Department of Mathematics and Computer Science

Heidelberg University

Master thesis

in Scientific Computing

submitted by

Yu Xiang

born in November 10, 1989

2022

Numerical methods for optimal control problems, with a case study in state constrained rocket car

This master thesis has been carried out by Yu Xiang

at the

Heidelberg University

under the supervision of

Professor Dr. Ekaterina A. Kostina

Contents

1 Introduction

Many real-life problems can be modeled as an optimal control problem (OCP), for example, launching a rocket to the moon with minimum fuel expenditure as the objective, or maximizing the profit from the factory production, with constraints in resources available and uncertain market demand. This paper focus on solving optimal control problems with numerical approaches, particularly with multiple shooting and (quasi) Newton style method.

In general, optimal control deals with the problem of finding the control over the state for a dynamic system over a period of time such that an objective function is optimized. Generally, an optimal control problem can be formulized as follows:

$$\begin{aligned} \min_{x(t),u(t)} \quad & F(x(t),u(t)) \\ s.t. \quad & \dot{x}(t) = f(x(t),u(t)) \\ & x(t) \in \Omega \\ & u(t) \in \mathbb{U} \\ & t \in [t_0, t_f] \end{aligned} \tag{1.1}$$

Here t is the independent variable (generally speaking, time), usually using t_0 and t_f representing the initial and terminal time respectively. $x(t)$ is the state variables, and $u(t)$ is the control variables, $F(\cdot)$ is the objective function, also called the cost function. $\dot{x}(t) = f(x(t), u(t))$ represents the underlying dynamic system, $x(t) \in \Omega$ and $u(t) \in \mathbb{U}$ represents the constraints for which the state variables $x(t)$ and control variable $u(t)$ must satisfy respectively. Sometimes, the constraints are expressed as functions of $x(t)$ and $u(t)$ together.

Generally speaking, there are three basis approaches to address optimal control problems (a) dynamic programming (b) indirect, and (c) direct approaches. (ref Moritz Diehl [2005]). This paper on hand, focus on the direct approaches, which are one of the most widespread and successfully used techniques. Direct approaches transform the original infinite optimal control problem into a finite dimensional nonlinear programming problem (NLP). This NLP is then solved by variants of state-of-the-art numerical optimization methods, and the approach is therefore often sketched as "first discretize, then optimize."

Mutilple shooting method can be used in the "first discretize" part of the direct approaches. The main idea is to divide the whole interval $t \in [t_0, t_f]$ into multiple subintervals, and introduce initial guess for each subinterval, solve the problem in each subinterval with the initial guess, and impose additional matching conditions at the boundary of each subinterval to form a solution of the whole interval. In each subinterval, with mutiple shooting methods applied, the system of differential equations will be turned into a system of ordinary non-linear algebraic equations. Therefore, the original OCP is transferred into piecewise nonlinear programming problem (NLP). We can use the Karush–Kuhn–Tucker (KKT) approach to combine the constraints and the original objective function into a new Lagrange function.

Minimizing this new Lagrangian function becomes an optimization problem without any constraints, and its optimum value can be determined via derivative methods. To

solve any unconstrained NLP, we can work within an iterative sequential quadratic programming (SQP), or Newton style framework. Newton style methods are second-order derivatives iterative optimization algorthm and generally, but not always, converges faster than first-order derivatives method (e.g. gradient descent). The Newton method needs to calculate the second order derivatives, i.e. the Hessian matrix and its inverse in each iteration, which is very expensive to compute. Quasi Netwon methods employs an approximation to the original Hessian matrix and takes an efficient way to update the approximation matrix. Therefore, quasi Newton method is generally faster than Newton method.

Besides the control variables u and state variables x, some optimal control problems may have uncertain parameters whose value are priori unknown, and the optimal objective value depends on the parameter value. This kind of problem is called the parameterized optimal control problems and is of the form

$$\begin{aligned} \min_{x(t),u(t)} \quad & F(x(t;p),u(t)) \\ s.t. \quad & \dot{x}(t;p) = f(x(t;p),u(t)) \\ & x(t;p) \in \Omega \\ & u(t) \in \mathbb{U} \\ & p \in \mathbb{P} \\ & x = x(;,p^\star) \; if \; p = p^\star \\ & t \in [t_0, t_f] \end{aligned} \tag{1.2}$$

where $p^\star$ is a fixed value in the feasible uncertainty set $\mathbb{P}$, where the parameter p can take value from. Equation 1.2 derives from the equation 1.1, with the paramter p added.

Parameterized optimal control problems are very difficult to solve due to the uncertainty in the parameter p. Since the paramter p can take different values, so does the corresponding objective function $F(\cdot)$. Then, the solution of 1.2, i.e. $min \; F(\cdot;p)$ can be regarded as a function of parameter p. One p value corresponds to one solution, if such solution can be found. Since p is priori unknown, in real life problems, it usually makes sense to solve the parameterized optimal control problems in a conservative way, so that a robust solution can be obtained. For example, in the paper Schlöder [2022], when modelling the therapy design of Cerebral Palsy (CP) for the patient, a conservative solution is a desriable result. Two different ways of solving the parameterized optimal control problem will be discussed in details, i.e. the classical approach and the training approach. Both are in the form of a bilevel optimization roblem, i.e. an optimization problem in which another optimization problem enters the constraints. Details about these two approaches will be discussed in Chapter 3. The approaches discussed above will be demonstrated with a case study in state constrained rocket car, with the description of the case, and its numerical result given in Chapter 4.

The structure of this paper is as follows. Following this introduction Chapter 1, in next Chapter 2, we focus on expaining in details how to solve optimial control problems with direct approaches using mutiple shooting and quasi Newton method. In Chapter 3, we discuss the approaches for solving parameterized optimal control problem, i.e. the classic approach and the training approach. In Chapter 4, we give the description of our case study, i.e. the state constrained rocket car case, and compare the numerical solutions of the classical approach and training approach. In the final Chapter 5, we conclude the analysis with the numerical results.

2 Numerical methods

Optimal control theory deals with systems that can be controlled, i.e whose evolution can be influenced by some external agent. In this paper we only consider the case that the control variable u is a function of only time t, and not function of the state variable x. This type of problem is known as open loop, or controllability problem. The system dynamics of the optimal control problem can be generalized as a system of differential equations

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1}$$

We would like to have such dynamic system to run in an optimal way, subject to the constriants that are applicable in real life. This indicates that the problem will have a clearly defined objective function, with the dynamic system and the constraints expressed in explicit formula. With the example of launching the rocket to the moon, the objective function can be mimizing the fuel used or minimize the time horizon of the system, subject to the contraints, e.g. gravity, fuel efficiency, speed limitation etc. The trajactories (dynamic system), as well as some of the contraints, are expressed in differential equations.

In Chapter 1, equation 1.1 gives a high-level formulation of an optimal control problem. Here we augument equation 1.1 with mathematical details, with the objective function and the constraints expressed in explicit formulas. For real-life problems, the optimal control formulation can typically be generalized in the following form

$$\min_{x(t),u(t)} \; F(x(t), u(t)) = \int_{t_0}^{t_f} L(x(t), u(t))dt + E(x(t_f)) \tag{2.2a}$$

$$s.t. \;\; \dot{x}(t) = f(x(t), u(t)), \quad (system\ dynamics) \tag{2.2b}$$

$$g(x(t), u(t)) = 0, \quad (path\ equality\ constraints) \tag{2.2c}$$

$$h(x(t), u(t)) \leq 0, \quad (path\ inequality\ constraints) \tag{2.2d}$$

$$x(t_0) = x_0, \quad (initial\ value) \tag{2.2e}$$

$$r(x(t_f)) \leq 0, \quad (terminal\ constraints) \tag{2.2f}$$

$$x^{lower} \leq x(t) \leq x^{upper} \tag{2.2g}$$

$$u^{lower} \leq u(t) \leq u^{upper} \tag{2.2h}$$

$$t \in [t_0, t_f] \tag{2.2i}$$

Here $L(\cdot)$ and $E(\cdot)$ are called the running cost and end cost, with their sum $F(\cdot)$ the cost/objective function. $g(\cdot)$, $h(\cdot)$ and $r(\cdot)$ are functions representing equaltity constraints, inequality constraints, and terminal constraints respectively. The time horizon is $[t_0, t_f]$, with the initial value known as x_0.

Depending on nature of the underlying optimal control problems, the mathematical expression can take a modifed form of equation 2.2. For certain problems, some of the constraints defined in equation 2.2 may not play a role, while for other problems,

addtional constraints may be needed or existing constraints need to be modified. Nevertheless, Equation 2.2 gives us a general mathematical formulation of the typical optimal control problems in real life, and we do not go further discussion with possible (minor) modification to Equation 2.2.

Various methods exist for solving optimal control problems. The paper Moritz Diehl [2005] summarizes three general approaches to address optimal control problems: (a) dynamic programming, (b) indirect, and (c) direct approaches. Dynamic programming uses the principle of optimality to compute recursively a feedback control for all time t and all x_0. Indirect methods use the necessary conditions of optimality of the infinite problem to derive a boundary value problem (BVP) in ordinary differential equations.

