5-2015

Numerical Methods for Solving Optimal Control Problems

Garrett Robert Rose
University of Tennessee - Knoxville, grose3@vols.utk.edu

Numerical Methods for Solving Optimal Control Problems

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Garrett Robert Rose
May 2015

Dedication

To my parents and brother
"If you're into that sort of thing."

Acknowledgements

I would like to take a second to thank those who help with either this thesis, or my academic career in general. First and foremost, I would like to thank Dr. Chuck Collins for being my advisor throughout this entire process; guiding, helping, and critiquing me when needed. His enthusiasm helped drive and push me towards reaching my goals. Also, a special thanks to the others on my thesis committee: Dr. Suzanne Lenhart and Dr. Abner Salgado. I really appreciate them taking time out of their schedules and going over the all the work I did and giving me their input.

Next, I would like to think the professors I have had at the University of Tennessee. They expanded my knowledge and understanding of mathematics well beyond what I thought possible when I entered in 2012. Though it was difficult at times, I have emerged from the other side a better mathematician for it.

Though the professors I have had in the last three years helped shape my mathematical knowledge into what it is today, none of that would be possible without the professors I had at my undergrad, Tennessee Wesleyan College: Dr. David Ashe, Dr. Sharon Brown, Mr. Kurt Mclaughlin, and Dr. Jianbing Niu. These four professors gave me the mathematical knowledge needed to succeed at the graduate level.

Finally, a thanks to my fellow students at the University of Tennessee for going over my code, helping me work out ideas, and just giving their input. In math, as in life in general, other perspectives are good and help flush out ideas. As well as my fellow cohorts in the graduate school learning atmosphere, I want to thank those I went to undergrad with. They taught me other perspectives on certain problems as well as mathematics that helped shaped the way I think and approach problems.

Abstract

There are many different numerical processes for approximating an optimal control problem. Three of those are explained here: The Forward Backward Sweep, the Shooter Method, and an Optimization Method using the MATLAB Optimization Tool Box. The methods will be explained, and then applied to three different test problems to see how they perform. The results show that the Forward Backward Sweep is the best of the three methods with the Shooter Method being a competitor.

Table of Contents

Chapter 1: Introduction 1

Chapter 2: General Set up 2

Section 1: Optimization 2

Section 2: Optimal Control Theory 3

Section 3: Numerical Processes 7

Chapter 3: Test Problems 9

Section 1: Problem 1 9

Section 2: Problem 2 10

Section 3: Problem 3 12

Chapter 4: Forward Backward Sweep 14

Section 1: Analytical Process 14

Section 2: Convergence 18

Chapter 5: Shooter Method 19

Section 1: Analytical Process 19

Section 2: Convergence 23

Chapter 6: Direct Optimization Process 27

Section 1: Analytical Process 27

Section 2: Convergence 29

Chapter 7: Processes Applied to Problems 30

Section 1: Forward Backward Sweep 30

Section 2: Shooter Method 40

Section 3: Optimization 46

Section 4: Comparison of Results 50

Chapter 8: Conclusion 53

List of References 54

Appendices 56

Appendix I: Glossary 57

Appendix II: Code Explanation 59

Vita.. 64

List of Tables

Table 1: Analytical Process 6
Table 2: Comparison of convergence for the root finding methods 25
Table 3: Accuracy of the root finding methods 26
Table 4: l^1-norms of Errors in State, Adjoint, Control for FBS 38
Table 5: l^1-norms of Errors in State, Adjoint, Control for Shooter 44
Table 6: Iteration results from the Optimization Method 47
Table 7: l^1-norms of Errors in the State, and Control for Optimization 49
Table 8: Comparison of methods table 51

List of Figures

Figure 1: Graphs for $A = B = C = 5$ $x_0 = 1$ to problem (P1). 17

Figure 2: 3-D visualization of how A, B, and C effect the number of iterations 31

Figure 3: The lower iteration rate region 32

Figure 4: The middle iteration rate region 32

Figure 5: The highest iteration rate region.. 33

Figure 6: The slice along the back left edge to the front right edge. 34

Figure 7: The slice along the bottom left edge to the top right edge. 34

Figure 8: Data represented in two dimensions... 35

Figure 9: Graphs produced by FBS to (P2) 36

Figure 10: Comparison of the State for FBS 37

Figure 11: Comparison of the Adjoint for FBS 37

Figure 12: Comparison of Control for FBS 38

Figure 13: FBS Solution to (P3) 39

Figure 14: The parameters that work or do not work 40

Figure 15: The parameters that work or do not work with previous information......................... 41

Figure 16: Results of Shooter Method applied to (P2) 42

Figure 17: Comparison of State for Shooter 43

Figure 18: Comparison of Adjoint for Shooter 43

Figure 19: Comparison of Control for Shooter 44

Figure 20: Results to (P3) produced by the Shooter Method... 45

Figure 21: Graphs of (P2) from Optimization Method 47

Figure 22: Comparison of State for Optimization 48

Figure 23: Comparison of Control for Optimization 48

Figure 24: Optimization results for (P3) 49

Chapter 1:
Introduction

Numerical mathematics is study of quantitative approximations to the solutions of mathematical problems including consideration of and bounds to the errors involved. Optimal control theory is no exception to this rule. The purpose here is to implement three different numerical algorithms in MATLAB to approximate the solution to an optimal control problem. Once the methods are developed, the concept of convergence for each method will be discussed as well as any flaws or problems with each specific method. After this, the three methods will be used to find the solution to three different test problems in order to see how the methods work and compare their results to each other. Each of three problems is chosen for specific reasons which will be explained in detail later on. Finally, a ‘winner’ will be chosen, if possible, from the results of each method applied to the three test problems, in order to see which method is best.

Chapter 2: General Set up

This thesis is dedicated to comparing different numerical processes for solving an optimal control problem. Though only a few specific problems will be studied, some general theory and processes must be established first before any specific details can be discussed. This chapter will be broken into three separate sections. The first section will be dedicated to discussing general optimization; the second will cover optimal control theory; and the third will discuss the specific details needed for the family of problems in question.

Section 1: Optimization

The first idea that needs to be set up and defined is what an optimization problem is and its relevance. In mathematics, **optimization** is the process in which the best feasible solution for a problem is found. This usually entails finding either a maximum or minimum, which are called **extrema**, of the possible solutions. This can be done in various ways, though the most common involves using some version of the derivative of the function.

In optimization, when discussing extrema, a point needs to be made to determine if the extrema in question is over the whole domain of the function or just over a certain interval or region. If f has a maximum (or minimum) over the entire domain, D, of the function, this is called the absolute maximum (or minimum). This means is that, for some c in the domain of f, $f(c) \geq f(x)\ \forall x \in D$ (or $f(c) \leq f(x)\ \forall x \in D$). These extrema values are referred to as **global** extrema. However, these are not the only type of extreme; there are **local** extrema are when there exist a maximum (or minimum) on a small interval, I, such that $I \subset D$. This means that for some $d \in I$, $f(d) \geq f(x)\ \forall x \in I$ (or $f(d) \leq f(x)\ \forall x \in I$).

When it comes to whether or not there even exists an extrema value, a reference can be made back to the Extreme Value Theorem [5], which states: If $f: U \to \mathbb{R}$, where $U \subset \mathbb{R}^n$, is continuous over a closed interval, $[a, b]$, then f attains an absolute maximum value, $f(c)$, and a absolute minimum value, $f(d)$, for some numbers $c, d \in [a, b]$. For more on this, see [6] and [7].

Section 2:
Optimal Control Theory

From a general perspective, an optimal control problem is an optimization problem. The difference between the two is that, in optimal control theory, the optimizer is a function, not just a single value. This function that optimizes is called the optimal control. The technical definition of an **optimal control problem** is the process of determining control and state trajectories for a dynamic system over a period of time to minimize a performance index. The **state** variable (or function) is the set of variables (functions) used to describe the mathematical state of the system. The **control** or **control function** is an operation that controls the recording, processing, or transmission of data. These two functions drive how the system works and how the desired control is found. With these definitions, a basic optimal control problem can be defined. This basic problem will be referred to as our standard problem (SP).

Standard Problem

(SP)

$$\max_u J(u) \text{ where } J(u) = \int_{t_0}^{t_1} f\big(t, x(t), u(t)\big)\, dt \tag{2.01}$$

$$x'(t) = g\big(t, x(t), u(t)\big) \tag{2.02}$$

$$x(t_0) = x_0, x(t_1) \text{ is free} \tag{2.03}$$

The optimal control, u^*, is the function that optimizes the **objective function**, $J(u)$, as seen in (2.01). This control is not bounded. The other arguments in equation (2.01) are t, which is the time variable, and $x(t)$, which is the state equation. The relationship between u and x is defined by equations (2.02) and (2.03) and is denoted by the relationship in the map $u(t) \rightarrow x = x(u)$. Though this relationship does indeed exist, x is really just a function of the independent time variable, but in writing $x(u)$, the dependence that x has on u is shown. Equation (2.02) is the constraint equation on the state, and the **initial** and **terminal conditions** are given by (2.03). By setting $x(t_1)$ to be free, this simply means that the state can grow over time unconditionally.

To solve our basic optimal control problem, a set of what is called necessary conditions must be satisfied. In mathematics, a **necessary condition** is a condition that must be satisfied for a statement to be true, but that does not in and of itself make it true. In regards to (SP), there are such conditions that must be satisfied in order to solve the problem. In the 1950's, a Russian mathematician by the name of Lev Pontryagin and his co-workers in Moscow derived such conditions. Pontryagin introduced the **adjoint function** to affix to the differential equation to the objective functional. These functions serve a similar purpose as the **Lagrange multipliers** in multivariable calculus. The derivation of these results can be found in [1]. The next few paragraphs will summarize these results.

The necessary conditions needed to solve the basic problem are derived from what is referred to as the Hamiltonian, H, which is given by equation (2.04).

$$H(t,x,u,\lambda) = f(t,x,u) + \lambda g(t,x,u) \tag{2.04}$$

Here λ denotes the adjoint and is dependent on t, x, and u. Using this, Pontryagin determined that the following conditions are satisfied by the optimal control, denoted as u^*, when the Hamiltonian is nonlinear in u.

$$\frac{\partial H}{\partial u} = 0 \text{ at } u^* \Longrightarrow f_u + \lambda g_u = 0 \qquad \text{Optimality Condition} \tag{2.05}$$

$$\lambda' = -\frac{\partial H}{\partial x} \Longrightarrow \lambda' = h(t,x,\lambda,u) - (f_x + \lambda g_x) \qquad \text{Adjoint Equation} \tag{2.06}$$

$$\lambda(t_1) = 0 \qquad \text{Transversality Condition} \tag{2.07}$$

$$\begin{cases} x' = g(t,x,u) \\ x(t_0) = x_0 \end{cases} \qquad \text{Dynamics of the State Equation} \tag{2.08}$$

With these conditions, there is now a process on how to solve the standard problem defined by SP. This process can be found in Table 1.

Table 1: Analytical Process

(1) Form the Hamiltonian (2.04) for the problem.

(2) Write the adjoint differential equation, transversality boundary condition, and the optimality condition in terms of three unknowns, u^*, x^*, and λ.

(3) Use the optimality equation $H_u = 0$ to solve for u^* in terms of x^* and λ.

