SOLVING THE OPTIMIZATION CONTROL PROBLEM FOR LUNAR SOFT LANDING USING MINIMIZATION TECHNIQUE

by

LIZETH PATRICIA OCAMPO

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTERS OF SCIENCE IN MATHEMATICS

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2013

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Benito Chen-Charpentier for the supervision, useful observations and engagement through the learning process of this master thesis; I would not have been able to complete this thesis without his guidance. Furthermore I would like to thank my committee members Dr. Hristo Kojouharov, and Dr. Christopher Zaleta for their time in reviewing my thesis. Finally I would like to thank my loved ones, my husband Justin for his support and encouragement, my parents Helman and Rosalba for teaching me perseverance, my sister Deysi and my brother in-law Brandon for helping me when I first came to the U.S.A to pursue higher education. Each one of them has played a role in my education, and I thank God for giving me such a supportive family.

July 22, 2013

Abstract

SOLVING THE OPTIMIZATION CONTROL PROBLEM FOR LUNAR SOFT LANDING USING MINIMIZATION TECHNIQUE

Lizeth Patricia Ocampo, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Benito Chen-Charpentier

Minimizing fuel consumption in lunar missions has been a well studied and documented optimization problem. In this paper two cases of the lunar lander are studied. The first case is the one dimensional problem where the objective is to make a vertical soft landing using the minimum amount of fuel. The second case has the same objective but an initial tangential velocity greater than zero is given making it a two dimensional problem.

The first case is solved using Newton's shooting method, finite difference method (using MATLAB's embedded function *bvp4c*), and solving it explicitly. For the second case, a minimization technique is proposed for cases where the above methods fail to provide a solution.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

There has never been more interest in landing on the moon than during the development of the Apollo missions. Since then, a lot of the attention of manned exploration has been turned to Mars and the possibility for deep space missions. Although currently there is little interest in putting men back on the moon, there is great interest in landing different probes on the lunar surface that would provide more information on the interior structure and composition of the moon. Some examples of these missions are the International Lunar Network that intends to set a series of stations on the moon to study geophysical conditions [1], and the MoonLITE, a U.K. lead mission whose goal is to land a set of robotic instruments on the moon that will study seismic activities among other physical characteristics of the Moon [2]. In order to complete these missions, it is necessary to ensure the soft landing of the vehicle carrying these scientific instruments. In addition, people are continually looking for ways to reduce cost, so minimizing fuel is an important parameter when planning a mission to the moon.

This optimization problem has been approached in various ways by different authors. Some have attempted to optimize not only the descent phase, but the de-orbit phase as well [3]. Others have analyzed the optimal strategy for landing from lunar parking orbit [4], and others have designed guidance laws to land on a target while still minimizing fuel [5] to cite some examples. However, most articles do not explain the numerical implementation of their optimization methods. Although in some situations simple techniques as a shooting method provides a solution, it is not the case for more complicated problems. As a result, there is the need to develop a technique that can deal with more complex cases.

This paper solves two different cases of the fuel optimization problem of landing on the moon. The first case is the 1-D moon lander where the purpose is to make a soft vertical landing while controlling the thrust of the vehicle to minimize fuel consumption. This first case lays down the ground work for understanding the difficulties than can be encountered when actually implementing the optimization techniques in the code. The vertical soft landing problem is solved by solving the system explicitly, and using two common methods to solve two boundary value problems, Newton's shooting method and finite difference method. The later is implemented via MATLAB's embedded two boundary value solver *bvp4c.*

The second case is the 2-D moon lander. Here, the lander starts with tangential velocity greater than zero and the fuel consumption is minimized by controlling the attitude angle (the angle between the radial vector and the thrust direction). This problem is more complicated and exposes how the methods used in the first case are ineffective when faced with the challenges of a more complex problem. Thus, a minimization technique is proposed to overcome the difficulties that the other methods posed when solving for the optimization problem for the 2-D lander.

Chapter 2

Optimization and Control Theory

Two point boundary value problems such as the one studied in this paper; often arise in optimal control theory. This chapter intends to provide a short background on optimization control theory. Most of the information in this chapter can be found in [6] [7]. Assume that the dynamics of a system can be described by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathrm{x}(t)) \\ \mathbf{x}(t_0) &= \mathrm{x}_0 \end{aligned} \tag{1}$$

The initial point $\mathrm{x}_0 \in \mathbb{R}^n$ and the function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ are given. Next, assume one wants to control the system described above, so that the outcome of the system may be changed by changing a parameter(s) as the system evolves. Then, the dynamical system above becomes

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathrm{x}(t), \mathbf{u}(t)) \\ \mathbf{x}(t_0) &= \mathrm{x}_0 \end{aligned} \tag{2}$$

The function $\mathbf{u} : t \to \mathbf{u}(t) \in A$ is known as the control, where $t \in [0, \infty)$ and A is the set of all admissible controls. The function $\mathbf{f} : \mathbb{R}^n \times A \to \mathbb{R}^n$ now depends on the control $\mathbf{u}(t) \in A$, so that the solution $\mathbf{x}(\cdot) : t \to \mathbb{R}^n$ is predicated not only on x_0 but on $\mathbf{u}(\cdot)$ as well. To determine the best control, it is necessary to establish a performance criterion, as to quantify divergence from ideal behavior. This criterion is called the payoff functional J, also known as the cost functional or the performance index.

$$\mathrm{J}(\mathbf{u}(t)) = \varphi\left(\mathbf{x}(t_f), t_f\right) + \int_{\mathrm{t}_0}^{\mathrm{t}_\mathrm{f}} \mathrm{L}(\mathbf{x}(t), \mathbf{u}(t), t)\,\mathrm{dt} \tag{3}$$

The function $\varphi : \mathbb{R}^n \to \mathbb{R}$ is called the terminal payoff or terminal cost function, and $\mathrm{L} : \mathbb{R}^n \times A \to \mathbb{R}$ is called the intermediate cost function or the running payoff. In general, an optimization problem will be stated in the following way: find the admissible control $\mathbf{u}(t)$ that minimizes the cost functional from equation (3) subject to the constraints from equation (2). This can also be expressed as find the curve $\mathbf{x}^*(\cdot)$ that minimizes (3) among all $\mathbf{x}(\cdot)$ satisfying the given initial conditions (or boundary conditions depending on the problem). Recall that $\mathbf{x}(\cdot)$ depends on the control $\mathbf{u}(t)$, so that the optimal curve $\mathbf{x}^*(\cdot)$ depends on $\mathbf{u}(t)$ as well. The optimal state $\mathbf{x}^*(\cdot)$ is characterized in optimization theory through the Pontryagin's maximum principle.

The Pontryagin's maximum principle states that "*if* $\mathbf{u}^*(t)$ *is an optimal control, then there exists a function* $\boldsymbol{\lambda}^*(t)$*, called a costate, that satisfies a certain maximization principle*." [6, p. 47] The costate function arises from the constraints that are imposed on $\mathbf{x}^*(\cdot)$ by the system of ODEs. The costate function can also be seen as Lagrange multipliers that contain important information about the cost of violating the constraints that $\mathbf{x}^*(\cdot)$ must satisfy. The relationship between the state equations, the cost functional, and the costate function is given by the Hamiltonian which in control theory is described in the following way

$$H(x,\lambda,u) := \mathbf{f}(x,u) \cdot \lambda + \varphi(x,u) \tag{4}$$

Now, consider the following optimal control problem

$$\mathrm{J}(\mathbf{u}(\cdot)) = \varphi\left(\mathbf{x}(t_f), t_f\right) + \int_{t_0}^{t_f} \mathrm{L}(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{5}$$

given $\mathbf{u}(\cdot) \in A \subseteq \mathbb{R}^m$, find a control $\mathbf{u}^*(\cdot)$ such that $\mathrm{J}(\mathbf{u}^*(\cdot)) = \max_{\mathbf{u}(\cdot) \in A} \mathrm{J}(\mathbf{u}(\cdot))$ subject to

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathrm{x}(t), \mathbf{u}(t)) \tag{6}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$, $\mathbf{u}(t) \in A$, and the initial time t_0 has been specified. In addition, some $q \leq n$ conditions may be specified at the terminal time t_f

$$\psi_f(x_k(t_f), t_f) = 0 \quad \in \mathbb{R}^q \quad (k = 1, 2, ...q) \tag{7}$$

where subscript f is there to denote that the conditions must be satisfied at the final time. Then If such optimal control $\mathbf{u}^*(\cdot)$ exists and $\mathbf{x}^*(\cdot)$ is its corresponding trajectory, then by the Pontryagin's maximum principle, there exists a costate function $\boldsymbol{\lambda}^* : [t_0, t_f] \rightarrow \mathbb{R}^n$ such that

a) $\dot{\mathbf{x}}^*(t) = H_{\boldsymbol{\lambda}}(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t))$

b) $\dot{\boldsymbol{\lambda}}^*(t) = -H_{\mathbf{x}}(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t))$

c) $H(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t)) = \max_{a \in A} H(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), a(t))$ (8)

d) $\mathbf{x}_l(t_0)$ given or $\boldsymbol{\lambda}_l(t_0) + \left(\dfrac{\partial \varphi}{\partial x_l(t_0)} + \boldsymbol{\nu}_f \dfrac{\partial \psi}{\partial x_l(t_0)} \right) = 0 \quad (l = 1, ..., n)$

e) $\mathbf{x}_j(t_f)$ given or $\boldsymbol{\lambda}_j(t_f) - \left(\dfrac{\partial \varphi}{\partial x_j(t_f)} + \boldsymbol{\nu}_f \dfrac{\partial \psi}{\partial x_j(t_f)} \right) = 0 \quad (j = 1, ..., n)$

