HASKELL SPACEFLIGHT WORKSHOP

Jonathan Merritt, Luke Clifton

YOW! LambdaJam 2019, May 13–15

Version timestamp: 2019-05-12T01:44:48+0000

Table of Contents

1

Introduction

This workshop consists of these published notes and a set of problems contained in the associated GitHub Haskell project.

How much of the notes it will be necessary to read depends on the background of an individual participant. Those who are already familiar with numerical methods might be able to skip most of the notes and simply work on the problems directly. However, we have tried to make the notes fairly comprehensive for newcomers, so that people unfamiliar with numerical methods at least have a starting point.

2

ODE Integration and Initial Value Problems

The models we use for a spacecraft depend upon a set of variables that represent its state at an instant in time. These state variables typically include:

- Position
- Velocity
- Mass

They may be scalar quantities or vectors, as appropriate to the problem.

Our simulations are all examples of "Initial Value Problems". In an initial value problem, we know the starting state of the spacecraft, and we have a set of first-order ordinary differential equations (ODEs), which describe how its state evolves with time. We will integrate these ODEs to predict the state at future times. Using this approach, we can compute the time history of state variables that are critial to mission or maneuver planning. For example, we might find the trajectory of a spacecraft (its position as a function of time), and check whether it places the spacecraft in a desired orbit.

The motion of a spacecraft depends on multiple forces that might be acting on it. For example:

- Gravity
- Atmospheric drag
- Rocket thrust

Thrust from a rocket engine may be controlled, and control inputs can be modeled easily in our system. Testing the behavior of a control system, particularly under conditions of real-world variations, is a modern practical use of the methods we cover (eg. [1, 2]).

2.1 1D Euler's Method

We can write a set of coupled, first-order ODEs as:

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}} = f(t, \mathbf{x}) \tag{2.1}$$

Here, $\mathbf{x}$ is the state vector, t is time, and f is some function. In Euler's method, we approximate a step forward in time by adding the product of the gradient, $\dot{\mathbf{x}}$, and the time step, h, to the current state, $\mathbf{x}$:

$$\mathbf{x}(t+h) \approx \mathbf{x}(t) + \dot{\mathbf{x}}\, h \tag{2.2}$$

$$\approx \mathbf{x}(t) + f\left(t, \mathbf{x}(t)\right)\, h \tag{2.3}$$

2.1.1 Radioactive Decay

We will begin implementing Euler's method with a 1D state, specialized to `Double`, using the process of radioactive decay as an example. Radioactive decay has an analytical solution, thus providing a ground truth against which the numerical result can be compared. It only involves a single state variable, N, which can be represented as a `Double`. Specializing to `Double` gives us a simple starting point that is close to the approach used in many other programming languages.

In radioactive decay, the rate of decay, $\dot{N}$, is proportional to the number of moles of radioactive particles that remain at any instant in time, N:

$$\dot{N} = -\lambda N \tag{2.4}$$

where λ is called the decay constant. This equation can be solved by knowing in advance that an exponential function happens to fit exactly the expected equation:

$$N = N_0 \exp\left(-\lambda t\right) \tag{2.5}$$

So that:

$$\dot{N} = -\lambda \left(N_0 \exp\left(-\lambda t\right)\right) \tag{2.6}$$

$$= -\lambda N \tag{2.7}$$

as required. Conventionally, λ is specified in terms of the half-life of an isotope, $t_{(1/2)}$:

$$\text{at } t = 0,\ N = N_0 \tag{2.8}$$

$$\text{at } t = t_{(1/2)},\ N = \frac{N_0}{2} \tag{2.9}$$

thus:

$$\frac{N_0}{2} = N_0 \exp\left(-\lambda t_{(1/2)}\right) \tag{2.10}$$

$$\ln\left(\tfrac{1}{2}\right) = -\lambda t_{(1/2)} \tag{2.11}$$

$$\lambda = \frac{\ln 2}{t_{(1/2)}} \tag{2.12}$$

As an example, consider the isotope Plutonium-238 (^{238}Pu), which has been used in radioisotope thermoelectric generators (RTGs) for spacecraft such as the Voyager 1 and 2 probes. This isotope has a half-life of approximately 87.7 years.

Problem 1: Euler integration specialized to `Double`.

In the file `ODE.hs`,

- implement `eulerStepDouble`, which takes a single step of Euler integration
- implement `integrateEulerDouble`, which takes multiple steps

In `ODEExamples.hs`,

- run `plotEulerDoubleExpDecay Screen`, to view a plot of Euler integration applied to the radioactive decay example

Figure 2.1 shows the result of applying Euler integration to the radioactive decay example. In this figure, it is evident that when smaller time-steps are taken, the Euler method more closely approximates the analytical solution. This is usually the case practically with numerical integration, although there is a limit beyond which smaller time steps will begin to diverge from the correct solution due to accruing floating-point errors. We will see later that raising the polynomial order of the integration approximation can improve accuracy with greater computational efficiency than taking smaller time steps.

2.2 Euler's Method for an `AffineSpace` State

We will now generalize Euler's method using the abstractions available in the `vector-space` package. The necessary constraints are captured in Listing 1. Don't panic if this seems a lot to take in, since we'll see a few concrete examples.

The first concept we introduce is the difference between an `AffineSpace` and its associated `VectorSpace`. In the present context, points in the `AffineSpace` are points belonging to the state space of the problem, of type `state` (eg. position, velocity, etc). Vectors of the associated `VectorSpace`, of type `diff`, represent deltas or differences between the points (eg. an offset of position, a delta in velocity, etc). We can add a vector to a point to obtain a new point, but we don't sum points directly. Similarly, we can multiply a vector by a scalar, but we cannot multiply a point by a scalar. Most widely-used frameworks for numerical integration do not make this distinction.

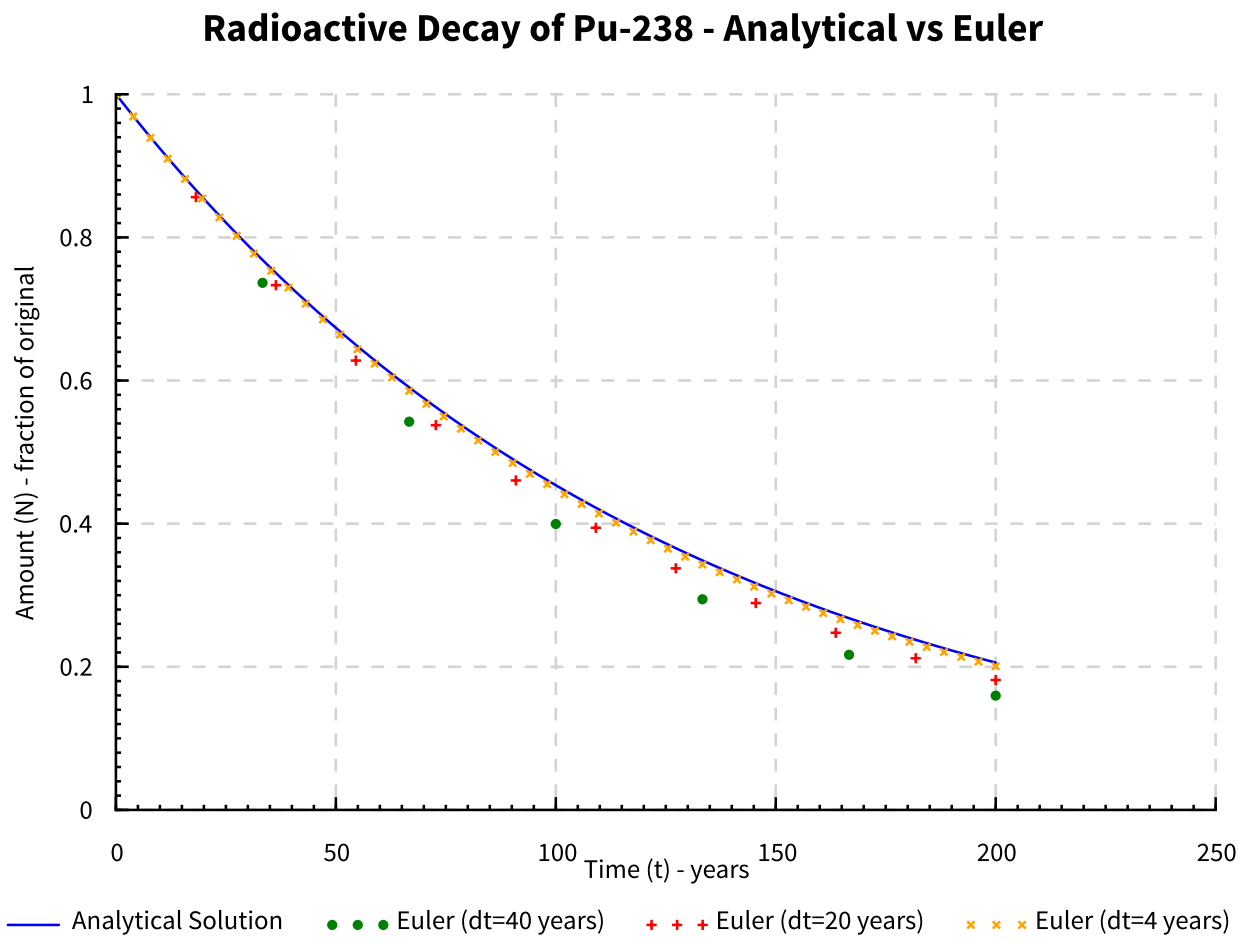

Figure 2.1: Comparison of Euler integration with the analytical result for radioactive decay of the isotope ^{238}Pu. The solid line shows the analytical solution while the points demonstrate the Euler approximation for different time steps.

```haskell
eulerStep
  :: ( AffineSpace state
     , diff ~ Diff state, VectorSpace diff
     , HasBasis time, HasTrie (Basis time)
     , s ~ Scalar diff, s ~ Scalar time )
  => time                              -- ^ Step size @dt@
  -> ((time, state) -> time :-* diff)  -- ^ Gradient function @f (x, t)@
  -> (time, state)                     -- ^ Before the step @(t, x)@
  -> (time, state)                     -- ^ After the step @(t, x)@
```

Listing 1: Constraints for Euler's method generalized by `vector-space`.

Next is the concept of a linear map representing the derivative: `time :-* diff`. In our case, where we deal with finite differences, this constructor can be considered analogous to a (linear) function of type `time -> diff`. What this function represents is the delta, `diff`, which arises from taking a time-step, h, of type `time`. A very simple illustration of the linear map is shown in Listing 2, with a version for vectors shown in Listing 3. The `linear` function assumes that the function it has been provided is linear, and it memoizes the values of that function along each basis vector of the vector space.

```
> :set -XFlexibleContexts
> import Data.LinearMap ((:-*), linear, lapply)
> f = (*) 5 :: Double -> Double
> lm = linear f
> :t lm
lm :: Double :-* Double
> :force lm
lm = Data.LinearMap.LMap (Just 5.0)
> lapply lm 1.0
5.0
> lapply lm 2.0
10.0
```

Listing 2: A scalar linear map. Once the map has been defined (by the `linear` function), the `lapply` function multiplies the input vector (a `Double`) by the memoized value along the unit basis vector.

```
> :set -XFlexibleContexts
> import Data.LinearMap ((:-*), linear, lapply)
> :{
| f :: (Double, Double) -> (Double, Double, Double)
| f (x, y) = (2*x + y, 3*x - y, 4*y)
| :}
> lm = linear f
> :force lm
lm = Data.LinearMap.LMap
     (Just
       (Data.MemoTrie.EitherTrie ((,,) 2.0 3.0 0.0) ((,,) 1.0 -1.0 4.0)))
> lapply lm (5, 6)
(16.0,9.0,24.0)
```

Listing 3: A vector linear map, using tuples for vectors. This provides a better view of the memoisation that is occurring under the hood. The construction of a matrix-like representation (but with automatic dimension checking) is evident.

Finally, we need to describe the operations that can be used on instances of `AffineSpace` and its associated `VectorSpace`:

`lapply m h` applies linear map `m` to vector `h`

`p .+^ v` adds vector `v` to point `p`
`a ^+^ b` adds vector `a` to vector `b`

These operations are sufficient to implement the generalized form of Euler's method.