(4) Solve the two differential equations for x^* and λ with two boundary conditions.

(5) After finding the optimal state and adjoint, solve for the optimal control using the formula derived by step (3).

If it is possible to solve for the optimal control in terms of x^* and λ, then the formula for u^* is called the characterization of the optimal control. The state equation and adjoint equations together with the characterization and boundary conditions are called the **optimality system**.

Now that the process on how to solve SP has been defined, it should be noted that it is not enough to simply solve the necessary conditions in order to solve the optimal control problem. Justification for the found solutions to be the actual solution for (SP) requires examining some **existence** and **uniqueness** conditions. A true existence results guarantees an optimal control, with finite objective functional. Such results usually require restrictions on either f or g or even possibly both. For the analysis of the methods, an assumption of existence will be made, but for reference on existence and uniqueness, refer back to [1].

Existence is only half of what is desired. Uniqueness of the optimal control is also needed. Suppose an optimal control exists, u^*, such that $J(u) \leq J(u^*)$ for all controls u. Now,

u^* is unique if and only if $J(u^*) = J(u)$. This implies that $u^* = u$ at all but finitely many points. In this case, the associated states will be identical. The state x^*, is the unique optimal state.

In most cases, if the solution to the state system is unique, then the corresponding optimal control is also unique. This, however, can only be said for small time intervals.

Now, in general, uniqueness of the optimal control does not always imply that there is a unique optimality system. To prove the uniqueness of the optimal control directly, the objective functional $J\big(t, x(u)\big)$ must have strict **concavity** established. However, this process is, in most cases, difficult to prove. Thus, other ways to prove uniqueness must be found, such as proving f,g and the right hand side of the adjoint equation are **Lipschitz** in their state and adjoint arguments. This only proves uniqueness for small time periods. Sometimes, one must bound the optimality system to get this property easily.

Section 3:
Numerical Processes

Though most problems have a theoretical answer, it is, in practice, very difficult to find explicitly. Hence the necessity of **numerical processes**. Like mentioned in Section 2.2, the main analytical technique is provided by Pontryagin's Maximum Principle which gives necessary conditions that the control and the state need to satisfy. These conditions can be solved explicitly sometimes; however, for most problems, the conditions are too complicated to be solved explicitly. This is especially true for problems that also involve additional constraints on the state or the control. Because of these, numerical approaches are used to construct approximations to these difficult equations.

One of these numerical processes is needed for all the methods. What is needed is a method to solve ordinary differential equations and systems of differential equations. For this, the Runge-Kutta algorithm will be used to solve such problems. Though there are many different adaptations of Runge-Kutta, only the method in its classical, fourth order will be used. The fourth order classical Runge-Kutta (RK4) method approximates the solution to the problem $y' = f(t, x)$.

Classical, fourth order Runge-Kutta Algorithm RK4

$$k_1 = f(t_n, x_n)$$
$$k_2 = f\left(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_2\right)$$
$$k_4 = f(t_n + h\,, x_n + hk_3)$$
$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Here, x_{n+1} is the RK4 approximation of $x(t_n + h)$; here h is the step size. x_{n+1} is calculated using the current value of x_n plus the weighted average of four values, k_i. Each of the k_i values are determined for each n step, then are overwritten for the next step; k_1 is the increment based on the slope of the beginning of the interval; k_2 and k_3 are both based on the midpoint of the interval, and lastly, k_4 is based on the slope at the end of the interval. The Runge-Kutta Method has an error that is $O(h^4)$, where h is the step size and also it is conditionally stable. The proof and further explanation of these ideas can be found in various texts, one being [2].

Chapter 3: Test Problems

Section 1: Problem 1

Now that the general set up is done, the discussion can be focused on the desired family of problems. This family can be found in [2] and will be referred back to as the Problem 1 (P1).

Problem 1 (P1)

$$\max_u \int_0^1 Ax(t) - Bu(t)^2 \, dt \tag{3.01}$$

$$\text{subject to } \begin{cases} x'(t) = -\frac{1}{2}x(t)^2 + Cu(t) \\ x(0) = x_0 > -2 \\ A \geq 0, B > 0 \end{cases} \tag{3.02}$$

The restriction on B is so that this is indeed a maximization problem. Before any method can be developed, there are a few key ideas that will be needed through all methods. The first thing that is needed is the Hamiltonian, as defined by (2.04).

$$H = Ax - Bu^2 - \frac{1}{2}\lambda x^2 + C\lambda u \tag{3.03}$$

Using this, the optimality condition, as defined in (2.05), for this specific problem is

$$0 = \frac{\partial H}{\partial u} = -2Bu + C\lambda \Longrightarrow u^* = \frac{C\lambda}{2B} \tag{3.04}$$

This clearly gives us an explicit formulation for the optimal control, which is only directly depends on the adjoint, though the state affects it through the state's relationship to the adjoint. The final piece of setup is the two differential equations that will be used to solve for our optimal control. One solves for the state and the second in turn solves the adjoint.

$$\begin{cases} x'(t) = \frac{1}{2}x^2 + Cu \\ x(0) = x_0 \end{cases} \tag{3.05}$$

$$\begin{cases} \lambda'(t) = -A + x\lambda \\ \lambda(1) = 0 \end{cases} \tag{3.06}$$

Note that the ODE in (3.06) was derived from (2.06) and (2.07). The solution is now completely described by these two ODE's and the equation for u^* in (3.04).

This problem is used to initially test the three methods due to its changeable parameters and initial state value. Also because of this fact, it produced many more results to discuss later in Chapter 7.

Section 2: Problem 2

The second problem that will be used to test the process can be found in [3]. This problem will be referred to later to as Problem 2 (P2).

Problem 2 (P2)

$$\min_u \frac{1}{2}\int_0^1 x(t)^2 + u(t)^2\, dt \tag{3.07}$$

$$\text{subject to } \begin{cases} x'(t) = -x(t) + u(t) \\ x(0) = 1 \end{cases} \tag{3.08}$$

Once again, to construct the adjoint ODE, the Hamiltonian must be constructed. Remember that from the Hamiltonian, not only is the adjoint ODE derived, but how to use it to find the approximated optimal control as well. The Hamiltonian for (P2) is derived to be:

$$H = \frac{1}{2}x^2 + \frac{1}{2}u^2 - \lambda x + \lambda u \tag{3.09}$$

Using the Hamiltonian in (3.09), as defined by equation (2.06) and (2.07), the state and adjoint ODE's are given by equation (3.10) and (3.11).

$$\begin{cases} x'(t) = -x + u \\ x(0) = 1 \end{cases} \tag{3.10}$$

$$\begin{cases} \lambda'(t) = x - \lambda \\ \lambda(1) = 0 \end{cases} \tag{3.11}$$

Once again, we use the optimality condition defined in (2.05) to find the formula for the optimal control, u^*.

$$0 = \frac{\partial H}{\partial u} = u + \lambda \Rightarrow u^* = -\lambda \tag{3.11}$$

Thus defining everything to find the solution to (P2). This problem is important because from [3], the real solution is given. With the actual solution to (P2), the accuracy of the three methods can be tested. The real solution for the state and adjoint are given in equations (3.12) and (3.13).

$$x(t) = \frac{\sqrt{2}\cosh\left(\sqrt{2}(t-1)\right) - \sinh\left(\sqrt{2}(t-1)\right)}{\sqrt{2}\cosh(\sqrt{2}) + \sinh(\sqrt{2})} \tag{3.12}$$

$$\lambda(t) = -\frac{\sinh\left(\sqrt{2}(t-1)\right)}{\sqrt{2}\cosh(\sqrt{2}) + \sinh(\sqrt{2})} \tag{3.13}$$

Section 3:
Problem 3

The last problem can be found in [1]. This problem will be referred back to as Problem 3 (P3).

Problem 3 (P3)

$$\min_{u} \int_0^1 x(t) + u(t)dt \tag{3.14}$$

$$\text{subject to } \begin{cases} x'(t) = 1 - u(t)^2 \\ x(0) = 1 \end{cases} \tag{3.15}$$

It needs to be stated that Problem 3 is a minimization problem, so when the methods are applied later, the negative of the objective function will be used since the algorithms are designed to find the maximum. Other than that, the construction of all the necessary pieces to solve for the solution are found the same way. First is the Hamiltonian, then the optimality condition, then finally the state and adjoint ODE's.

$$H = x + u + \lambda - \lambda u^2 \tag{3.16}$$

$$0 = \frac{\partial H}{\partial u} = 1 - 2\lambda u \Rightarrow u^* = \frac{1}{2\lambda} \tag{3.17}$$

$$\begin{cases} x'(t) = 1 - u^2 \\ x(0) = 1 \end{cases} \tag{3.18}$$

$$\begin{cases} \lambda'(t) = -1 \\ \lambda(1) = 0 \end{cases} \tag{3.19}$$

One thing to note about this problem is the relationship of the control to the adjoint. The optimal control is inversely related to the adjoint, which causes the control to have issues as time approaches 1. Thus this problem does not have a solution. This problem was used to see how the three methods handle this fact: to see what the methods do when there is not supposed to be an optimal control.

Chapter 4: Forward Backward Sweep

Section 1: Analytical Process

The first method that will be discussed is the Forward Backward Sweep (FBS). This iterative method is named based on how the algorithm solves the problem's state and adjoint ODE's. Given an approximation of the control function, FBS first solves the state 'forward' in time (from t_0 to t_1) then solves the adjoint 'backward' (from t_1 to t_0). Once it has found the state and adjoint functions, the control is updated based on (2.05) and then the state, control, and adjoint are tested for **convergence** against a user provided **tolerance** and depending on that, the algorithm eithers starts the process over using the updated control or the algorithm terminates with the final approximations for the state, adjoint, and control functions considered as the solution to the optimal control problem. The code developed is based heavily on the code listed in [1], which was based on work from [8], but it has been generalized so that it can be used to solve other problems, not just the problem (P1), for which it was built for.

Before starting, an initial value is needed for the control vector. In every case, this initial value is a $N + 1$ vector of zeros. With this, the FBS can begin and it does so with the state ODE. To solve the state ODE, a simple RK4 method is applied, but to solve the adjoint ODE, the RK4 method has to be adapted to account for solving backwards in time. This however is the only difference between the two RK4 algorithms. The first algorithm below is a translation of the RK4 to work for 3 inputs, and the second is from the RK4 outfitted for 4 inputs and to solve backwards. In both algorithms, the i represents the i^{th} element of the vector.

<u>Runge-Kutta 4 (with 3 input update) Algorithm</u> URK4

$$K_1 = f(t_i, x_i, u_i)$$
$$K_2 = f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_1, \frac{1}{2}\,(u_i + u_{i+1})\right)$$
$$K_3 = f\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_2, \frac{1}{2}\,(u_i + u_{i+1})\right)$$
$$K_4 = f(t_i + h, x_i + hK_3, u_{i+1})$$

$$x_{i+1} = x_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

<u>Backward Runge-Kutta 4</u> BRK4

$$j = N + 2 - i$$

$$K_1 = f(t_j, \lambda_j, x_j, u_j)$$
$$K_2 = f\left(t_j - \frac{h}{2}, \lambda_j - \frac{h}{2}K_1, \frac{1}{2}\,(x_j + x_{j-1}), \frac{1}{2}(u_j + u_{j-1})\right)$$
$$K_3 = f\left(t_j - \frac{h}{2}, \lambda_j - \frac{h}{2}K_2, \frac{1}{2}\,(x_j + x_{j-1}), \frac{1}{2}(u_j + u_{j-1})\right)$$
$$K_4 = f(t_j - h, \lambda_j - hK_3, x_{j-1}, u_{j-1})$$

$$\lambda_{j-1} = \lambda_j - \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

Looking at the algorithms, it can be seen that the major difference in URK4 and BRK4 is that the index counts down towards one instead of counting forward and all the time steps are negative.