The second part of equations e) and f) are called transversality conditions and $\boldsymbol{\nu}_f \in \mathbb{R}^q$ are the Lagrange multipliers that arise from the terminal constraints imposed by equation (7). Transversality conditions are necessary conditions for optimality problems with free boundary points. The solution of the *2n* differential equations from the state

equations a) and the adjoint equations b) as well as the value of the *q* parameter $\boldsymbol{\nu}_f$ are determined by the *2n+q* boundary conditions from e) and f) and (7). The Hamiltonian does not need to be differentiable with respect to the control; however, if the Hamiltonian is differentiable with respect to the control $\mathbf{u}(\cdot)$ and the control variables are unconstrained, i.e. $\mathbf{u}(t) \in \mathbb{R}^m$ instead of $\mathbf{u}(t) \in A \subset \mathbb{R}^m$ where A is the set of admissible controls, then condition c) can be replaced by $\partial H / \partial u = 0$ where the minimum can be guaranteed by $\partial^2 H / \partial u^2$ being positive definite.

There are some additional conditions that must be satisfied depending on whether or not the final time is given. If the final time is fixed and the Hamiltonian does not depend explicitly on the time then for a given t_0 and t_f

$$H(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t)) = C \quad \text{for} \quad t \in [t_0, t_f] \tag{9}$$

Also, if the final time is free and the Hamiltonian does not depend explicitly on the time then for a given t_0

$$H(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t)) = 0 \quad \text{for} \quad t \in [t_0, \tau] \tag{10}$$

where τ denotes the first time when the terminal conditions are met. These last two equations state that when the Hamiltonian is evaluated along the optimal trajectory it must be equal to a constant if the final time is fixed and equal to zero if the final time is free.

Chapter 3

Lunar Lander: Soft Vertical Landing

Problem Setup

The problem will be set up as described in [6, pp. 55-59]. The objective is to minimize fuel consumption during a vertical landing on the moon, so that the remaining fuel at landing $m(\tau)$ is maximized, where τ denotes the first time the height and velocity is zero, so $h(\tau) = v(\tau) = 0$. Since the change in mass is directly proportional to the thrust applied to the engine, the problem is equal to minimizing the total thrust $u(t)$ applied. Then the optimal control problem can be stated as

$$\min J(u(\cdot)) = -\int_0^{\tau} u(t)dt \tag{11}$$

subject to the dynamical system

$$\begin{cases} \dot{h}(t) = v(t) \\ \dot{v}(t) = -g + \dfrac{u(t)}{m(t)} \\ \dot{m}(t) = -ku(t) \end{cases} \tag{12}$$

with initial conditions

$$\begin{cases} h(0) = h_0 \\ v(0) = v_0 \\ m(0) = m_0 \end{cases} \tag{13}$$

and $u(t) \in A = [0,1]$, so that the thrust is constrained to $0 \le u(t) \le 1$. In addition $h(t) \ge 0$ and $m(t) \ge 0$. Also, m_0 denotes the total mass of the spacecraft, and the

constant k is the mass flow rate given by the total mass of the fuel m_{fuel} divided by the maximum burn time of the fuel.

The Hamiltonian of the system is

$$\begin{aligned} H(x,\lambda,u) &= \mathbf{f}\cdot\lambda + L \\ &= v(t)\lambda_h - g\lambda_v + \frac{u(t)}{m(t)}\lambda_v - ku(t)\lambda_m - u(t) \end{aligned} \tag{14}$$

Note that Equation (8.a) is equivalent to the state equations from (12). The adjoint equations from (8.b) are

$$\begin{cases} \dot{\lambda}_h = -\dfrac{\partial H}{\partial h} = 0 \\ \dot{\lambda}_v = -\dfrac{\partial H}{\partial v} = \lambda_h \\ \dot{\lambda}_m = -\dfrac{\partial H}{\partial m} = \dfrac{u(t)}{m(t)^2}\lambda_v \end{cases} \tag{15}$$

and the maximization condition from c) is

$$\begin{aligned} H(\mathbf{x}(t),\boldsymbol{\lambda}(t),\mathbf{u}(t)) &= \max_{0\le a\le 1} H(\mathbf{x}(t),\boldsymbol{\lambda}(t),a) \\ &= v(t)\lambda_h(t) - g\lambda_v(t) + \max_{0\le a\le 1}\left[a\left(\frac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 \right)\right] \\ &= \begin{cases} v(t)\lambda_h(t) - g\lambda_v(t) + \dfrac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 & \text{if} \quad \dfrac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 > 0 \\ v(t)\lambda_h(t) - g\lambda_v(t) & \text{if} \quad \dfrac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 < 0 \end{cases} \end{aligned} \tag{16}$$

so the optimal control is

$$u(t) = \begin{cases} 1 & \text{if} \quad \dfrac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 > 0 \\ 0 & \text{if} \quad \dfrac{1}{m(t)}\lambda_v(t) - k\lambda_m(t) - 1 < 0 \end{cases} \tag{17}$$

Since all initial conditions are given, $\lambda_h(0)$, $\lambda_v(0)$, and $\lambda_m(0)$ cannot be determined by the adjoint equation from (8.e). In the same way, since it is known that the desired state at τ is $h(\tau)=v(\tau)=0$ the adjoint equation from (8.f) cannot be used to find $\lambda_h(\tau)$ and $\lambda_v(\tau)$. However, $m(\tau)$ is a free variable, so that by the adjoint equation of (8.f) $\lambda_m(\tau)=0$.

From equation (17) it can be deduced that the there is a switching time in which the control value changes from 0 to 1 (since these are the values provided by the optimality conditions). Denote the switching time t_s and $t_s<\tau$. Assume that the rocket engine is off at $t_0=0$ and that at t_s the engine is turn on at full power where $u=0$ and $u=1$ are the thrust values for the off and full power engine respectively. To prove this claim, it is necessary to solve the adjoint equations from (15). Given that the optimal control problem in case one is a well documented problem, the proof for the above claim will be omitted but refer to [6, p. 59] [8] for details. For the purpose of this paper, it will be accepted that the above assumption satisfies the Pontryagin's maximum principle. Therefore,

$$u(t)=\begin{cases}0 & \text{if} \quad 0\le t\le t_s\\ 1 & \text{if} \quad t_s\le t\le \tau\end{cases} \tag{18}$$

The problem is then divided in two parts, a free fall trajectory and a powered trajectory. The free fall trajectory, $u(t)=0$, becomes an initial value problem with state equations

$$\begin{cases}\dot{h}(t)=v(t)\\ \dot{v}(t)=-g\\ \dot{m}(t)=0\end{cases} \tag{19}$$

initial conditions from (13), and unknown final time t_s. The powered trajectory, $u(t)=1$, becomes a two boundary value problem with state equations

$$\begin{cases} \dot{h}(t) = v(t) \\ \dot{v}(t) = -g + \dfrac{1}{m(t)} \\ \dot{m}(t) = -k \end{cases} \tag{20}$$

and boundary conditions

$$\begin{cases} h(\tau) = 0 \\ v(\tau) = 0 \\ m(t_s) = m_0 \end{cases} \tag{21}$$

where the initial time t_s is the same as the final time from the free fall trajectory and unknown final time τ corresponds to the first time when $h(\tau)=0$ and $v(\tau)=0$.

Numerical Analysis

MATLAB was used to implement all the routines for the solution of the fuel optimization problem for the moon lander. The vertical lunar landing optimization problem was solved with three different approaches: solving the system explicitly, using Newton's shooting method, and finite difference method via MATLAB embedded function for two boundary value problems *bvp4c*.

Method One: Solving the System Explicitly

Implementation

The optimization problem described in the previous section is one of those rare cases for which an explicit solution is available. The solution of system (19) subject to initial conditions given in (13) is

$$\begin{cases} h_{free}(t) = -\frac{1}{2}gt^2 + tv_0 + h_0 \\ v_{free}(t) = -gt + v_0 \\ m_{free}(t) = m_0 \end{cases} \tag{22}$$

and the solution of the state equations from (20) subject to boundary conditions given in (21) is

$$\begin{cases} h_{powered}(t) = \tau gt - \frac{1}{2}gt^2 - \frac{\tau - t}{k} - \frac{1}{k^2}\log\left(\frac{m_0 - k(\tau - t_s)}{m_0 - k(t - t_s)}\right)(m_0 - k(t - t_s)) - \frac{1}{2}g\tau^2 \\ v_{powered}(t) = g(t - \tau) + \frac{1}{k}\log\left(\frac{m_0 + k(t_s - \tau)}{m_0 + k(t_s - t)}\right) \\ m_{powered}(t) = m_0 + k(t_s - t) \end{cases} \tag{23}$$

Both trajectories, free fall and powered descend, must intersect at the time when the engine is turned on at full power, i.e. the solution of the state equations for both trajectories are equal at t_s. Then, the solutions (22) and (23) can be used to solve for the switching time t_s and final time τ. Since $m(t_s) = m(0) = m_0$, $h(t)$ and $v(t)$ are the only solutions of interest.