2.2.1 Simple Harmonic Motion

To motivate the generalized form of Euler's method, let's consider simple harmonic motion (SHM), with two scalar state variables:

- Position, r
- Velocity, v

The data type `StateSHM` in `ODEExamples.hs` describes this state, which is the `AffineSpace` of the problem. It introduces statically type-checked units from the `units` package for length and velocity. The data type `DStateSHM` is the correspondng `VectorSpace` of the problem, representing deltas in the state.

In SHM, a linear spring force, F, is proportional to the position, r, with a spring constant, k:

$$F = -kr \tag{2.13}$$

This is combined with the equations of motion of a point mass to produce the following governing ODE:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{r} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ -kr/m \end{bmatrix} \tag{2.14}$$

where v is the velocity and m is the mass. If the initial conditions of the problem at $t = 0$ are $r = r_0$ and $v = 0$ then the analytical solution is:

$$\mathbf{x}(t) = \begin{bmatrix} r_0 \cos(\omega t) \\ -\omega r_0 \sin(\omega t) \end{bmatrix} \tag{2.15}$$

in which ω is the angular velocity, given by:

$$\omega = \sqrt{\frac{k}{m}} \tag{2.16}$$

This analytical solution can be differentiated twice manually to confirm that it satisfies the governing ODE, and values substituted to confirm that it satisfies the initial conditions.

Problem 2: Generalized Euler integration.

In the file `ODE.hs`,

- implement `eulerStep`
- implement `integrate` and `integrateWithDiff`

In `ODEExamples.hs`,

- run `plotEulerSHM Screen`, to view a plot of the position state variable, computed by Euler integration of SHM equations

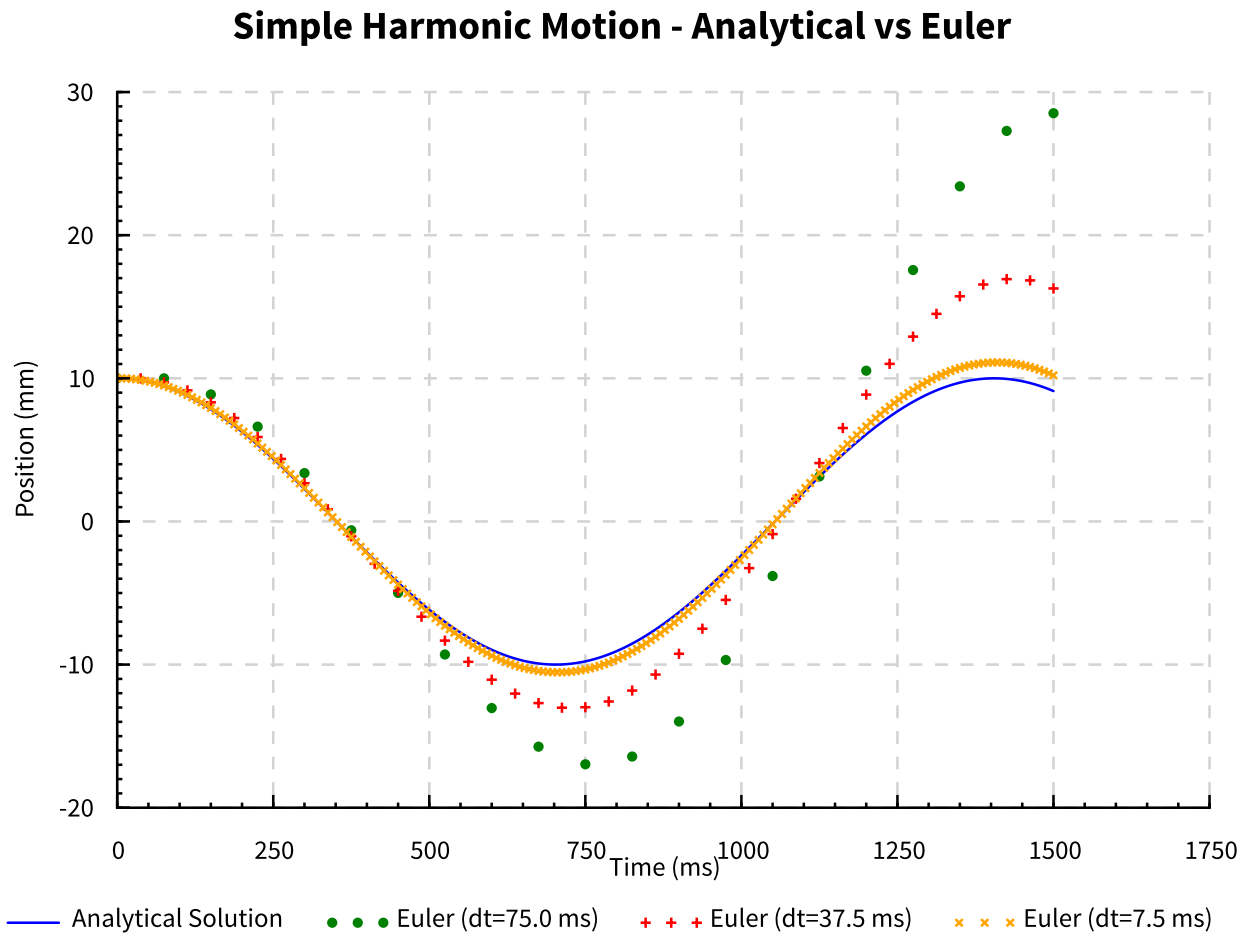

Figure 2.2: Comparison of Euler integration with the analytical result for the position variable of a simple harmonic oscillator. The solid line shows the analytical solution while the points demonstrate the Euler approximations for different time steps.

Figure 2.2 shows the position variable of a simple harmonic oscillator example. Once again, the smaller the time step, the more closely the result tracks the analytical solution.

2.3 4th-Order Runge-Kutta Integration

We have used Euler integration so far because it introduced the concepts we rely on from `vector-space`. However, there is a more common default for numerical integration of ODEs: the 4th-Order Runge-Kutta method (RK4). RK4 is the default in Matlab and SciPy (in those packages it is typically used with an embedded 5th-order approximation for step-size control; something we won't implement here). RK4 is usually a much better choice than the Euler method in terms of the accuracy/performance tradeoff.

RK4 is definitely not the final word though! The popular Numerical Recipes textbook recommends an 8th-order method (Dopr853) for general production use in non-stiff systems [3]. Specialized integrators may also be used for particular problems. The Apollo Guidance Computer used Nyström's Method to perform integration for efficiency and because of the dominant effect of a single, central gravitational force in most situations [4].[1] Long-duration astrodynamics problems, such as those concerning solar-system formation, or the behaviour of orbits over thousands of years, may have to use symplectic integrators to achieve reasonable accuracy (eg. [5]). We don't investigate these methods here because of time limitations, and because RK4 is both easy to implement and entirely sufficient for the examples.

We will only supply the equations for RK4 here and refer readers elsewhere (eg. [3, 6]) for a complete derivation:

$$\mathbf{k_1} = h\, f(t, \mathbf{x}) \tag{2.17}$$
$$\mathbf{k_2} = h\, f(t + \tfrac{1}{2}h, \mathbf{x} + \tfrac{1}{2}\mathbf{k_1}) \tag{2.18}$$
$$\mathbf{k_3} = h\, f(t + \tfrac{1}{2}h, \mathbf{x} + \tfrac{1}{2}\mathbf{k_2}) \tag{2.19}$$
$$\mathbf{k_4} = h\, f(t + h, \mathbf{x} + \mathbf{k_3}) \tag{2.20}$$
$$\mathbf{x}(t + h) \approx \mathbf{x} + \tfrac{1}{6}\mathbf{k_1} + \tfrac{1}{3}\mathbf{k_2} + \tfrac{1}{3}\mathbf{k_3} + \tfrac{1}{6}\mathbf{k_4} \tag{2.21}$$

The vectors $\mathbf{k_1} \ldots \mathbf{k_4}$ can be treated as stages of the computation, and are good candidates for let-floating. However, be aware that these equations do not directly represent the Haskell code. Instead, when implementing them, some care must be taken to consider what components are computed by `lapply`, what operations are adding vectors, and what operations are offsetting a point by a vector. Determining these are left as part of the exercise.

[1] The Draper Lab Apollo documents do refer to RK4 though, as "The usual fourth-order Runge-Kutta integration". Apparently it has a long history as the "go to" approach!

Problem 3: Runge-Kutta Integration.

In the file `ODE.hs`,

- implement `rk4Step`

In `ODEExamples.hs`,

- run `plotSHMComparison Screen`, to view a plot of Euler vs RK4 for the SHM example, using the same number of function evaluations

Figure 2.3 shows the comparison of Euler's method and RK4, for the same number of function evaluations. It is clear that RK4 much more closely approximates the analytical result.

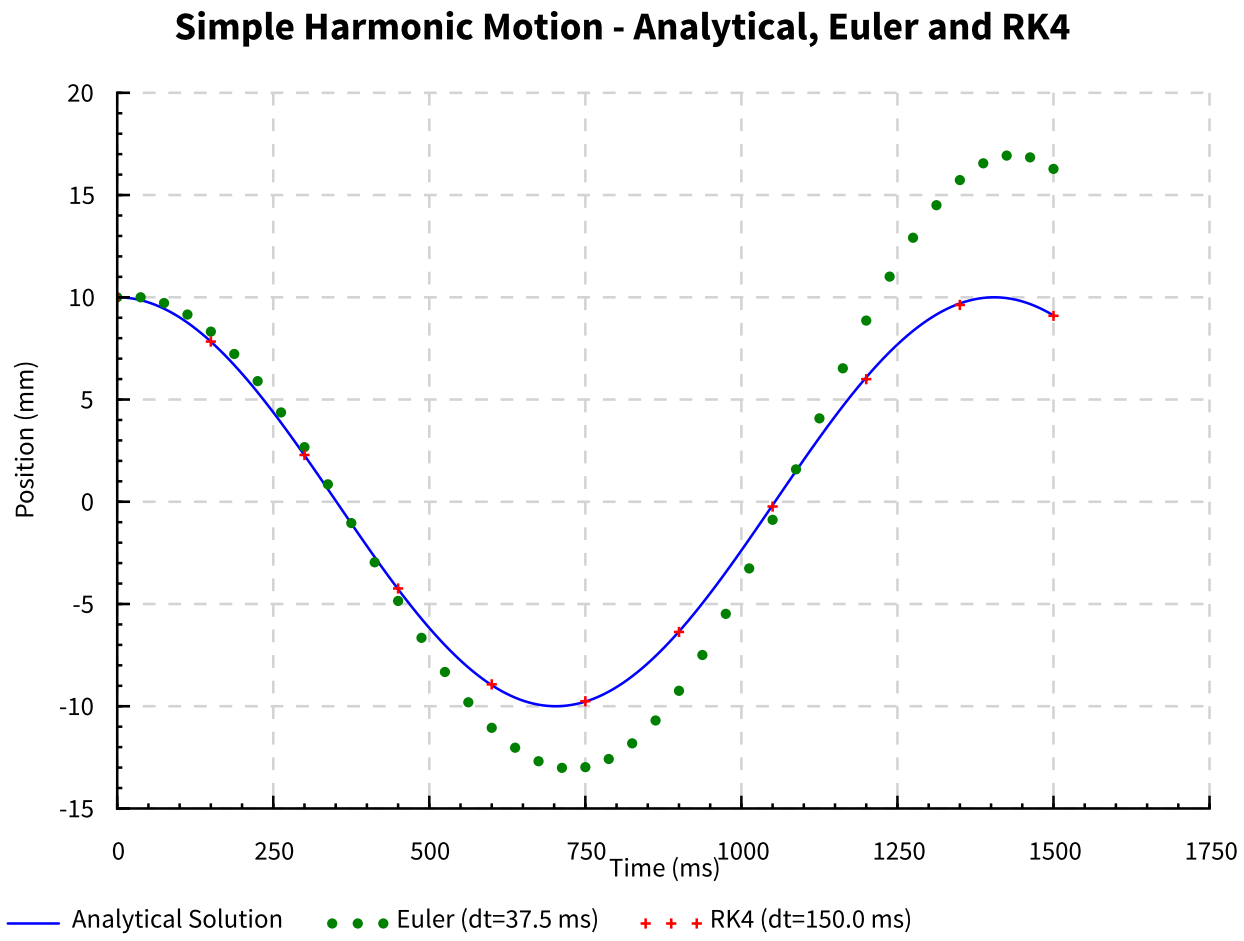

Figure 2.3: Comparison of Euler and RK4 integrations, for the same number of function evaluations, with the analytical result of a simple harmonic oscillator.