This paper on hand, focus on the direct approaches, which are one of the most widespread and successfully used techniques. Direct approaches transform the original infinite optimal control problem into a finite dimensional nonlinear programming problem (NLP). This NLP is then solved by variants of state-of-the-art numerical optimization methods, and this approach is therefore often sketched as "first discretize, then optimize." One of the most important advantages of direct approaches is that they can easily treat inequality constraints, e.g. the inequality path constraints as in equation 2.2d. This is because structural changes in the active constraints during the optimization procedure are treated by well developed NLP methods, which can deal with inequality constraints and active set changes. (ref Moritz Diehl [2005]).

In the text that follows, we first explain the mutilple shooting method, which can be used in the "first discretize" part of the direct approaches. Afterwards, we explain the KKT conditions, which can easily treat equality and inequality constraints of the optimal control problems. With the background knowledge of mutiple shooting methods and KKT conditions, we continue to explain the Newton-style method and sequential quadratic programming. These methods, together, can serve as the "then optimize" part of the direct approaches, and solve the optimal control problems of the form as in equation 1.1, or in more mathematical details as in equation 2.2.

2.1 Multiple shooting

Multiple shooting method was initially introduced to solve boundary value problem (BVP) in differential equation scope David Morrison [1962]. This method, with some adjustment, therefore, is well suited for solving optimal control problem with constraints in differential equations. In the text that follows, we first explain how the multiple shooting can be used for solving BVP in the differential equation scope. After that, we explain how mutilple shooting can be applied to a general optimal control problem, and particularly how it can be used for solving optimal control problem of the form as in equation 2.2.

To illustrate the concept of shooting method to solve boundary value problem (BVP), we use the following example.

$$\dot{x} = x(t), t_0 \leq t \leq t_f$$

The analytical solution of above equation is

$$x(t) = x(t_0)e^{t-t_0}$$

where e is the exponential number. Then $x(t_0) = x_0$ will be determined such that it will satisfy $x(t_f) = b$ for a given value b. Therefore, the equation $x(t_f) - b = 0$ or

$x_0 e^{t-t_0} - b = 0$ is obtained. This derivation is called as shooting method. Generally, the shooting method can be summarized as follow

shooting method

- Step 1, choose an initial value $x_0 = x(t_0)$
- Step 2, form a solution of the differential equation from t_0 to t_f
- Step 3, evaluate the error at the boundary, if $x(t_f) - b = 0$, then stop, otherwise continue to Step 4
- Step 4, update the guess for x_0 based on some updating schema, go to Step 2

In multiple shooting method, the "shoot" interval is partitioned into some short subintervals. We define a general differential equation with boundary value of the following form

$$\begin{aligned} &\dot{y} = f(t, y) \\ &y(t_f) = y_f \end{aligned} \tag{2.3}$$

where y denotes the differential variables, $t \in [t_0, t_f]$, y_f is the boundary value at t_f. With multiple shooting method, one chooses a suitable grid of multiple shooting nodes $\tau_j \in [t_0, t_f]$, where $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$, i.e. m subintervals covering the whole interval. At the beginning of each subinterval, $[\tau_k, \tau_{k+1}], k = 0, 1, ..., m-1$, we have the initial guess of the starting value $\hat{y}_k$. Then in each subinterval, we have the initial value problem of the following form:

$$\begin{aligned} &\dot{y} = f(t, y, p), t \in [\tau_k, \tau_{k+1}],\ k = 0, 1, ..., m-1 \\ &y(\tau_k) = \hat{y}_k, k = 0, 1, ..., m-1 \end{aligned} \tag{2.4}$$

In each subinterval, we introduce the new unkown parameter $\hat{y}_k$, we solve an initial value problem and will get a solution $y(t), t \in [\tau_k, \tau_{k+1}]$. The piecewise solution is not necessary continuous and also not necessarily satisfy the boundary condition $y(t_f) = y_f$. The continuity has to be enforced by additional matching conditions at each subinterval boundary, i.e.

$$\begin{aligned} &y(\tau_{k+1}; \hat{y}_k) = \hat{y}_{k+1},\ k = 0, 1, ..., m-1 \\ &\hat{y}_m (i.e.\ \hat{y}_{\tau_m} = \hat{y}_{t_f}) = y_f \end{aligned} \tag{2.5}$$

The procedure of multiple shooting method can then be summarized as

Mutilple shooting method

- Step 1, choose multiple shooting nodes $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$
- Step 2, choose initial guess for $\hat{y}_k, k = 0, 1, ..., m-1$
- Step 3, form solutions of the differential equation in each subinterval $[\tau_k, \tau_{k+1}], k = 0, 1, ..., m-1$
- Step 4, evaluate the error at the boundary of each subinterval. If $y(\tau_{k+1}; \hat{y}_k) - \hat{y}_{k+1} = 0,\ k = 0, 1, ..., m-1$ and $\hat{y}_m - y_f = 0$, then stop, otherwise continue to Step 5
- Step 5, update the guess for $\hat{y}_k, k = 0, 1, ..., m-1$ based on some updating schema, go to Step 3

In practice, there are many details to be decided when using (multiple) shooting methods, which we will discuss briefly without giving a comprehensive description. In Step 1 & 2, when choosing the shooting nodes and the initial guess $\hat{y}_k$, how they are chosen usually depends on nature of the problem as well as a balance between accuracy and computational cost. For example, the nodes can be equally spaced and the $\hat{y}_k$ can be initialized with the same value, or they can be addressed based on our initial knowledge of the problem. In Step 3, polynomial functions can be used as the approximate solutions, leveraging the fact that Taylor expansion can be used to approximate any continuous functions. In Step 4, "evaluate the error" is usually in the form of evaluating an objective function, which, e.g. can be defined as the sum of quadratic errors. In Step 5, the "updating schema" can be defined to so that the $\hat{y}_k$ can move in a direction that decreases the objective function. The (quasi) Newton method, for example, can be used as an updating method in Step 5.

The multiple shooting method can be used for many problems. For example, it can be also applied to the twice differential system of the following form

$$y''(t) = f(t, y(t), y'(t)) \quad y(t_0) = y_0, \quad y(t_f) = y_f, \quad t \in [t_0, t_f] \tag{2.6}$$

The problem 2.6 is similar to the problem 2.3. In each subinterval, a boundary value problem (BVP) is to be solved and matching conditions at the boundary of each subinterval are to be enforced so that the final solution is continuous and applicable to the whole interval.

2.1.1 Mutiple shooting method for OCP

Mutiple shooting method is well suited for solving optimal control problems of the form in equation 2.2, as it can serve as the "first discretize" part of the direct approaches. For problems in the form of equation 2.2, we can discretize the time $t \in [t_0, t_f]$ and the original problem is split into multiple subintervals, with the constraints discretized and applied to each subinterval. We also need to introduce the initial guess for each subinterval and add the matching condition at each boundary. In the end, we turn the original problem in the form of equation 2.2 into piecewise OCPs with augmented parameters and constraints.

The dynamic systems of the OCP in eqation 2.2 is defined in the subequation 2.2b, which we show here independently

$$\dot{x}(t) = f(x(t), u(t)), \quad t \in [t_0, t_f] \tag{2.7}$$

If the initial value x_0 for 2.7 is known, then equation 2.7 becomes an initial value problem (IVP) and the solution can be found via numerical method. If analytical solutions exist for equation 2.7, then we can use the analytical solution directly. Nevertheless, for most real-life problems, the analytical solution either does not exist or is very difficult to find, and numerical method is the only practical way to find the solution. Given the initial value x_0, equation 2.7 will have solution as

$$\begin{aligned} x(t) - x_0 &= \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau, \quad t \in [t_0, t_f] \\ x(t) &= x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau, \quad t \in [t_0, t_f] \end{aligned} \tag{2.8}$$

The integral in equation 2.8 can be approximated by numerical methods, and one simple approximation to the integral in equation 2.8 can be obtained via Euler method.

The Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. Using a step size equal to 1, the approximation to equation 2.8 via Euler method is of the form

$$x(t) = x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau))d\tau \;\approx\; x_0 + tf(x(t_0), u(t_0)) \tag{2.9}$$

Euler method is a first order and explicit method for approximating intergrals, another widely used method is the trapezoidal rule, which is an implicit second-order method. The trapezoidal rule can approximate equation 2.8 as

$$x(t) = x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau))d\tau \;\approx\; x_0 + \frac{t}{2}[f(x(t_0), u(t_0)) + f(x(t), u(t))] \tag{2.10}$$

Notice, the soultion $x(t)$ becomes an input in the approximation in equation 2.10, and that is why this method is classified as an implicit method. The trapezoidal rule belongs to the family of Runge–Kutta method, which is a family of implicit and explicit iterative methods, with various order of derivatives used. Among them, the Runge–Kutta RK4 is the most widely used, which we describe in the following text. Let an initial value problem be specified as follows:

$$\dot{y} = f(t, y), \quad y(t_0) = y_0 \tag{2.11}$$

Here y is an unknown function (scalar or vector) of time t, which we would like to approximate; we are told that $\dot{y} = \frac{dy}{dt}$, the rate at which y changes, is a function of t and of y itself. At the initial time t_0 the corresponding y value is y_0. The function f and the initial conditions t_0, y_0 are given. Now we pick a step-size $h > 0$ and define:

$$y_{n+1} = y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)h, \tag{2.12}$$

$$t_{n+1} = t_n + h \tag{2.13}$$

for $n = 0, 1, 2, 3, ...$, using

$$k_1 = \; f(t_n, y_n), \tag{2.14}$$

$$k_2 = \; f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \tag{2.15}$$

$$k_3 = \; f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \tag{2.16}$$

$$k_4 = \; f(t_n + h, y_n + hk_3)\,. \tag{2.17}$$

Here y_{n+1} is the RK4 approximation of $y(t_{n+1})$, and the next value (y_{n+1}) is determined by the present value (y_n) plus the weighted average of four increments, where each increment is the product of the size of the interval, h, and an estimated slope specified by function f on the right-hand side of the differential equation.