Substitute t with t_s in equations (22) and (23),

$$\begin{cases} h(t_s) = -\frac{1}{2} g t_s^{\,2} + t_s v_0 + h_0 \\ v(t_s) = -g t_s + v_0 \\ h(t_s) = -\frac{1}{2} g (t_s - \tau)^2 + \frac{t_s - \tau}{k} - \frac{m_0}{k^2} \log\left(\frac{m_0 - k(t_s - \tau)}{m_0} \right) \\ v(t_s) = g(t_s - \tau) + \frac{1}{k} \log\left(\frac{m_0 + k(t_s - \tau)}{m_0} \right) \end{cases} \tag{24}$$

which provides a system of four equations and four unknowns. The values of t_s, τ, $h(t_s)$, and $v(t_s)$ can now be substituted into the solutions (22) and (23) and be solved for the intervals $t \in [0, t_s]$ and $t \in [t_s, \tau]$ respectively. The amount of fuel that was burned during descend can be solved as well with $m(\tau) = m_0 + k(t_s - \tau)$.

Results

MATLAB's symbolic toolbox has embedded functions that can be used to provide explicit formulae for the solution of ordinary differential equations and systems of equations such as the one from (24). However, using the symbolic toolbox can be computationally expensive, in particular when the number of steps for the simplification has to be increased to allow the solver to find an explicit solution. On the other hand, it is perhaps the easiest and most intuitive way to solve the optimization problem in case one. The results below were obtained with the inputs from the following table

Table 1 Constants and Initial Conditions

$u_{\max}$	m_0	m_{fuel}	k	v_0	h_0	g_{moon}
400N	224kg	204kg	2.833x10^{-3}kg/s	100m/s	100000m	1.622m/s^2

Recall $u(t) \in A = [0,1]$, so that the values from Table 1 must be rescaled when entering them into the code. In Figure 1, the red line denotes a non-optimal case where at the switching time, the engine was turned on at half power $u = 0.5$, as opposed to full power $u = 1$ as the optimization analysis suggested. In both Figure 1 and 2 the freefall trajectory is denoted by the solid blue line, and the powered descend trajectory by the dotted blue line. Table 2 summarizes the results for the optimal solution.

Table 2 Results Using Symbolic Toolbox

t_s	τ	$h(t_s)$	$v(t_s)$	m_{burned_fuel}
312.859s	481.849s	51904.471m	-407.458m/s	191.522 kg

As an interesting remark, the burned fuel when the engine has turned on at half thrust was 200.46 kg, about 9 kg more than the optimal solution. Although this particular case can be solved explicitly, the other methods used in this paper serve as an example for cases for which the explicit solution is not found.

