
123
124         runtime = time.time()-runtime
125         print("# Search runtime   = %3.2fs" % (runtime))
126
127     # Print best result
128     print("################ Best ################")
129     print("# Best dV(total)   = %f km/s" % (best_total_dv*unit_vel))
130     print_search_results(best_stat,best_pos,best_ang,best_burn,best_x0,
            best_y0,best_px0,best_py0,best_dv,best_toa)
131
132     # Initialize arrays
133     tlist = np.linspace(0,duration,n)
```

```
134     xlist = np.zeros(n)
135     ylist = np.zeros(n)
136     pxlist = np.zeros(n)
137     pylist = np.zeros(n)
138     errlist = np.zeros(n)
139     hlist = np.zeros(n)
140     info = np.zeros(2)
141 
142     # Do trajectory
143     duration = 10/unit_time
144     status = symplectic(n,duration,x0,y0,px0,py0,xlist,ylist,pxlist,pylist,
            errlist,hlist,info)
145 
146     return tlist,xlist,ylist,pxlist,pylist,errlist,hlist
147 
148 def low_energy(threads,n):
149     """Finding low energy transfer trajectory for the reduced 3-body problem
            .
150 
151     Args:
152         n (int): Positions stored.
153 
154     Returns:
155         Tuple of time-, x-, y-, px- and py lists.
156 
157     """
158 
159     print("# Running low_energy.")
160 
161     # Low-energy trajectory < 200 days
162     duration = 200/unit_time
163     best_total_dv = 1e9
164     positions = 100
165     angles = 1
166     burns = 200
167     pos = -3*pi/4
168     ang = 0
169     burn = 3.12/unit_vel
170     pos_range = pi
171     ang_range = 0
172     burn_range = 0.01/unit_vel
173 
174     # Start search
175     searches = 0
176     max_searches = 1
177     while searches < max_searches:
178         runtime = time.time()
179         searches += 1
180         print("############### Search %i ###############" % (searches))
181         print("# pos               = %f" % (pos))
182         print("# ang               = %f" % (ang))
183         print("# burn              = %f" % (burn))
```

```
184         pos_low = pos-pos_range
185         pos_high = pos+pos_range
186         ang_low = ang-ang_range
187         ang_high = ang+ang_range
188         burn_low = burn-burn_range
189         burn_high = burn+burn_range
190         stat,pos,ang,burn,x0,y0,px0,py0,dv,toa = search_mt(threads,1,
                duration,positions,angles,burns,pos_low,pos_high,ang_low,ang_high
                ,burn_low,burn_high)
191
192         if stat < 0:
193             total_dv = abs(burn)+dv
194             if best_total_dv > total_dv:
195                 best_total_dv = total_dv
196                 best_stat = stat
197                 best_pos = pos
198                 best_ang = ang
199                 best_burn = burn
200                 best_x0 = x0
201                 best_y0 = y0
202                 best_px0 = px0
203                 best_py0 = py0
204                 best_dv = dv
205                 best_toa = toa
206             else:
207                 break
208
209         pos_range *= 0.1
210         ang_range *= 0.1
211         burn_range *= 0.1
212
213         runtime = time.time()-runtime
214         print("# Search runtime    = %3.2fs" % (runtime))
215
216     # Initialize arrays
217     tlist = np.linspace(0,duration,n)
218     xlist = np.zeros(n)
219     ylist = np.zeros(n)
220     pxlist = np.zeros(n)
221     pylist = np.zeros(n)
222     errlist = np.zeros(n)
223     hlist = np.zeros(n)
224     info = np.zeros(2)
225
226     # Do trajectory
227     duration = toa+(2.0*pi*lunar_orbit/lunar_orbit_vel)/(unit_time*day)
228     status = symplectic(n,duration,x0,y0,px0,py0,xlist,ylist,pxlist,pylist,
            errlist,hlist,info)
229     exit()
230     return tlist,xlist,ylist,pxlist,pylist,errlist,hlist
```

.

E.4 Search Helper Functions

Search helper functions, also implements all the paralellization (filename: search.py).