- k_1 is the slope at the beginning of the interval, using y (Euler's method);
- k_2 is the slope at the midpoint of the interval, using y and k_1;
- k_3 is again the slope at the midpoint, but now using y and k_2;

- k_4 is the slope at the end of the interval, using y and k_3.

In the numerical implementation for our chosen case, the Runge–Kutta RK4 method will also be used.

To solve the OCP as in equation 2.2, we first discretize the time horizon $\in [t_0, t_f]$ into multiple subintervals $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$, then we introduce the initial guess for x and u in each subinterval, i.e. we have $x_0, x_1, ..., x_{m-1}$ and $u_0, u_1, ..., u_m$. Then in each subinterval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, the following step should be taken:

- Solve the dynamic system as in equation 2.7 with the initial guess (x_{j-1}, u_{j-1}) within subinterval $\mathbb{I}_j$, and obtain a solution $x(\tau; x_{j-1}, u_{j-1}),\ \tau \in \mathbb{I}_j = [\tau_{j-1}, \tau_j]$
- Evaluate the cost function 2.2a within subinterval $\mathbb{I}_j$ with the solution $x(\tau; x_{j-1}, u_{j-1})$. We use the notation F_j as the objective function value that is applicable in subinterval $\mathbb{I}_j$, with the definition given in equation 2.19.
- Evaluate the error with the constraints 2.2c, 2.2d, 2.2f, 2.2g, and 2.2h, which are applicable in subinterval $\mathbb{I}_j$.

Within subinterval $\mathbb{I}_j$, with initial guess $X_j = (x_{j-1}, u_{j-1})$, the result from solving the dynamic system equation 2.7 might be an infeasiable solution, i.e. a solution which does not satisfy the constraints 2.2c, 2.2d, 2.2f, 2.2g, and 2.2h. In this case, we can update the initial guess iteratively until feasiable solutions are found and one optimal solution is reached. The solution is optimal if a minimum cost function value in the subinterval $\mathbb{I}_j$ is obtained while satisfying all the constraints 2.2c, 2.2d, 2.2f, 2.2g, and 2.2h. In practice, we do not aim to find the optimal solution in each subinterval, instead we aim to find the optimal solution of the whole interval, i.e. $min \sum_{j=1}^{m} F_j$. With the matching codition added, then the original OCP is transferred into NLP of the following form

$$\min_{w} \sum_{j=1}^{m} F_j \tag{2.18a}$$

$$s.t.\ \ x(\tau_j; x_{j-1}, u_{j-1}) - x_j = 0 \quad j = 1, 2, ..., m \tag{2.18b}$$

$$g(x_{j-1}, u_{j-1}) = 0 \quad j = 1, 2, ..., m \tag{2.18c}$$

$$h(x_{j-1}, u_{j-1}) \leq 0, \quad j = 1, 2, ..., m \tag{2.18d}$$

$$g(x(\tau_m; x_{m-1}, u_{m-1}), u_m) = 0 \ (\ the\ final\ point\ t_m = t_f) \tag{2.18e}$$

$$h(x(\tau_m; x_{m-1}, u_{m-1}), u_m) \leq 0 \ (\ the\ final\ point\ t_m = t_f) \tag{2.18f}$$

$$x(t_0) = x_0, \tag{2.18g}$$

$$x^{lower} \leq x_j \leq x^{upper} \quad j = 0, 1, 2, ..., m \tag{2.18h}$$

$$u^{lower} \leq u_j \leq u^{upper} \quad j = 0, 1, 2, ..., m \tag{2.18i}$$

$$t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f \tag{2.18j}$$

where $w = (x_0, u_0, x_1, u_1, ..., x_{m-1}, u_{m-1}, u_m)$, and x_{j-1}, u_{j-1} is the initial guess for interval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, the solution $x(\tau_j; x_{j-1}, u_{j-1})$ comes from solving the dynamic system 2.7 numerically for the subinterval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, and F_j is defined as in 2.19

$$\begin{aligned} F_j &= \int_{\tau_{j-1}}^{\tau_j} L(x(t), u(t))dt \qquad\qquad if \quad j = 1, ..., m-1 \\ F_j &= \int_{\tau_{j-1}}^{\tau_j} L(x(t), u(t))dt + E(x(t_f)) \quad if \quad j = m\ (the\ last\ subinterval) \end{aligned} \tag{2.19}$$

2.2 KKT condition

As stated in the introduction Chapter 1, direct approaches transform the original infinite optimal control problem into a finite dimensional nonlinear programming problem (NLP). We have shown in previous section 2.1.1 that an OCP of the form 2.2 can be transferred into NLP of the form 2.18. This NLP is then solved by variants of state-of-the-art numerical optimization methods. KKT condition is one of the widespread and sucessfully used techniques to address the constraints of NLP. Before we explain the KKT condition, we need to introduce several definitions and theorems first. We consider a general NLP of the following form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \\ & h(x) \leq 0 \end{aligned} \tag{2.20}$$

Definition 1 *(Feasible set) The feasible set Ω is the set*

$$\Omega := \{x \in \mathbb{R}^n \mid g(x) = 0,\ h(x) \leq 0\} \tag{2.21}$$

Definition 2 *(Active Constraints and Active Set) An inequality constraint $h_i(x) \leq 0$ is called active at $x^\star \in \Omega$ iff $h_i(x^\star) = 0$ and otherwise inactive. The index set $\mathcal{A}(x^\star) \subset \{1, ..., n_h\}$ of active inequality constraint indices is called the "active set".*

Often, the name active set also includes all equality constraint indices, as equalities could be considered to be always active.

Definition 3 *(LICQ) The linear independence constraint qualification (LICQ) holds at $x^\star \in \Omega$ iff all vectors $\nabla g_i(x^\star)$ for $i \in \{1, ..., n_g\}$ and $\nabla h_i(x^\star)$ for $i \in \mathcal{A}(x^\star)$ are linearly independent.*

To give further meaning to the LICQ condition, let us combine all active inequalities with all equalities in a map $\tilde{g}$ defined by stacking all functions on top of each other in a column vector as follows:

$$\tilde{g}(x) = \begin{pmatrix} g(x) \\ h_i(x),\ i \in \mathcal{A}(x^\star) \end{pmatrix} \tag{2.22}$$

With the definitions above, we are ready to formulate the famous KKT condition.

Theorem 1 *(KKT Conditions) If $x^\star$ is a local minimizer of the problem 2.20 and the LICQ holds at $x^\star$, then there exist so called multiplier vectors $\lambda \in \mathbb{R}^n_g$ and $\mu \in \mathbb{R}^n_h$ with*

$$\nabla f(x^\star) + \nabla g(x^\star)\lambda^\star + \nabla h(x^\star)\mu^\star = 0 \tag{2.23a}$$
$$g(x^\star) = 0 \tag{2.23b}$$
$$h(x^\star) \leq 0 \tag{2.23c}$$
$$\mu^\star \geq 0 \tag{2.23d}$$
$$\mu_i^\star h_i(x^\star) = 0,\ i = 1, ..., n_h \tag{2.23e}$$

The "KKT Conditons" are also known as "First-Order Necessary Conditons".

Definition 4 *(KKT Point) We call a triple $(x^\star, \lambda^\star; \mu^\star)$ a "KKT Point" if it satisfies LICQ (Definition 3) and the KKT conditions (Theorem 1).*

Definition 5 *(Lagrangian Function) We define the so called "Lagrangian function" to be*

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^\top g(x) + \mu^\top h(x) \tag{2.24}$$

Here, we have used the so called "Lagrange multipliers" $\lambda \in \mathbb{R}^n_g$ and $\mu \in \mathbb{R}^n_h$. The last three KKT conditions 2.23c - 2.23e are called the complementarity conditions. For each index i, they define an $L-$shaped set in the (h_i, μ_i) space. This set is not a smooth manifold but has a non-differentiability at the origin, i.e. if $h_i(x^\star) = 0$ and also $\mu_i^\star = 0$. This case is called a weakly active constraint. Often we want to exclude this case. On the other hand, an active constraint with $\mu_i^\star > 0$ is called strictly active.