2.4 Simulating Apollo Lunar Ascent

Now that we have a working integrator, it's possible to look at an example of a somewhat realistic simulation: the algorithm used for Lunar Ascent Guidance during the Apollo missions.

This algorithm is described in summary by a NASA technical report [7] and in much greater detail by an MIT Draper Lab document [4], which contains sufficient detail to implement the algorithm.

This section contains a "pre-baked" scenario that uses the code written in this chapter to run a prepared simulation of the Apollo Lunar Ascent. We do not expect participants to be able to cover all the details in the available time, so much of the information in this section is provided only to give a high-level overview.

The approach we use for simulation involves checking-in with our version of the Apollo Guidance Computer (AGC) every 2 s of simulation time, matching the polling that was used in the original AGC. The ascent guidance returns a commanded thrust angle and an optional engine shutoff time. We take the commanded thrust angle and compute an angular acceleration that will point the ascent stage at that angle by the end of the 2 s period. This approximates the behaviour of the original Digital Autopilot, which achieved the same thing by firing the Reaction Control System (RCS) thrusters. We then integrate the equations of motion forward for 2 s before polling the AGC once again. This swapping between calls to the AGC and forward integration continues until we reach the commanded engine shutoff time, at which point we stop the thrust from the Ascent Propulsion System (APS) engine. Following the burn phase, we take its final state as the initial conditions of a new integration, and integrate the equations of motion for a further 10 000 s to numerically compute the coasting trajectory.

A rare feature of this simulation is that it incorporates statically-checked units, from the `units` package. Whether or not that was a worthwhile exercise is debatable, as the overhead and extra code complexity is somewhat overwhelming. However, we do have static confirmation that almost all aspects of the algorithm use consistent units.

The ascent guidance used the following parameters as a target:

- target velocity (a 2D vector in our version; 3D in the original)
- target radius (a scalar distance measured from the center of the moon)
- distance from the Command Service Module (CSM) orbital plane (removed in our version)

During the Apollo missions, the desired insertion orbit of the ascent stage was computed by Misson Control in advance of starting the ascent. That orbit determined the target parameters. In our simulation, we did not perform those calculations, but instead used the target parameters for a "quick, early takeoff", which were programmed by default into the AGC, and would typically be overridden by the astronauts under nominal conditions [4]. The phase of the orbit relative to the CSM would be set approximately, by knowing the nominal ascent duration and scheduling the takeoff accordingly [4]. Rendezvous with the CSM was an entirely different manouvre, performed after the ascent, and not part of the ascent guidance [4].

The inner workings of the guidance algorithm are somewhat outside the scope of this workshop, and in fact are not derived in the Draper Lab document [4]. We can roughly describe our inferred understanding of the approach based upon other texts. A detailed derivation of multiple types of guidance is provided by Battin [6], who was a core member of the Draper Lab during the Apollo era. A freely-available derivation that quite closely matches the Apollo algorithm is also provided by Townsend et al. [8], although those authors do not specifically mention Apollo.

Given the ascent target parameters, the guidance begins each polling loop by computing a velocity-to-be-gained. This is the difference between the target velocity and the current velocity, corrected for any velocity losses during the burn from gravitational acceleration. The change in velocity due to gravity was given by an expression obtained by integrating the gravitational acceleration analytically. The upper limit of that integration required an approximation of the remaining burn time (time-to-go estimate). The initial time-to-go estimate was a hard-coded value, but subsequent iterations obtained an estimate from a Taylor Series expansion of the well-known Tsiolkovsky Rocket Equation (see Section 3.4), using the velocity-to-be-gained as the delta-V. Finally, a control law, referred to as the "linear guidance concept" [4], was used to choose the commanded thrust direction, so that the velocity-to-be-gained would fall to zero at the end of the burn, thus matching the target velocity, and the target radius would also be achieved. In addition, the algorithm contained several, somewhat more ad-hoc tweaks, such as prioritising radial thrust and performing bang-bang directional control of radial acceleration if the target radius was reached early.

Given the Draper Lab's prior involvement with ballistic missile projects, such as the Minuteman ICBMs [9], the ascent guidance algorithm seems quite likely to have been influenced by missile systems. It also seems likely that earlier practical tests of those systems and other rockets lent some confidence to the early choice of guidance approaches. However, it is naturally extremely difficult to track down any details of large weapons systems for final confirmation.[2]

In our version of the algorithm, we have made some minor simplifications to suit this workshop, none of which are substantial changes to the core algorithm or guidance concept. The main changes and approximations are:

- We projected the problem into 2D, by removing the parameter specifying the distance from the CSM orbital plane, and removing the associated control parameters. Mathematically, this is identical to launching in the CSM orbital plane and remaining there, so it is simply a special case of the full guidance algorithm.

- We removed the thrust filter computations and associated pre-launch initializations, since we don't model the behaviour and noise characteristics of the inertial guidance unit. Instead, we used the nominal initial value for the thrust, based on stored values for the APS exhaust velocity and mass flow rate.

[2] I tried. Hopefully I'm not on any (additional) watch lists now. - J. Merritt.

- We did not allow the RCS to substitute for the APS, and we did not consider modeling abort scenarios.
- We did not include lunar rotation in the initial conditions (our initial velocity is zero).
- The Average-G routine is substituted with a simpler version.
- Window Pointing Direction (WDP) is unused and not specified (it always points toward the center of the moon in the original version anyway).
- The Digital Autopilot (not technically part of the Ascent Guidance itself) is not modelled in full, and is instead substituted by a piecewise constant angular acceleration.
- We did not take into account additional thrust produced by +X firing of the RCS for attitude control. Due to the regular angular acceleration, and thus regular RCS burns during ascent, this may result in our slightly underestimating the net thrust.
- We did not compute any of the user-feedback parameters that were only computed to be displayed on the AGC Display and Keyboard (DSKY), rather than participating in guidance.

Although the details above are unlikely to be investigated deeply during the workshop, the larger point is to demonstrate that a relatively involved control algorithm can be simulated using the integration approaches developed in this chapter.

Problem 4: Using the RK4 Integrator to Simulate Apollo Lunar Ascent Guidance.

In the file `LunarAscent.hs`,

- run `plotLunarAscentVerticalRise Screen`, to view the vertical rise phase of the ascent
- run `plotLunarAscentBurnOnly Screen`, to view the entire burn phase
- run `plotLunarAscentMoonView Screen`, to display a moon-centred view of the burn and coasting

Figure 2.4 shows the initial vertical rise phase of the lunar ascent. This closely matches the published vertical rise phase reported in Bennett [7] (Figure 13). Figure 2.5 shows the entire orbital insertion phase, which corresponds to Figures 14 and 15 from Bennett [7]. Finally, Figure 2.6 shows a moon-centred view of the burn and later coasting orbit, computed numerically. In Figure 2.6, the radial altitude above the moon is magnified by a factor of 20 to better distinguish the orbits. This is common in many illustrations of lunar orbits (for example, Figure 12 of Bennett [7]).

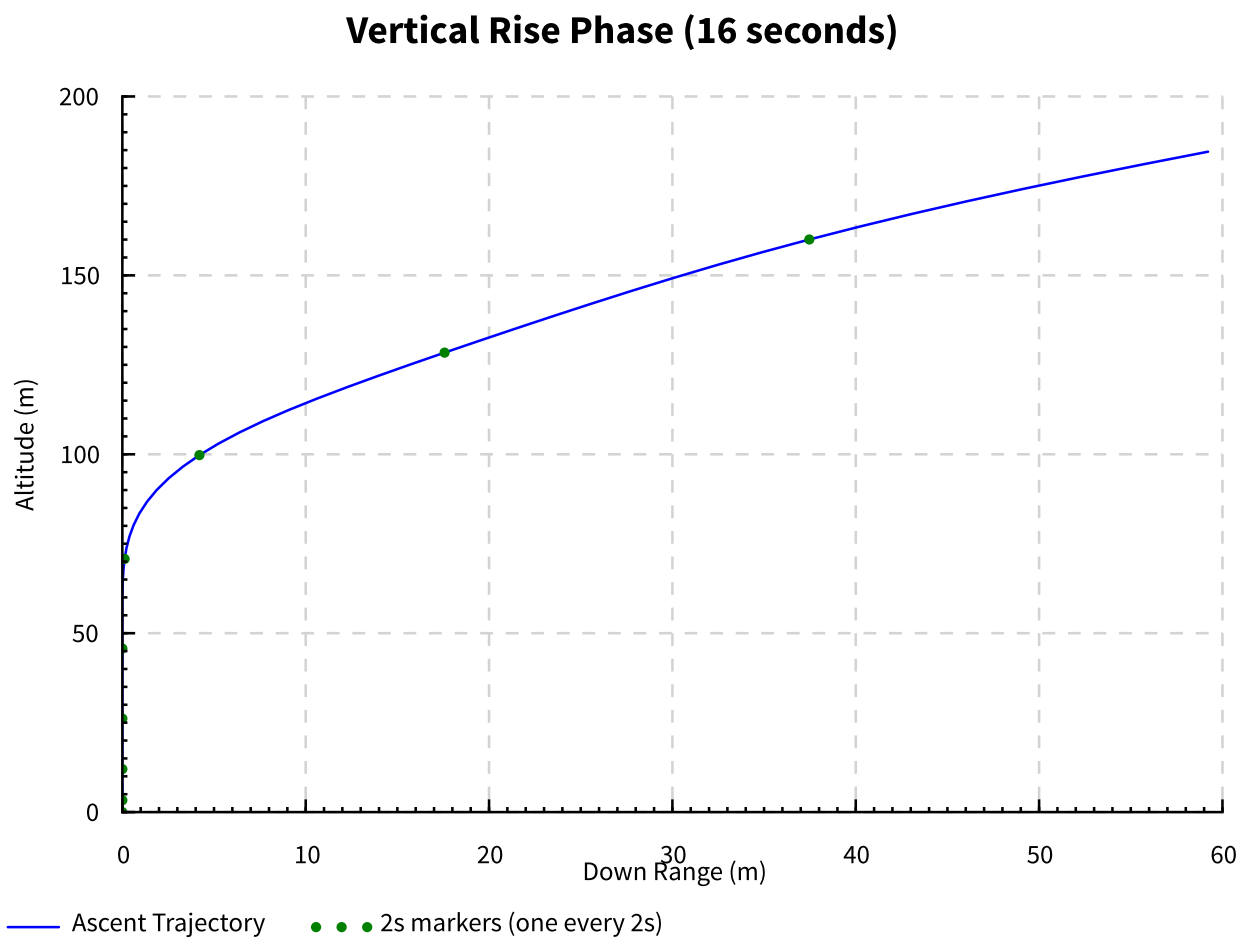

Figure 2.4: The first 16 s of lunar ascent, including the vertical rise phase.