```
"""
Brute-force search Module for reduced 3-body solver
===================================================

"""

from __future__ import print_function
import sys
import time
from math import ceil
import numpy as np
from multiprocessing import Pool
from const import *
from symplectic import symplectic

def print_search_results(stat,pos,ang,burn,x0,y0,px0,py0,dv,toa):
    print("#
       ------------------------------------------------------------------------
       ")
    print("duration = %i/unit_time" % (max(1,int(ceil(toa*unit_time)))))
    print("pos      = %.15lf" % (pos))
    print("ang      = %.15lf" % (ang))
    print("burn     = %.15lf/unit_vel" % (burn*unit_vel))
    print("x0       = %.15lf" % (x0))
    print("y0       = %.15lf" % (y0))
    print("px0      = %.15lf" % (px0))
    print("py0      = %.15lf" % (py0))
    print("#
       ------------------------------------------------------------------------
       ")
    print("# dV(earth-escape) = %f km/s" % (abs(burn)*unit_vel))
    if stat > 0 and stat < 100:
        print("# Min moon distance= %f" % (stat))
    elif stat < 0:
        print("# dV(moon-capture) = %f km/s" % (dv*unit_vel))
        print("# dV(total)        = %f km/s" % ((abs(burn)+dv)*unit_vel))
        print("# Flight-time      = %f days" % (toa*unit_time))
    else:
        print("# Crashed on earth!")
    print("#
       ------------------------------------------------------------------------
       ")
    sys.stdout.flush()

def search(thread,threads,n,duration,positions,angles,burns):

    #print("Start thread=%i" % (thread))

    # Initialize arrays
```

```
    xlist = np.zeros(n)
    ylist = np.zeros(n)
    pxlist = np.zeros(n)
    pylist = np.zeros(n)
    errlist = np.zeros(n)
    hlist = np.zeros(n)
    info = np.zeros(2)

    # Search for orbits
    trials = len(positions)*len(angles)*len(burns)
    ld1 = len(angles)*len(burns)
    ld2 = len(burns)
    trial = 0
    hit_earth = 0
    hit_moon = 0
    best_status = 1e9
    progress = -1
    i = thread
    while i < trials:

        # One-to-one mapping of i -> (pos_i,ang_i,burn_i)
        pos_i = i//ld1
        ang_i = (i-pos_i*ld1)//ld2
        burn_i = i-pos_i*ld1-ang_i*ld2
        i += threads

        # Find launch setup
        pos = positions[pos_i]
        ang = angles[ang_i]
        burn = burns[burn_i]

        # Calculate initial conditions
        r = leo_orbit/unit_len
        x0 = np.cos(pos)*r
        y0 = np.sin(pos)*r
        rx = -y0/r
        ry = x0/r
        vx = (leo_orbit_vel/unit_vel)*rx
        vy = (leo_orbit_vel/unit_vel)*ry
        x0 += earth_pos_x
        bx = np.cos(ang)*rx-np.sin(ang)*ry
        by = np.sin(ang)*rx+np.cos(ang)*ry
        px0 = (burn/abs(burn))*vx+burn*bx-y0 # Sign of burn decides
            rotational direction of launch
        py0 = (burn/abs(burn))*vy+burn*by+x0

        # Call symplectic integration
        # status > 0     : Closest distance to moon achieved
        # status < 0     : Hit the moon using status=dV(moon)-10000 to get
            into orbit
        # status == 100  : Collided with earth
        #if thread == 1:
```

```
94          #    print(n,duration,x0,y0,px0,py0)
95          status = symplectic(n,duration,x0,y0,px0,py0,xlist,ylist,pxlist,
                pylist,errlist,hlist,info)
96          if status == 100:
97              hit_earth += 1
98          if status < 0:
99              hit_moon += 1
100         if status < best_status:
101             best_status = status
102             best_pos = pos
103             best_ang = ang
104             best_burn = burn
105             best_x0 = x0
106             best_y0 = y0
107             best_px0 = px0
108             best_py0 = py0
109             best_dv = info[0]
110             best_toa = info[1]
111
112         # Show progress
113         if thread == 0: # only thread 0
114             if (100*trial/(1+trials//threads))//10 > progress:
115                 progress = (100*trial/(1+trials//threads))//10
116                 print(progress*10,end="% ")
117                 sys.stdout.flush()
118             trial += 1
119         #if thread == 13:
120          #   print("thread=%i status=%f best_status=%f trial=%i(%i) pos=%f
                ang=%f burn=%f" % (thread,status,best_status,trial,trials,pos,
                ang,burn))
121
122     #print("End thread=%i" % (thread))
123
124     return best_status,best_pos,best_ang,best_burn,best_x0,best_y0,best_px0,
            best_py0,best_dv,best_toa,hit_earth,hit_moon
125
126 def search_worker(args):
127     return search(args[0], args[1], args[2], args[3], args[4], args[5], args
            [6])
128
129 def search_mt(threads,n,duration,num_pos,num_ang,num_burn,pos_low,pos_high,
        ang_low,ang_high,burn_low,burn_high):
130
131     # Time search
132     runtime = time.time()
133
134     # Set search space
135     positions = np.linspace(pos_low,pos_high,num_pos)
136     angles = np.linspace(ang_low,ang_high,num_ang)
137     burns = np.linspace(burn_low,burn_high,num_burn)
138     if num_pos == 1:
139         positions[0] = (pos_high+pos_low)/2.0
```