Definition 6 *Regard a KKT point $(x^\star, \lambda^\star; \mu^\star)$. We say that strict complementarity holds at this KKT point iff all active constraints are strictly active.*

Theorem 2 *(Second Order Optimality Conditions) Let us regard a point $x^\star$ at which LICQ holds together with multipliers $\lambda^\star, \mu^\star$ so that the KKT conditions are satisfied and let strict complementarity hold. Regard a basis matrix $\mathbb{Z} \in \mathbb{R}^{n*(n-n_g)}$ of the null space of $\frac{\partial \tilde{g}}{\partial x}(x^\star) \in \mathbb{R}^{n_{\tilde{g}}*n}$, i.e., $\mathbb{Z}$ has full column rank and $\frac{\partial \tilde{g}}{\partial x}(x^\star)\mathbb{Z} = 0$. Then the following two statements hold:*

- If $x^\star$ is a local minimizer, then $\mathbb{Z}^\top \nabla^2_x \mathcal{L}(x^\star, \lambda^\star, \mu^\star)\mathbb{Z} \succeq 0$.(Second Order Necessary Condition, short : SONC)
- If $\mathbb{Z}^\top \nabla^2_x \mathcal{L}(x^\star, \lambda^\star, \mu^\star)\mathbb{Z} \succ 0$, then $x^\star$ a local minimizer. This minimizer is unique in its neighborhood, i.e., a strict local minimizer, and stable against small differentiable perturbations of the problem data. (Second Order Sufficient Condition, short: SOSC).

The matrix $\nabla^2_x \mathcal{L}(x^\star, \lambda^\star, \mu^\star)$ plays an important role in optimization algorithms and is called the Hessian of the Lagrangian, while its projection on the null space of the Jacobian, $\mathbb{Z}^\top \nabla^2_x \mathcal{L}(x^\star, \lambda^\star, \mu^\star)\mathbb{Z}$, is called the reduced Hessian.

For an optimization problem, if we can start with an initial guess x_0 and find a updating schema which decreases the objective value while maintain the Hessian matrix positive, if such updating schema can converge, it will converge to a local minimizer. If the original problem is convex, the local miminizer is, therefore, the global minimizer. Taking advantage of these properties, we can reformulate the NLP as in equation 2.20 as well as the optimal control problem as in equation 2.2, and then re-write in the Lagrangian form 2.24, optimize the (objective) Lagrangian function with quasi Newton method.

2.3 quasi Newton method

As explained in Section 2.2, KKT condtion can be applied to include the constraints into a Lagrangian function, and we can employ an updating schema with an initial guess to solve NLP as in equation 2.20 and optimal control problem as in equation 2.2. Newton

and quasi Netwon method can be used as the updating method to find the optimal solution. For the sake of simplicity, we focus on explaining the Newton and quasi Newton method without constraints in this chapter. How the quasi Newton method, together with multiple shooting, can be used to solve optimal control problem, will be exlained in more details in Chapter 4 when we discuss our numerical solutions for the rocket car case.

2.3.1 Newton method

A general optimization problem is typically of the form

$$\begin{aligned} &min \quad f(x) \\ &s.t. \quad x \in \Omega \end{aligned} \tag{2.25}$$

Here $x \in \Omega$ represents the constraints for which x must satisfy, it may be in the form of $\{g(x) = 0, h(x) \leq 0\}$ as in the equation 2.20, i.e. the feasible set $\Omega = \underset{x}{arg}\ \{g(x) = 0, h(x) \leq 0\}$. Before discussing the optimization problem with constraints, we first focus on the problems without constraints, i.e. problem of the form

$$\min_{x \in \mathbb{R}^n} \qquad f(x) \tag{2.26}$$

The problem 2.26 can be solved via Newton's method, which attempts to solve this problem by constructing a sequence $\{x_k\}$ from an initial guess (starting point) x_0 that converges towards a minimizer $x^\star$ of $f(x)$ by using a sequence of second-order Taylor approximations of $f(x)$ around the iterates. The second-order Taylor expansion of $f(x)$ around x_k is

$$f(x_k + \delta_x) \approx h(x_k) := f(x_k) + f'(x_k)\delta(x_k) + \frac{1}{2}f''(x_k)\delta(x_k)^2$$

where δ represents a small change (with respect to x), and f', f'' are the first and second order derivatives of the original function $f(x)$. The notation f', f'' are usually expressed as ∇f and H (the Hessian matrix) respectively when x is a vector of variables. In the text that follows, we will use the symbol ∇f and H directly. Therefore, the Talyor expansion can be written as

$$f(x_k + \delta_x) \approx h(x_k) := f(x_k) + \nabla f(x_k)^T\delta(x_k) + \frac{1}{2}H(x_k)\delta(x_k)^2$$

The next iterate x_{k+1} is defined so as to minimize this quadratic approximation $h(\cdot)$. The function $h(\cdot)$ is a quadratic function of $\delta(x)$, and is minimized by solving $\nabla h(\cdot) = 0$. The gradient of $h(\cdot)$ with respect to $\delta(x_k)$ at point x_k is

$$\nabla h(x_k) = \nabla f(x_k) + H(x_k)\delta(x_k)$$

We are motivated to solve $\nabla h(x_k) = 0$, which turns out to solve a linear system

$$\nabla f(x_k) + H(x_k)\delta(x_k) = 0 \tag{2.27}$$

Therefore, for the next iteration point x_{k+1}, we can just add the small change $\delta(x_k)$ to the current iterate, i.e.

$$x_{k+1} = x_k + \delta(x_k) = x_k - H^{-1}(x_k)\nabla f(x_k),$$

here $H^{-1}(\cdot)$ represents the inverse of the Hessian matrix $H(\cdot)$. The Newton method performs the iteration until the convergence, i.e. x_k and $f(x_k)$ converge to $x^\star$ and $f(x^\star)$, respectively[1]. The details of the Newton method is as follows:

Newton method

- Step 0, $k = 0$, choose an initial value x_0
- Step 1, $\delta(x_k) = -H^{-1}(x_k)\nabla f(x_k)$, if $\delta(x_k) = 0$, then stop
- Step 2, choose a step-size α_k (typically $\alpha_k = 1$)
- Step 3, set $x_{k+1} = x_k + \alpha_k\delta(x_k)$, let $k = k + 1$. Go to Step 1

The parameter α_k is introduced to augment the Newton method such that a line-search of $f(x_k + \alpha_k\delta(x_k))$ is applied to find an optimal value of the step size parameter α_k.

Though the Newton method is straightforward and easy to understand, it has two main limitations. Firstly, it is sensitive to initial conditions. This is especially apparent if the objective function is non-convex. Depending on the choice of the starting point x_0, the Newton method may converge to a global minimum, a saddle point, a local minimum or may not converge at all. In another word, due to the sensitivity with respect to the initialization, the Newton method may be not able to find the global solution. Secondly, the Newton method can be computationally expensive, with the second-order derivatives, aka, the Hessian matrix $H(\cdot)$ and its inverse very expensive to compute. It may also happen that the Hessian matrix is not positive definite, therefore, Newton method can not be used at all for solving the optimization problem. Due to these limitations of the Newton method, the quasi Netwon method is, therefore, usually preferred for solving optimal control or general optimisation problems.

2.3.2 quasi Newton method

We have stated that one limitation or the downside of the Newton method, is that Newton method can be computationally expensive when calculating the Hessian (i.e. second-order derivatives) matrix and its inverse, especially when the dimensions get large. The quasi-Newton methods are a class of optimization methods that attempt to address this issue. More specifically, any modification of the Newton methods employing an approximation matrix B to the original Hessian matrix H, can be classified into a quasi-Newton method.

The first quasi Newton algorithm, i.e. the Davidon–Fletcher–Powell (DFP) method, was proposed by William C. Davidon in 1959 Davidon [1959], which was later popularized by Fletcher and Powell in 1963 Fletcher R. [1963]. Some of the most common used quasi-Newton algorithms currently are the symmetric rank-one (SR1) method A. R. Conn [1991] and the Broyden–Fletcher–Goldfarb–Shanno(BFGS) method. The family of the quasi Newton algorithms are similar in nature, with most of the difference arising in the part how the approximation Hessian matrix is decided and the updating distance $\delta(x_k)$ is calculated. One of the main advantages of the quasi Newton methods over Newton method is that the approximation Hessian matrix B can be chosen in a way that no matrix needs to be directly inverted. The Hessian approximation B is chosen to satisfy the equation 2.27, with the approximation matrix B replacing the original Hessian matrix H, i.e.

$$\nabla f(x_k) + B_k\delta(x_k) = 0 \tag{2.28}$$

[1] In another word, the Newton mehtod has converged when the small change $\delta(x_k) = 0$ or $\delta(x_k)$ is small enough that the change in the objective function is below a pre-defined tolerance level.