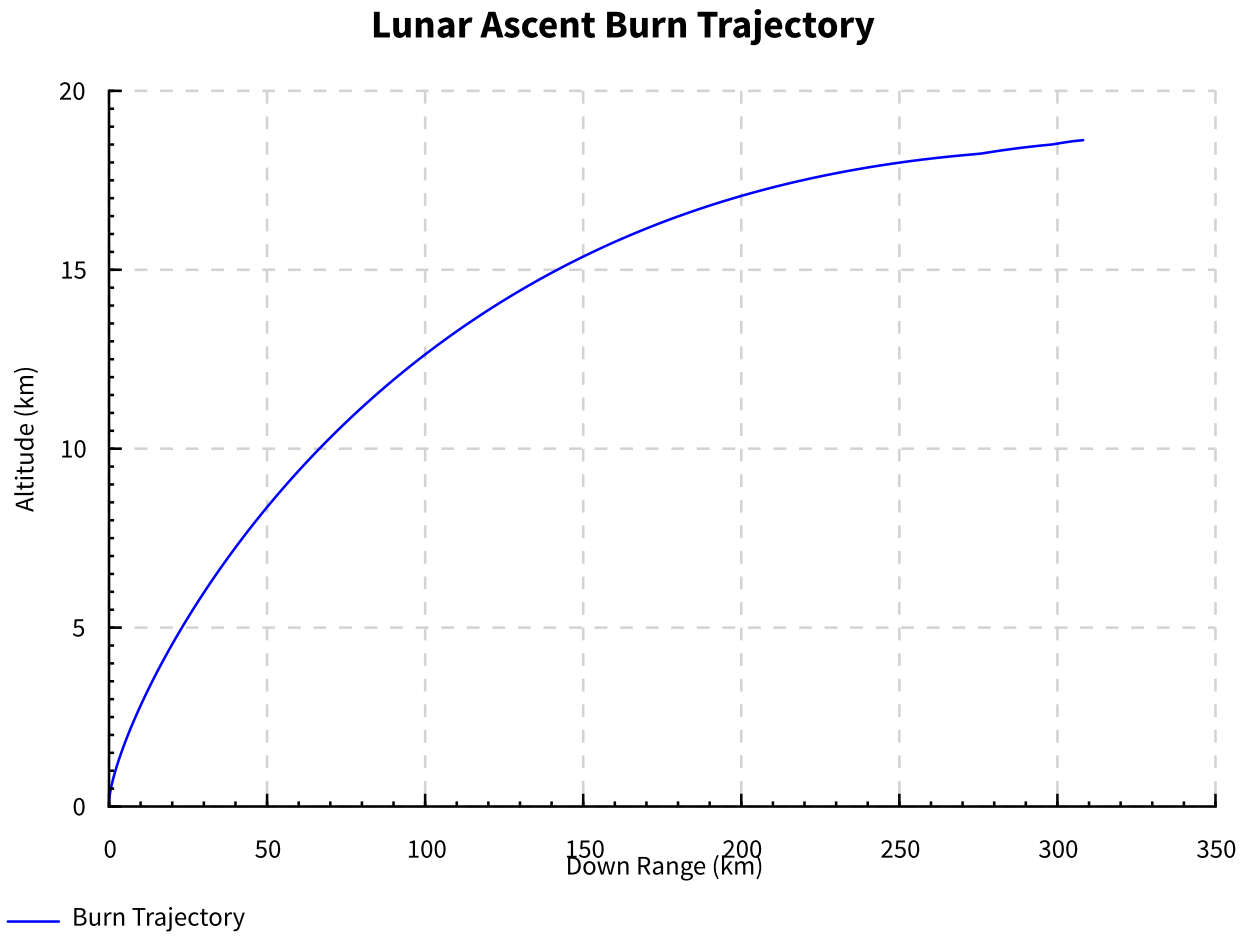

Figure 2.5: The entire burn phase of the lunar ascent, shown using altitude-downrange coordinates. The altitude is the radial height above the moon surface, while downrange is the arc-length distance travelled when projected radially downward onto the lunar surface.

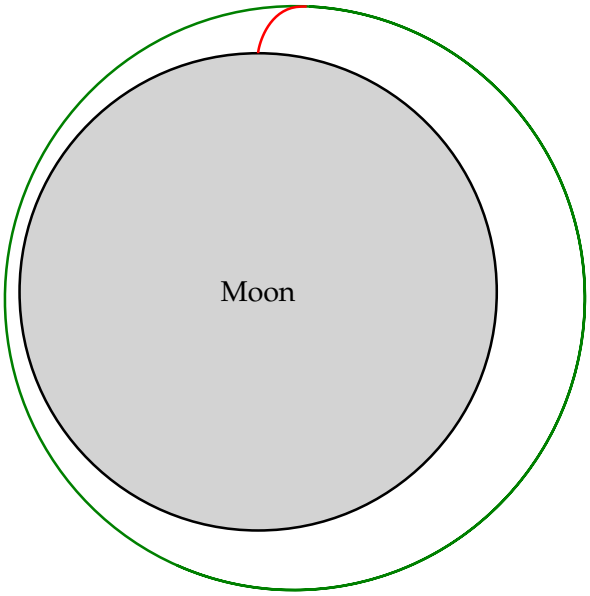

Figure 2.6: Moon-centred view of the burn and coasting phases. The radial (altitude) scale above the lunar surface is magnified by a factor of 20.

3

Rocket Staging and the Tsiolkovsky Rocket Equation

Rockets that deliver payloads to Earth orbit usually have multiple stages. There are several reasons for using multi-stage rockets, including:

- Reduced fuel use. This is achieved by discarding portions of the rocket that were only used to support the fuel that was previously being carried.
- Selection of rocket engines that are optimised for different parts of the atmosphere or a vacuum. For example, the optimal nozzle geometry of liquid fuel rocket engines depends on the ambient pressure [10, 11].
- Economic factors. Earlier rocket stages can often be returned to Earth, recovered and re-used more cheaply than a single stage, due to lower mass and less downrange displacement.

We will examine the first of these benefits by considering the Tsiolkovsky Rocket Equation. This equation is often used as the basis of high-level discussions of mission planning and rocket capabilities.

This chapter of the workshop introduces many basic approximations that are frequently used for simulating rockets. We will build a simulation comparing the behaviour of a single-stage and two-stage rocket, to gain experience setting up and solving equations of motion. Then we will analyse the situation analytically to derive the Tsiolkovsky Rocket Equation.

3.1 Propellant Mass Fraction

We shall define the propellant mass fraction of a rocket stage, ζ, as the available mass of propellant, m_p, divided by the total mass of the stage:

$$\zeta = \frac{m_p}{m_p + m_d} \tag{3.1}$$

where m_d is the "dry mass" of the stage (ie. the mass without any propellant). For real rocket systems, this equation may need to be modified to incorporate factors such as liquid propellant residuals and reserves [12], but we ignore those components here for simplicity. Rearranging to solve for m_d gives:

$$m_d = m_p \left(\frac{1}{\zeta} - 1 \right) \tag{3.2}$$

The dry mass, m_d, is the mass of the entire stage without fuel, and thus accounts for the mass of components such as the engines, the fuel tanks, and the payload. However, the payload mass is only approximately 30% of the dry mass on average [13], so most of the dry mass is consumed by non-payload components. This percentage depends heavily on the mission type [14], with wide variation, but it demonstrates the importance of the overall structure of the rocket compared with the payload.

Holt and Monk [12] provide some plots of typical propellant mass fractions for various launch systems. As a nominal figure we will choose $\zeta = 0.9$ for our simulations.

3.2 Specific Impulse

In our simulation, we will specify that the rocket is burning a bipropellant mixture of liquid oxygen (LOx) and Rocket Propellant-1 (RP1). RP1 is a refined kerosene. This is the propellant system reported to be used by SpaceX Merlin engines.

A commonly-used measure for the performance of a rocket engine is its specific impulse [11], I_s. This represents not just the propellant but the entire engine design. Thus, an I_s figure could vary somewhat between engines even if they were to use identical propellant. However, bearing this in mind, specific impulse values can be used in the design phase as representative of the performance of a particular propellant, assuming some near-optimal engine were to be provided later. The magnitude of the thrust force, F_T, is related to the specific impulse by:

$$F_T = g_0 I_s \dot{m} \tag{3.3}$$

where g_0 is the standard acceleration due to gravity and $\dot{m}$ is the mass flow rate of propellant (the rate at which propellant, including both the base fuel and oxidiser, is being burned).

Parameter	Symbol	Value	Units
Total propellant mass	m_p	500 000	kg
Mass flow rate	$\dot{m}$	290	kg/s
Specific impulse	I_s	300	s
Propellant mass fraction	ζ	0.9	

Table 3.1: Rocket parameters for staging simulation.

Parameter	Symbol	Value	Units
Propellant mass	m_p	500 000	kg
Dry mass	m_d	55 556	kg

Table 3.2: Single-stage starting mass parameters.

In the current discussion, this figure is valid only for a vacuum. Under conditions of atmospheric pressure at sea level, the effective I_s, and thus the thrust produced, is typically reduced by around 10% [15].

We will specify a LOx-RP1 engine with an $I_s = 300$ s [11].[1]

3.3 Simulating Staging

Now let's compare the effect of using one stage vs two for a rocket carrying the same total amount of propellant, with a fixed propellant mass fraction for each stage. We will simulate the rocket burning all its propellant from a zero-velocity start, in a vacuum with no gravitational effects. Table 3.1 shows the parameters for the simulation. In the single stage simulation, the propellant mass will be carried in just one stage (Table 3.2), while for the two-stage simulation, it will be split evenly between stages (Table 3.3).

[1] There are sources on the web, particularly relating to the SpaceX Merlin engines, that quote a figure more like 350 s. However, we were unable to find any high-quality academic or NASA-endorsed references to support this in the time prior to the workshop. The reference we quote has a foreword written by Wernher von Braun, so it seems rather more trustworthy albeit possibly out-of-date.

		Value		
Parameter	Symbol	Stage 1	Stage 2	Units
Propellant mass	m_p	250 000	250 000	kg
Dry mass	m_d	27 778	27 778	kg

Table 3.3: Two-stage starting mass parameters.

The state space for the simulation will represent the propellant mass, position and velocity:

$$\mathbf{x} = \begin{bmatrix} m_p \\ r \\ v \end{bmatrix} \tag{3.4}$$

While the equation of motion of the system is:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{m}_p \\ \dot{r} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} -\dot{m} \\ v \\ g_0 I_s \dot{m}/(m_p + m_d + m_r) \end{bmatrix} \tag{3.5}$$

where the new symbol, m_r, is the total mass of all remaining stages. The rate of change of the propellant is the mass flow rate, the rate of change of position is velocity, and the rate of change of velocity is the thrust force (Eq 3.3) divided by the total instantaneous mass. The burn for each stage will be conducted for $(m_p/\dot{m})$ s, which is the time required to burn all the propellant.

Problem 5: Comparison of staging scenarios.

In the file `Staging.hs`,

- implement `equationOfMotion`
- implement `burnStage`
- implement `burnSingleStage`
- implement `burnTwoStage`
- run `plotVelocityComparison Screen`, to view a comparison of the velocity profiles achieved by the two different scenarios

Figure 3.1 shows the effect of staging on the velocity profiles of the two scenarios. As expected, both scenarios are identical until the time at which the first stage of the two-stage scenario is jettisoned. After ejection of the first stage dry mass, the velocity profile of the two-stage rocket increases compared with the single stage rocket.