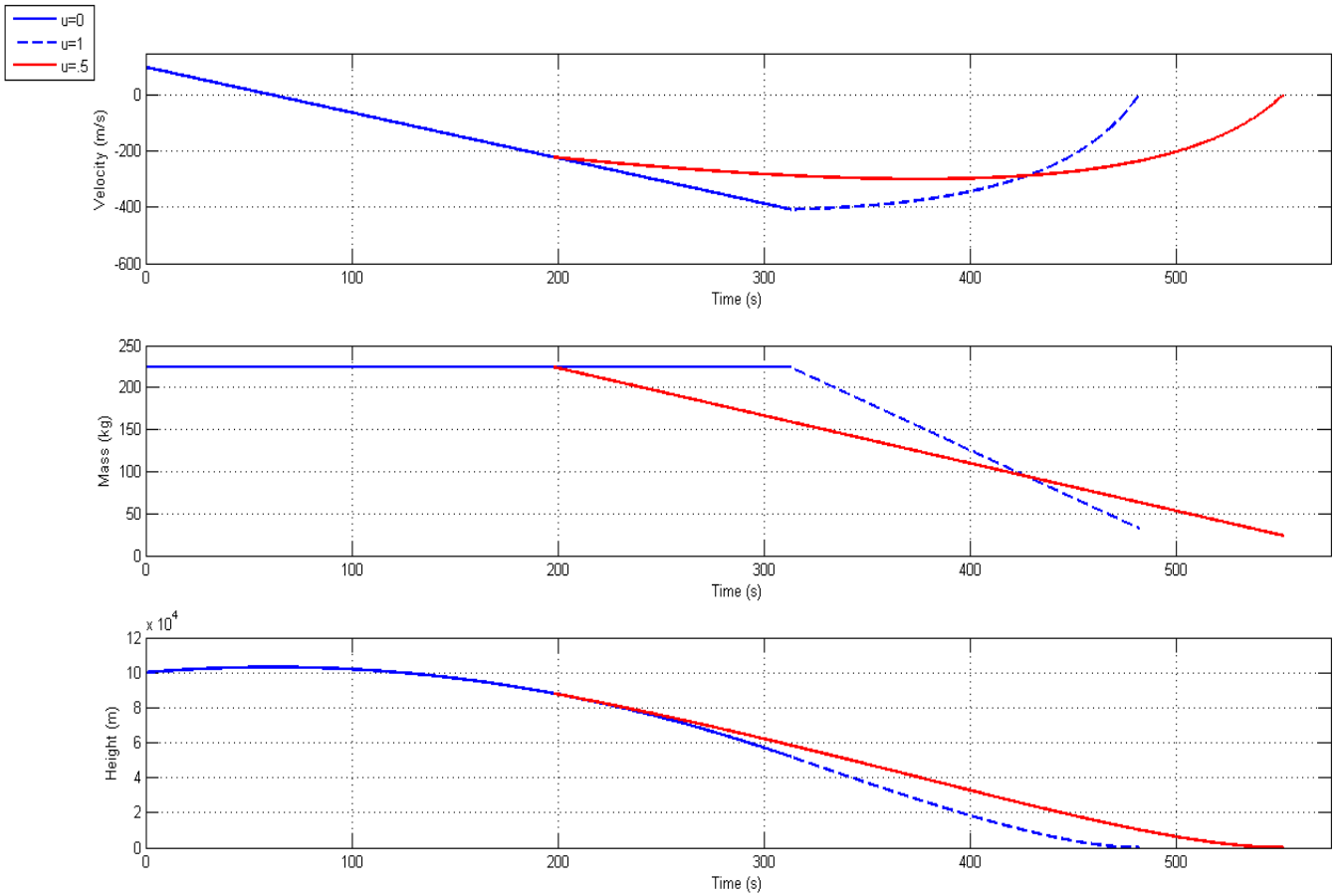

Figure 1 Evolution of Velocity, Mass, and Altitude during Flight Time

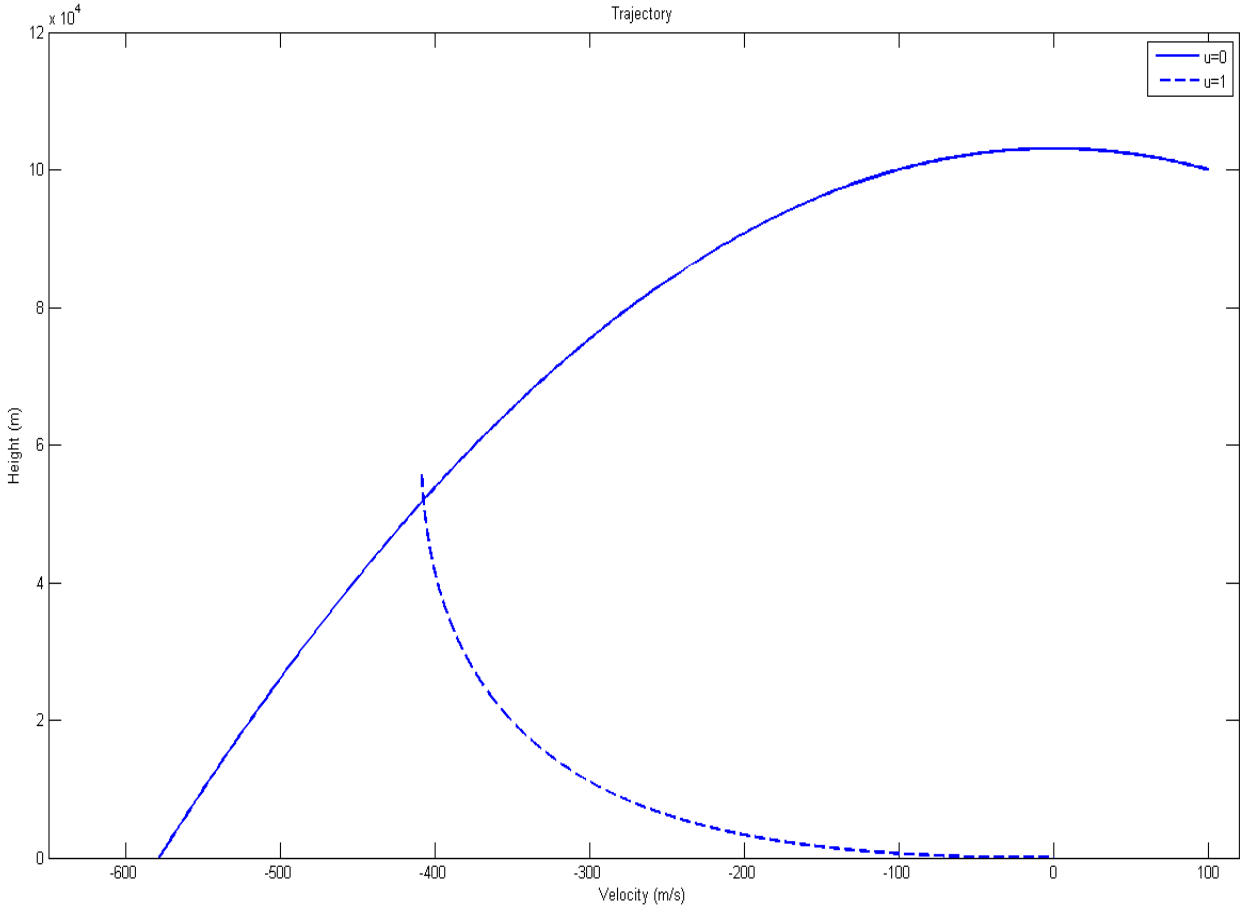

Figure 2 Height vs. Velocity Trajectory

Method 2: Newton's Shooting Method

Implementation

The shooting method converts a two boundary value problem into an initial value problem. The values that are not given at the initial time are guessed and corrections are made so that when the ODE system is integrated using the initial conditions it reaches the terminal conditions that were set by the two boundary value problem [9]. Since the free fall trajectory is an initial value problem, the shooting method will be used only on solving the powered trajectory. However, the equations for the free fall trajectory will be used to constrain the guesses for the two boundary value problem.

As before the free fall trajectory state equations from (19) are solved with initial conditions from (13) so that the same result as in (22) is obtained. The powered descent state equations are now solved with initial conditions. Given a system of ODE from (20) with boundary conditions (21), replace terminal conditions with (guessed) initial conditions

$$\begin{cases} h(t_s) = h_s \\ v(t_s) = v_s \\ m(t_s) = m_0 \end{cases} \tag{25}$$

and initial guesses h_s and v_s, so that when solving the state equations from (20) along with the initial conditions from (25) satisfies the terminal conditions given in (21). Then, the solution to the powered descend trajectory is

$$\begin{cases} h_{powered}(t) = h_s + v_s(t-t_s) + \log\left(\frac{m_0 - k(t-t_s)}{m_0}\right)\left(\frac{m_0}{k^2} - \frac{t-t_s}{k}\right) - \frac{1}{2}g(t^2 + {t_s}^2) + \frac{t-t_s}{k} + gtt_s \\ v_{powered}(t) = v_s - g(t-t_s) - \frac{1}{k}\log\left(1 - \frac{k(t-t_s)}{m_0}\right) \end{cases} \quad (26)$$

To assess how far off the evaluated terminal conditions are from the desired values of the terminal conditions from (21), it is necessary to evaluate the discrepancy function $L = [L_1 \quad L_2]^T$

$$\begin{aligned} L_1 &= h(\tau, h_s) - h(\tau) \\ &= h_s + v_s(\tau - t_s) + \log\left(\frac{m_0 - k(\tau - t_s)}{m_0}\right)\left(\frac{m_0}{k^2} - \frac{\tau - t_s}{k}\right) - \frac{1}{2}g(\tau^2 + {t_s}^2) + \frac{\tau - t_s}{k} + g\tau t_s - 0 \\ L_2 &= v(\tau, v_s) - v(\tau) \\ &= v_s - g(\tau - t_s) - \frac{1}{k}\log\left(1 - \frac{k(\tau - t_s)}{m_0}\right) - 0 \end{aligned} \quad (27)$$

where $h(\tau, h_s)$ and $v(\tau, v_s)$ are equations (26) evaluated at the time where the boundary conditions are met. Note that h_s and v_s have been included as variables of $h(\tau)$ $v(\tau)$ to denote their dependence on the guessed initial conditions. Now it is necessary to find a correction that will improve the guessed values for h_s and v_s. Let J denote the Jacobian of the discrepancy functions

$$J = \begin{bmatrix} \frac{\partial L_1}{\partial h_s} & \frac{\partial L_1}{\partial v_s} \\ \frac{\partial L_1}{\partial h_s} & \frac{\partial L_1}{\partial v_s} \end{bmatrix} \quad (28)$$

so that by Newton's method, the correction of h_s and v_s is given by $J \cdot \delta X = -L$ where $L = [L_1 \quad L_2]^T$ and $X = [h_s \quad v_s]^T$. Then the correction matrix is

$$\delta X = -J^{-1}L \tag{29}$$

and the new improved guess is given by adding the correction to the old guessed value

$$X_{new} = X_{old} + \delta X \tag{30}$$

There are some important remarks that need to be made in regard to the specific two boundary value problem in question. First, the powered descend trajectory is constrained by the free fall trajectory meaning that these two curves must intersect at t_s as mentioned in the previous section. So far, nothing that has been done provides this constraint to the solution. Second, the initial time for the powered descend trajectory t_s is unknown as well, moreover, the discrepancy functions are dependent on the terminal time τ which is also unknown. Consequently, t_s and τ need to be guessed (and corrected) along with h_s and v_s to satisfy the terminal conditions from the two boundary value problem.

Newton's method is not globally convergent meaning that the initial guesses must be somewhat close to the solution to get convergence, so adding two variables more that need to satisfy some terminal conditions poses a problem. To improve the likelihood of making initial guesses that will converge start by guessing a value for t_s

Given some value for t_s, h_s and v_s can be evaluated using the free fall trajectory equations so

$$t_s \approx t_{s\_guess} \tag{31}$$

$$h_{free}(t_s) \approx h_{s\_guess}(t_{s\_guess}) = -\frac{1}{2} g t_{s\_guess}{}^2 + t_{s\_guess} v_0 + h_0 \tag{32}$$

$$v_{free}(t_s) \approx v_{s\_guess}(t_{s\_guess}) = -g t_{s\_guess} + v_0 \tag{33}$$

which not only improve the probability of making a good guess, but it constrains the powered descend curve to being equal to the free fall descend curve at t_s. Now the guess for τ can be evaluated from either L_1 or L_2. For simplicity, L_2 equation is used

$$v_{s\_guess} - g(\tau - t_{s\_guess}) - \frac{1}{k}\log\left(1 - \frac{k(\tau - t_{s\_guess})}{m_0}\right) = 0 \tag{34}$$

Now that all the necessary guesses have been made, Newton's method can be applied. Since from the correction of one guessed variable the other variables can be determined, use the correction for $h_{s\_guess} = h_{s\_old}$ to improve on the other guesses, so

$$h_{s\_new} = h_{s\_old} + \delta h_s \tag{35}$$

solving for t_{s_guess} in equation (32) and substituting the improve height value (35)

$$t_{s\_new} = v_0 \pm \frac{\sqrt{{v_0}^2 - 2g(h_{s\_new} - h_0)}}{g} \quad \text{and} \quad t_s > 0 \tag{36}$$

then

$$v_{s\_new} = -gt_{s\_new} + v_0 \tag{37}$$

and τ_{new} is attained by solving for τ in L_2 and substituting the new values. If the tolerance is not met then make the old values the new ones and continue the iteration until convergence is achieved to the given tolerance.

Results

The results were attained using the same initial conditions and constants in Table 1, the initial guess $t_{s\_guess} = 100s$, and a tolerance of 10^{-6}. After 60 iterations Newton's method converges to the same answers as with the explicit solutions up to the 4th

decimal place for $h(t_s)$ and m_{burned_fuel}, and up to the 7th decimal place for t_s, τ, and $v(t_s)$. Refer to Table 2 for these values.

In this case, the Newton's shooting method is more difficult and less intuitive to implement. In addition, it is less time efficient that than the explicit solution that can be obtained using MATLAB's symbolic toolbox alone. Moreover, for the specific values provided in Table 1 the problem provided a way to estimate good guesses that allow for convergence, but it is important to note that equation (36) can give an imaginary number depending on the initial values because of the square root term. If instead equation (33) is used to solve for t_{s_new}, then L_1 instead of L_2 must be used to solve for τ_{guess} (as well as τ_{new}), but the possibility of an imaginary number is again encountered when solving for τ in L_1. In conclusion, using Newton's shooting method is not the most efficient method for the optimization problem in case one not only because it is more computationally expensive, but also because of the difficulties met when attempting to find suitable initial guesses that will lead to convergence.

Method 3: Finite Difference Method (MATLAB's function bvp4c)

MATLAB's two boundary value solver *bvp4c* is a "Residual control based, adaptive mesh solver" [10] that uses a three point Lobatto method of order 4 called the Simpson formula because it reduces to the Simpson's quadrature rule. The function computes a cubic spline on each subinterval of a mesh by requiring it to be continuous, to satisfy the boundary conditions, and satisfy the ODEs at the endpoints and at the midpoint of each subinterval [11].

Implementation

Only the powered descent trajectory is solved using the solver *bvp4c* because the free fall trajectory is setup as an initial value problem and not as a boundary value problem. The information gathered from *bvp4c*, however, is essential for solving the free fall trajectory in the appropriate time interval. The syntax of the *bvp4c* function is:

```
solinit = bvpinit(t, xinit, params)
sol = bvp4c(odefun,bcfun,solinit,options)
```

"Solinit" is a structure that contains the initial guesses for the solution. It is built with the help of the function *bvpinit* with the following fields: "*t"* which represents the points at which the boundary conditions are imposed, "*xinit"* which provides the initial guesses to the solver, and finally "*params"* which gives an initial guess for unknown parameters [12].