In the text that follows, we explain how the iteration is performed in the BFGS method, as an example illustrating the quasi Newton method. In the BFGS method, instead of computing B_k afresh at every iteration, it has been proposed to update it in a simple manner to account for the curvature measured during the most recent step. To determine an update scheme for B, we will need to impose additional constraints. One such constraint is the symmetry and positive-definiteness of B, which is to be preserved in each update for $k = 1, 2, 3, ...$ Another desirable property is that B_{k+1} is sufficiently close to B_k at each update $k + 1$, and such closeness can be measured by the matrix norm, i.e. the quantity $\|B_{k+1} - B_k\|$. We can, therefore, formulate our problem during the $k + 1$ update as

$$\begin{aligned} &\min_{B_{k+1}} \|B_{k+1} - B_k\| \\ &s.t. \;\; B_{k+1} = B_{k+1}^T, \;\; B_{k+1}\delta(x_k) = y_k \end{aligned} \tag{2.29}$$

where $\delta(x_k) = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. In the BFGS method, the norm is chosen to be the Frobenius norm:

$$\|B\|_F = \sqrt{\sum_i^m \sum_j^n |b_{ij}|^2}$$

Solving the problem 2.29 directly is not trivial, but it has been proved (ref Jorge Nocedal [2006]) that problem ends up being equivalent to updating our approximate Hessian B at each iteration by adding two symmetric, rank-one matrices U and V:

$$B_{k+1} = B_k + U_k + V_k$$

where the update matrices can then be chosen of the form $U = auu^T$ and $V = bvv^T$, where u and v are linearly independent non-zero vectors, and a and b are constants. The outer product of any two non-zero vectors is always rank one, i.e. U_k and V_k are rank-one. Since u and v are linearly independent, the sum of U_k and V_k is rank-two, and an update of this form is known as a rank-two update. The rank-two condition guarantees the "closeness" of B_k and B_{k+1} at each iteration.

Besides, the condition $B_{k+1}\delta(x_k) = y_k$ has to be imposed.

$$B_{k+1}\delta(x_k) = B_k\delta(x_k) + auu^T\delta(x_k) + bvv^T\delta(x_k) = y_k$$

Then, a natural choice of u and v would be $u = y_k$ and $v = B_k\delta(x_k)$, we then have

$$\begin{aligned} B_k\delta(x_k) + ay_ky_k^T\delta(x_k) + bB_k\delta(x_k)\delta(x_k)^TB_k^T\delta(x_k) &= y_k \\ y_k(1 - ay_k^T\delta(x_k)) &= B_k\delta(x_k)(1 + b\delta(x_k)^TB_k^T\delta(x_k)) \\ \Rightarrow a = \frac{1}{y_k^T\delta(x_k)}, \;\; b &= -\frac{1}{\delta(x_k)^TB_k\delta(x_k)} \end{aligned}$$

Finally, we get the update formula as follows:

$$B_{k+1} = B_k + \frac{y_ky_k^T}{y_k^T\delta(x_k)} - \frac{B_k\delta(x_k)\delta(x_k)^TB_k}{\delta(x_k)^TB_k\delta(x_k)}$$

Since B is positive definite for all $k = 1, 2, 3, ...$, we can actually minimize the change in the inverse B^{-1} at each iteration, subject to the (inverted) quasi Newton condition and the requirement that it is symmetric. Applying the Woodbury formula (ref Woodbury [1950] for more details), we can show that the updating formula of inverse B^{-1} is as follows

$$B_{k+1}^{-1} = (I - \frac{\delta(x_k)y_k^T}{y_k^T\delta(x_k)})B_k^{-1}(I - \frac{y_k\delta(x_k)^T}{y_k^T\delta(x_k)}) + \frac{\delta(x_k)\delta(x_k)^T}{y_k^T\delta(x_k)} \tag{2.30}$$

As shown in the formula 2.30, at each iteration, we update B^{-1} by using $\delta(x_k) = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. Since an update of B^{-1} depends on the previous value, we need to initialize B^{-1}, either as an identity matrix or as the true Hessian matrix $H(x_0)$, calculated based on the starting point x_0.

We have shown that with quasi Newton method, with an initial guess x_0, we can interatively update x_k until convergence so that an optimal solution to the unconstrained optimization problem as in 2.26 can be obtained. In the next section, we explain how the quasi Netwon method can be applied to optmization problems with contstraints, as well as the OCP problems of the form 2.2.

2.4 Sequential quadrative programming

With all the background knowlege of mutiple shooting, KKT conditions and Newton-style method, now we are ready to explain how we can solve the OCP with the form 2.2 in details, i.e. working within an iterative Sequential Quadratic Programming (SQP) approach.

As explained in section 2.1.1, the original OCP with the form 2.2 can be transferred into a NLP of the form 2.18. Within the NLP 2.18, the independent variables are the initial guess of x and u for each subinterval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$ and the final control value u_m, i.e. $w = (x_0, u_0, x_1, u_1, ..., x_{m-1}, u_{m-1}, u_m)$. The constraints in formulation 2.18 are applied to the independent variables $w = (x_0, u_0, x_1, u_1, ..., x_{m-1}, u_{m-1}, u_m)$ piecewise, and they are in the form as equality constraints or inequality constraints. Together with the matching condtion 2.18b, then the NPL problem 2.18 can be re-written as the form in NLP 2.20, with the x in NLP 2.20 replaced by $w = (x_0, u_0, x_1, u_1, ..., x_{m-1}, u_{m-1}, u_m)$.

To solve NLP of the form 2.20, we can use the KKT apporach, i.e. including the constraints and the original objective function into a new objective function, defined as a Lagrangian function as 2.24, which we re-rewite here

$$\mathcal{L}(w, \lambda, \mu) = f(w) + \lambda^\top g(w) + \mu^\top h(w) \tag{2.31}$$

Based on our discussion regarding Newton-style method, to find the minimum of $\mathcal{L}(w, \lambda, \mu)$, we are motivated to solve a linear system as in equation 2.27, where the first order derivatives and Hessian matrix is calculated for the objective function $\mathcal{L}(w, \lambda, \mu)$. Starting with an initial guess of (w_0, λ_0, μ_0), then we are motivated to update (w, λ, μ) so as to decrease the objective function $\mathcal{L}(\cdot)$ by solving a subproblem of quadrative pro-

gramming of the following form

$$\min_{\Delta w} \quad \frac{1}{2}\Delta w^\top \nabla_w^2 \mathcal{L}(\cdot)\Delta w + \nabla_w f(w_i)^T \Delta w \tag{2.32a}$$

$$\text{s.t.} \quad g(w_i) + \nabla g(w_i)^T \Delta w = 0 \tag{2.32b}$$

$$h(w_i) + \nabla h(w_i)^T \Delta w \leq 0 \tag{2.32c}$$

where $\nabla_w^2 \mathcal{L}(\cdot)$ is the Hessian matrix of function $\mathcal{L}(\cdot)$, whose updating can be calculated with quasi Newton method. With a solution $\Delta w_k, \lambda_k^{QP}, \mu_k^{QP}$ found from the subproblem 2.32, we can therefore update the w, λ, μ as

$$w_{k+1} = w_k + \Delta w_k \tag{2.33a}$$

$$\lambda_{k+1} = \lambda_k^{QP} \tag{2.33b}$$

$$\mu_{k+1} = \mu_k^{QP} \tag{2.33c}$$

until convergence.

We have shown that the original OCP of the form 2.2 can be transferred into a NLP of the form 2.18 using mutiple shooting method. The NLP 2.18 can then be re-written in the form 2.20, and can be solved using Netwon-style framework, i.e. sequential quadrative programming and quasi Netwon method. The numerical implementation details to a case study of rocket car will be given in Chapter 4.

3 Optimal control under uncertainty

Besides the OCP as in the formulation 2.2, some OCP may have uncertain parameters whose value are priori unknown, and the optimal objective value depends on the parameter value, as shown in the formulation 1.2 in Chapter 1. This kind of problem is called the parameterized optimization problems and the formulation 1.2 can be augumented with mathematical details as well, leading to a OCP under uncertainty of the following form

$$\min_{x(t;p),u(t)} \; F(x(t;p),u(t)) = \int_{t_0}^{t_f} L(x(t;p),u(t))dt + E(x(t_f;p)) \tag{3.1a}$$

$$s.t. \;\; \dot{x}(t;p) = f(x(t;p),u(t)) \;\; (system\ dynamics) \tag{3.1b}$$

$$g(x(t;p),u(t)) = 0, \;\; (path\ equality\ constraints) \tag{3.1c}$$

$$h(x(t;p),u(t)) \leq 0, \;\; (path\ inequality\ constraints) \tag{3.1d}$$

$$x(t_0) = x_0, \;\; (initial\ value) \tag{3.1e}$$

$$r(x(t_f;p)) \leq 0, \;\; (terminal\ constraints) \tag{3.1f}$$

$$x^{lower} \leq x(t;p) \leq x^{upper} \tag{3.1g}$$

$$u^{lower} \leq u(t) \leq u^{upper} \tag{3.1h}$$

$$p \in \mathbb{P} \tag{3.1i}$$

$$x = x(;,p^\star)\ if\ p = p^\star \tag{3.1j}$$

$$t \in [t_0, t_f] \tag{3.1k}$$

In other words, the state variables x depenpds not only on the system dynamics 3.1b, the control variables u, but also an uncertain parameter p. Parameterized optimization problems are very difficult to solve due to the uncertainty in the parameter p. Since different parameter p will lead to different solutions, it makes sense to solve the parameterized optimal control problems in a conservative way. For simplicity and the sake of generalization, we use the more compact formulation 1.2 in Chapter 1 as the representation of a parameterized optimal control problem for our subsequent discussion,which we show here again

$$\begin{aligned} \min_{x(t),u(t)} \quad & F(x(t;p),u(t)) \\ s.t. \quad & \dot{x}(t;p) = f(x(t;p),u(t)) \\ & x(t;p) \in \Omega \\ & u(t) \in \mathbb{U} \\ & p \in \mathbb{P} \\ & x = x(;,p^\star)\ if\ p = p^\star \\ & t \in [t_0, t_f] \end{aligned} \tag{3.2}$$

In the paper Schlöder [2022], multiple methods of solving the parameterized optimal control problem have been discussed. The main idea of solving the parameterized optimal control problem 3.2 in a conservative way is to transform the problem into another form. Two different ways of solving the parameterized optimal control problem will be discussed in details in the following sections, i.e. the classical approach and the training approach.