3.4 Tsiolkovsky Rocket Equation

We can derive the famous Tsiolkovsky Rocket Equation from Eq 3.5. Start by subsituting the multiple mass components in the acceleration term by a single instantaneous mass:

$$\dot{v} = \frac{g_0 I_s \dot{m}}{m} \tag{3.6}$$

If we burn mass at a uniform rate then the instantaneous mass, m, can be written as:

$$m(t) = m_i - \dot{m}t \tag{3.7}$$

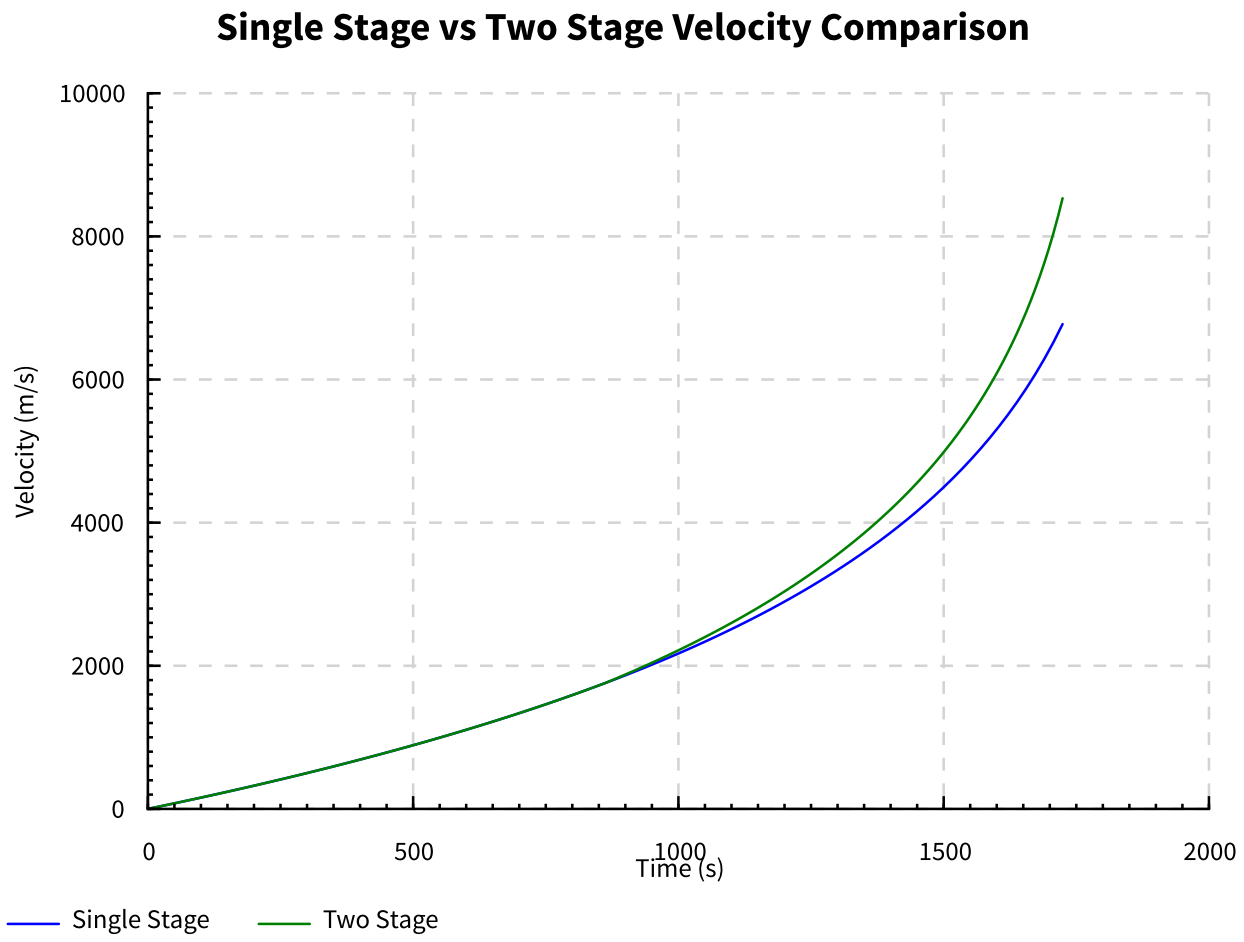

Figure 3.1: Velocity profile of the different staging scenarios. The two-stage rocket achieves a substantially higher final velocity despite the same propellant mass.

where m_i is the initial mass. Then we can integrate this equation to find the change in velocity due to the burn (final velocity, v, minus initial velocity, v_i) as follows:

$$v = \int_0^{t_f} \dot{v}\, dt \tag{3.8}$$

$$= \int_0^{t_f} \frac{g_0 I_s \dot{m}}{m_i - \dot{m}t}\, dt \tag{3.9}$$

$$= g_0 I_s \dot{m} \left[\frac{1}{-\dot{m}} \ln\left(m_i - \dot{m}t\right) \right]_0^{t_f} + v_i \tag{3.10}$$

$$= g_0 I_s \left(\ln m_i - \ln(m_i - \dot{m}t_f)\right) + v_i \tag{3.11}$$

$$v - v_i = \Delta v = g_0 I_s \ln\left(\frac{m_i}{m_f}\right) \tag{3.12}$$

where t_f is the duration or final time of the burn. m_f is the final mass, given by $m_f = m_i - \dot{m}t_f$. Eq 3.12 is the famous Tsiolkovsky Rocket Equation.

The quantity Δv ("Delta Vee") is the change in vehicle velocity that would occur under the conditions we have described above: a vacuum burn with no gravity. This quantity expresses the "magnitude" of a spacecraft manouevre without referencing the mass of the system, which is of course always changing. Thus, it's a popular way to gauge the requirements of various manouevres in mission planning (eg. [7]). Due to the logarithm that appears in the equation, Δv requirements of multiple manouevres compose by addition.

We can thus verify our numerical Δv findings from the one- and two-stage rocket configurations. For the single-stage rocket:

$$\Delta v = g_0 I_s \ln\left(\frac{m_p + m_d}{m_d}\right) \tag{3.13}$$

$$= 9.81\,\text{m/s}^2 \times 300\,\text{s} \times \ln\left(\frac{500\,000\,\text{kg} + 55\,556\,\text{kg}}{55\,556\,\text{kg}}\right) \tag{3.14}$$

$$= 6776\,\text{m/s} \tag{3.15}$$

And for the two-stage rocket, we simply add the Δv from burning each stage (bearing in mind the slightly-different definitions of m_p and m_d):

$$\Delta v = g_0 I_s \left[\ln\left(\frac{2m_p + 2m_d}{m_p + 2m_d}\right) + \ln\left(\frac{m_p + m_d}{m_d}\right) \right] \tag{3.16}$$

$$= 8536\,\text{m/s} \tag{3.17}$$

4

Hohmann Transfers

In this chapter, we will set up our first 2D spacecraft manoeuvre in which the available analytical solution is only approximate (but still close enought to provide a guide).

We will set up a Hohmann transfer, which is an elliptical transfer between two coplanar circular orbits [6, 16]. The geometry of the manoeuvre is shown below in Figure 4.3. The spacecraft will start in the inner circular orbit. A short burn is performed which changes the velocity of the spacecraft by Δv_1, placing it into an elliptical transfer orbit of exactly the right size to intercept the outer orbit. Finally, when the intersection with the outer orbit is reached, another short burn is performed, changing the velocity by Δv_2 to enter the outer orbit.

Our analytical treatment of this manoeuvre will allow us to determine the approximate values of Δv_1 and Δv_2. In doing so, we make the approximations that an instantaneous impulse can be delivered to the spacecraft, and that its mass does not change, so that its motion obeys Kepler's laws at all times. In reality, this instantaneous impulse becomes a burn that is merely very short compared with the orbital periods involved, and the reaction mass used is small compared with the vehicle mass.

The analytical derivation in this chapter involves some mathematics that is more involved than previous chapters. Feel free to skip it if necessary, since only the results for Δv_1 and Δv_2 are important for the numerical simulation. We provide it because a short yet complete treatment is somewhat difficult to find online.

4.1 Acceleration in Polar Coordinates

To examine the Kepler problem in the next section, it is convenient to use polar coordinates. This is because the only force acting on the body is gravity, which is exerted only in the radial direction, and no force is exerted along the angular or circumferential direction. This results in more tractable solutions to the equations of motion.

Figure 4.1 shows a pair of 2D coordinate systems: an inertial system with orthogonal unit vectors $\mathbf{e_x}$ and $\mathbf{e_y}$, and a non-inertial system, with instantaneous orthogonal unit vectors $\mathbf{e_r}$ and $\mathbf{e}_\theta$, which correspond to the instantaneous directions of a polar coordinate system. A point is illustrated at a distance r from the origin, at angle θ from the x axis. Notice that if the point moves, then the basis vectors $\mathbf{e_r}$ and $\mathbf{e}_\theta$ would also change. In order to express equations of motion in these coordinates, we must account for the time-dependence of $\mathbf{e_r}$ and $\mathbf{e}_\theta$, which is described in this section.

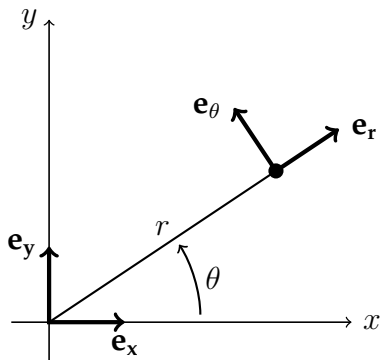

Figure 4.1: Polar coordinates and polar unit vectors.

We can write $\mathbf{e_r}$ and $\mathbf{e}_\theta$ in terms of $\mathbf{e_x}$ and $\mathbf{e_y}$ as follows:

$$\mathbf{e_r} = \mathbf{e_x}\cos\theta + \mathbf{e_y}\sin\theta \tag{4.1}$$

$$\mathbf{e}_\theta = -\mathbf{e_x}\sin\theta + \mathbf{e_y}\cos\theta \tag{4.2}$$

This provides our fundamental link to an inertial frame of reference, which is the purpose of using $\mathbf{e_x}$ and $\mathbf{e_y}$. Now consider the derivatives of these unit vectors with respect to the angle θ (and notice that they have no dependence on r):

$$\frac{d\mathbf{e_r}}{d\theta} = -\mathbf{e_x}\sin\theta + \mathbf{e_y}\cos\theta = \mathbf{e}_\theta \tag{4.3}$$

$$\frac{d\mathbf{e}_\theta}{d\theta} = -\mathbf{e_x}\cos\theta - \mathbf{e_y}\sin\theta = -\mathbf{e_r} \tag{4.4}$$

Relative to the inertial frame of reference, these unit basis vectors depend only on θ, so the time derivatives can be expressed using the chain rule as follows:

$$\dot{\mathbf{e}}_\mathbf{r} = \frac{d\mathbf{e_r}}{d\theta}\frac{d\theta}{dt} = \dot{\theta}\mathbf{e}_\theta \tag{4.5}$$

$$\dot{\mathbf{e}}_\theta = \frac{d\mathbf{e}_\theta}{d\theta}\frac{d\theta}{dt} = -\dot{\theta}\mathbf{e_r} \tag{4.6}$$

To describe equations of motion in this coordinate system, we will require an expression for acceleration, which we will derive from expressions for position and velocity. Position, **r**, is given very simply by:

$$\mathbf{r} = r\mathbf{e_r} \tag{4.7}$$

Velocity, **v**, is the time derivative of position, which we find using the chain rule and then substituting Eq 4.5 and Eq 4.6:

$$\begin{aligned}
\mathbf{v} &= \frac{d\mathbf{r}}{dt} && (4.8)\\
&= \frac{d}{dt} r\mathbf{e_r} && (4.9)\\
&= \dot{r}\mathbf{e_r} + r\dot{\mathbf{e}}_\mathbf{r} && (4.10)\\
&= \dot{r}\mathbf{e_r} + r\dot{\theta}\mathbf{e}_\theta && (4.11)
\end{aligned}$$

Finally, acceleration, **a**, is the time derivative of velocity, again substituting Eq 4.5 and Eq 4.6:

$$\begin{aligned}
\mathbf{a} &= \frac{d\mathbf{v}}{dt} && (4.12)\\
&= \frac{d}{dt}\left(\dot{r}\mathbf{e_r} + r\dot{\theta}\mathbf{e}_\theta\right) && (4.13)\\
&= \ddot{r}\mathbf{e_r} + \dot{r}\dot{\mathbf{e}}_\mathbf{r} + \dot{r}\dot{\theta}\mathbf{e}_\theta + r\ddot{\theta}\mathbf{e}_\theta + r\dot{\theta}\dot{\mathbf{e}}_\theta && (4.14)\\
&= \ddot{r}\mathbf{e_r} + \dot{r}\dot{\theta}\mathbf{e}_\theta + \dot{r}\dot{\theta}\mathbf{e}_\theta + r\ddot{\theta}\mathbf{e}_\theta - r\dot{\theta}^2\mathbf{e_r} && (4.15)\\
&= \left(\ddot{r} - r\dot{\theta}^2\right)\mathbf{e_r} + \left(r\ddot{\theta} + 2\dot{r}\dot{\theta}\right)\mathbf{e}_\theta && (4.16)
\end{aligned}$$

4.2 Kepler Problem and Elliptical Orbit Solutions

The Kepler problem concerns the motion of a particle under the action of a central force whose magnitude varies according to an inverse-square law. In the current context, the central force is gravity, exerted by a dominant mass such as a planet, acting on a much less massive object such as a space vehicle.