```
140     if num_ang == 1:
141         angles[0] = (ang_high+ang_low)/2.0
142     if num_burn == 1:
143         burns[0] = (burn_high+burn_low)/2.0
144     if num_burn == 2:
145         burns[0] = burn_low
146         burns[1] = burn_high
147     trials = num_pos*num_ang*num_burn
148     print(positions)
149     print(angles)
150     print(burns*unit_vel)
151 
152     # Set threads
153     threads = min(threads,num_pos)
154 
155     # Do multi-threaded search
156     print("#
            ------------------------------------------------------------------------
            ")
157     print("# Threads:           %6i" % (threads))
158     print("# Trials:            %6i (" % (trials), end="")
159     if threads == 1:
160         # Single thread
161         best_status,best_pos,best_ang,best_burn,best_x0,best_y0,best_px0,
                best_py0,best_dv,best_toa,hit_earth,hit_moon = search(0,num_pos,n
                ,duration,positions,angles,burns)
162     else:
163         # Multi-threading
164         chunk = num_pos/threads
165         args = [[i,threads,n,duration,positions,angles,burns] for i in range
                (threads)]
166         pool = Pool()
167         result = pool.map(search_worker, args)
168         pool.close()
169         pool.join()
170 
171         # Reduce results from all threads
172         best_status = 1e9
173         hit_earth = 0
174         hit_moon = 0
175         for i in range(threads):
176             status = result[i][0]
177             if status < best_status:
178                 best_status = status
179                 best_pos = result[i][1]
180                 best_ang = result[i][2]
181                 best_burn = result[i][3]
182                 best_x0 = result[i][4]
183                 best_y0 = result[i][5]
184                 best_px0 = result[i][6]
185                 best_py0 = result[i][7]
186                 best_dv = result[i][8]
```

```
                best_toa = result[i][9]
                hit_earth += result[i][10]
                hit_moon += result[i][11]

    print("100%)")
    print("# No interception:  %6i (%i%%)" % (trials-hit_earth-hit_moon
        ,100*(trials-hit_earth-hit_moon)/trials))
    print("# Crashed on earth: %6i (%i%%)" % (hit_earth,100*hit_earth/trials
        ))
    print("# Hit moon:         %6i (%i%%)" % (hit_moon,100*hit_moon/trials))
    runtime = time.time()-runtime
    print("# Runtime:          %6.2fs" % (runtime))
    if best_status < 100:
        print_search_results(best_status,best_pos,best_ang,best_burn,best_x0
            ,best_y0,best_px0,best_py0,best_dv,best_toa)
        return best_status,best_pos,best_ang,best_burn,best_x0,best_y0,
            best_px0,best_py0,best_dv,best_toa
    else:
        return best_status,0,0,0,0,0,0,0,0,0
```

.

E.5 Symplectic Adaptive Verlet and Symplectic Euler Integrators

Implemeting the symplectic algorithms from Chapter 4 (filename: symplectic.py).

```
"""
Reduced 3-Body Problem Solver Module
====================================
"""

from math import pi,sqrt
import numpy as np
from numbapro import *
from const import *

@jit
def F(x,y):
    mux = mu+x
    mux2 = mux*mux
    mumx = 1-mux
    mumx2 = mumx*mumx
    y2 = y*y
    denum1 = 1.0/((mux2+y2)*sqrt(mux2+y2))
    denum2 = 1.0/((mumx2+y2)*sqrt(mumx2+y2))
    Fx = (mu-1.0)*mux*denum1+mu*mumx*denum2
    Fy = (mu-1.0)*y*denum1-mu*y*denum2
    return Fx,Fy
```