3.1 Classical approach

The classical approach belongs to the family of robust optimization. Robust optimization is an important subfield of optimization that deals with optimization problems under uncertainty. These uncertainties may influence the satisfication of constraints but also the objective function value. The aim of roubust optimization is to robustify or immunize a solution against uncertainty in terms of feasibility and optimality. In this paper on hand, we focus on problems with deterministic uncertainty, i.e. the uncertain parameters lie in a so-called uncertainty set. For deterministic uncertain problems, robustifying the solution of the considered problem means that the robustified solution yields feasible parameter-dependent variables for all possible realizations of the uncertain parameters, and the robustified solution is optimal with regard to the worst possible value the objective function can take due to uncertainty. Therefore, the robustified solution is conservative. The dominating paradigm in this area of robust optimization is *minmax* model, namely

$$\underset{x\in\mathbb{R}^n}{min}\ \underset{p\in\mathbb{P}}{max}\ f(x,p) \tag{3.3}$$

This is the classic format of the generic model, and is often referred to as *minmax* approach, also called classical appraoch. In this paper on hand, we are particular interested in the robustification of OCPs under uncertainties. In our chosen case, the uncertainty appears in the form of an uncertain parameter which enters the differential equations.

For the original problem 3.2, assuming parameter $p^\star$ lies in an uncertainty set $\mathbb{P}$, we can firstly reach one objective, i.e. identifying a worst possible solution with respect to $p^\star$, i.e. solving a lower level problem. Based on the result of lower level, we can continue to find

the best solution with respect to x, i.e. solving a upper level problem. The "worst-case treatment planning by bilevel optimal control", i.e. a bilevel optimization problem, is an optimization problem in which another optimization problem enters the constraints. Mathematically, the problem 3.2 is transformed into another form, as following

$$\begin{aligned} \min_{x(:,p),u(\cdot)} \quad & \max_{p\in\mathbb{P}} \quad F(x(t;p),u(t)) \\ s.t. \quad & \dot{x}(t;p) = f(x(t;p),u(t)) \\ & x(t;p) \in \Omega \\ & u(t) \in \mathbb{U} \\ & p \in \mathbb{P} \\ & x = x(;,p^\star) \ if \ p = p^\star \\ & t \in [t_0, t_f] \end{aligned} \tag{3.4}$$

In the classical approach, the set of feasible controllable variables are given by $u(\cdot)$, which yield feasible trajectories $x(\cdot,p)$ for all $p \in \mathbb{P}$. The value of the objective function in the lower level does not depend on p and $x(\cdot,p)$. In other words, in this approach, the dynamic system has no prior knowledge about the value of the parameter p and gets no feedback during the process and has to decide the control strategy in advance.

In the papers Vasile [2014] and Konstantin Palagachev [2016], methods of solving bilevel problems have been discussed. In paper Konstantin Palagachev [2016], the lower level problem is a bang bang control problem and theoretical solution can be derived, then the original bilevel problem turns into a single level problem, which can be solved with numerical methods discussed in Chapter 2. In paper Vasile [2014], a general algorithm have been proposed for solving robust optimization or *minmax* problem. Basically, the idea is to break the bilevel problem into two parts and interatively solving these two parts one after another.

$$p^\star = \underset{p\in\mathbb{P}}{arg\ max} \quad F(x^\star(t;p),u^\star(t)) \tag{3.5}$$

where $x^\star(t;p), u^\star(t)$ comes from the solution of

$$x^\star(t;p), u^\star(t) = \underset{x(:,p^\star),u(\cdot)}{arg\ min} \quad F(x(t;p^\star),u(t)) \tag{3.6}$$

Such algorithm can be used to solve a general *minmax* problem. Details on the implementation can be found in the paper Vasile [2014].

3.2 Training approach

The paper Schlöder [2022] introduces the "Training Approach". It is based on the idea that in the real world, during the training period, an intervention is introduced and a certain, but a priori unknown, parameter $p \in \mathbb{P}$ is realized. What follows the training period (during which the parameter p is realized), the dynamic system is able to react to it in an optimal manner, i.e. an optimal value $F(\cdot)$ will be obtained given the realized parameter p. The paper Schlöder [2022] call this approach "worst case modeling Training

Approach", and it can be written as

$$\begin{aligned} \max_{p\in\mathbb{P}} \min_{x(:,p),\ u(t)} \quad & F(x(t;p),u(t)) \\ s.t.\ \ & \dot{x}(t;p) = f(x(t;p),u(t)) \\ & x(t;p) \in \Omega \\ & u(t) \in \mathbb{U} \\ & p \in \mathbb{P} \\ & x = x(;,p^\star)\ if\ p = p^\star \\ & t \in [t_0, t_f] \end{aligned} \tag{3.7}$$

Due to the *max mix* notation, this approach of solving the bilevel problem can be also called *maxmin* approach. The solution of the Training Approach in paper Schlöder [2022] is given by a gradient-free method, more precisely, a so-called model-based Derivative-Free Optimization (DFO) approach for box-constrained optimization problems is used. The BOBYQA algorithm is chosen for such approach to solve problems of the form

$$\begin{aligned} &\min_{x\in\mathcal{R}^n}\ F(x) \\ &s.t.\ a_i \le x_i \le b_i, i = 1, ..., n \end{aligned} \tag{3.8}$$

The name BOBYQA is an acronym for "Bound Optimization By Quadratic Approximation", and is used to solve lower level problem of 3.7. In the general DFO method, the objective function $F(\cdot)$ is considered a black box. For a given p, the parametric lower level OCP of the Training Approach 3.7 is solved with a direct DFO approach and the resulting (finite dimensional) solution is viewed as dependent variable. Furthermore, the uncentainty set $\mathbb{P}$ is chosen to be box-shaped, and hence the BOBYQA algorithm is applicable to the problem in the Training Approach. The BOBYQA algorithm has been introduced in details in the paper Powell [2009], and we reiterate the main idea in the text that follows.

The method of BOBYQA is iterative, k and n being reserved for the iteration number and the number of variables, respectively. Further, m is reserved for the number of interpolation conditions that are imposed on a quadratic approximation $Q_k(x) \to F(x),\ x \in \mathcal{R}^n$, with m is a chosen constant integer from the interval $[n+2, \frac{1}{2}(n+1)(n+2)]$.

The approximation is available at the beginning of the k-th iteration, the interpolation equations have the form

$$Q_k(y_j) = F(y_j),\ j = 1, 2, ..., m. \tag{3.9}$$

We let x_k be the point in the set $\{y_j : j = 1, 2, ..., m\}$ that has the property

$$F(x_k) = min\ \{F(y_j),\ j = 1, 2, ..., m\}, \tag{3.10}$$

with any ties being broken by giving priority to an earlier evaluation of the least function value $F(x_k)$. A positive number Δ_k, called the "trust region radius", is also available at the beginning of the k-th iteration. If a termination condition[1] is satisfied, then the iteration stops. Otherwise, a step d_k from x_k is constructed such that $\|d_k\| \le \Delta_k$ holds,

[1]Typically, a termination condition is satified when the objective value can not be improved further after some iterations. For the termination condition of BOBYQA algorithm, please refer the paper Powell [2009] for more details.

$x = x_k + d_k$ is within the bounds of equation 3.8, and $x_k + d_k$ is not one of the interpolation points $y_j : j = 1, 2, ..., m$. Then the new function value $F(x_k + d_k)$ is calculated, and one of the interpolation points, y_t say, is replaced by $x_k + d_k$, where y_t is different from x_k. It follows that x_{k+1} is defined by the formula

$$x_{k+1} = \begin{cases} x_k, & F(x_k + d_k) \geq F(x_k) \\ x_k + d_k, & F(x_k + d_k) < F(x_k) \end{cases} \tag{3.11}$$

Further, Δ_{k+1} and Q_{k+1} are generated for the next iteration, Q_{k+1} being subject to the constraints

$$Q_{k+1}(\hat{y}_j) = F(\hat{y}_j), \ j = 1, 2, ..., m, \tag{3.12}$$

at the new interpolation points

$$\hat{y}_j = \begin{cases} y_j, & j \neq t, \\ x_k + d_k, & j = t \end{cases}, \ j = 1, 2, ..., m. \tag{3.13}$$

The operations of BOBYQA algorithm requires the user to provide an initial vector of variables $x_0 \in \mathcal{R}^n$, the initial trust region Δ_1, and the number m of interpolation conditions where $n + 2 \leq m \leq \frac{1}{2}(n + 1)(n + 2)$. Two different ways have been proposed for constructing the step d_k from x_k and updating procedures from the k-th iteration to the $k + 1$-th iteration in the paper Powell [2009], with both methods having utilized the "quadratic" nature of the approximation function $Q(\cdot)$.