Gravity acts only in the radial direction, with a force $\mathbf{F_G}$:

$$\mathbf{F_G} = -\frac{GMm}{r^2}\mathbf{e_r} \tag{4.17}$$

in which G is the gravitational constant, M is the dominant mass (eg. the mass of a planet or moon), and m is the much smaller mass, such as a space vehicle. To simplify the equation, we can introduce the standard gravitational parameter for a celestial body, μ:

$$\mu = GM \tag{4.18}$$

This parameter is used frequently in astrodynamics formulas (eg. [4]):

$$\mathbf{F_G} = -\frac{\mu m}{r^2}\mathbf{e_r} \tag{4.19}$$

Since we neglect all forces except gravity, the entire equation of motion of the vehicle can be expressed in polar coordinates using Equations 4.16 and 4.19:

$$-\frac{\mu}{r^2}\mathbf{e_r} = \left(\ddot{r} - r\dot{\theta}^2\right)\mathbf{e_r} + \left(r\ddot{\theta} + 2\dot{r}\dot{\theta}\right)\mathbf{e}_\theta \tag{4.20}$$

We can now perform some mathematical substitutions to solve this equation of motion. We shall solve for the two different vector components in sequence.

4.2.1 Circumferential Component $\mathbf{e}_\theta$

The circumferential component has no force acting on it:

$$0 = r\ddot{\theta} + 2\dot{r}\dot{\theta} \tag{4.21}$$

At this point we can observe that:[1]

$$\begin{aligned} 0 &= r\ddot{\theta} + 2\dot{r}\dot{\theta} && (4.22)\\ &= \frac{1}{r}\frac{d}{dt}\left(r^2\dot{\theta}\right) && (4.23)\\ &= \frac{d}{dt}\left(r^2\dot{\theta}\right) && (4.24) \end{aligned}$$

This implies that the quantity whose time derivative is zero is constant:

$$r^2\dot{\theta} = h_\omega \text{ is constant} \tag{4.25}$$

where h_ω happens to correspond to the specific angular momentum:

$$\begin{aligned} h_\omega &= r^2\dot{\theta} && (4.26)\\ &= \frac{|\mathbf{r}\times\mathbf{p}|}{m} && (4.27)\\ &= |\mathbf{r}\times\mathbf{v}| && (4.28) \end{aligned}$$

in which $\mathbf{p}$ is the linear momentum, and the cross-product $(\mathbf{r}\times\mathbf{p})$ is the angular momemtum. Thus, the angular momentum of bodies in the Kepler problem is constant.

4.2.2 Radial Component $\mathbf{e_r}$

The radial component is acted upon by the central gravitational force:

$$-\frac{\mu}{r^2} = \ddot{r} - r\dot{\theta}^2 \tag{4.29}$$

[1] These substitutions are a bit *deus ex machina*, but bear with it; they have a physical meaning.

In order to solve this equation, we will aim write it as a differential equation for $(1/r)$ as a function of θ. This requires some substitutions. First, notice that:

$$\frac{d}{dt}\left(\frac{1}{r}\right) = -\frac{1}{r^2}\dot{r} \tag{4.30}$$

Rearranging:

$$\dot{r} = -r^2\frac{d}{dt}\left(\frac{1}{r}\right) \tag{4.31}$$

But from Equation 4.25, we have that $r^2 = h_\omega/\dot{\theta}$, so:

$$\dot{r} = -r^2\frac{d}{dt}\left(\frac{1}{r}\right) \tag{4.32}$$

$$= -\frac{h_\omega}{\dot{\theta}}\frac{d}{dt}\left(\frac{1}{r}\right) \tag{4.33}$$

$$= -h_\omega\frac{dt}{d\theta}\frac{d}{dt}\left(\frac{1}{r}\right) \tag{4.34}$$

$$= -h_\omega\frac{d}{d\theta}\left(\frac{1}{r}\right) \tag{4.35}$$

Now we can differentiate this expression with respect to time to find an expression for $\ddot{r}$:

$$\ddot{r} = -h_\omega\frac{d^2}{d\theta^2}\left(\frac{1}{r}\right)\dot{\theta} \tag{4.36}$$

and again from Eq 4.25, we have $\dot{\theta} = h_\omega/r^2$, which we substitute to obtain:

$$\ddot{r} = -\frac{{h_\omega}^2}{r^2}\frac{d^2}{d\theta^2}\left(\frac{1}{r}\right) \tag{4.37}$$

Now we substitute this back into the expression for the radial component, Eq 4.29, also using the identity $r\dot{\theta}^2 = {h_\omega}^2/r^3$ from Eq 4.25:

$$-\frac{\mu}{r^2} = \ddot{r} - r\dot{\theta}^2 \tag{4.38}$$

$$= -\frac{{h_\omega}^2}{r^2}\frac{d^2}{d\theta^2}\left(\frac{1}{r}\right) - \frac{{h_\omega}^2}{r^3} \tag{4.39}$$

This is finally rearranged into the form we require:

$$\frac{d^2}{d\theta^2}\left(\frac{1}{r}\right) + \frac{1}{r} = \frac{\mu}{{h_\omega}^2} \tag{4.40}$$

This is a linear, *second order* ODE, of the form:

$$\frac{d^2}{dx^2}f(x) + f(x) = C \tag{4.41}$$

where C is a constant. This equation has the general solution:

$$f(x) = C\left(1 + e\cos(x + \psi)\right) \tag{4.42}$$

where e and ψ are constants of integration. It can be confirmed that this is a solution to Equation 4.41 by directly differentiating and substituting. Thus the solution for Equation 4.40 is:

$$\frac{1}{r} = \frac{\mu}{{h_\omega}^2}\left(1 + e\cos(\theta + \psi)\right) \tag{4.43}$$

We can choose to define the datum for angle θ such that $\psi = 0$ (which corresponds to $\theta = 0$ when r is minimum), and rearrange to find the solution for r:

$$r = \frac{{h_\omega}^2/\mu}{1 + e\cos\theta} \tag{4.44}$$

This equation happens to be the equation for a conic section.

4.2.3 Circular and Elliptical Orbits

The solutions to Equation 4.44 can be categorized according to the value of the parameter, e, which is called the eccentricity:

$e = 0$	circular orbit
$e < 1$	elliptical orbit
$e = 1$	parabolic orbit
$e > 1$	hyperbolic orbit

For Hohmann transfers, we are only concerned with circular and elliptical orbits.

4.3 Hohmann Transfer Velocities

Figure 4.2 illustrates the geometry of an ellipse. The dimensions for its semi-minor axis, a, and semi-major axis, b, are shown along with the radial distances of the periapsis (closest distance), r_p, and apoapsis (furthest distance), r_a, of an orbit with the more massive body at the focus.

From Equation 4.44, we can write the radius of an elliptical orbit at perapsis, r_p (when $\theta = 0$), and apoapsis, r_a (when $\theta = \pi$), in terms of the orbital parameters, as:

$$r_p = \frac{{h_\omega}^2}{\mu(1+e)} \qquad\qquad r_a = \frac{{h_\omega}^2}{\mu(1-e)} \tag{4.45}$$

From Equation 4.25, we can express the velocity at the apses, when the radius and velocity are exactly perpendicular, as:

$$v^2 = \frac{{h_\omega}^2}{r^2} \text{ at the apses only} \tag{4.46}$$

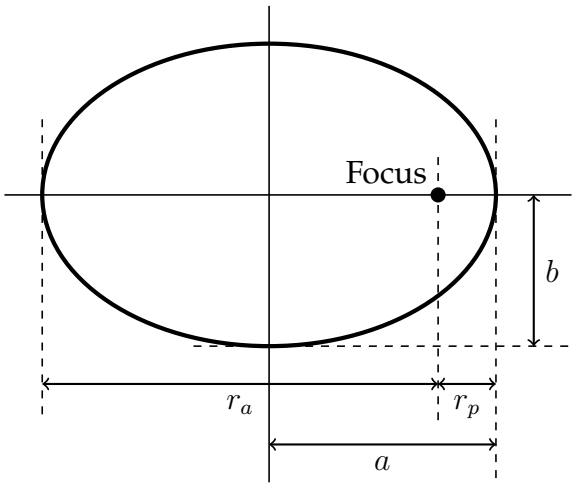

Figure 4.2: Ellipse geometric parameters.

Substituting Eq 4.45 into this expression yields velocities at periapsis and apoapsis:

$$v_p{}^2 = \frac{\mu(1+e)}{r_p} \qquad\qquad v_a{}^2 = \frac{\mu(1-e)}{r_a} \tag{4.47}$$

We now introduce geometric relationships between the apsis radii and the eccentricity:

$$e = 1 - \frac{r_p}{a} \qquad\qquad e = \frac{r_a}{a} - 1 \tag{4.48}$$

Which can be substituted to yield:[2]

$$v_p{}^2 = \mu\left(\frac{2}{r_p} - \frac{1}{a}\right) \qquad\qquad v_a{}^2 = \mu\left(\frac{2}{r_a} - \frac{1}{a}\right) \tag{4.49}$$

Figure 4.3 illustrates the geometry of the Hohmann transfer itself. We will initially consider a transfer from the inner orbit to the outer one. A velocity change, Δv_1, is applied to the inner circular orbit to enter the elliptical transfer orbit at its periapsis. Then, once the apoapsis of the transfer orbit is reached, another velocity change, Δv_2, is applied to enter the outer orbit.

The velocity of the transfer orbit at its periapsis is given by the relationship for periapsis velocity in an elliptical orbit, substituting the radii in the diagram:

$$v_p{}^2 = \mu\left(\frac{2}{r_1} - \frac{2}{r_1 + r_2}\right) \tag{4.50}$$

Similarly for the transfer orbit velocity at apoapsis:

$$v_a{}^2 = \mu\left(\frac{2}{r_2} - \frac{2}{r_1 + r_2}\right) \tag{4.51}$$

[2]This relationship actually holds for any radius (it is referred to as the *vis-viva equation*), but we chose to derive it at these geometric points only, because this allows us to forego a description of orbit energies, making our derivation here slightly shorter.

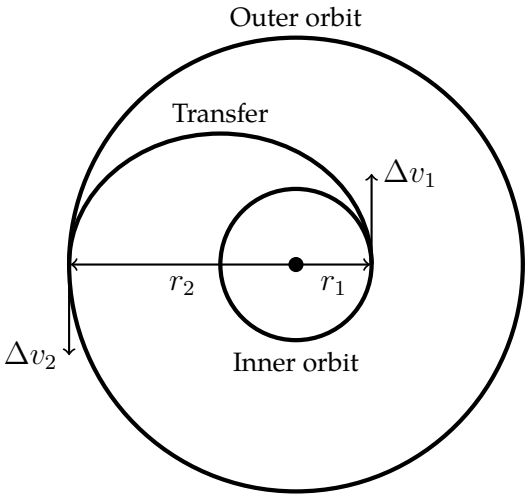

Figure 4.3: Hohmann transfer geometry. The outer and inner orbits are circular, while the transfer orbit is elliptical. All three orbits share the same focus.

The velocity in both circular orbits is given by the same velocity relationships when $e = 0$:

$$v^2 = \frac{\mu}{r} \tag{4.52}$$

Thus, we can write the required velocity change in the first burn, Δv_1, as:

$$\Delta v_1 = v_p - v_i \tag{4.53}$$

$$= \sqrt{\mu \left(\frac{2}{r_1} - \frac{2}{r_1 + r_2} \right)} - \sqrt{\frac{\mu}{r_1}} \tag{4.54}$$

$$= \sqrt{\mu \frac{2r_1 + 2r_2 - 2r_1}{r_1(r_1 + r_2)}} - \sqrt{\frac{\mu}{r_1}} \tag{4.55}$$

$$= \sqrt{\frac{\mu}{r_1}} \left(\sqrt{\frac{2r_2}{r_1 + r_2}} - 1 \right) \tag{4.56}$$

In the same way, the required velocity change in the second burn, Δv_2, is:

$$\Delta v_2 = v_f - v_a \tag{4.57}$$

$$= \sqrt{\frac{\mu}{r_2}} \left(1 - \sqrt{\frac{2r_1}{r_1 + r_2}} \right) \tag{4.58}$$

These final two equations, Eq 4.56 and Eq 4.58, are the results we will require in order to simulate the Hohmann transfer manoeuvre.

4.4 Simulating the Hohmann Transfer

To simulate the Hohmann transfer, we will separate the manoeuvre into several parts. We will examine the behaviour of the inner and outer orbits as coasting manoeuvres, and we

will split the transfer orbit into three parts as shown in Figure 4.4: a first burn, followed by a coasting trajectory, followed by a second burn.