```
23
24 @jit
25 def explicit_euler_step(h,x,y,px,py):
26     Fx,Fy = F(x,y)
27     vx = px+y
28     vy = py-x
29     vpx = Fx+py
30     vpy = Fy-px
31     x += vx*h
32     y += vy*h
33     px += vpx*h
34     py += vpy*h
35     return x,y,px,py
36
37 @jit
38 def symplectic_euler_step(h,x,y,px,py):
39     # Step 1
40     vx = px+y
41     x = (x+(vx+py*h)*h)/(1.0+h*h)
42     vy = py-x
43     y += vy*h
44     # Step 2
45     Fx,Fy = F(x,y)
46     vpx = Fx+py
47     vpy = Fy-px
48     px += vpx*h
49     py += vpy*h
50     return x,y,px,py
51
52 @jit
53 def symplectic_verlet_step(h,x,y,px,py):
54     hh = 0.5*h
55     denum = 1.0/(1.0+hh*hh)
56     # Step 1
57     vx = px+y
58     x = (x+(vx+py*hh)*hh)*denum
59     vy = py-x
60     y += vy*hh
61     # Step 2
62     Fx,Fy = F(x,y)
63     vpx = Fx+py
64     vpy = Fy-px
65     px = (px+(2.0*vpx+(vpy+Fy)*hh)*hh)*denum
66     py += (vpy+Fy-px)*hh
67     # Step 3
68     vx = px+y
69     vy = py-x
70     x += vx*hh
71     y += vy*hh
72     return x,y,px,py
73
74 @jit #('void(int64, float64, float64, float64, float64, float64, float64,
```

```
    float64[:], float64[:], float64[:], float64[:])')
def symplectic(n,duration,x0,y0,px0,py0,xlist,ylist,pxlist,pylist,errlist,
    hlist,info):

    # Initialize initial conditions
    h = 1e-6
    hmin = 1e-10
    # tol = 1e-9
    tol = 10
    maxsteps = duration
    x = x0
    y = y0
    px = px0
    py = py0
    err = 1e-15
    status = 1
    target_dist = 1
    target = 1; target_pos_x = moon_pos_x
    #target = 2; target_pos_x = L1_pos_x
    target_pos_y = 0

    # Time reset
    t = 0
    for i in range(n):

        # Store position
        xlist[i] = x
        ylist[i] = y
        pxlist[i] = px
        pylist[i] = py
        errlist[i] = err
        hlist[i] = h

        # Integrate time period
        dt = duration*(i+1)/n
        count = 0
        while t < dt:
            # Safety on iterations
            count += 1
            if count > 10000000:
                count = 0
                hmin = 2*hmin

            # Adaptive symplectic euler/midpoint
            x1,y1,px1,py1 = symplectic_euler_step(h,x,y,px,py)
            x2,y2,px2,py2 = symplectic_verlet_step(h,x,y,px,py)

            # Relative local error of step
            err = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)/(x2*x2+y2*y2))

            # Accept the step only if the weighted error is no more than the
            # tolerance tol.  Estimate an h that will yield an error of tol
```

```
            on
# the next step and use 0.8 of this value to avoid failures.
if err < tol or h <= hmin:

    # Accept step
    x = x2
    y = y2
    px = px2
    py = py2

    # Forward time by step
    t = t+h
    # h = max(hmin, h*max(0.1, 0.8*sqrt(tol/err)))

else:
    # No accept, reduce h to half
    h = max(hmin, 0.5*h)

# How close are we to the moon?
rx = x-target_pos_x
ry = y-target_pos_y
r = sqrt(rx*rx+ry*ry)
target_dist = min(target_dist,r)

# Check if we hit the target
if status == 1:
    if target == 1:
        r_low = (lunar_orbit-orbit_range)/unit_len
        r_high = (lunar_orbit+orbit_range)/unit_len
    else:
        r_low = 0
        r_high = orbit_range/unit_len

    if r > r_low and r < r_high:

        # Current velocity
        vx = px+y
        vy = py-x

        if target == 1:

            # Project velocity onto radius vector and subtract
            # so velocity vector is along orbit
            vr = (vx*rx+vy*ry)/r
            vx = vx-vr*rx/r
            vy = vy-vr*ry/r

            # Now ajust velocity to lunar orbit velocity
            vt = sqrt(vx*vx+vy*vy)
            px = (lunar_orbit_vel/unit_vel)*vx/vt-y
            py = (lunar_orbit_vel/unit_vel)*vy/vt+x

```