The lower level OCP of the Training Approach 3.7 can be solved with the BOBYQA algorithm for a given p, since the lower level problem can be re-written into the form of 3.8 and the uncentainty set $\mathbb{P}$ is chosen to be box-shaped. With the BOBYQA algorithm computing local extrema, the upper level problem still needs to be solved globally. In our rocket car case in Chapter 4, this is straight-forward, i.e. maximizing over all p.

Nevertheless, the BOBYQA algorithm has limitations with several strong assumptions being made. Firstly, it has been assumed the uncertainty set is of moderate size and is box-shaped. Secondly, it has been assumed that there is only one local extrema, i.e. the lower level problem has only one solution for each $p \in \mathbb{P}$. In general, we cannot expect the second assumption to be valid. The BOBYQA algorithm is a gradient-free method with repect to the objective function $F(\cdot)$, it still utilizes the gradient of the approximation function $Q(\cdot)$ while updating the iteration. Therefore, this BOBYQA algorithm, or a general DFO approach, is still subject to the numerical errors and computational costs while calculating the gradients of the approximation function $Q(\cdot)$ and updating them in each iteration.

The paper at hand, instead, is utilizing the gradient of the objective function $F(\cdot)$ directly, with some approximation applied as well. We have used the multiple shooting and Newton-style framework for solving the lower level problem of the Training Approach 3.7 with a specific p_i value. We discretize the whole uncertainty set $\mathbb{P}$ into multiple points in an increasing order $[p_0, p_1, ..., p_n] \subset \mathbb{P}$ so that $p_i, i = 0, 1, ..., n$ can approximately representing the whole set $\mathbb{P}$ when n is large enough. Then we take the max over all the lower level solution (i.e. one T_i corresponding to one p_i).

In the next Chapter 4, we discuss in details how the numerical methods discussed in Chapter 2 can be solved the bilvel problem in the classical and training approach, with a case study (i.e. state constrained rocket car) as an example for illustration.

4 Numerical solution

In this chapter, we use a case study as an example to illustrate how the numerical methods discussed before can be used for solving OCPs and OCPs under uncertainty. In the next section we first give a description of the case we have chosen, the state constrained rocket car. After that, we present how the numerical methods are implemented for the chosen case and the numeric results.

4.1 Introduction to the rocket car case

The description of the rocket car case is mostly coming from the paper Schlöder [2022], with content either verbatim or in a modified form. We consider the rocket car case with state constraints, i.e. the one-dimensional movement of a mass point under the influence of some constant acceleration/deceleration, e.g. modeling head-wind or sliding friction, which can accelerate and decelerate in order to reach a desired position. The mass of the car is normalized to 1 unit[1] and the constant acceleration/deceleration enters the model in form of an unknown parameter $p \in \mathbb{P} \subset \mathbb{R}$ suffering from uncertainty, with the uncertainty set $\mathbb{P}$ convex and compact. We consider a problem in which the rocket car shall reach a final feasible position and velocity in a minimum time:

$$
\begin{aligned}
\underset{T,u(\cdot),x(:,p)}{min} \quad & T && && (4.1a)\\
s.t. \quad & x = (x_1, x_2), && && (4.1b)\\
& \dot{x} = T \begin{pmatrix} x_2(t;p) \\ u(t) - p \end{pmatrix}, && t \in [0,1], && (4.1c)\\
& x(0,p) = 0, && && (4.1d)\\
& x_1(1;p) \geq 10, && && (4.1e)\\
& x_2(t;p) \leq 4, && t \in [0,1], && (4.1f)\\
& x_2(1;p) \leq 0, && && (4.1g)\\
& T \geq 0, && && (4.1h)\\
& u(t) \in [-10, 10], && t \in [0,1]. && (4.1i)
\end{aligned}
$$

where x represents the variables of the rocket car, and it has two components $x = (x_1, x_2)$. The first component x_1 is the (time-transformed) position of the rocket car. The second component x_2 is (time-transformed) velocity of the rocket car. The control function $u : [0,1] \to \mathbb{R}$ in equation 4.1 represents the acceleration/deceleration value. The condition 4.1d, i.e. $x(0,p) = 0$, indicates that at starting time $t = 0$, both the position and velocity of the car is 0. The condition 4.1e, i.e. $x_1(1;p) \geq 10$, indicates that the

[1] We do not specify the unit on purpose since the actual unit, either one kilogram or meter, does not play a role in the modeling. We are more concerned about the scale.

position of the car at $t = 1$ must be greater than or equal to 10. The condition 4.1f, i.e. $x_2(t;p) \leq 4$, indicates that the velocity of the car is always smaller or equal to 4 across the whole period. The condition 4.1g, i.e. $x_2(1;p) \leq 0$, indicates that the velocity of the car at ending time $t = 1$ is always smaller or equal to 0. Here, a negative velocity means that the car is moving in a direction that decreases the position. To make the rocket car case even simpler, we can limit the size of the uncertainty set, as following

$$p \in \mathbb{P} = [p_l, p_u] = [0, 9], \tag{4.2}$$

where $p_l < p_u$, with p_l and p_u the lower and upper boundary of the parameter p.

Because our objective is to minimize the time between starting state and ending state, i.e. the variable T, which is unknown, we cannot define a time horizon over which we will discretize and optimize. Therefore, a new variable t, as in the problem 4.1, is defined as follows:

$$t = \frac{\tau}{T} \in [0, 1] \quad \tau \in [0, T] \tag{4.3}$$

where τ is the real time between starting time 0 and ending time T, and t is the relative time between 0 and 1. The equation 4.1c can be also written as

$$\dot{x} = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = T \begin{pmatrix} x_2(t;p) \\ u(t) - p \end{pmatrix} = \begin{pmatrix} T x_2(t;p) \\ T(u(t) - p) \end{pmatrix} \tag{4.4a}$$

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau}\frac{\partial \tau}{\partial t} \\ \frac{\partial x_2}{\partial \tau}\frac{\partial \tau}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau} T \\ \frac{\partial x_2}{\partial \tau} T \end{pmatrix} = \begin{pmatrix} T x_2(t;p) \\ T(u(t) - p) \end{pmatrix} \tag{4.4b}$$

$$\begin{pmatrix} \frac{\partial x_1}{\partial \tau} \\ \frac{\partial x_2}{\partial \tau} \end{pmatrix} = \begin{pmatrix} x_2(t;p) \\ u(t) - p \end{pmatrix} \tag{4.4c}$$

The equation $\frac{\partial x_1}{\partial \tau} = x_2(t;p)$ means the change in the position in real time is proportional to the speed at that moment. The equation $\frac{\partial x_2}{\partial \tau} = u(t) - p$ means the change in speed is proportional to the acceleration/deceleration value at that moment. The variable $x(t;p)$ is a dependent variable, and is uniquely determined by $T, u(\cdot)$ and p. The goal is to minimize T such that the variable $x(t;p)$ satisfies all the conditions in 4.1.

4.2 Theoretical solution to rocket car case

We have chosen the rocket car case for two main reasons. First, this case is easy to understand, and second a theoretical solution exists, which is useful for evaluating the result of the numerical methods. The optimization problem 4.1 has a unique global solution, and no further local solution exists. The optimal solution based on the uncertain parameter p is given by

$$T^\star = T^\star(p) = 2.5 + \frac{40}{100 - p^2}, \tag{4.5}$$

and the optimal control function $u^\star(\cdot)(= u^\star(\cdot;p))$ by

$$u^\star(\cdot) = \begin{cases} 10, & for\ 0 \leq t < \frac{4}{(10-p)T^\star} \\ p & for\ \frac{4}{(10-p)T^\star} \leq t < 1 - \frac{4}{(10+p)T^\star} \\ -10 & for\ 1 - \frac{4}{(10+p)T^\star} \leq t \leq 1 \end{cases} \tag{4.6}$$

In words, we accelerate as strongly as possible (the acceleration value $u^\star(t) = 10$) until the velocity $x_2^\star(t;p) = 4$, and then keep the velocity $x_2^\star(t;p)$ constant for a certain period of time[2], and eventually decelerate as as strongly as possible until the velocity reaches the state $x_2(1;p) \leq 0$ and $x_1(1;p) \geq 10$. The first moment that these two conditions have been reached, is the moment that we finds the optimal/smallest T.