In this case the points at which the boundary conditions are met are the initial time t_s and the terminal time τ. The problem is that both times are unknown. To get around this problem the terminal time is normalized and a linear space is created so that

```
t = linspace(0, 1)
```

A similar approached to the one in the shooting method is taken when deciding on the guesses for the *bvp4c* solver. Provided a guess for t_s, equations (32) and (33) can be used as guesses for the height and velocity solutions. Note that it is also necessary to provide a guess for the solution of mass. In addition, because it is a free terminal time problem, there is an additional parameter that needs to be determined which is the final time τ. The code for the structure of guesses would look something like

```
t_s = t_guess;
```

```
t = linspace(0, 1);
yinit = [h_free(t_s); v_free(t_s); m_guess];
solinit = bvpinit(t, xinit, T_guess);
```

where the values of h_free(t_s) and v_free(t_s) are evaluated from (32) and (33) using t_guess.

The solver first needs a function (*odefun*) that provides the system of ordinary differential equations from equation (20). Remember that the time has been normalized, so if t_n is the normalized time such that $t_n = t/\tau$ then the ODE,

$$\frac{d\mathbf{x}}{dt_n} = \frac{d\mathbf{x}}{d\left(t/\tau\right)} = \tau \frac{d\mathbf{x}}{dt} \tag{38}$$

Second, the solver needs a function (*bcfun*) that provides the boundary conditions from (21). As mentioned before, because it is a free terminal time problem and the terminal time needs to be found, a fourth boundary condition must be added. Also, in the previous section it was mentioned that the powered descent trajectory had to be constrained to the free fall equation so that they intersect at the switching time. The fourth boundary condition addresses this constraint by using the free fall equations, so from (22)

$$\begin{aligned} t_s &= \frac{v_0 - v(t_s)}{g} \\ \therefore\ h(t_s) &= h(t_s, v(t_s)) = -\frac{1}{2} g \left(\frac{v_0 - v(t_s)}{g} \right)^2 + \left(\frac{v_0 - v(t_s)}{g} \right) v_0 + h_0 \end{aligned} \tag{39}$$

then the equations for *bcfun* are

$$\begin{cases} h(\tau) = 0 \\ v(\tau) = 0 \\ m(t_s) = m_0 \\ h(t_s) = -\frac{1}{2} g \left(\frac{v_0 - v(t_s)}{g} \right)^2 + \left(\frac{v_0 - v(t_s)}{g} \right) v_0 + h_0 \end{cases} \tag{40}$$

Finally, the solver needs some initial guesses that are provided by "*solinit"* as it has already been explained. MATLAB offers a list of options such as error tolerance and mesh size that can be supply to the solver using *bvpset*. For more details refer to [13].

With the solutions from *bvp4c* the switching time can be solved using equation (39). Then, given the interval $[0, t_s]$ the solution for the free fall trajectory can be easily solved using an ODE solver such as *ode45.*

Results

The inputs for the initial conditions and constants used are the same ones given on Table 1. The guesses provided to the solver were

Table 3 Guesses Provided to *bvp4c* Solver

$h_s = 101890m$	v_s =-62.2m/s	$m = 204kg$	$\tau = 400$
Equation (32) evaluated at $t_{s\_guess} = 100s$	Equation (33) evaluated at $t_{s\_guess} = 100s$	m_{fuel}	Reasonable guess

Table 4 Results Using *bvp4c* Solver

t_s	τ	$h(t_s)$	$v(t_s)$	m_{burned_fuel}
312.834s	481.894s	51915.058m	-407.417m/s	191.601 kg

With the exception of $h(t_s)$, all results in Table 4 do not differ by more than 10^{-1} from the results given in Table 1. There is a difference of about 10.5 m between the results for $h(t_s)$ which results from the small difference in the found times that creates a large change in the height at high velocities. In this case using the *bvp4c* solver is the most efficient way to solve the optimization problem considering it was possible to find guesses that provided convergence. In addition, MATLAB's two boundary value solver was able to calculate and display the results in about 1/3 of the time that it took when solving the system explicitly.

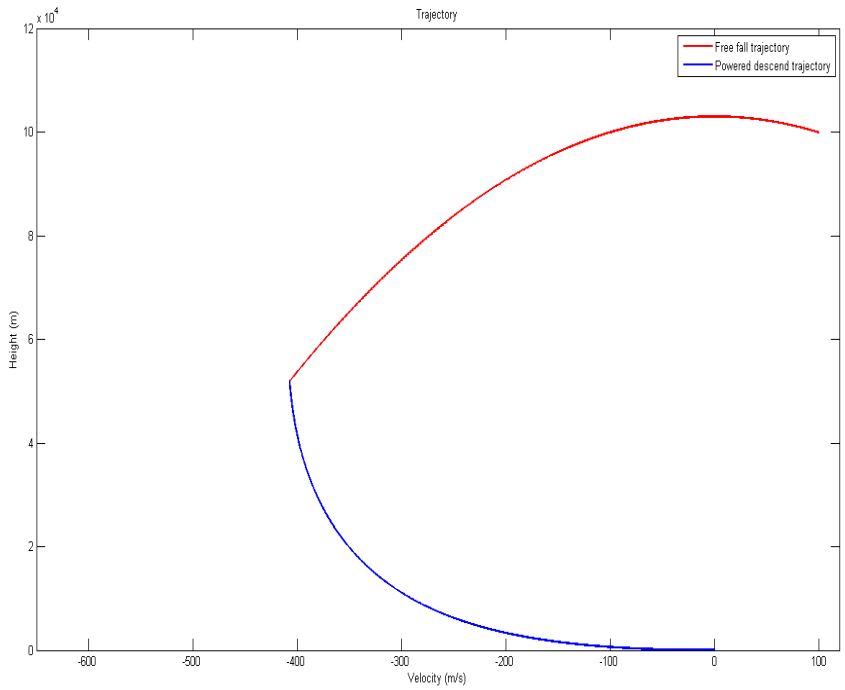

Figure 3 Height vs. Velocity Trajectory Using *bvp4c* Solver

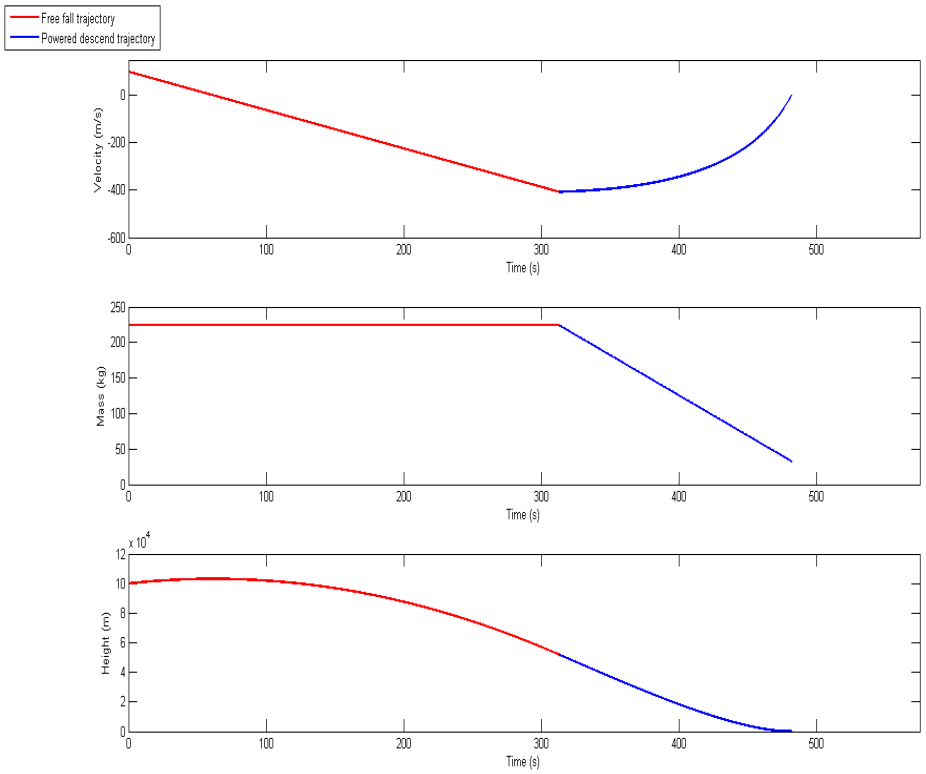

Figure 4 Evolution of Velocity, Mass, and Altitude during Flight Time Using *bvp4c* Solver

Chapter 4

Lunar Lander: Soft Landing with Initial Tangential Velocity

Problem Setup

The problem will be set up as described in [5, p. 5]. Given the coordinate system from Figure 5 where O is the lunar center, L is the position of the lander, s is the surface of the moon, and given the notation

r = radial distance

θ = position angle

ψ = control angle (angle measured from the radial vector to the thrust vector)