Now the algorithm has a state and a control for the current step, but before the program can test for convergence, the actual control needs to be calculated. This means the actual control for the current step is some mixture of the current control, u_{new}, and the control from the past step, u_{old}. This can be done in many ways. One can simply take all of u_{new} and disregard u_{old} all together. Another is taking the average of the u_{new} and u_{old} and the last is an adaptive scheme. This adaptive scheme is seen in equation (4.01). In (4.01) the variable c^k is a constant such that $0 < c < 1$ and k is the iteration number, not an exponent.

$$u = u_{new} * (1 - c^k) + u_{old} * c^k \tag{4.01}$$

Generally when this method is used, the larger k gets, the less and less of the current control is used in the mixture. Generally by doing this, the algorithm will converge faster, however in the three test problems, the difference in convergence was not substantial, thus the algorithm is set to take an average of the old control and the current control, though the code can easily be adapted to use the equation set up in (4.01)

Once these two processes are done and u has been calculated, the code calculates the error terms in order to check for convergence. In the FBS, at the end of each iteration, it tests the change between the newly calculated state, control and adjoint vector against the old state, control, and adjoint to see if the difference in each is small enough to stop the algorithm. In the FBS function, this is done when the test variable becomes positive. The test variable is the minimum of all of the relative errors of the state, adjoint, and control. The **relative error**, for the state vector, x, is given below. Note the k represents the iteration step, not the k^{th} element of x.

$$\frac{\left\|x^{(k)} - x^{(k+1)}\right\|_1}{\|x^{(k)}\|_1} \leq \delta \tag{4.02}$$

The relative error, as seen in equation (4.02) is then solved so that there is no division because it is possible that $\left\|x^{(k)}\right\|_1 \approx 0$. When this is done, the result is equation (4.03)

$$\delta\left\|x^{(k)}\right\|_1 - \left\|x^{(k)} - x^{(k+1)}\right\|_1 \geq 0 \tag{4.03}$$

When this is true for all three vectors being tested, the algorithm stops and the current control is the optimal control approximation.

As an example of the outputs, the FBS was applied to the (P1), and the results are displayed in Figure 1. In Figure 1, there are three graphs; the State, Control, and the Adjoint.

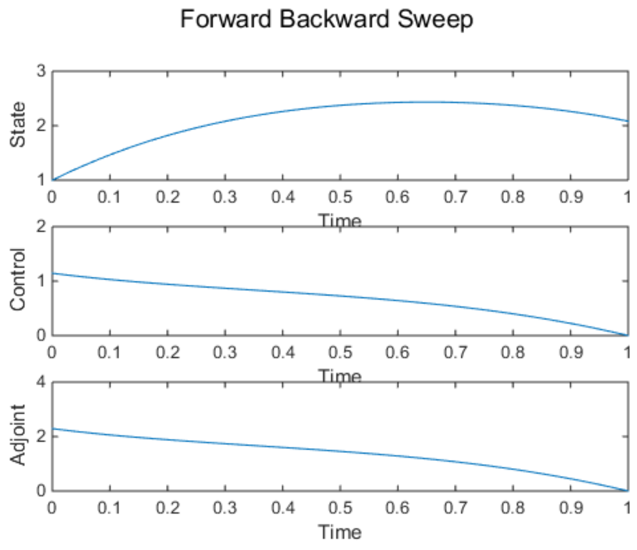

Figure 1: Graphs for $A = B = C = 5$ $x_0 = 1$ to problem (P1).

Section 2:
Convergence

Now that the process has been presented, a study of the convergence of the FBS is necessary. One result is from the paper [3]. The theorem states that if a Lipschitz condition is assumed for the integrand of (SP) and the equations for the state (2.02) and adjoint (2.03) ODE’s, and that there exists a constant c_0 (defined in the paper), then the FBS will converge if the c_0 is small enough. Another set of restrictions are that either the FBS works only if the Lipschitz constants for the state, adjoint, and control is small enough or the time interval is small. Because of these restrictions, this method does not work in most cases.

Chapter 5:
Shooter Method

Section 1:
Analytical Process

The Shooter Method (SM) is another way to solve an optimal control problem, like (SP). This method still solves the ODE's like the FBS with two exceptions: this method takes an initial value for the adjoint equation and solves it forward, and then using a root finding method for convergence, finds the initial time value that makes the adjoint equal to zero at time t_1.

Though the process of picking a new starting value for this process can be different, the overall algorithm works the same. A different take on this can be found in [1]. The algorithm first takes an initial interval. This interval is the range that contains an initial value for the adjoint (at t_0) will produce the desired end result of zero ($\lambda(t_1) = 0$). The algorithm tests the end points of the interval as well as the test value determined by the root finding method. If the test value does not produce a $\lambda(t_1)$ that is within tolerance of zero, it will use this information as well as the $\lambda(t_1)$ data about the endpoint to produce a new test value. The three ways that the algorithm does that is either by doing a bisection, secant, or regula falsi root finding scheme.

The Runge-Kutta algorithm here is actually slightly different than the one used in the FBS. This Runge-Kutta takes the vector formed by the state and adjoint ODE'S and runs the Runge-Kutta process once with both terms at the same time, thus it is solving the differential in equation (5.01).

$$\begin{cases} \Delta(t,\phi,u) = \begin{bmatrix} x' \\ \lambda' \end{bmatrix} = \begin{bmatrix} g(t,\phi_1,u) \\ h(t,\phi_1,\phi_2,u) \end{bmatrix} \\ \begin{bmatrix} x(0) \\ \lambda(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ \lambda_0 \end{bmatrix} \end{cases} \tag{5.01}$$

Here, one thing to note is what ϕ represents. It is a vector of the state and adjoint variable i.e. $\phi = \begin{bmatrix} x \\ \lambda \end{bmatrix}$.

Referring back to RK4, between each k_i values, the algorithm computes the value for the control with the current state and adjoint values, then used that to find the value of the next K_i value. This can be seen by observing the algorithm in SRK4.

<u>Runge-Kutta for Shooter Method</u> SRK4

$$X = \begin{bmatrix} x_i \\ \lambda_i \end{bmatrix} \qquad U = u(t_i, X_1, X_2)$$

$$K_1 = \Delta(t, X, U)$$

$$X = \begin{bmatrix} x_i \\ \lambda_i \end{bmatrix} + \frac{h}{2}K_1 \qquad U = u\left(t_i + \frac{h}{2}, X_1, X_2\right)$$

$$K_2 = \Delta\left(t + \frac{h}{2}, X, U\right)$$

$$X = \begin{bmatrix} x_i \\ \lambda_i \end{bmatrix} + \frac{h}{2}K_2 \qquad U = u\left(t_i + \frac{h}{2}, X_1, X_2\right)$$

$$K_3 = \Delta\left(t + \frac{h}{2}, X, U\right)$$

$$X = \begin{bmatrix} x_i \\ \lambda_i \end{bmatrix} + hK_3 \qquad U = u(t_i + h, X_1, X_2)$$

$$K_4 = \Delta(t + h, X, U)$$

$$\bar{X} = \begin{bmatrix} x_i \\ \lambda_i \end{bmatrix} + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

$$x_{i+1} = \bar{X}_1 \qquad \lambda_{i+1} = \bar{X}_2$$

By inspection, for each K_i value needed for the process, the algorithm computes the changes in the state and adjoint vector, then updates the control, and then computes the current K_i value. When this process is finished, it computes the next term for the state and adjoint, and then runs the algorithm again until it has computed each element of the corresponding vector.

Once the Shooter Method has successfully calculated the state and adjoint values—including the values using the left and right endpoints of the interval of initial adjoint values—a zero-finding method of the users choice will determine if the initial guess produces a value close enough to zero, or if an updated initial guess for the adjoint needs to be found. As mentioned before, there are three different **root finding methods** used for this algorithm: Bisection, Secant, and Regula-Falsi. For all three algorithms, let $\Lambda(\lambda_0)$ denote the process that sets the initial value for the adjoint as λ_0, i.e. $\lambda(t_0) = \lambda_0$, computes the adjoint and then sets $\Lambda(\lambda_0)$ as the value of the adjoint at t_1, i.e. $\Lambda(\lambda_0) = \lambda(t_1)$. In the Bisection and Regula-Falsi methods, an initial interval is needed. This interval, $[a_0, b_0]$, needs to exist such that ideal $\lambda_0 \in [a_0, b_0]$ and $\Lambda(a_0) \cdot \Lambda(b_0) < 0$. The Secant Method is a strict update of the value that moves closer to $\Lambda = 0$.

In the Bisection method, x_k is the value being tested to see if $\Lambda(x_k)$is close to zero. The Bisection method takes x_k and the interval $[a_k, b_k]$, determines which half the solution lies in, and then uses the midpoint of the half-interval as the next test value and updates the interval endpoints. This process can be found in ZF1. The Bisection method terminates when $|\Lambda(x_{k+1})| < \bar{\delta} \ll 1$.

Bisection ZF1

```
If $\Lambda(a_k) \cdot \Lambda(x_k) < 0$
    $a_{k+1} = a_k$
    $b_{k+1} = x_k$
    $x_{k+1} = \frac{1}{2}(a_{k+1} + b_{k+1})$
Else
    $a_{k+1} = x_k$
    $b_{k+1} = b_k$
    $x_{k+1} = \frac{1}{2}(a_{k+1} + b_{k+1})$
```

The next breakdown is for the Secant Method. It differs from Bisection and Regula Falsi because it is an update of the value, not of the interval. The way it does that is by taking the previous two values, x_k and x_{k+1}, and constructs the secant line between these two values. The point in which the secant line is zero is the next value in the sequence, x_{k+2}. The formula for this is found in ZF2. This method terminates when $|\Lambda(x_{k+2})| < \bar{\delta} \ll 1$.

Secant ZF2

$$x_{k+2} = x_{k+1} - \Lambda(x_{k+1})\frac{x_{k+1} - x_k}{\Lambda(x_{k+1}) - \Lambda(x_k)}$$

The last method is the Regula Falsi method. This method is a blend of the last two. It updates the interval like Bisection, but instead uses the Secant Method value instead of the midpoint. The method can be found in ZF3. The Regula Falsi method terminates, like the last two methods, when $|\Lambda(x_{k+1})| < \bar{\delta} \ll 1$.