```
176                         # Total velocity change
177                         dv = sqrt(vr*vr+(vt-lunar_orbit_vel/unit_vel)*(vt-
                                lunar_orbit_vel/unit_vel))
178                     else:
179                         dv = sqrt(vx*vx+vy*vy)
180 
181                     # Store info
182                     info[0] = dv
183                     info[1] = t
184 
185                     # Finish?
186                     status = -10000+dv
187                     if n == 1:
188                         return status
189 
190             # Check if we hit the earth
191             r = (x-earth_pos_x)*(x-earth_pos_x)+y*y
192             r_high = earth_radius/unit_len
193             if r < r_high*r_high:
194                 return 100 # Hit earth surface
195 
196     if status >= 0:
197         status = target_dist
198 
199     return status
```

.

APPENDIX F

Mathematica Data

In[42]:=
```
Quiet@Remove["`*"]
```

In[43]:=
```
SetOptions[SelectedNotebook[],
 PrintingStyleEnvironment → "Printout", ShowSyntaxStyles → True]
```

Date and version

Notebook run / all data acquired from Wolfram's servers at time:

In[44]:=
```
Now[]
```

Out[44]= Wed 10 Jun 2015 19:42:38 [ ]

In[45]:=
```
$Version
```

Out[45]= 10.0 for Mac OS X x86 (64-bit) (June 29, 2014)

Raw data

Universal

In[46]:=
```
UnitConvert[Quantity[1, "GravitationalConstant"], "SIBase"]
```

Out[46]= $6.67 \times 10^{-11}\ \mathrm{m}^3/(\mathrm{kg\,s}^2)$

Earth

In[47]:=
```
PlanetData[Earth (planet), "Mass"]
```

Out[47]= 5.9721986×10^{24} kg

In[48]:=
```
PlanetData[Earth (planet), "Radius"]
```

Out[48]= 6367.4447 km

Moon

In[49]:=
```
PlanetaryMoonData[Moon (planetary moon), "Mass"]
```

Out[49]= 7.3459×10^{22} kg

In[50]:=
```
PlanetaryMoonData[Moon (planetary moon), "Radius"]
```

Out[50]= 1737.5 km

In[51]:=
```
PlanetaryMoonData[Moon (planetary moon), "Inclination"]
```

Out[51]= 5.16°

Earth-Moon

In[52]:=
```
PlanetaryMoonData[Moon (planetary moon), "AverageDistanceFromEarth"]
```

Out[52]= 3.850×10^5 km

In[53]:=
```
PlanetaryMoonData[Moon (planetary moon), "OrbitPeriod"]
```

Out[53]= 27.322 days

Derived Units

Unit velocities

In[54]:=
```
unitTime = 1/(2 Pi) PlanetaryMoonData[Moon (planetary moon), "OrbitPeriod"]
```

Out[54]= 4.3484 days

In[55]:=
```
unitLength = PlanetaryMoonData[Moon (planetary moon), "AverageDistanceFromEarth"]
```

Out[55]= 3.850×10^5 km

In[56]:=
```
unitVelocity = UnitConvert[unitLength/unitTime, "km/s"]
```

Out[56]= 1.025 km/s

Earth sphere of influence

In[72]:=
```
earth sphere of influence / average moon orbit distance
PlanetData[Earth (planet), "SphereOfInfluenceRadius"] /
 PlanetaryMoonData[Moon (planetary moon), "SemimajorAxis"]
```

Out[72]= 2.41

Parking Velocities

Earth

In[57]:=
```
earthParkingVelocity =
 UnitConvert[Sqrt[(G PlanetData[Earth (planet), "Mass"])/(PlanetData[Earth (planet), "Radius"] + 160 km)], "km/s"]
```

Out[57]= 7.814 km/s

In[58]:= `earthParkingVelocity / unitVelocity`

Out[58]= 7.63

Moon

In[59]:= `moonParkingVelocity = UnitConvert[ Sqrt[ ( G PlanetaryMoonData[ Moon (planetary moon), "Mass" ] / PlanetaryMoonData[ Moon (planetary moon), "Radius" ] + 100 km ) ], "km/s" ]`

Out[59]= 1.633 km/s

In[60]:= `moonParkingVelocity / unitVelocity`

Out[60]= 1.594

Miscellaneous

In[61]:= `μ = G * PlanetData[ Earth (planet), "Mass" ]`

Out[61]= 5.9721986×10^{24} kg G

In[62]:= `0.1 * unitVelocity`

Out[62]= 0.102471 km/s

Apollo delta-v