T = thrust

u = tangential velocity

v = radial velocity

m = mass of the lander

μ = moon's gravitational constant

$c = \dfrac{1}{g_{earth} I}$ and I = specific impulse

the equations governing the motion of the lunar lander are

$$\begin{cases} \dot{r} = v \\ \dot{\theta} = \dfrac{u}{r} \\ \dot{u} = \dfrac{T}{m}\sin\psi - \dfrac{uv}{r} \\ \dot{v} = \dfrac{T}{m}\cos\psi - \dfrac{\mu}{r^2} + \dfrac{u^2}{r} \\ \dot{m} = -cT \end{cases} \tag{41}$$

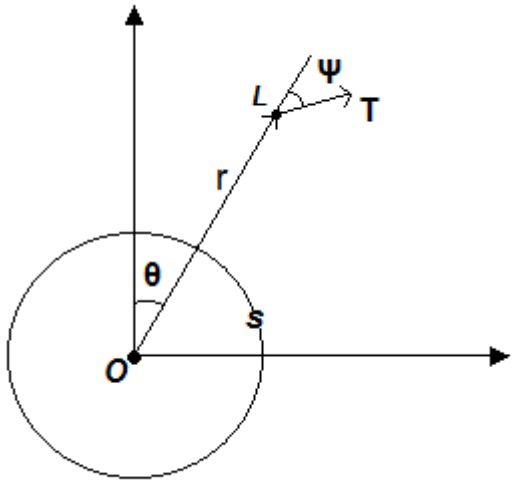

Figure 5 Lunar Polar Coordinate System

The objective is to minimize fuel consumption during soft landing by controlling the direction of the thrust $\psi(t)$. The problem ignores the de-orbit phase, and minimizes the fuel consumption during the powered descent phase only. The problem is simplified by assuming that during the descent phase the engine is turned on at full power. Note that optimizing the control $\psi(t)$ is equivalent to optimizing the distance of travel, i.e. the time the engine is on. The cost function must depend on the control, and since only the velocities depend on the direction of the thrust, the optimal control problem is stated as

$$\min J(\psi(\cdot)) = (u(t_f) - u_f)^2 + (v(t_f) - v_f)^2 \tag{42}$$

Subject to the state equations given in (41) with boundary conditions

$$\begin{cases} r(0) = y_0 + r_{moon} \\ \theta(0) = \theta_0 \\ u(0) = u_0 \\ v(0) = v_0 \\ m(0) = m_0 \\ r(t_f) = r_{moon} \\ u(t_f) = 0 \\ v(t_f) = 0 \end{cases} \tag{43}$$

where y_0 denotes the altitude of the lander at initial time. The values u_f and v_f in (42) are the target values at landing; that is, they are the boundary values at t_f given in (43). Then, the Hamiltonian of the system is

$$\begin{aligned} H(x,\lambda,u) &= \mathbf{f}\cdot\lambda + L \\ &= v(t)\lambda_r + \frac{u(t)}{r(t)}\lambda_\theta + \left(\frac{T}{m}(\sin\psi) - \frac{u(t)v(t)}{r(t)}\right)\lambda_u \\ &\quad + \left(\frac{T}{m}(\cos\psi) - \frac{\mu}{r(t)^2} + \frac{u(t)^2}{r(t)}\right)\lambda_v - cT\lambda_m \end{aligned} \tag{44}$$

Equation (8.a) is satisfied by the state equations from (41). The costate equations from (8.b) are

$$\begin{cases} \dot{\lambda}_r = -\dfrac{\partial H}{\partial r} = \dfrac{u}{r^2}\lambda_\theta - \dfrac{uv}{r^2}\lambda_u - 2\dfrac{\mu}{r^2}\lambda_v + \dfrac{u^2}{r^2}\lambda_v \\ \dot{\lambda}_\theta = -\dfrac{\partial H}{\partial \theta} = 0 \\ \dot{\lambda}_u = -\dfrac{\partial H}{\partial u} = -\dfrac{1}{r}\lambda_\theta + \dfrac{v}{r}\lambda_u - 2\dfrac{u}{r}\lambda_v \\ \dot{\lambda}_v = -\dfrac{\partial H}{\partial v} = -\lambda_r + \dfrac{u}{r}\lambda_u \\ \dot{\lambda}_m = -\dfrac{\partial H}{\partial m} = \dfrac{T}{m^2}(\sin\psi)\lambda_u + \dfrac{T}{m^2}(\cos\psi)\lambda_v \end{cases} \tag{45}$$

and the condition (8.c) can be reduced to

$$\frac{\partial H}{\partial \psi} = 0 \tag{46}$$

since there are no constraints imposed on the control $\psi(t)$. Then the optimal control is

$$\psi(t) = \tan^{-1}\left(\frac{\lambda_u}{\lambda_v}\right) \tag{47}$$

Since all initial conditions are given, $\lambda_r(0)$, $\lambda_\theta(0)$, $\lambda_u(0)$, $\lambda_v(0)$, and $\lambda_m(0)$ cannot be determined by the adjoint equation from (8.e). In the same way, the adjoint equation from (8.f) cannot be used for $\lambda_r(t_f)$, $\lambda_u(t_f)$, and $\lambda_v(t_f)$. However, $\theta(t_f)$ and $m(t_f)$ are free variables, so that by the adjoint equation of (8.f) $\lambda_\theta(t_f) = \lambda_m(t_f) = 0$. Hence the optimization analysis results in a two boundary value problem with 10 equations, 5 state equations (41) and 5 costate equations (45), and 10 boundary conditions, 8 given by equation (43) and 2 resulting from the adjoint equations.

Numerical Analysis

It is easy to see that the system of equations given by (41) and (45) cannot be solved in the same explicit way as the equations from the vertical landing, so that Method 1 from the previous section will not work in this problem, as it is expected in most cases. Several complications arise from trying to use Newton's shooting method or MATLAB's function *bvp4c* as discussed in the following paragraphs.

As mentioned before, Newton's method requires initial guesses that are not too far off from the solution to allow for convergence. In the previous case the state equations

did not depend on the costate equations. In this case, on the other hand, the radial and tangential velocity depend on the costate equations due to optimization of the control ψ. Making guesses for the adjoint equations is not easy because they do not provide the physical intuition that can be derived when making an initial guess for some of the variables in the state equations. The shooting method could be used with a globally convergent method, but these are harder to implement and as mentioned in [5, p. 8] there may not exist an exact solution to the problem with the given boundary conditions.

It is important to remember that this is a free terminal time problem, which is more difficult to implement. For example to be able to solve it using *bvp4c*, the solver needs and extra boundary condition to solve for t_f. In the previous section it was necessary to constrain the descent curve to the powered descent curve so that the extra condition rose up naturally. In this case, however, there is no extra condition that can be imposed on the problem and still make physical sense. In addition, *bvp4c* requires guesses that capture the general behavior of the solution. Although the solver adapts the mesh after obtaining convergence on a given mesh, making guesses about the behavior of adjoint equations is extremely difficult.

The proposed solution to these problems is to reformulate the two boundary value problem as a minimization problem with the intention of satisfying the boundary conditions as well as possible.

Method 4: Minimization Technique

Implementation

Augment the cost functional as a sum of squares of the terminal conditions so that the objective function to be minimized is

$$F = (u(t_f) - u_f)^2 + (v(t_f) - v_f)^2 + (r(t_f) - r_f)^2 \tag{48}$$

and u , v , r are solved by using the system of equations provided by (41) and (45) and optimal control (47). Since F is being minimized by using the equations that give the optimal control, its solution provides an optimal landing curve that guarantees that the final conditions are met.

The evaluation of F , and therefore its minimization, requires the solution of (41) and (45) where the initial conditions for all the state equations are known, but none of the initial conditions for the costate equations are. Recall that this is a free terminal time problem, so the objective of the optimization problem can be restated as find some

$$\begin{cases} \lambda_r(0) = \lambda_{r0} \\ \lambda_\theta(0) = \lambda_{\theta 0} \\ \lambda_u(0) = \lambda_{u0} \\ \lambda_v(0) = \lambda_{v0} \\ \lambda_m(0) = \lambda_{m0} \end{cases} \tag{49}$$

and t_f that minimizes F where u_f , v_f , and r_f are the conditions imposed at t_f in equation (43).

The routine used to solve this minimization problem is given in [14] which attempts to find a minimum of a function of several variables. It works in the same way that *fminsearch* from MATLAB does, using a Nelder-Mead optimizer [14] along with a derivate-free method [15], with the difference that bounds are applied internally using a transformation of the variables so that the initial guesses can be constrained to an interval. The syntax of the function is

```
x = fminsearchbnd(minfun,xg,lbdd,ubdd)
```

where "*minfun"* is the function to be minimized, "*xg"* is some starting estimate, and as explained before *"lbdd"* and *"ubdd"* are upper and lower constraints imposed on the

variables that are trying to minimize F. These constraints limit the areas where the minimizing function looks to obtain a minimum, not only increasing the possibility of finding an adequate answer, but also reducing the computational cost.