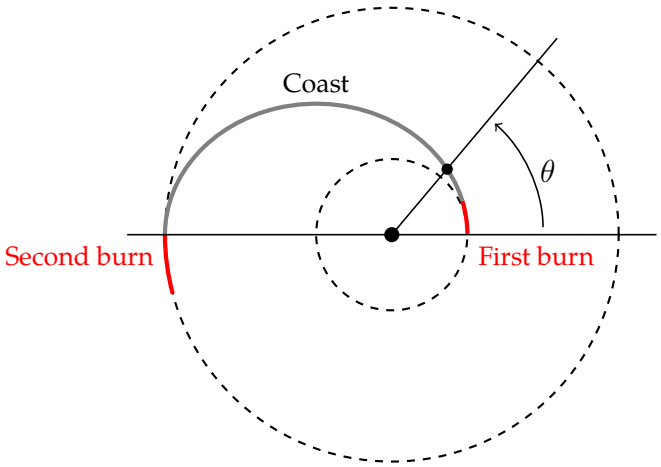

Figure 4.4: Simulated burns for the Hohmann transfer (red) and coasting trajectory (black). The first burn will start at $\theta = 0$, while the second burn will start at $\theta = \pi$. Both burns will be oriented along the instantaneous velocity vector of the craft.

We will model a rocket engine which has a constant thrust (ie. it is not throttleable), so that the burn time is determined by reaching the pre-computed velocity for each given orbit.

4.4.1 Terminating ODE Integration

So far, when integrating ODEs, we have supplied a list of times at which the ODE should be evaluated, from which the ODE driver has computed a list of time steps. Now, we want to simulate a scenario in which we don't know the termination time. Instead, the termination of ODE integration will be determined by the ODE state; in this case, when the magnitude of the velocity reaches a pre-determined value.

To achieve this, we will modify the function being integrated to return an *optional* result: `(time, state) -> Maybe (time :-* diff)`. A return value of `Nothing` indicates that the ODE has run past its termination.

In addition, we will supply a means to ensure we determine the termination time with a specified accuracy, t_ϵ. This will be achieved by halving the time step successively at the end of the integration until the time step that is taken, h, satisfies $h < t_\epsilon$.

Problem 6: Terminating ODE Integration.

In the file `ODE.hs`,

- implement `rk4StepTerminating`, which modifies the RK4 step to handle the possibility of the gradient function returning `Nothing` at any step
- implement `integrateTerminating`, which performs terminating integration, bisecting the step at the end to achieve a given accuracy in termination time

In `ODEExamples.hs`,

- run `plotVerticalThrow Screen`, to view a plot of integrating an ODE representing a vertical throw, which terminates when the velocity reaches zero

Figure 4.5 shows an example of an ODE terminating itself using `integrateTerminating`. The ODE simulates throwing an object vertically upward in a constant gravitational field. The initial conditions supply a positive vertical velocity, and the ODE is set up to terminate when the velocity reaches zero. The time step of the simulation is set to 0.5 s, but near the end, the halving of the time step is visible as it iterates toward a more accurate completion time. In this simulation, $t_\epsilon = 0.001$ s.

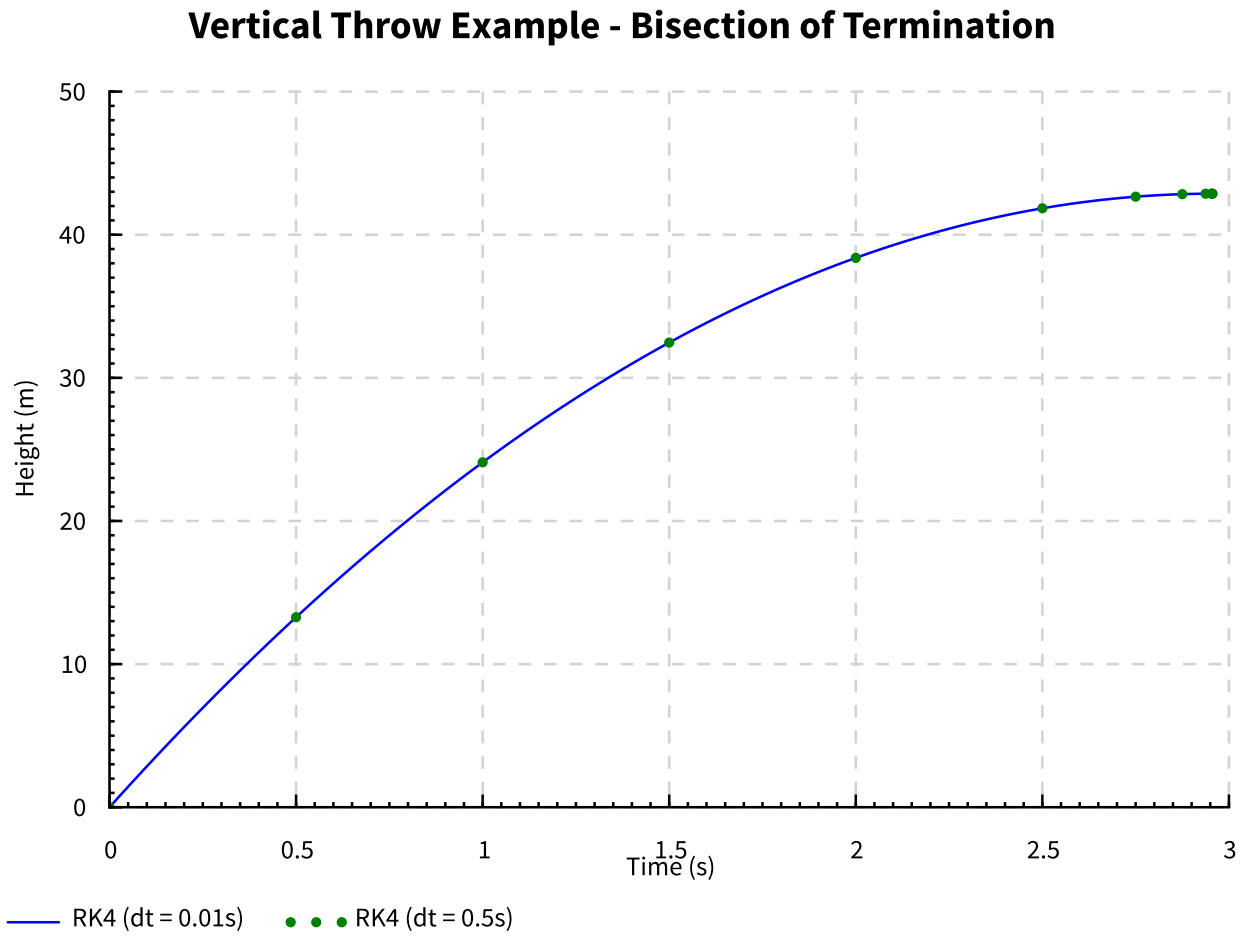

Figure 4.5: Illustration of halving the integration time step to find a more accurate completion time of a simulation.

In this case, we are using a binary signal to indicate termination. There are other possibilities, such as supplying a locally-monotonic function, which allow more efficient isolation of termination by root finding.

4.4.2 Equations of Motion

The state for the Hohmann transfer simulation will include mass, m, distance travelled, d, vector position, **r**, and vector velocity, **v**:

$$\mathbf{x} = \begin{bmatrix} m \\ d \\ \mathbf{r} \\ \mathbf{v} \end{bmatrix} \tag{4.59}$$

Distance travelled is tracked so that we can more easily determine when to terminate the numerical simulations of the circular orbits. We terminate the circular orbit simulations when they reach particular angles, but since angles are periodic (ie. $\theta = 0 = 2\pi$), we use the distance traveled to roughly isolate which orbital pass we happen to be on.

Coasting Trajectory

The equation of motion for a coasting trajectory is given as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{m} \\ \dot{d} \\ \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} 0 \\ |\mathbf{v}| \\ \mathbf{v} \\ \mathbf{F_G}/m \end{bmatrix} \tag{4.60}$$

where $\mathbf{F_G}$, the force due to gravity, is given by:

$$\mathbf{F_G} = -\frac{\mu m}{|\mathbf{r}|^2}\frac{\mathbf{r}}{|\mathbf{r}|} \tag{4.61}$$

Burn Trajectory

Each burn will be aligned with the direction of the vehicle's current velocity. The equation of motion during a burn is given as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{m} \\ \dot{d} \\ \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} -\dot{m} \\ |\mathbf{v}| \\ \mathbf{v} \\ (\mathbf{F_G} + \mathbf{F_T})/m \end{bmatrix} \tag{4.62}$$

in which the thrust force, $\mathbf{F_T}$, is:

$$\mathbf{F_T} = g_0 I_s \dot{m} \frac{\mathbf{v}}{|\mathbf{v}|} \tag{4.63}$$

$\dot{m}$ is constant during the burn.

Problem 7: Hohmann transfer numerical simulation.

In the file `Hohmann.hs`,

- implement `angle`
- implement `gravity`
- implement `thrust`
- implement `terminateWhen`
- implement `coast`
- implement `burn`

In `Hohmann.hs`,

- run `plotHighImpulseBurn Screen`, to view a plot of a simulated normal high-impulse Hohmann transfer
- run `plotlowimpulseburn screen`, to view a plot of a simulated low-impulse hohmann transfer
- run `plotLowUltraImpulseBurn Screen`, to view a plot of a simulated ultra-low-impulse Hohmann transfer

Figure 4.6 shows a numerical simulation of a Hohmann transfer as it would normally be performed. The burns in this case are representative of a chemical rocket of approximately the same size as the Ascent Propulsion System (APS) of the lunar ascent stage of the Apollo missions, and a vehical of a similar mass ($I_s = 310\,\text{s}$, $\dot{m} = 5.13\,\text{kg/s}$, $m = 3000\,\text{kg}$). The burn periods are so short that they are negligible compared with the rest of the orbit, and are thus well-approximated as instantaneous impulses.

Figure 4.7 is a low-impulse scenario ($I_s = 200\,\text{s}$, $\dot{m} = 0.2\,\text{kg/s}$, $m = 3000\,\text{kg}$), in which the impulse is low enough that the burn periods are visible in the trajectory. However, a similar manouvre is still performed, and the analytical prediction is quite representative of the numerical result.

Finally, Figure 4.8 shows a much lower-impulse scenario (only 75 N acting on a 3000 kg spacecraft), resulting in an outcome which is clearly different from the analytical solution. This is expected, since the thrust is definitely not instantaneous, and large portions of the trajectory are produced under thrust conditions.

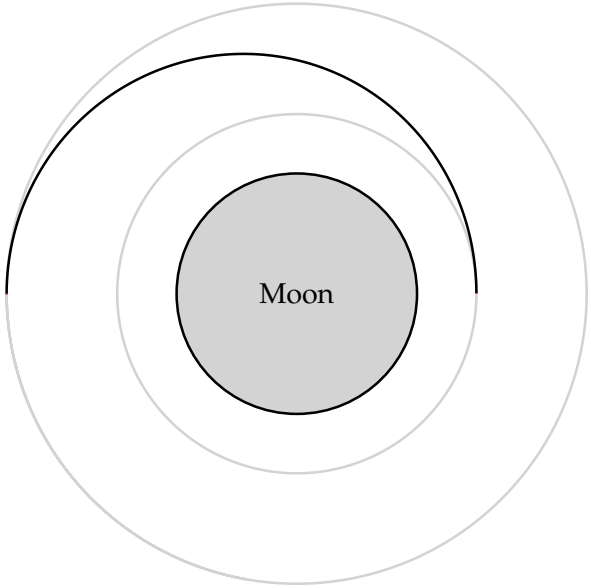

Figure 4.6: High impulse Hohmann transfer numerical simulation, using parameters representative of a chemical rocket. The nominal transfer is from 20 nautical miles to 60 nautical miles of altitude. The burn trajectories (shown in red) are so short as to be invisible. The altitude scale is magnified by a factor of 20.