Regula-Falsi ZF3

```
If $\Lambda(a_k) \cdot \Lambda(x_k) < 0$
    $a_{k+1} = a_k$
    $b_{k+1} = x_k$
    $x_{k+1} = b_{k+1} - \frac{\Lambda(b_{k+1})(b_{k+1} - a_{k+1})}{\Lambda(b_{k+1}) - \Lambda(a_{k+1})}$
Else
    $a_{k+1} = x_k$
    $b_{k+1} = b_k$
    $x_{k+1} = b_{k+1} - \frac{\Lambda(b_{k+1})(b_{k+1} - a_{k+1})}{\Lambda(b_{k+1}) - \Lambda(a_{k+1})}$
```

In regards to the Shooter Method, once the root finding method has found a value, it tests it to see if it is small enough. If it is, then the algorithm terminates and the current approximations for the state, adjoint, and control are the solution. If not, it loops back through the algorithm with updated initial conditions and starts the process over again.

Section 2: Convergence

The convergence of the Shooter Method depends on three things. The first two are the two numerical processes that make up the method: Runge-Kutta and a root finding method. The last dependence is initial data set. This section will discuss how each method affects the convergence. When it comes to converging, it is known from the theory discussed in Chapter 2 that Runge-Kutta will find an approximate solution for small enough h. To make sure h is small enough, the number of mesh points, N, needs to be large. Thus the root finding method convergence is what needs to be shown. From [4], the proofs of convergence for all three methods are given. All three methods convergence is based on the Intermediate Value Theorem, which states that if a function, $p(x)$, is continuous over a closed interval, $[a, b]$, and if $p(a) \cdot p(b) < 0$, then there exists a value $\xi \in [a, b]$ such that $p(\xi) = 0$. Thus, the convergence of the Shooter Method will depend on the correct initial interval for the adjoint. If the Shooter Method does indeed have the correction initial interval, then the Shooter can approximate the state, adjoint, and control. The Shooter Method terminates when the l^1 –norm of the change in the control from the last control is below a tolerance, δ.

To find the initial interval, two methods were implemented. Mathematically, these intervals have to have certain properties. The first thing the interval needs to satisfy is the Intermediate Value Theorem so that it satisfies the zero method. What is meant by this is that there needs to be an interval that contains a value that, if set to $\lambda(t_0)$, using Runge-Kutta, will produce an adjoint vector such that $\lambda(t_1) = 0$. To find this interval, two different MATLAB functions are used to find this interval two different ways.

The first, which is the `lambda0_finder`, is used when no previous information about the interval is found. The MATLAB functions starts at -100 and counts up until it finds a value that causes Runge-Kutta to produce an adjoint vector whose last value that can be computed successfully. When it finds one, the function then keeps counting up until it finds another value that has the opposite sign. Once it finds this value, it uses a bisection-like process to narrow the interval. This small interval is the initial interval that will be used for the Shooter Method.

The second MATLAB function, which is called `lambda0_finder_adjusted`. This function is used when there is previous information given about the interval, for example the adjoint produced by FBS. This function takes this approximation to the initial value and moves left and right until it finds the desired interval. This interval is then used as the initial interval for the Shooter Method. These two functions were created to help find the interval needed to run the Shooter Method. These methods are used mostly for the initial interval for (P1). For the other two problems, information from FBS is found, then the interval is built around it.

Since the Shooter Method has three different options for finding zeros, a comparison needs to be made among the three of them. The difference can be seen in Table 2. The figure has a few different parameter sets for (P1) as well as (P2) and (P3). For each of the root finding methods used in the Shooter Method, the work to find the initial interval is not accounted for.

Table 2: Comparison of convergence for the root finding methods

Problem	Bisection	Secant	Regula Falsi
(P1) $A = B = C = 5$	28	5	5
(P1) $A = 81, B = 91, C = 13$	32	6	6
(P1) $A = 91, B = 63, C = 10$	32	6	6
(P2)	25	3	3
(P3)	2	2	2

As can be seen by Table 2, generally, the bisection method takes more iterations to converge at the answer while the Secant and Regula Falsi take the same number of iterations. Next the accuracy of the Shooter with the three root finding methods needs to be seen. By using the Shooter Method with the three root finding methods and applying them to (P2), the accuracy of the root finding methods can be seen in Table 3.

Table 3: Accuracy of the root finding methods

	State	Adjoint	Control
Bisection	2.0617×10^{-7}	6.8507×10^{-7}	6.8507×10^{-7}
Secant	1.0847×10^{-11}	1.1756×10^{-12}	1.1756×10^{-12}
Regula Falsi	1.0908×10^{-11}	9.4798×10^{-13}	9.4798×10^{-13}

With the results from Tables 2 and 3, it can be concluded that Regula Falsi is the better root finding method, thus for the comparisons in Chapter 7, it will be used as the root finding method when the Shooter Method is compared to the other methods.

Chapter 6:
Direct Optimization Process

Section 1:
Analytical Process

For this process, no adjoint equation is necessary. Instead, the $J(u)$ functional will be converted into an integral approximation then use an optimization process to solve for the maximizing or minimizing control u by use of the MATLAB Optimization Toolbox (MOT).

The first step is to convert our integral functional, J, from (2.09) into a function that the MOT can work with. Though are many ways of doing just that, the Trapezoid Rule of integration approximation will be the only one we use. The algorithm is not dependent upon this fact and can be adapted easily to incorporate other integration approximations. The Trapezoid Rule is defined in equation (6.01).

Trapezoid Rule (6.01)

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i)\right]$$

$$\text{where } x_i = a + ih$$

Note that in (6.01), f does not have to be a function of a single variable. Here x can represent a single value or a collection of variables. A thing to note, that equation (6.01) is continuous as long as f is continuous. This will play a part when the convergence of the Direct Optimization is discussed in the next section.

Now that the Trapezoid Rule has been defined, the process for solving for the optimal control, u^*, by optimization algorithm can be explained. The algorithm starts by first converting

J into an appropriate function. In doing this, the algorithm creates a function of the vector u so that the MOT finds the minimum. This function proceeds by first computing the state vector using Runge-Kutta given the current u, then it uses the Trapezoid Rule with the state and control in the objective functional to create the final value. The last step is to negate the function. This is because the MOT can only find minimum, and from theory, the maximum of a function is the minimum of the negative of the function.

The next step is to actually use the MOT. The MOT provides functions for finding parameters that minimize objectives while satisfying constraints. The toolbox includes solvers for linear programming, mixed-integer linear programming, quadratic programming, nonlinear optimization, and nonlinear least squares. They can be used to find the optimal solutions to continuous and discrete problems, perform tradeoff analyses, and incorporate optimization methods into algorithms and applications.

The first thing that needs to be set up before optimizing is the options for the MOT. These options determines the type of numerical optimization that will be done. Experimenting with these options would make one of the test problem produce a better result while causing the opposite effect for the other two test problems. Thus when the algorithm was run to test the three problems, all of these are left to default, with the exception of Algorithm, which is set to 'quasi-Newton'. This refers to how it computes the Hessian in the optimization process.

The MOT has many different minimizing methods. The one that was used here is the function `fminunc`. This particular function ends depending certain parameters and reports the result using a certain output, called `exitflag`. This variable indicates why the algorithm terminates. One can find ways to interpret the `exitflag` from the function from MATLAB. In the case for the three test problems, this variable is equal to 1. What this means is that the condition met for the algorithm to terminate and call the value it has the ‘solution’ is when the magnitude of the gradient is small enough.

Section 2: Convergence

This method is going to converge because of the Extreme Value Theorem. As mentioned with the Trapezoid Rule, it can be seen that equation (6.01) is continuous as long as the f function in the objective function J is continuous on the interval $[t_0, t_1]$. When it comes to iteration rates, MOT keeps track of the number of iterations it takes to find a minimum. Each time it finds a value and tests it to be a potential minimum, the MOT counts that as an iteration step. In order to compare it to the other two methods, our implementation of the algorithm keeps track of the number of function evaluations.

Chapter 7:
Processes Applied to Problems

In the first three sections of this chapter, there will be a detailed look at each process applied to the three problems defined in Chapter 3. Once each is broken down and explained, a comparison will be made to see how the methods compare against each other. For all three methods, the number of mesh points is set to one thousand.

Section 1:
Forward Backward Sweep

The discussion will start with the results from applying the FBS to (P1), moving from there to (P2), and then finishing up looking at how the FBS does on (P3). Since that convergence of the FBS has been shown in Section 2 of Chapter 4, a discussion can be made about the iteration rate of the (P1) given the parameters A, B, and C. To do this, MATLAB ran the FBS on the (P1) varying A, B, and C each from 5 to 100 by steps of 5 each time with x_0 fixed at 1. For each step, it saved the parameters and the number of iterations it took FBS to converge. In doing this, MATLAB constructed a 8000×4 matrix. Using this matrix, MATLAB then plotted a three dimensional graph of the parameters and had the color of the corresponding point depend on how many iterations the FBS. The results of this can be found in Figure 2.

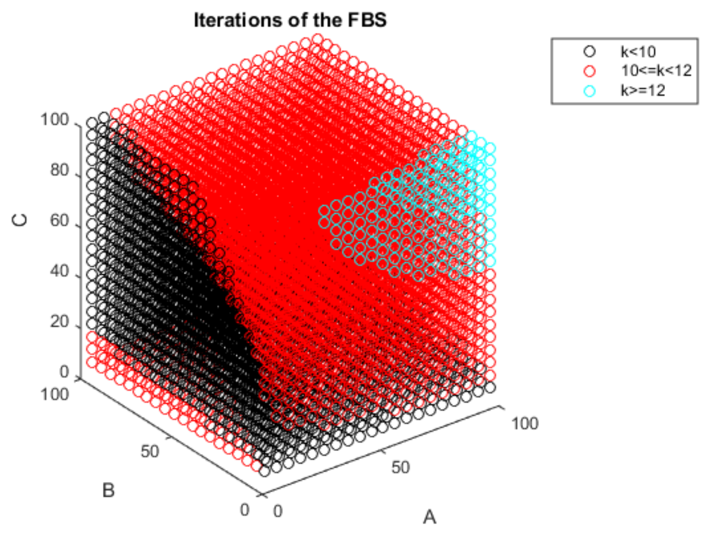

Figure 2: 3-D visualization of how A, B, and C effect the number of iterations

With this graph, three distinct regions are shown. The first region, which is black, shows all the parameters sets that converge in less than ten iterations. The second region are all the parameter sets that converge at least ten but no more than twelve, and this region is plotted with red points. The light blue region is the last region and it represents all the parameter sets that converge with more than twelve iterations. A better look can be seen of these regions by referring to Figures 3, 4, and 5.