Given equations (41) and (45), initial conditions

$$\begin{cases} r(0) = r_0 = y_0 + r_{moon} \\ \theta(0) = \theta_0 \\ u(0) = u_0 \\ v(0) = v_0 \\ m(0) = m_0 \end{cases} \tag{50}$$

and guesses $x_g = [t_f, \lambda_{r0}, \lambda_{\theta 0}, \lambda_{u0}, \lambda_{v0}, \lambda_{m0}]$, the ODEs can be solved using a standard ode solver and from them attain the necessary information to evaluate F. The idea is that when *fminsearchbnd* calls the "*minfun"* function, it will evaluate the ODEs that F needs. So the code for "*minfun*" looks something like

```
function F = minfun(xg)
x0 = [r0, theta0, u0, v0, m0, xg(2:end)];
time = [0, xg(1)];
G = odesolver(@odefun, time, x0);
F = ((G(end,1)-rf)a)^2 + ((G(end,3)-uf)b)^2 + ((G(end,4)-vf)c)^2 +…
(min(G(:,1))<1)*1e7
```

The last term of F prevents the lander from crashing into the ground by imposing a high cost penalty in the minimization if at any time the radius is less than the radius of the moon. The constants a, b, and c are weights that aid in prioritizing how the minimization function alters the variables to reach the desired terminal state. Using the found initial conditions of the costate equations and the t_f that minimize F, the two boundary value problem can be solved as an initial value problem using a standard ode solver routine.

Results

The results in Figure 6 were attained by scaling equations (41) and (45), and by accordingly scaling the values from the Tables below.

Table 5 Constants and Initial Conditions for 2-D Lander

$T_{\max}$	I	r_{moon}	y_0	θ_0	u_0	v_0	m_0
500N	224s	1737.4x10^{3}m	20000m	π/2	100m/s	-100m/s	224kg

Table 6 Terminal Conditions and Weights for 2-D Lander

r_f	u_f	v_f	a	b	c
r_{moon}	0m/s	0m/s	10^3	1	1

Table 7 Initial Values Given to *fminsearchbnd*

t_f	λ_{r0}	$\lambda_{\theta 0}$	λ_{u0}	λ_{v0}	λ_{m0}
500s	.02	.8	.01	-1.2	-3

It was assumed that the value of the initial costate equations are small, so the boundary constraints applied to all initial guesses was the interval $[-5,5]$ and $[100,800]$ for the time. It was also assumed that the gravitational force is constant equal to moon's gravity. The differential equations were solved using *ode23s*. Figure 4 shows the evolution of the velocities, the control angle, mass, and altitude through time, and Table 8 and 9 summarize the results obtained using the values above.

Table 8 Results Attained Using Minimization Technique

r_f	θ_f	u_f	v_f	m_f
1737.4x10^{3}m	1.581rad	1.51x10^{-3}m/s	1.850x10^{-3}m/s	172.71kg

Table 9 Values that Minimize the Objective Function

t_f	λ_{r0}	$\lambda_{\theta 0}$	λ_{u0}	λ_{v0}	λ_{m0}
225.41s	2.203x10^{-2}	2.443	6.134x10^{-1}	9.620x10^{-1}	-1.709

The results in Figure 7 were attained with the same values from Table 6 and 7 with the difference that the altitude was reduced to $y_0 = 10000m$. The results are summarized in the following tables.

Table 10 Results Attained for Initial Altitude of 10000m

r_f	θ_f	u_f	v_f	m_f
1737.4x10^{3}m	1.574rad	2.125 x10^{-8}m/s	5.123x10^{-8}m/s	188.642kg

Table 11 Minimization Values for Initial Altitude of 10000m

t_f	λ_{r0}	$\lambda_{\theta 0}$	λ_{u0}	λ_{v0}	λ_{m0}
155.394s	6x10^{-2}	1.714	9.064x10^{-1}	-2.124x10^{-1}	-2.2843

Notice in Figure 6 that the direction of the angle changes rapidly at about 40 seconds. As the denominator in the arctangent approaches zero the angle changes from pi/2 to –pi/2. The tangential velocity increases in the first 40 seconds, then the angle of

the engine changes to start breaking. Recall that the engine is turned on at full power, so if the direction of the angle was such that the lander started decelerating at t_0 it would reach the target velocity before reaching the target altitude. It could be said that the engine was turned on too early, so that the optimizer finds the best route that satisfies the terminal conditions with the starting altitude. Notice in figure 7 where the initial altitude was decreased that the angle of the thrust direction does not change suddenly as when the initial altitude was higher. In this case, the direction of the thrust is such that the lander starts decelerating when the engine is turn on, so that it was able to find a landing curve that satisfied all terminal conditions without having to accelerate at the beginning.