In[69]:=
```
(*Trans-lunar injection*)
apolloEarth = UnitConvert[Quantity[10 000, "Feet per second"], "km/s"] // N
```

Out[69]= 3.048 km/s

In[71]:=
```
(*Lunar orbit insertion (PDF p. 27 in source*)
apolloMoon = UnitConvert[Quantity[3500, "Feet per second"], "km/s"] // N
```

Out[71]= 1.0668 km/s

In[67]:= `UnitConvert[apolloEarth + apolloMoon, "Km/s"] // N`

Out[67]= 4.1148 km/s

Bibliography

[And] Ross Andersen. *The Elon Musk interview on Mars colonisation – Ross Andersen – Aeon*. URL: http://aeon.co/magazine/technology/the-elon-musk-interview-on-mars/ (visited on June 4, 2015).

[Ast] Astrobotic. *Lunar Delivery | Astrobotic*. URL: https://www.astrobotic.com/lunar-delivery (visited on June 4, 2015).

[Bar] Hubert Bartkowiak. *File:Hohmann transfer orbit.svg - Wikimedia Commons*. URL: http://commons.wikimedia.org/wiki/File:Hohmann%7B_%7Dtransfer%7B_%7Dorbit.svg (visited on June 6, 2015).

[BBC] BBC. *BBC News - Obama cancels Moon return project*. URL: http://news.bbc.co.uk/2/hi/science/nature/8489097.stm (visited on June 4, 2015).

[BC00] Edward a Belbruno and John P Carrico. "Calculation of Weak Stability Boundary Ballistic Lunar Transfer Trajectories". In: *Astrodynamics specialist conference*. Denver, Colerado, 2000. DOI: doi:10.2514/6.2000-4142.

[BJ00] R Biesbroek and G Janin. "Ways to the Moon?" In: *Earth* august (2000), pages 92–99.

[Bra12] Jim Branson. *Recalling Lagrangian Mechanics*. 2012. URL: http://hepweb.ucsd.edu/ph110b/110b%7B_%7Dnotes/node86.html (visited on June 8, 2015).

[Buc] Andrew Buck. *File:Bi-elliptic transfer.svg - Wikimedia Commons*. URL: http://commons.wikimedia.org/wiki/File:Bi-elliptic%7B_%7Dtransfer.svg (visited on June 6, 2015).

[CNN] CNN. *CNN.com - Bush unveils vision for moon and beyond - Jan. 15, 2004*. URL: http://edition.cnn.com/2004/TECH/space/01/14/bush.space/ (visited on June 4, 2015).

[Coo] Jonathan Coopersmith. *Affordable Access to Space | Issues in Science and Technology*. URL: http://issues.org/29-1/jonathan/ (visited on June 4, 2015).

[Cop13] Copperheadtnp. *I was curious about the delta v requirements for different transfers. So I made this plot. At best, a bi-elliptic transfer will require 92% of the delta v of a Hohmann transfer. : KerbalSpaceProgram*. 2013. URL: http://www.reddit.com/r/KerbalSpaceProgram/comments/1ajru7/i%7B_%7Dwas%7B_%7Dcurious%7B_%7Dabout%7B_%7Dthe%7B_%7Ddelta%7B_%7Dv%7B_%7Drequirements%7B_%7Dfor/ (visited on June 6, 2015).

[ESA14] ESA. *Soyuz rendezvous and docking explained*. 2014. URL: https://www.youtube.com/watch?v=M2%7B_%7DNeFbFcSw%7B%5C&%7Dfeature=youtu.be%7B%5C&%7Dt=9m55s.

[Fau] Kjell Magne Fauske. *Global nodes | TikZ example*. URL: http://www.texample.net/tikz/examples/global-nodes/ (visited on May 29, 2015).

[Fra83] Craig Fraser. "J. L. Lagrange's early contributions to the principles and methods of mechanics". In: *Archive for History of Exact Sciences* 28.3 (1983), pages 197–241. ISSN: 00039519. DOI: 10.1007/BF00328268. URL: http://link.springer.com/10.1007/BF00328268.

[GG07] F. García and G. Gómez. "A note on weak stability boundaries". In: *Celestial Mechanics and Dynamical Astronomy* 97.2 (2007), pages 87–100. ISSN: 09232958. DOI: 10.1007/s10569-006-9053-6.

[Gol80] Herman Heine Goldstine. *A history of the calculus of variations from the 17th through the 19th century*. First Edit. Springer-Verlag, 1980, page 410. ISBN: 3540905219. URL: https://books.google.com/books?id=ymQPAQAAMAAJ%7B%5C&%7Dpgis=1.