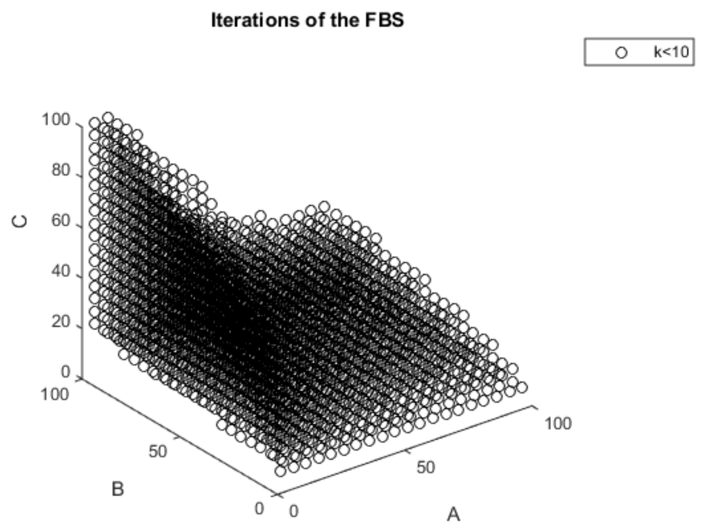

Figure 3: The lower iteration rate region

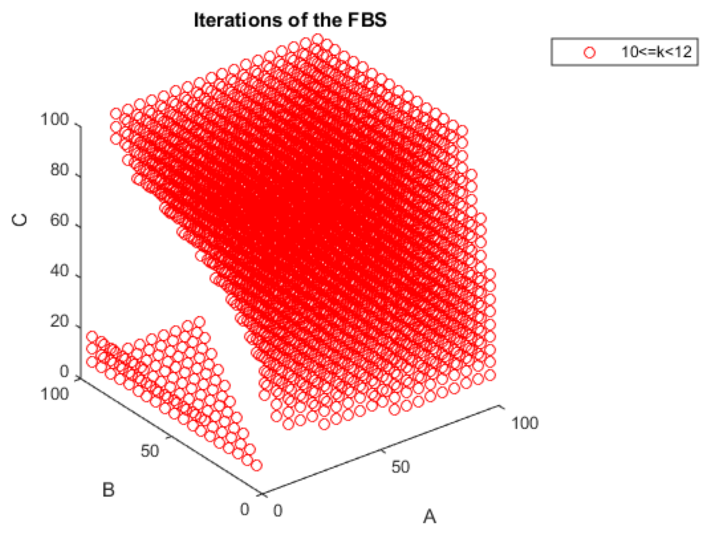

Figure 4: The middle iteration rate region

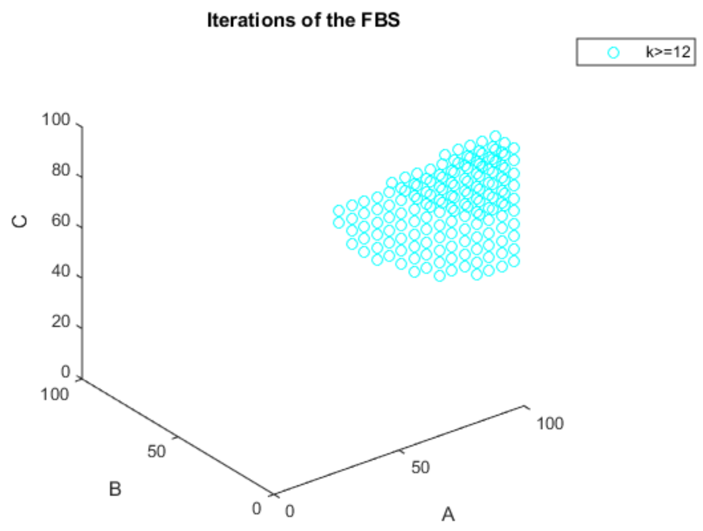

Figure 5: The highest iteration rate region

Now it is easier to see and discuss the three different regions, and a few observations can be made. Notice that as A stays close to zero while B and C grow towards a hundred, the iteration rate is below ten, which is best shown in Figure 3. Also shown in Figure 3, if the parameters are flipped – A goes to 100 and B and C stay near zero – the FBS converges at the same rate. Now if all three parameters grow towards a hundred, the iteration rate is somewhere between ten and twelve and is shown in Figure 4. Lastly if we let A and C grow towards a hundred, but keep B small, the iterations rate is greater than twelve, as shown in Figure 5. To get a better look on how the iterations rate changes as the parameters change, MATLAB took two slices out of the graph in Figure 2. These two slices, shown in Figures 6 and 7, better visualize the dramatic changes in iteration rate as A, B, and C vary.

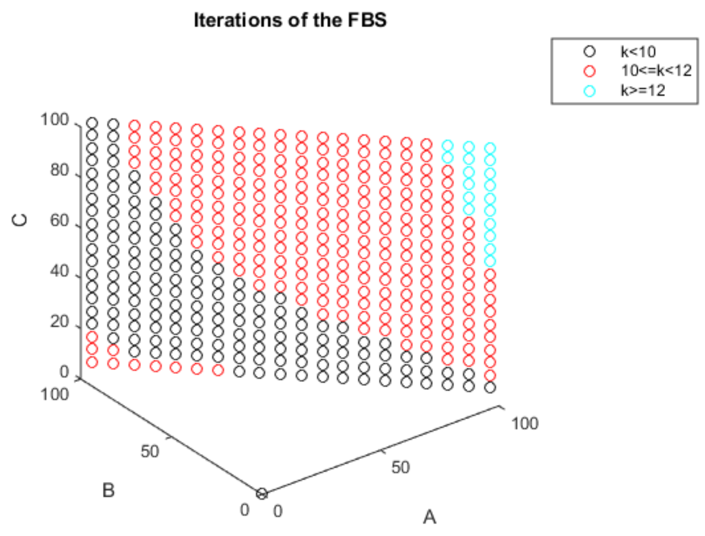

Figure 6: The slice along the back left edge to the front right edge.

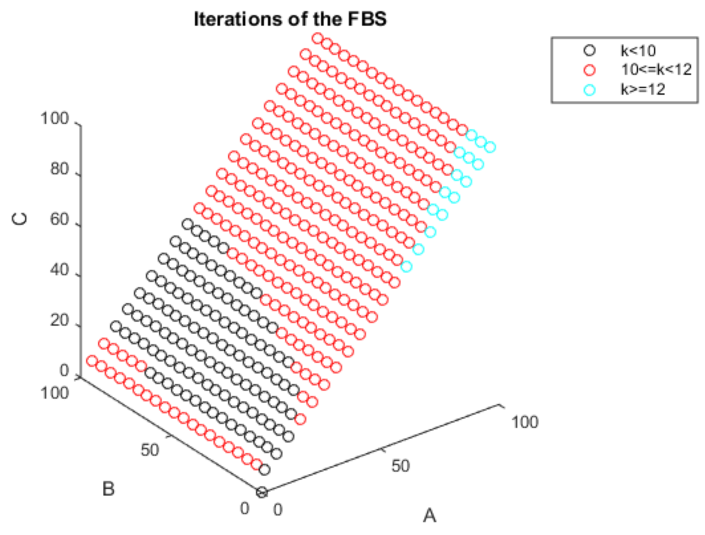

Figure 7: The slice along the bottom left edge to the top right edge.

With these two slices, an interesting result can be observed: That there are two regions of red. This can also be seen by referring to the separated regions in Figure 3 and 4. Notice that along the B axis, as long as A and C stay relatively small, the iteration rate is actually higher than it would be if the parameters move away from the corresponding axes. This can again be seen in Figure 8, by looking at a two dimensional flattening of that data by plotting A/B against C.

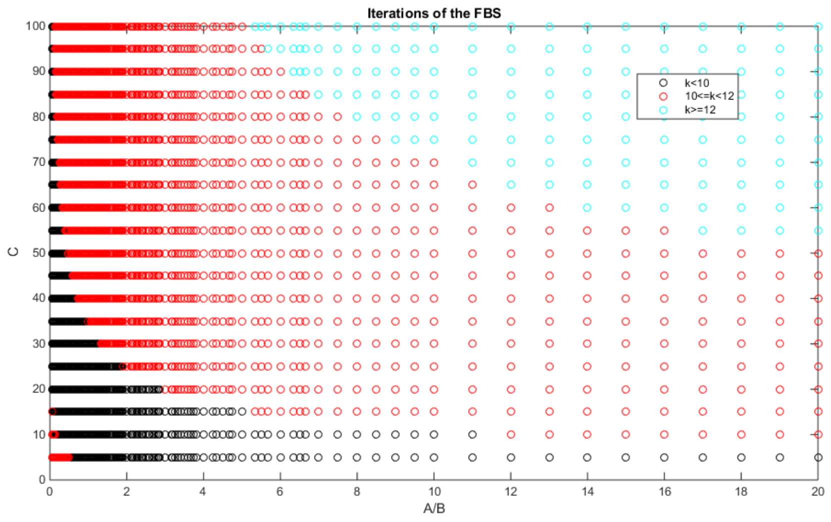

Figure 8: Data represented in two dimensions

For (P2) the analysis of the results is simpler due to the lack of changing parameters. The FBS applied to (P2) converges in nine iterations and produces the graph in Figure 9.

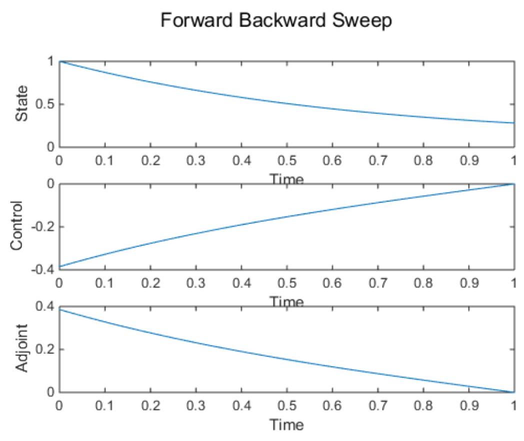

Figure 9: Graphs produced by FBS to (P2)

Now, as discussed in Section 2 of Chapter 3, the real solution is stated in [3], thus a comparison between the actual and the approximation can be made. Using the actual solutions, MATLAB graphed the approximation against the actual and also gave the l^1-norm of the difference in each of the state, adjoint, and control. The graphs can be seen in Figures 10, 11, and 12 while the results from the l^1 norm can be found in Table 4.