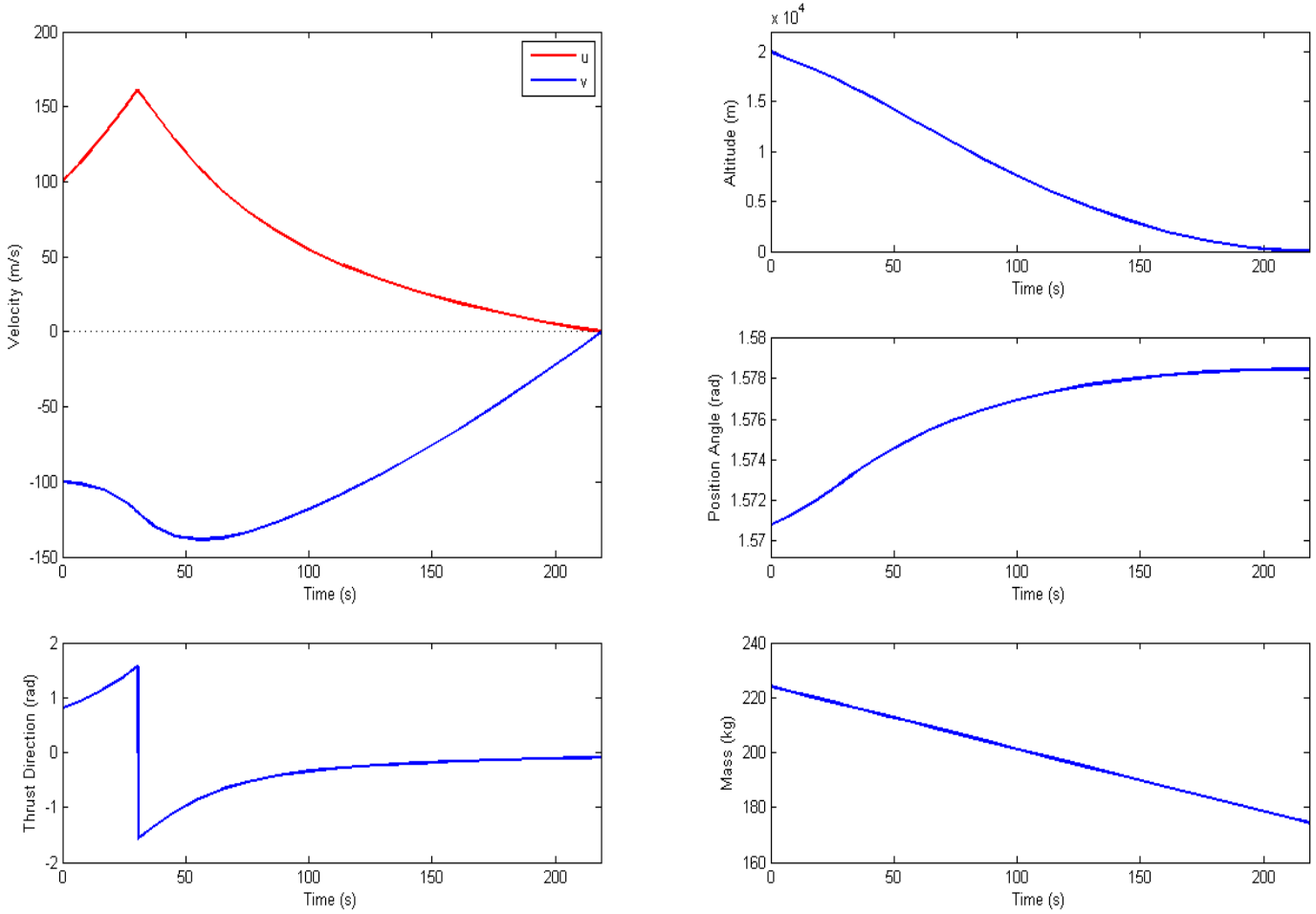

Figure 6 Results Using Minimization Technique

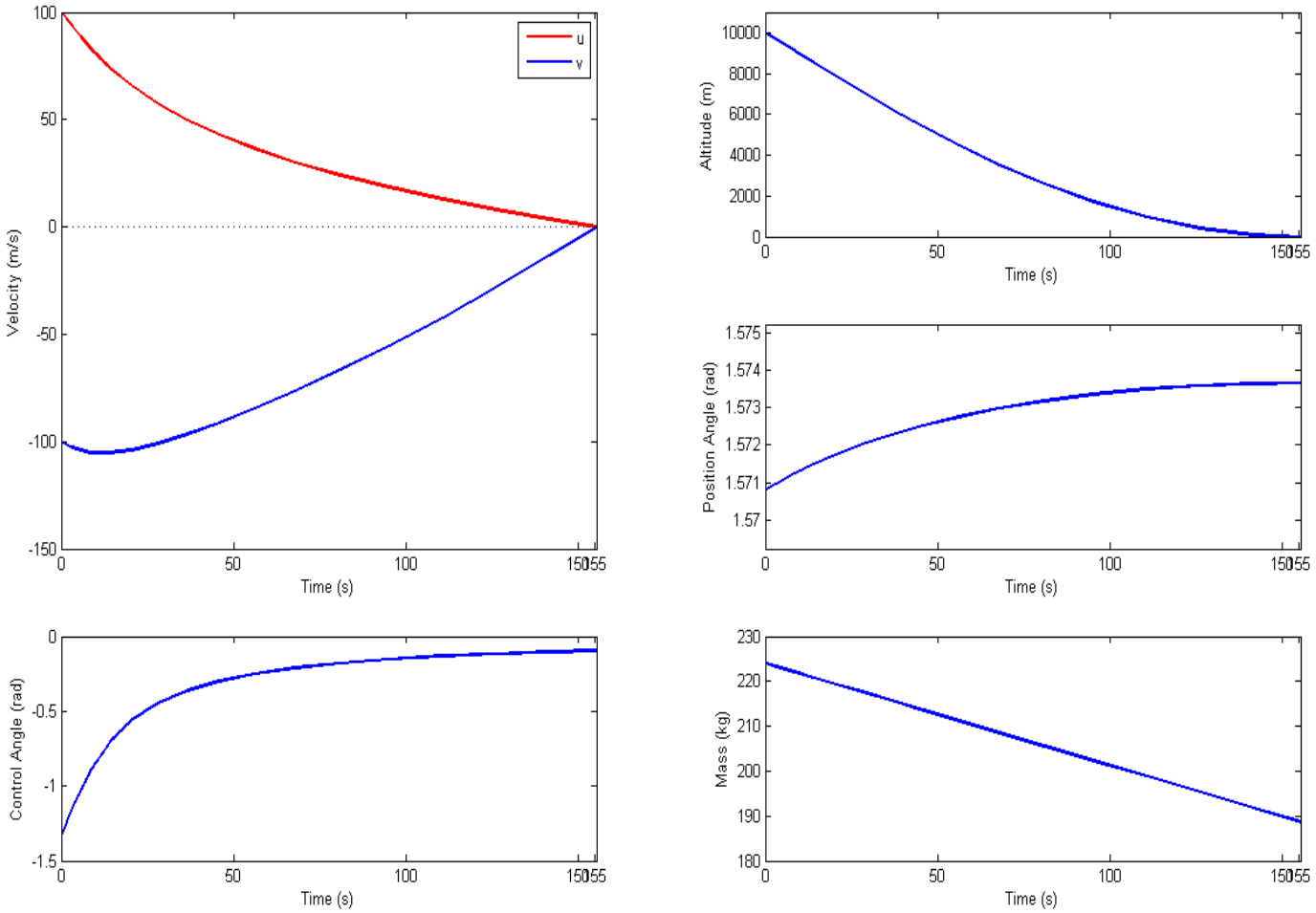

Figure 7 Results with an Initial Altitude of 10000m

Note from Table 8 that the results are very close to the desired terminal states, with a difference of 10^{-3} for the velocities, and 10^{-6} for the radius. In first trials *ode45* was used to solve the system of ODEs, but after noticing some stiff behavior, as shown in Figure 6 in the Velocity and Control Angle graphs, it was decided to use a stiff solver (*ode23s*) to better capture the behavior in the areas where rapid changes occurred. Also, it is important to note the relevance of scaling the equations due to the large difference among the values that the function is trying to minimized, the radius is in the order of 10^6 while the velocities are in the order of 10^2. Another important remark is that although some guesses had to be made; convergence was achieved with different guesses attaining acceptable results, meaning reaching the desired radius (r_{moon}) with velocities <2 m/s [5, p. 9], showing that this method allows more flexibility than Newton's shooting method or the two boundary value solver *bvp4c* when it comes to making those initial guesses. It is worth noting that other minimization methods such as simulated annealing and swarm type global methods also failed to provide convergence to a satisfactory answer.

Chapter 5

Summary and Conclusion

The simple case of the vertical soft landing on the moon was solved with common methods for solving two boundary value problems; however, in a more complicated case when the tangential velocity is greater than zero, the methods used for the vertical landing case proved insufficient. A minimization technique was proposed in which the two boundary value problem resulting from the optimization and control analysis was reformulated as a minimization problem. The cost function was augmented to include all terminal conditions, so that the objective function was the sum of squares of the evaluated and desired terminal states. The two boundary value problem was treated as an initial value problem where the variables that minimized the objective function (the augmented cost functional) were the unknown initial values and the final time. Most papers do not make it clear that it is likely that the two boundary value problem will not have an exact solution and thus should be considered as a minimization problem. Some authors such as [5] used a similar approach to the one proposed in this paper but minimizing the final position alone and subjecting it to velocity constrains of <2m/s, however, to formulate the problem in this way and obtain a solution, the author had to approximate the solution of the state equations as a first order Taylor series. Another author [4] uses controlled random search to minimize an objective function such as the cost function (42), arriving to successful results when the vertical velocity constraint is relaxed to a target velocity of 5m/s. Thus, it has been shown that reformulating the objective function as in (48) and implementing it to a Nelder-Mead optimizer with unknown initial conditions bounded is a successful new approach to solving the problem

in case two. It is important to note that minimization problems are hard to solve and may fail depending on the initial guesses, and it is essential to understand the limitations of the routine used. An important limitation of the function used to find the unknown initial values is that it might only give local solutions. Another limitation is that the solution varies depending on the given starting point (in this case the initial unknown values), on the other hand, it converges to adequate solutions given different initial values which shows that it is more advantageous than Newton's shooting method or *bvp4c* when making initial guesses that will lead to convergence. In conclusion, in spite of the limitations the results from Table 8 show that reformulating the two boundary value problem as a minimization problem as described in (48) is an effective technique for solving the optimization problem described in chapter 4.

References

[1] NASA, "International Lunar Network," [Online]. Available: http://iln.arc.nasa.gov/welcome. [Accessed 23 6 2013].

[2] Y. Gao, A. Phipps, M. Taylor, J. Clemmet, D. Parker, C. Ian, B. Andrew, W. Lionel, d. S. C. Alex, P. Davies, M. Sweeting and A. Baker, "UK Lunar Science Missions: MoonLITE and Moonraker," in *DGLR International Symposium "To Moon and Beyond"*, Bremen, 2007.

[3] B. Jeong, D. Lee and H. Bang, "Optimal Perilune Altitude of Lunar Trajectory," *International Journal of Aeronautical and Space Sciences,* vol. 10, no. 1, pp. 67-74, 2009.

[4] R. V. Ramanan and M. Lal, "Analysis of optimal strategies for soft landing on the Moon from lunar parking orbits," *Journal of Eath System Science,* vol. 114, no. 6, pp. 807-813, 2005.

[5] J. Guo and C. Han, "Design of Guidance Laws for Lunar Pinpoint Soft Landing," *Advances in the Astronautical Sciences,* vol. 135, no. 9, pp. 2133-2145, 2009.

[6] L. C. Evans, "An Introduction to Mathematical Optimal Control Theory. Version 0.2," [Online]. Available: http://math.berkeley.edu/~evans/control.course.pdf. [Accessed 23 6 2013].

[7] A. E. Bryson and Y.-C. Ho, Applied Optimal Control: Optimization, Estimation and Control, Washington D.C.: Hemisphere, 1975.

[8] J. Meditch, "On the problem of optimal thrust programming for a lunar soft landing," *Automatic Control, IEEE Transactions,* vol. 9, no. 4, pp. 477-484, 1964.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, "Two Boundary Value Problem," in *Numerical Recepies in C*, Cambridge, Cambridge University Press, 1992, p. 754.

[10] J. Shampine and L. Kierzenka, "A Sixth-Order Extension to the MATLAB bvp4c," Deparment of Mathematics. Imperial College London, London, 2006.

[11] L. F. Shanpine, I. Gladwell and S. Thompson, in *Solving ODEs with MATLAB*, Cambridge, Cambridge University Press, 2003, pp. 161-164.

[12] The MathWorks Inc., "bvp4c," 1994-2013. [Online]. Available: http://www.mathworks.com/help/matlab/ref/bvp4c.html. [Accessed 15 07 2013].

[13] The MathWorks Inc., "bvset," 1994-2013. [Online]. Available: http://www.mathworks.com/help/matlab/ref/bvpset.html. [Accessed 15 07 2013].

[14] J. D'Errico, "MATLAB CENTRAL: File Exchange. fminsearchbnd, fminsearchcon," The MathWorks Inc, 06 02 2012. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd. [Accessed 17 07 2013].

[15] MathWorks Inc, "fminsearch," 1994-2013. [Online]. Available: http://www.mathworks.com/help/matlab/ref/fminsearch.html. [Accessed 17 07 2013].

[16] M. Manoranjan, J. D. Turner and J. L. Junkins, "Solution of Two-Point Boundary-Value Problems Using Lagrange Implicit Function," *Journal of Guidance, Control and Dynamics*, vol. 32, no. 5, pp. 1684-1687, 2009.

Biographical Information

Lizeth Ocampo graduated from Embry Riddle Aeronautical University in Fall 2010 with a B.S. in Engineering Physics. In Fall 2011 she started her graduate studies in University of Texas at Arlington to pursue a degree in M.S. in Mathematics.

Lizeth plans to pursue a career in the field of engineering and mathematics.