[GPS02] Herbert Goldstein, Charles P. Poole, and John L. Safko. *Classical Mechanics*. 3rd. Addison Wesley, 2002, page 638. ISBN: 0201657023. URL: https://books.google.dk/books/about/Classical%7B_%7DMechanics.html?id=tJCuQgAACAAJ%7B%5C&%7Dpgis=1.

[Hjo15] P G Hjorth. *A few Concepts from Analytical Mechanics, DTU (unpublished)*. 2015.

[Hoc08] Marlis Hochbruck. "Symplectic Integrators". In: (2008). URL: https://na.math.kit.edu/marlis/download/meetings/08Oberwolfach/symplectic1-print.pdf.

[HT10] Ernst Hairer and M Tu. *Lecture 1 : Hamiltonian systems*. 2010. URL: http://www.unige.ch/%7B~%7Dhairer/poly%7B_%7Dgeoint/week1.pdf (visited on June 8, 2015).

[Juu08] Jeppe Søgaard Juul. *Low Energy Trajectories to the Moon*. Technical report. University of Copehagen, 2008, page 36. URL: http://jeppejuul.com/research/projects/bachelors-project-low-energy-trajectories-to-the-moon/.

[KH02] Jens Martin Knudsen and Poul G. Hjorth. *Elements of Newtonian Mechanics*. 3rd Editio. Springer-Verlag, 2002, page 451. ISBN: 3-540-67652-X.

[Koo+01] W.S. Koon et al. "Low Energy Transfer to the Moon". In: *Celestial Mechanics and Dynamical Astronomy* 81.1-2 (2001), pages 63–73. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.307.2362.

[Lan70] Cornelius Lanczos. *The Variational Principles of Mechanics*. Fourth Edi. Dover Publications, 1970, page 418. ISBN: 978-0486650678.

[Lau] Laursen. *_laursen / Laursen's XeLaTeX thesis template — Bitbucket*. URL: https://bitbucket.org/%7B_%7Dlaursen/laursens-xelatex-thesis-template/overview (visited on May 29, 2015).

[MD99] Carl D. Murray and Stanley F. Dermott. *Solar System Dynamics*. 1st. Cambridge University Press, 1999, page 592. ISBN: 0-521-57295-9.

[Min] J.R. Minkel. *Is the International Space Station Worth $100 Billion?* URL: http://www.space.com/9435-international-space-station-worth-100-billion.html (visited on June 9, 2015).

[NASa] NASA. *Apollo 11 Mission Overview*. URL: http://www.lpi.usra.edu/lunar/missions/apollo/apollo%7B_%7D11/overview/ (visited on June 11, 2015).

[NASb] NASA. *Launch Windows Essay*. URL: http://history.nasa.gov/afj/launchwindow/lw1.html (visited on June 8, 2015).

[NASc] NASA. *NASA - NSSDC - Spacecraft - Details*. URL: http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1990-007A (visited on June 8, 2015).

[NAS66] NASA. "Apollo Lunar Landing Mission Symposium". In: 1966. URL: http://www.hq.nasa.gov/alsj/LunarLandingMIssionSymposium1966%7B_%7D1978075303.pdf.

[NF02] Michiyo Nakane and Craig G. Fraser. "The Early History of Hamilton-Jacobi Dynamics 1834?1837". In: *Centaurus* 44.3-4 (December 2002), pages 161–227. ISSN: 0008-8994. DOI: 10.1111/j.1600-0498.2002.tb00613.x. URL: http://doi.wiley.com/10.1111/j.1600-0498.2002.tb00613.x.

[Oka15] Jacob Akira Okada. *Painting the Way to the Moon*. 2015. URL: http://www.paintingthewaytothemoon.com/%20http://www.imdb.com/title/tt4294246/.

[Pee] Matthew M. Peet. *Spacecraft and Aircraft Dynamics - Lecture 9: Bi-elliptics and Out-of-Plane Maneuvers*. URL: http://mmae.iit.edu/%7B~%7Dmpeet/Classes/MMAE441/Spacecraft/441Lecture21.pdf.

[Pis05] Vincent L. Pisacane. *Fundamentals of Space Systems*. 2nd. Oxford University Press, 2005, page 828. ISBN: 978-0-19-516205-9.

[Pru92] John E. Prussing. "Simple proof of the global optimality of the Hohmann transfer". en. In: *Journal of Guidance, Control, and Dynamics* 15.4 (July 1992), pages 1037–1038. ISSN: 0731-5090. DOI: 10.2514/3.20941. URL: http://arc.aiaa.org.globalproxy.cvt.dk/doi/abs/10.2514/3.20941.