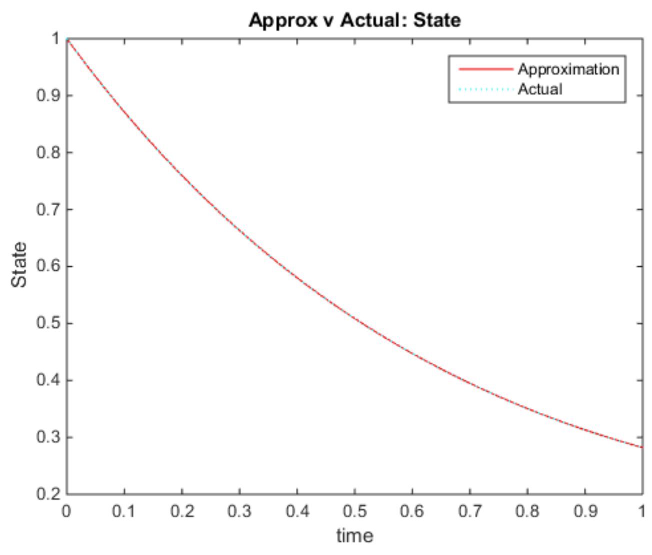

Figure 10: Comparison of the State for FBS

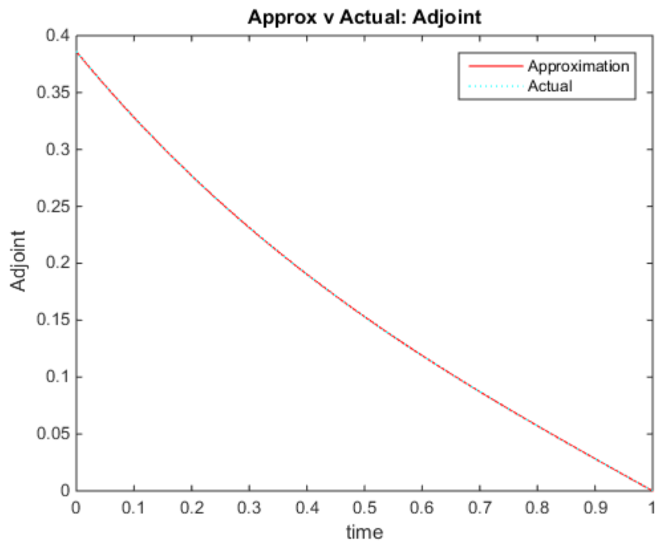

Figure 11: Comparison of the Adjoint for FBS

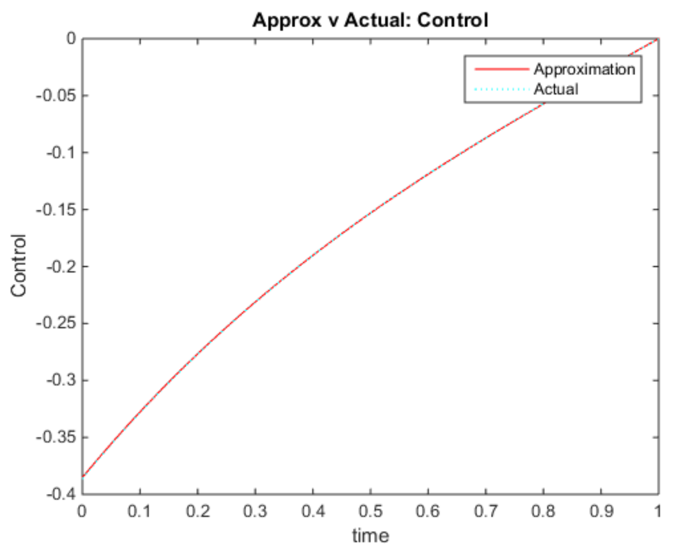

Figure 12: Comparison of Control for FBS

Table 4: l^1-norms of Errors in State, Adjoint, Control for FBS

State	Adjoint	Control
0.1505	0.0558	0.1289

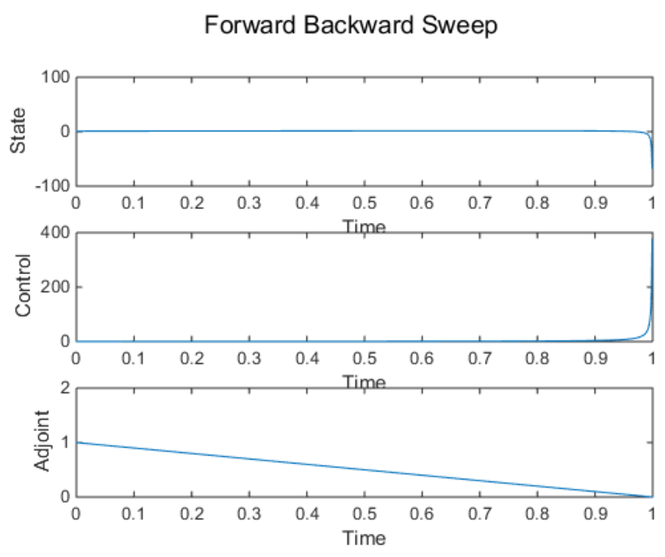

Figure 13: FBS Solution to (P3)

When the FBS is used to solve (P3), it converges in three iterations and the graphs can be seen in Figure 13. From equation (3.17) the control is inversely related by a factor of $\frac{1}{2}$ to the adjoint. Thus as the time progresses towards 1 and the adjoint goes to zero, the control grows exponentially to infinity, as seen by the control graph in Figure 13. This is what was to be expected.

Section 2:
Shooter Method

When the Shooter Method is applied to (P1), an initial interval is needed. Now as discussed in Section 2 of Chapter 5, given the correct initial interval, the method converges. However, there is a major problem: Due to the conditional stability of Runge-Kutta, (P1) with certain parameter choices, no interval can be found. Figure 14 shows what parameters work and which do not.

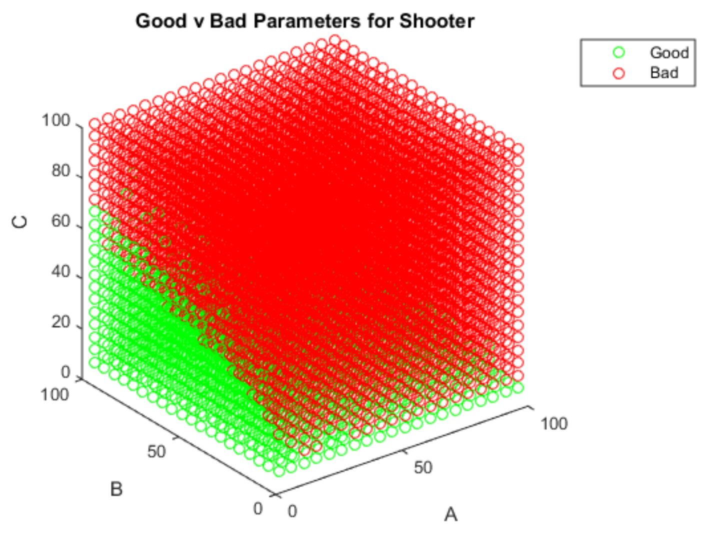

Figure 14: The parameters that work or do not work

This was determined using the MATLAB function `lambda0_finder`. This function tries to find an interval for the Shooter Method with no previous knowledge on where that interval should be. To do this, the function starts at a large negative number and counts forward along the number line. This value is used as the initial guess for the adjoint vector to solve the ODE defined in (4.01). If the ODE gives an adjoint vector where the last value is not a number, then it moves on to a new initial value for the adjoint. However, if it does find a value that works, it will then search for a different initial guess that produces an adjoint whose finial value is of a different sign. Once it has accomplished this, it has found a working initial interval for the Shooter Method.

Using `lambda0_finder`, though it did find values, out of the 8000 tested, it only found 1780 that worked. Thus an adjustment must be made to finding this initial interval. The function `lambda0_findier_adjusted` does this. It uses information on where the interval should be from the results from using FBS and moves away from the correct value. This function takes the first value from the adjoint vector produced from FBS and moves to the left and the right and tests this interval. It then adjusts where needed until the left and right initial adjoint values produce end adjoint values with different signs. Though this method should produce more working results, it still failed more times than not. Referring to Figure 15, this result can be seen.

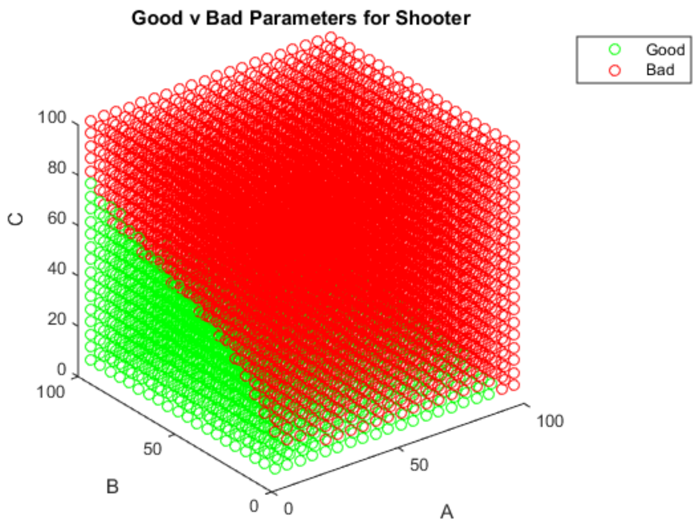

Figure 15: The parameters that work or do not work with previous information

Therefore there is something interesting here. The fact that even with the correct information from FBS, an appropriate interval cannot be found in some cases. This was an interesting problem that if time allowed, would have been delved into more deeply; however, due to time constraints, it had to be looked over for now.

When the Shooter Method is applied to (P2), the process converges in three iterations and the results are given in Figure 16. To get that convergence, the initial interval used is formed from data from the FBS applied to (P2).

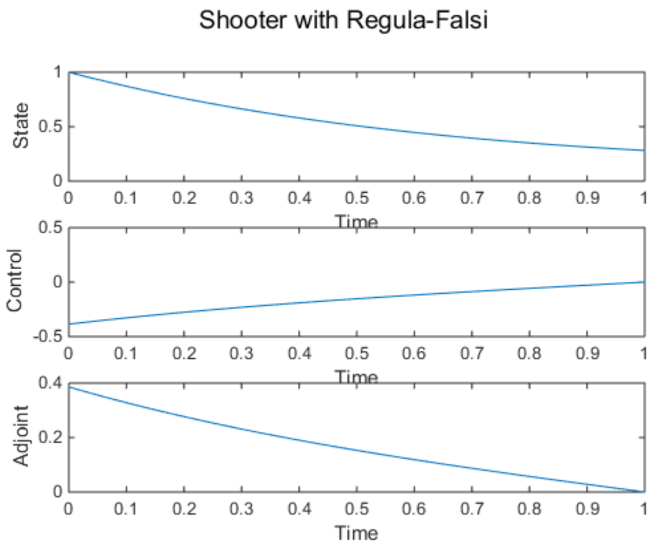

Figure 16: Results of Shooter Method applied to (P2)

Like with (P1), a comparison can be made with the approximation by the Shooter Method to the real solution. In Figure 17, 18, and 19, the state, adjoint, and control produced by both are displayed. Like with FBS, a look at the l^1 norms comparison computed. These values are found in Table 5. This data is just a repeat from the data in Table 3.