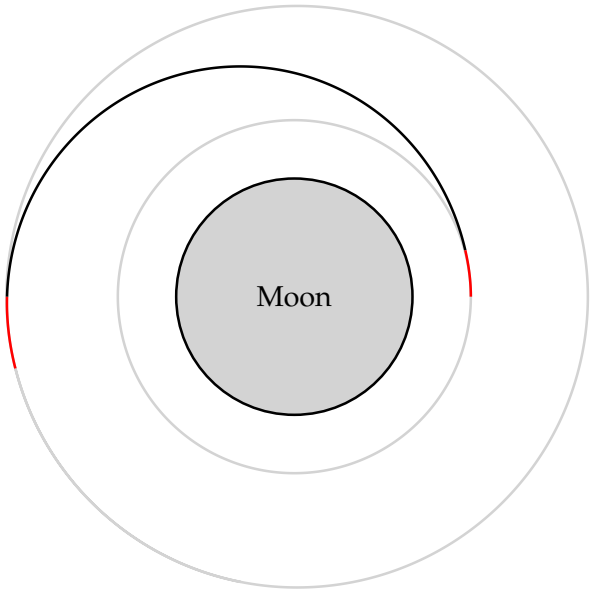

Figure 4.7: Low impulse Hohmann transfer simulation. The nominal transfer is from 20 nautical miles to 60 nautical miles of altitude. The burn trajectories are visible but still short enough that the analytical solution is valid. The altitude scale is magnified by a factor of 20.

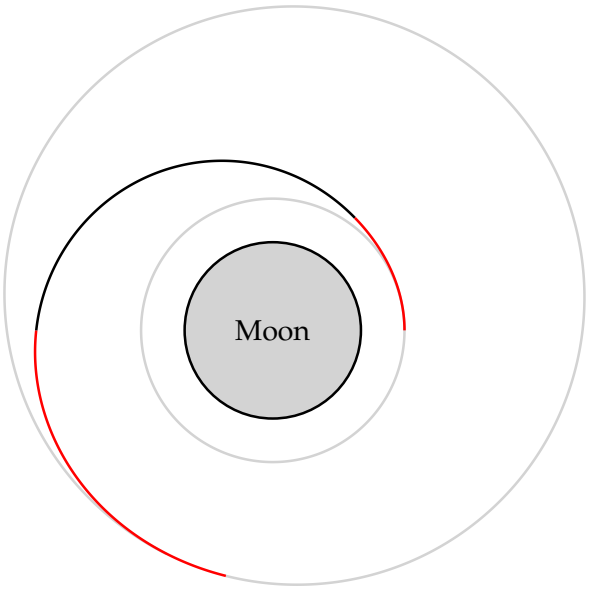

Figure 4.8: Ultra-low impulse Hohmann transfer numerical simulation. The nominal transfer is from 20 nautical miles to 60 nautical miles of altitude. Now the burn trajectories are more substantial and detrimentally affect the final orbit, so that the analytical solution is entirely non-representative of the outcome. The altitude scale is magnified by a factor of 20.

5

Suggested Projects

This chapter lists some potential projects which could follow on from the workshop.

5.1 Simulate a Launch from Earth

Simulating launches from Earth's surface is an interesting problem, since it is necessary to model atmospheric drag forces and to have some kind of guidance approach. Kephart [15] provides an excellent template for the overall simulation. Many options for guidance are possible, but an interesting approach is to use numerical optimisation to determine a desired trajectory (eg. [17, 18]). Numerical optimisation could allow for objective functions that not only minimise fuel use but also avoid other undesirable trajectory charateristics, such as high aerodynamic pressures and large down-range displacement prior to first-stage separation.

5.2 Simulate Guidance for an Asteroid Rendezvous

Guelman [19] provides a complete guidance approach for rendezvous with an asteroid. Such a rendezvous is challenging because there may be many unknowns, such as the gravitational constant, rate of rotation and shape of the asteroid. The approach taken in this paper involves a spiralling manouvre into a direct parking orbit, using a relatively low-thrust, throttleable engine. The entire guidance algorithm and dynamics of the approach can be simulated using the methods from this workshop.

5.3 Simulate a Halo Orbit

A proposal for NASA's Deep Space Gateway involves an orbit around a Lagrange point, called a Halo Orbit. Numerical simulation of these orbits is necessary for mission planning, and is a more challenging, multi-body problem. Tackling this project will involve substantial independent reading, but a good summary of the objectives and approaches is provided by Davis [20].

5.4 FRP Simulation

Investigate solving ODEs using an FRP framework instead of the more imperative approach we have taken in this workshop.

It is possible to solve ODEs using existing FRP libraries (eg. `Yampa`, `netwire`, etc), but these libraries typically only use Euler integration, which is usually regarded as too inaccurate even for basic spaceflight problems. What would be required for general scientific use is an FRP library with a pluggable integrator, with adaptive step size, which would also have to work correctly with whatever event or behaviour framework that library used. This appears to be an open problem.

Some initial work on an FRP framework specifically for numerical solution of ODEs has been done in `dynamical`.

Symbols

$\mathbf{a}$	Acceleration (m/s/s).
a	Ellipse semi-major axis (m).
b	Ellipse semi-minor axis (m).
d	Distance travelled (m).
$\mathbf{e_r}$	Unit vector in the r direction.
$\mathbf{e_x}$	Unit vector in the x direction.
$\mathbf{e_y}$	Unit vector in the y direction.
$\mathbf{e}_\theta$	Unit vector in the θ direction.
e	Eccentricity.
E	Total orbital energy (J).
f	A function.
F	Total force, scalar (N).
$\mathbf{F_G}$	Force due to gravity (N).
$\mathbf{F_T}$	Thrust force (N).
F_T	Thrust force, scalar (N).
g_0	Standard Earth gravity ($9.806\,65\,\mathrm{m/s^2}$).
G	Gravitational constant ($6.674\,08 \times 10^{-11}\,\mathrm{m^3kg/s^2}$).
h	Time step (s).
h_ω	Specific angular momentum ($\mathrm{m^2/s}$).
I_s	Vacuum specific impulse (s).
k	Spring constant (N/m).
m	Mass (kg).
m_d	Dry mass; mass of rocket stage without propellant (kg).
m_f	Final mass (kg).
m_i	Initial mass (kg).
m_p	Propellant mass (kg).
m_r	Total mass of remaining stages (kg).
M	Mass of dominant body (eg. a planet) (kg).
N	Number of moles of a substance.
N_0	Number of moles of a substance at $t = 0$.
$\mathbf{p}$	Momentum (kgm/s).
$\mathbf{r}$	Position (m).
r	Position, scalar (m).

r_0	Position, scalar, at $t = 0$ (m).
r_a	Apoapsis radius (m).
r_p	Periapsis radius (m).
r_1	Inner radius of Hohmann transfer(m).
r_2	Outer radius of Hohmann transfer(m).
t	Time (s).
t_f	Final time (s).
t_ϵ	Allowable error in time (s).
$t_{(1/2)}$	Radioactive half life (s).
$\mathbf{v}$	Velocity (m/s).
v	Velocity, scalar (m/s).
v_a	Apoapsis velocity, scalar (m/s).
v_p	Periapsis velocity, scalar (m/s).
v_i	Initial velocity, scalar (m/s).
v_f	Final velocity, scalar (m/s).
$\mathbf{x}$	System state vector.
Δv	Delta-V (m/s).
Δv_1	Velocity change for Hohmann transfer. (m/s).
Δv_2	Velocity change for Hohmann transfer. (m/s).
λ	Radioactive decay constant (1/s).
μ	Standard gravitational parameter (m^3/s^2).
θ	Angle (radians).
ω	Angular frequency (1/s).
ζ	Propellant mass fraction.

References

[1] J. L. Prince, P. N. Desai, E. M. Queen, and M. R. Grover, "Mars phoenix entry, descent and landing simulation design and modelling analysis," *Journal of Spacecraft and Rockets*, vol. 48, no. 5, pp. 754–764, 2011. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080033126.pdf

[2] G. L. Brauer, D. E. Cornick, and R. Stevenson, *Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST). Program Summary Document. NASA CR-2770*. NASA, 1977. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770012832.pdf

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[4] G. M. Levine, Ed., *Apollo: Guidance, Navigation and Control*. Charles Stark Draper Laboratory, MIT, 1971, vol. Section 5: Guidance Equations (Rev. 11). [Online]. Available: https://www.ibiblio.org/apollo/Documents/j2-80-R-567-SEC5-REV11_text.pdf

[5] H. Rein and D. Tamayo, "JANUS: a bit-wise reversible integrator for N-body dynamics," *Monthly Notices of the Royal Astronomical Society*, vol. 473, no. 3, pp. 3351–3357, 09 2017. [Online]. Available: https://arxiv.org/abs/1704.07715

[6] R. H. Battin, *An introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*. American Institute of Aeronautics and Astronautics, 1999.

[7] F. V. Bennett, "Apollo Lunar Descent and Ascent Trajectories. NASA TM X-58040," *AIAA 8th Aerospace Sciences Meeting, New York*, 1970. [Online]. Available: https://www.hq.nasa.gov/alsj/nasa58040.pdf

[8] G. E. Townsend, A. S. Abbott, and R. R. Palmer, *Guidance, Flight Mechanics and Trajectory Optimization. NASA CR-1007*. NASA, 1968, vol. VIII – Boost Guidance Equations. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19680010980.pdf

[9] M. A. Dennis, "Encyclopaedia Britannica: Charles Stark Draper." [Online]. Available: https://www.britannica.com/biography/Charles-Stark-Draper

[10] F. E. Marble, Ed., *Spacecraft Propulsion*. Space Technology Summer Institute. California Institute of Technology, 1964. [Online]. Available: https://authors.library.caltech.edu/62399/1/ST-3.pdf

[11] D. K. Huzel and D. H. Huang, *Design of Liquid Propellant Rocket Engines, 2nd Edition. NASA SP-125*. Office of Technology Utilization, NASA, 1967 (may be 1971 – date uncertain). [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19710019929.pdf

[12] J. B. Holt and T. S. Monk, "Propellant Mass Fraction Calculation Methodology for Launch Vehicles and Application to Ares Vehicles," in *AIAA Space 2009 Conference and Exposition. Session ST-2: Advanced Vehicle Systems II*, 2009. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090037584.pdf

[13] P. N. Springmann and O. L. de Weck, "Parametric scaling model for nongeosynchronous communications satellites," *Journal of Spacecraft and Rockets*, vol. 41, no. 3, pp. 472–477, 05 2004. [Online]. Available: http://web.mit.edu/deweck/www/PDF_archive/2%20Refereed%20Journal/2_3_JSR_parametric_NGSO.pdf

[14] M. W. Gerberich and S. R. Oleson, "Estimation Model of Spacecraft Parameters and Cost Based on a Statistical Analysis of COMPASS Designs," in *AIAA Space 2013 Conference and Exposition*, 2013. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140011472.pdf

[15] D. C. Kephart, "BOOST: On-Line Computer Program for Estimating Powered-Rocket Performance. R-670-PR," 1971. [Online]. Available: https://www.rand.org/content/dam/rand/pubs/reports/2006/R670.pdf

[16] S. Widnall and J. Peraire, *16.07 Dynamics, Lecture L17 – Orbit Transfers and Interplanetary Trajectories. MIT Open Courseware*. MIT, 2008. [Online]. Available: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec17.pdf

[17] M. V. Dileep, S. Kamath, and V. G. Nair, "Particle swarm optimization applied to ascent phase launch vehicle trajectory optimization problem," in *11th International Multi-Conference on Information Processing*, 2015. [Online]. Available: https://core.ac.uk/download/pdf/82182140.pdf

[18] G. A. Dukeman and A. D. Hill, "Rapid Trajectory Optimization for the ARES I Launch Vehicle," *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, Hawaii*, 2008. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080048217.pdf

[19] M. Guelman, "Guidance for asteroid rendezvous," *Journal of Guidance, Control and Dynamics*, vol. 14, no. 5, pp. 1080–1083, 1991. [Online]. Available: https://sci-hub.se/10.2514/3.20759

[20] D. C. Davis, S. M. Phillips, K. C. Howell, S. Vutukuri, and B. P. McCarthy, "Stationkeeping and Transfer Trajectory Design for Spacecraft in Cislunar Space," *AAS/AIAA Astrodynamics Specialist Conference, Columbia River Gorge, Stevenson, Washington*, 2017. [Online]. Available: https://engineering.purdue.edu/people/kathleen.howell.1/Publications/Conferences/2017_AAS_DavPhiHow.pdf