[Ram15] Sarah Ramsey. *NASA's Space Launch System Booster Passes Major Ground Test*. 2015. URL: http://www.nasa.gov/press/2015/march/nasas-space-launch-system-booster-passes-major-ground-test.

[See02] Wolfgang Seefelder. *Lunar Transfer Orbits utilizing Solar Pertubations and Ballistic Capture*. 1st. Herbert Utz Verlag, 2002, page 171. ISBN: 3-8316-0155-0.

[Sta] Stackexchange. *Stackexchange Mathematics - Who came up with the Euler-Lagrange equation first?* URL: http://math.stackexchange.com/questions/177243/who-came-up-with-the-euler-lagrange-equation-first (visited on June 2, 2015).

[Swe91] Theodore H. Sweetser. "Estimate of the global minimum DV needed for earth-moon transfer". In: *Advances in the Astronautical Sciences* 75.pt 1 (1991), pages 111–120. ISSN: 00653438.

[Tay09] Travis S. Taylor. *Introduction to Rocket Science and Engineering*. 1st. CRC Press, 2009, page 310. ISBN: 1420075292. URL: https://books.google.com/books?id=dQHMBQAAQBAJ%7B%5C&%7Dpgis=1.

[TB14] Francesco Topputo and E Belbruno. "Earth–Mars Transfers with Ballistic Capture". In: *arXiv preprint arXiv:1410.8856* (2014), pages 1–21. ISSN: 0923-2958. DOI: 10.1007/s10569-015-9605-8. arXiv: 1410.8856. URL: http://arxiv.org/abs/1410.8856.

[TVB05] Francesco Topputo, Massimiliano Vasile, and Franco Bernelli-Zazzera. "Earth-to-moon low energy transfers targeting L1 hyperbolic transit orbits". In: *Annals of the New York Academy of Sciences*. June. 2005. DOI: 10.1196/annals.1370.025. URL: https://www.researchgate.net/profile/Francesco%7B_%7DTopputo/publication/45417269%7B_%7DEarth-to-Moon%7B_%7DLow%7B_%7DEnergy%7B_%7DTransfers%7B_%7DTargeting%7B_%7DL1%7B_%7DHyperbolic%7B_%7DTransit%7B_%7DOrbits/links/0046351f230274d70a000000.pdf.

[Use] Stanmar (Wikipedia User). *File:EllipseInPolarCoords2.svg - Wikimedia Commons*. URL: http://commons.wikimedia.org/wiki/File:EllipseInPolarCoords2.svg (visited on June 6, 2015).

[Wei] Eric W. Weisstein. *Noether's Symmetry Theorem*. en. URL: http://mathworld.wolfram.com/NoethersSymmetryTheorem.html.

[Whe] Robin Wheeler. *Apollo lunar landing launch window: The controlling factors and constraints*. URL: http://history.nasa.gov/afj/launchwindow/lw1.html%7B%5C#%7DEPO (visited on June 4, 2015).

[Wika] Wikipedia. *Wikipedia - Bi-elliptic transfer*. URL: http://en.wikipedia.org/wiki/Bi-elliptic%7B_%7Dtransfer%7B%5C#%7DExample (visited on June 6, 2015).

[Wikb] Wikipedia. *Wikipedia: Lagrangian - Advantages over other methods.* URL: http://en.wikipedia.org/wiki/Lagrangian%7B%5C#%7DAdvantages%7B_%7Dover%7B_%7Dother%7B_%7Dmethods (visited on June 1, 2015).

[Wikc] Wikipedia. *Wikipedia: List of missions to the Moon.* URL: http://en.wikipedia.org/wiki/List%7B_%7Dof%7B_%7Dmissions%7B_%7Dto%7B_%7Dthe%7B_%7DMoon (visited on June 4, 2015).

[Wikd] Wikipedia. *Wikipedia: Quadratic form.* URL: http://en.wikipedia.org/wiki/Quadratic%7B_%7Dform (visited on June 2, 2015).

[Wike] Wikipedia. *Wikipedia - SpaceX.* (Visited on June 4, 2015).

[Wil] Matt Williams. *Making the Trip to Mars Cheaper and Easier: The Case for Ballistic Capture.* URL: http://www.universetoday.com/117615/making-the-trip-to-mars-cheaper-and-easier-the-case-for-ballistic-capture/ (visited on June 8, 2015).

[Wol] Inc. Wolfram Research. *Mathematica (see appendix "Mathematica Data" for details).* Champaign, Illinois.

{2.7182818284}ϑωερτυθιοπσδφγηξκλ