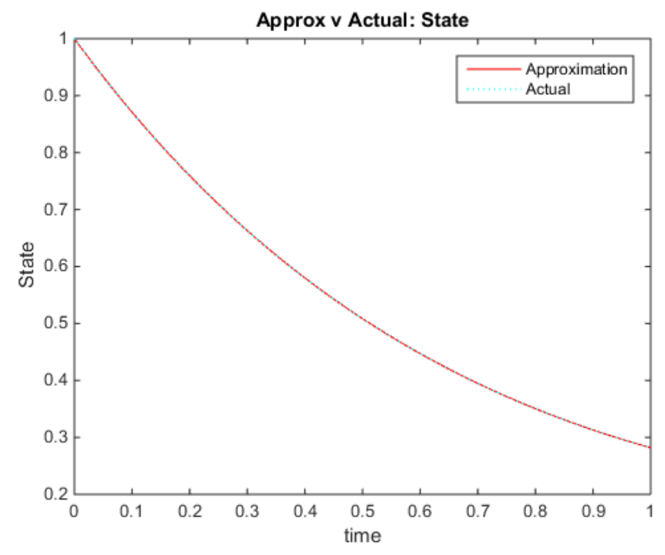

Figure 17: Comparison of State for Shooter

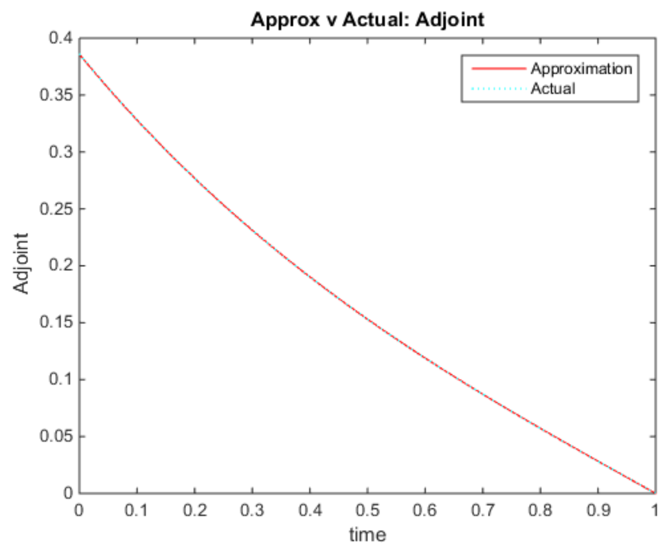

Figure 18: Comparison of Adjoint for Shooter

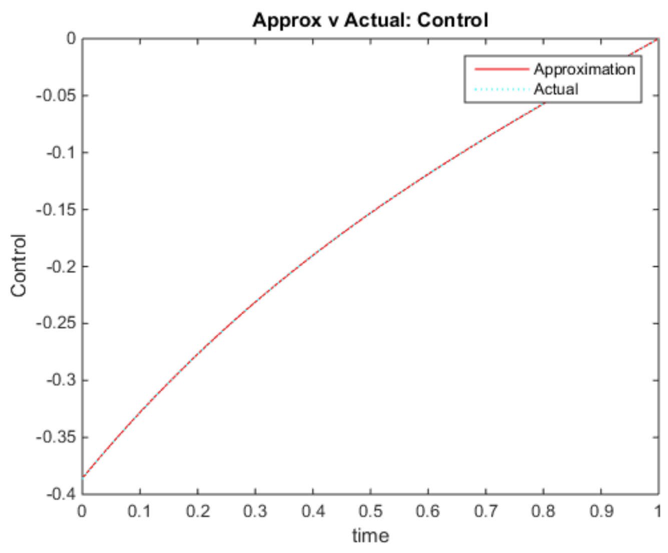

Figure 19: Comparison of Control for Shooter

Table 5: l^1-norms of Errors in State, Adjoint, Control for Shooter

State	Adjoint	Control
1.0908×10^{-11}	9.4798×10^{-13}	9.4798×10^{-13}

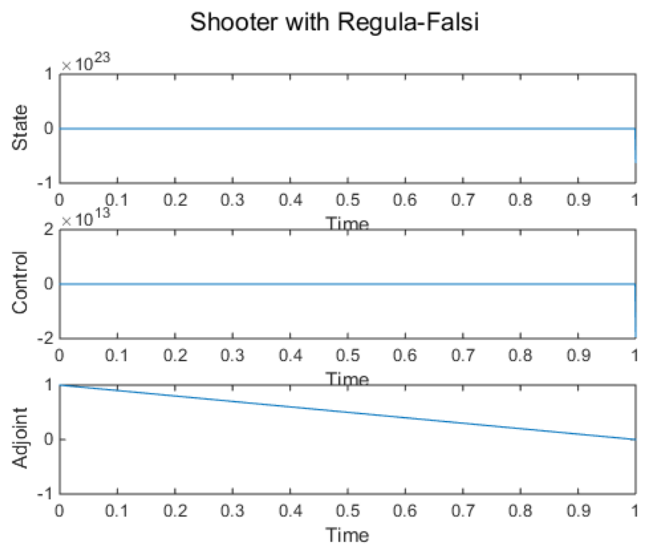

Figure 20: Results to (P3) produced by the Shooter Method

Lastly, looking at the Shooter Method applied to (P3), the results are given in Figure 20. Again, because of (3.17), the control is relatively small, but this time, the Shooter cannot account for the asymptotic behavior of the relationship between the adjoint and control, thus it does not compute a control as accurate as the other two methods.

Section 3:
Optimization

When the Optimization method is applied to the three different test problems, once again note that it does take a while for the algorithm to produce results, but it does indeed produce results. Also since the Optimization Method does not use the adjoint equation, there will be no information about it produced from this method.

First it is applied to Problem 1. Due to the length it time that this algorithm takes to converge, not as many parameter sets were used. Because of the difference in the eight corners from the graph in Figure 2, these were the parameters used when applying the Optimization Method to (P1). The iteration rates and function evaluations can be seen in the Table 6. The reason for reporting both values is to get an understanding of the actual work this method does. The function evaluations are so high is because of all of the work the method does to compute the Hessian matrix for each iteration.

Figure 21 shows the graphs produced by the Optimization method for (P2). The results are then compared to the real solutions to see how accurate the solutions are. This can be seen in the graphs in Figures 22 and 23. Then the l^1 norm between the solutions is given in the table in Table 7. Like with (P1), the MOT thinks it only takes 5 iterations, however it takes 511,520 function evaluations to get there.

Table 6: Iteration results from the Optimization Method

A	B	C	Iteration Rates	Function Evaluations
5	5	5	17	535,568
5	5	100	47	563,614
5	100	5	8	519,536
5	100	100	22	538,574
100	5	5	41	565,628
100	5	100	62	584,666
100	100	5	19	547,592
100	100	100	28	543,584

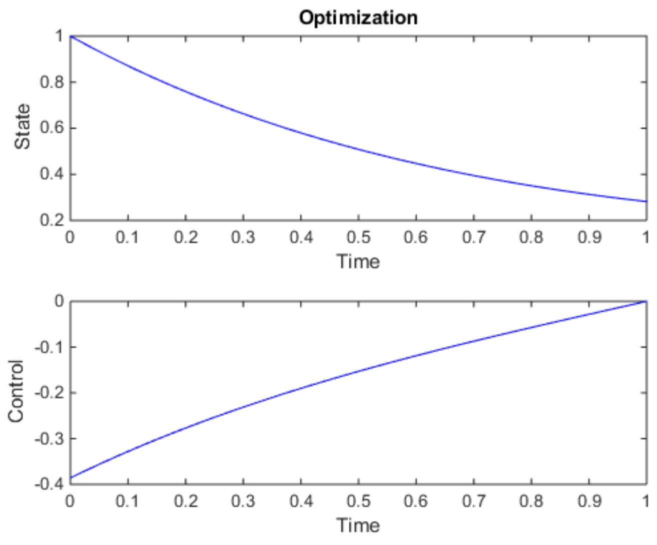

Figure 21: Graphs of (P2) from Optimization Method

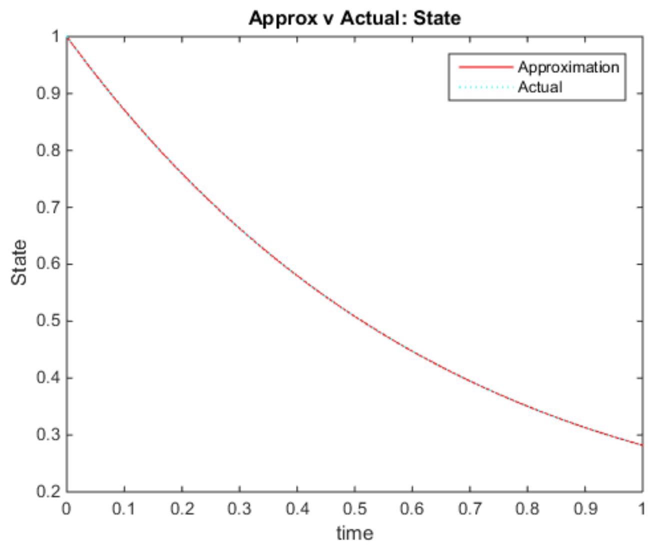

Figure 22: Comparison of State for Optimization

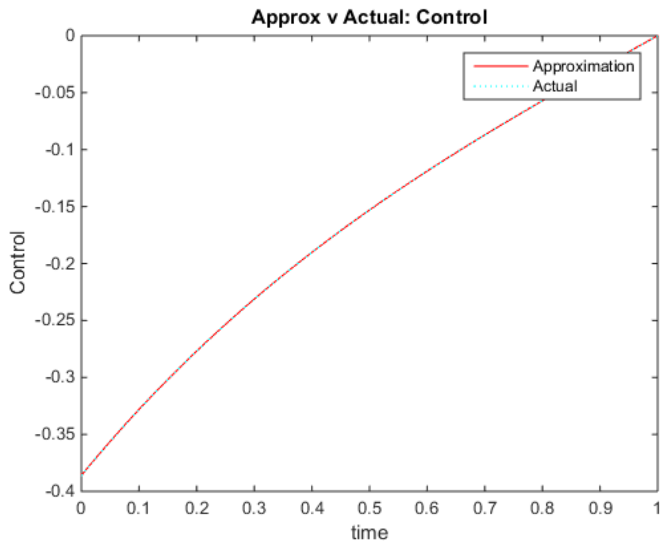

Figure 23: Comparison of Control for Optimization

Table 7: l^1-norms of Errors in the State, and Control for Optimization

State	Control
0.0075	0.0363

Lastly is the results for (P3) using the Optimization Method. The results can be seen in Figure 24. The Optimization Method can register the asymptotic behavior of the control in (P3). This can be seen by the control graph in Figure 24. The MOT takes 96 iterations with 603,704 function evaluations.

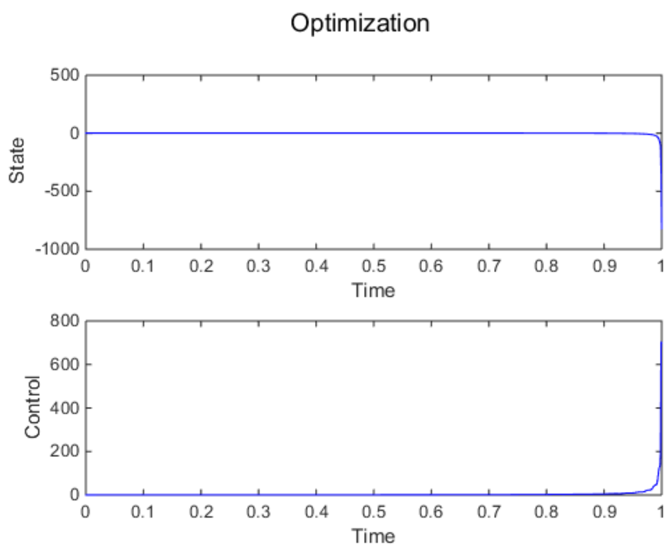

Figure 24: Optimization results for (P3)

Section 4:
Comparison of Results

For this section, a direct comparison will be made among the three methods applied to the three problems. Some things to note before looking at Table 8: The first is about the problems. So that (P1) is properly represented, different parameters will be chosen to compare it to the other problems, but in all cases x_0 will be set to one. When it comes to the columns of Table 8, the number of iterations for the Shooter Method are when root finding method Regula Falsi is used with the interval using prior information. For the Optimization, the column is split with the number of iteration and the number of function evaluations to get a better comparison with the other two methods.

Table 8: Comparison of methods table

Problem	FBS	Shooter	Optimization
(P1) with $A = B = C = 5$	9	5	17 / 535,568
(P1) with $A = B = 5, C = 100$	11	N/A	47 / 563,614
(P1) with $A = C = 5, B = 100$	11	4	8 / 519,536
(P1) with $C = B = 5, A = 100$	9	N/A	22 / 538,574
(P1) with $A = 5, B = C = 100$	9	N/A	41 / 565,628
(P1) with $B = 5, A = C = 100$	12	N/A	62 / 584,666
(P1) with $C = 5, A = B = 100$	9	5	19 / 547,592
(P1) with $A = B = C = 100$	11	N/A	28 / 543,584
(P2)	9	3	5 / 511,520
(P3)	3	3	96 / 603,704

After studying Table 8, one can see that when the Shooter Method works, it is the quickest method, but the problem with it is finding that initial interval, hence why there are so many N/A's in the Shooter Method column. So because of the lack of results, we can only compare FBS and Direct Optimization, and FBS has the better iteration rate.

When it comes to accuracy, a reference needs to be made back to Tables 4, 5, and 7 to see the results from (P2) compared to the actual solution. Note that though the FBS might converge faster, it is actually the least accurate. The results show that the Shooter Method is the most accurate with the Direct Optimization method being in the middle.

Based on these results, though it is the least accurate, FBS is the most reliable method, therefore it is the 'winner'. It is the method that will work in most cases and produces an answer that is semi-accurate.

Chapter 8:
Conclusion

All in all, the process of implementing the three methods to solve an optimal control problem has been completed and those methods tested against the three test problems. As was seen in the previous chapter, a 'winner' was determined from comparing results in Table 8. Now the Forward Backward Sweep might be the 'winner' based on the work here, but there are potentially other ways to solve an optimal control problem numerically and other optimal control problems to test. Therefore, this work done here can be expanded on and updated depending on new methods and problems added into the competition.

Also if a more reliable and efficient way to find the initial interval is found for the Shooter Method, then it would become the method to beat out of the three presented here. Overall, the closing remark is this: This process is never done and there is still work to be found, and it will be interesting to see if others pick up from here and continue to find better methods to solve optimal control problems. But for now, the purpose of this thesis has been completed: Three methods found, implemented and then compared using the test problems.

Though the desired work here is done, there is a lot of potential future work to be done based on what was done here. One can work with making the FBS more accurate, finding more efficient ways to determine the initial interval for the Shooter Method, find better root finding methods for the Shooter Method to implement, or making the Optimization method more efficient. These are just a few examples of what can be done, but there are many ways to expand or refine all the work that has been done.

List of References

[1] S. Lenhart and J. T. Workman, Optimal Control Applied to Biological Models, Boca Raton, FL, Florida: Taylor & Francis Group, LLC, 2007.

[2] E. Süli and D. Mayers, An Introduction to Numerical Analysis, New York, NY: Cambridge University Press, 2006.

[3] M. McAsey, L. Mou and W. Han, "Convergence of the forward-backward sweep method for optimal control," pp. 207-227, 2012.

[4] J. Stoer and R. Bulirsch, Introduction to Numerical Analysis, New York, NY: Springer-Verlag, 1972.

[5] J. Stewart, Calculus: Early Transcendentals, Belmont, CA: Brooks/Cole, 2008.

[6] P. G. Ciarlet, Introduction to Numerical Linear Algebra and Optimization, New York, NY: Press Syndicate of the University of Cambridge, 1989.

[7] J. Nocedal and S. J. Write, Numerical Optimization, New York, NY: Springer Science+Buisness Media, Inc., 1999.

[8] W. Hackbush, "A numerical method for solving parabolic equations with opposite orientations," *Computing*, pp. 229-40, 1978.

Appendices

Appendix I: Glossary

adjoint function	The function designed by Pontryagin in the 1950's used to help solve optimal control problems
concavity	When the twice differential function's second derivative is negative
control or control function	An operation that controls the recording or processing or transmission of interpretation of data
convergence	When an iterative algorithm has a step that a stopping condition
existence conditions	When a condition or conditions are met that guarantee there is a solution to the problem
extrema	A maximum or minimum of a function
flops	floating point operations
global extrema	A maximum or minimum of the entire function. There is no point larger or smaller
initial condition	Any of a set of starting-point values belonging to or imposed upon the variables in an equation that has one or more arbitrary constants
Lagrange Multiplier	A strategy for finding the local maxima and minima of a function subject to equality constraints.
Lipschitz	A strong form of uniform continuity for functions
local extrema	A maximum or minimum of region of a function. There could be a point that is larger or smaller somewhere else in the function
necessary condition	When something is true, certain conditions must hold
numerical processes	A process used to approximate the solution of a mathematical problem

objective function	The function to be maximized or minimized
optimal control problem	The process of determining control and state trajectories for a dynamic system over a period of time to minimize a performance index
optimality system	The state equation and adjoint equations together with the characterization and boundary conditions
optimization	The process in which the best feasible solution for a problem. This usually in tells finding either a maximum or minimum of the possible solutions
root finding method	A numerical method to find where the function has a zero, i.e. $f(x) = 0$.
state variable (or function)	The set of variables (functions) that are used to describe the mathematical state of the system
terminal condition	Any of a set of ending-point values belonging to or imposed upon the variables in an equation that has one or more arbitrary constants
tolerance	The value used to determine convergence
uniqueness condition	When a condition or conditions are met that guarantee there is a only one solution to the problem

Appendix II:
Code Explanation

All MATLAB files can be found on the internet at https://sites.google.com/site/grmsthesis/home

Scripts

Script Name	**Script Explanation**
`clean`	This script has a very simplistic job: Clear our past values, the workspace, and close all past graphs.
`FBS_Interface`	The interface for the parameters of (P1)
`Interface`	Runs the user friends GUI
`Problem_1`	This script sets up the all the appropriate functions and values according to (P1)
`Problem_2`	This script sets up the all the appropriate functions and values according to (P2)
`Problem_3`	This script sets up the all the appropriate functions and values according to (P3)
`Problem_Create`	This script is used in tandem with the `Solve_Optimal_Control` script to help the use create a new problem file to solve
`Solve_Optimal_Control`	This script opens up the GUI used to make using all the functions user friendly

Variable Key

Variable	Explanation
`adjoint`	The imputable adjoint function
`control`	The imputable control function
`cr`	The collection of the control vectors created by `Control_Solver`
`err_control`	The error in the two given control vectors.
`err_state`	The error in the two given state vectors
`f`	The imputable objective function
`FBS`	Turns the Forward Back Sweep on in `Control_Solver`
`fbs_initial`	Data from the FBS to find the initial data for the Shooter
`Graph_switch`	Tells `Control_Solver` to graph the compared methods errors on
`h`	The step size of the mesh
`h2`	Half the step size of the mesh
`k`	The number of iterations the method takes
`lambda`	Adjoint vector
`lambda_data`	The initial data needed for the root finding method
`lr`	The collection of the adjoint vectors created by `Control_Solver`
`N`	Number of mesh points
`ode`	ODE to be solved.
`OPT`	Turns The Optimization method on in `Control_Solver`
`PM`	Tells `Control_Solver` which problem script to run

`positions`	This is a 1×3 vector containing the input values to update with respect to the results of `testers`
`SB`	Turns the Shooter with Bisection on in `Control_Solver`
`sr`	The collection of the state vectors created by `Control_Solver`
`SRF`	Turns the Shooter with Regula Falsi on in `Control_Solver`
`SS`	Turns the Shooter with Secant on in `Control_Solver`
`state`	The imputable state function
`state_adjoint`	The imputable vector function where the first function is the state ODE to solve and the second is the adjoint ODE
`t`	Time vector
`testers`	This is a 1×3 vector containing the output values to test if the middle value is the zero between the left and right values
`u`	Control vector
`u_func`	Determines how the FBS will update the control each iteration
`x`	State vector
`x0`	The initial value for the state ODE
`zero_choice`	Tells the Shooter what root finding method to use.

Functions

Function Name	Input	Output	Explanation

`backward_runge_kutta_4`	`t`
	`x`
	`lambda`
	`u`
	`N`
	`ode`
		`lambda`	Solve adjoint in FBS

`bisection`	`testers`
	`positions`
		`left`
		`right`
		`lambda0`
			The Bisection method of finding a root

`Control_Solver`	`PM`
	`FBS`
	`SB`
	`SS`
	`SRF`
	`OPT`
	`graph_switch`
	`N`
		`sr`
		`lr`
		`cr`
			Used with the GUI to run the users desired process on the problems chosen.

`error_calculator`	`t`
	`x1`
	`u1`
	`x2`
	`u2`
	`graph_switch`
		`err_state`
		`err_control`
			Computes the error in the two given state and control vectors and possibly display this in a graph.

`Forward_Backward_Sweep`	`x0`
	`t`
	`h`
	`h2`
	`u_func`
	`state`
	`adjoint`
	`control`
	`N`
		`x`
		`lambda`
		`u`
		`k`
			The Forward Backward Sweep to solve the control problem

`J_function`	`t`
	`x`
	`u`
	`f`
	`state`
		`J`	Used to approximate the integral in the objective function using the trapezoid rule

`lambda0_finder`	`x0`
	`control`
	`state_adjoint`
	`t`
		`lambda_data`
		`k`
			Used to find the initial information needed for the Shooter Method with no previous knowledge of it

`lambda0_finder_adjusted`	`x0`
	`control`
	`state_adjoint`
	`t`
	`fbs_initial`
		`lambda_data`
		`k`
			Used to find the initial information needed for the Shooter Method using information from the FBS

`midpoint`	`point_1` `point_2`	`m`	Find the midpoint between two n-dimensional points

`Optimization`	`t`
	`h`
	`N`
	`x0`
	`f`
	`state`
		`u`
		`x`
		`k`
			The Optimization method for solving the control problem

`regula_false`	`testers`
	`positions`
		`left`
		`right`
		`lambda0`
			The Regula Falsi Method of finding a root

`runge_kutta_4`	`t`
	`x`
	`u`
	`N`
	`ode`
		`x`	The Explicit RK4 Method

`runge_kutta_4_shooter`	`t`
	`x`
	`lambda`
	`N`
	`state_adjoint`
	`control`
		`x`
		`lambda`
			Solves the state and adjoint ODE's simultaneously

`secant`	`testers`
	`positions`
		`left`
		`right`
		`lambda0`
			The Secant method of finding a root

`shooter`	`x0`
	`lambda_data`
	`t`
	`h`
	`h2`
	`N`
	`state_adjoint`
	`control`
	`zero_choice`
		`x`
		`lambda`
		`u`
		`k`
			The Shooter Method to solve the Control given control problem

Vita

Garrett Robert Rose was born in Fort Pierce, FL, to Todd and Peggy Rose. He is the eldest of two sons, the youngest being Jordan Rose. In 1994, his family moved to Athens, TN. He attended City Park Elementary School for grades Kindergarten through third grade. From there, he moved to Westside Elementary and completed up to sixth grade. After Westside, he attended Athens Junior High School for grades seventh, eighth, and ninth. Lastly, he attended McMinn County High School and graduated in 2008.

From high school, he attended the local private Methodist college, Tennessee Wesleyan College to peruse a degree in Mathematics. He graduated in 2012 with a Bachelor of Science degree majoring in Mathematics with a minor in Music. The following fall, he moved to Knoxville, TN to attend the University of Tennessee in Knoxville. He was accepted there in the Master's Program and is conferred to graduate in 2015 with a Master of Science degree in Mathematics.