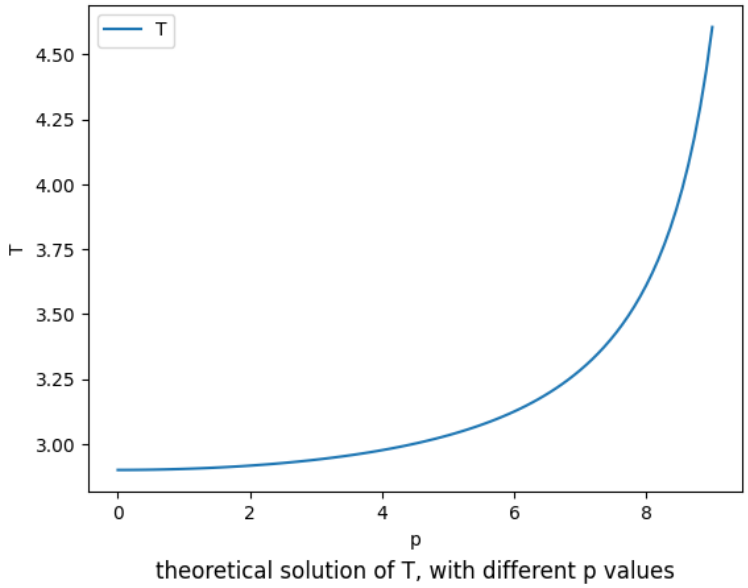


Figure 4.1: The theoretical optimal T values against p values in the uncertainty set, from the formulation 4.5

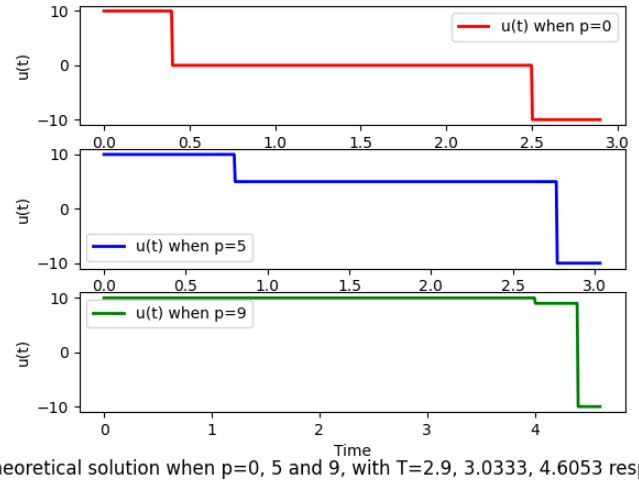
Our rocket car problem is a typical bang-bang control problem, its optimal solution is obtained when the trajectory takes the extreme points whenever possible. This means acceleration, speed and deceleration take the maximum value whenever possible. A more proof of general bang-bang control problem needs the background knowledge in calculus of variations, Hamilton–Jacobi–Bellman equation, and the Pontryagin Maximum Principle etc. The paper Konstantin Palagachev [2016] gives a proof and explanation of the solution to the bang bang problem within a bilevel OCP. We refrain from diving deep into these theories, readers can refer to papers McShane [1989], Gamkrelidze [1999] and Bertsekas [2005] for more information.

The theoretical solution (equations 4.5 and 4.6) to our problem 4.1 can also be obtained by solving the underlying partial differential equation 4.1c directly, whiling taking the constraints into consideration. The proof are given in Appendix B of Schlöder [2022]. The proof is tedious in detail, yet the underlying idea is simple and straightforward. Here we will briefly explain the idea behind the proof given in the paper Schlöder [2022] and show with one example that the theoretical solution in formulation 4.5 and 4.6 is indeed correct. We take a fixed parameter value $p = 0$ as an example for the explanation.

Because the objective is to minimize the time between the starting state and ending state, then it is optimal to run at the maximum allowable speed whenever possible, i.e with $x_2 = 4$ as in the constraint 4.1f. Otherwise, time is wasted if a car is running at a speed that is less than the maximum allowable speed, i.e. $x_2 = 4$. Since the starting state is at position 0 and speed 0, to reach the maximum allowable speed as soon as possible,

[2]The acceleration value cancels out with a inherent deceleration value so that the velocity can stay constant. The inherent deceleration value can be result of a friction or head wind.

the car should be accelerated as strongly as possible at the begining until the speed reaches $x_2 = 4$. Then the car should be running at this maximum speed for a certain period. After this period, the car should be decelerated as strongly as possible until the speed decreases to 0, and exactly at the moment that the speed reaches 0, the position should reach 10. It is this moment that the optimal/smallest T is achieved. How long the period should be running at maximum speed, depends on the acceleration/deceleration distance, which in turn depends on starting/ending position (in our case, it is 0 and 10 respectively) and starting/ending speed (in our case, both are 0).



theoretical solution when p=0, 5 and 9, with T=2.9, 3.0333, 4.6053 respectively

Figure 4.2: The theoretical u(t) values when p=0, p=5 and p=9, from the formulation 4.6

This is a qualitative explanation why the solution from formulation 4.5 and 4.6 is correct. Now, we show that with $p = 0$, the optimal time $T = 2.9$. At the beginning $\tau_0 = 0$, we should accelerate as much as possible until the real time $\tau_1 = 0.4$ (i.e. $t = \frac{\tau}{T} = 0.137931$). This number $\tau_1 = 0.4$ can be obtained by solving the partial differential equation 4.1c with boundary conditions $x_2(\tau_0) = 0$ and $x_2(\tau_1) = 4$. At $\tau_1 = 0.4$, the car speed reaches 4, and it keeps running at this speed $x_2 = 4$ till $\tau_2 = 2.5$. From $\tau = 2.5$, the car starts to decelerate as strongly as possible, and at $T = \tau_3 = 2.9$, the speed reaches 0 and the total distance traveled is exactly 10. Then all the contraints are satisfied, and the optimal time is $T = 2.9$.

We can prove that $T = 2.9$ is indeed the optimal solution when $p = 0$ by using the method proof by contradiction. In the optimal strategy above, it has three stages $[\tau_0, \tau_1]$, $[\tau_1, \tau_2]$, and $[\tau_2, \tau_3]$, corresponding to acceleration, constant and deceleration stages. We assume there is another strategy that differs the one we have described above, yet leading to a less time while satisfying all the constraints. First, we assume the alternative strategy does not speed as strong as possible, this means within this alternative strategy, the speed of the car is always less or equal to the speed of our optimal strategy before $\tau_1 = 0.4$. This means that before $\tau_1 = 0.4$, the travel distance in the alternative strategy is less than our optimal strategy, and therefore, it will takes more time for car to cover the remaining distance in the alternative strategy. This alternative strategy, therefore, can not take less time compared to the optimal strategy. Similarly, we can prove that any modication to the optimal strategy will lead a bigger T value. Therefore, by using the example of $p = 0$, we have shown the solution 4.5 and 4.6 is indeed optimal with respect to different p for all the p in the uncertainty set $\mathbb{P} = [p_l, p_u] = [0, 9]$. The details of such

proof can be found in paper Schlöder [2022].

We can also numerically verified that the theoretical solution is correct. First, we show the theoretical solution of T against different p values accross the whole uncertainty set $\subset \mathbb{P} = [0, 9]$, as shown in Figure 4.1. Then, we remove the uncertainty by showing the solution to the original problem 4.1 for several chosen p_i values, where $p_i \in [p_0, p_1, ..., p_n] \subset \mathbb{P} = [0, 9]$, here, we have chosen $p = 0$, $p = 5$, and $p = 9$ as the example for illustration. The T values (rounded to four decimals) and the theoretical $u(t)$ values corresponding to $p = 0$, $p = 5$, and $p = 9$, are shown in Figure 4.2.

When p takes fixed values, the OCP under uncertainty becomes a normal OCP problem and we can solve directly with the numerical methods discussed in Chapter 2, and no classical or training approach is needed. With $[0, 1]$ discretized into 500 subintervals $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, i.e. the discretizion points as $0 = t_0 =< t_1 < ... < t_m = t_f = 1$, and an error tolerance level as $1e - 6$, and Runge-Kutta RK4 method used, the result from solving a normal OCP with $p = 0$, $p = 5$ and $p = 9$ are shown in Figure 4.3, Figure 4.4 and Figure 4.5 respectively.

Based on these three Figures 4.3, 4.4 and 4.5, the results from $p = 0$, $p = 5$, and $p = 9$ are consistent with that from the theoretical results in formulation 4.5 and 4.6, as shown in Figure 4.1 and Figure 4.2. This confirms two points: first, the theoretical results are correct and second, the numerical methods we have chosen can solve our rocket car case with one chosen fixed p value. These methods can, therefore, be used to solve general OCPs.

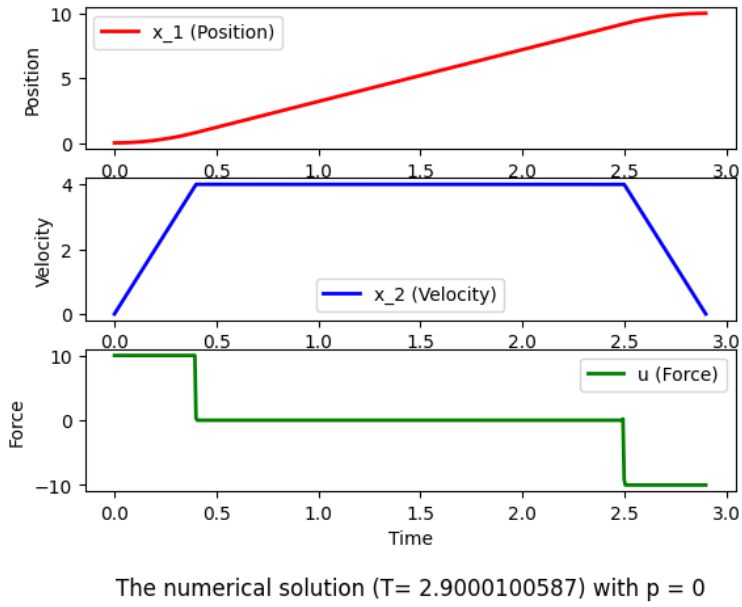


Figure 4.3: Orginal rocket car problem 4.1 solution when p=0

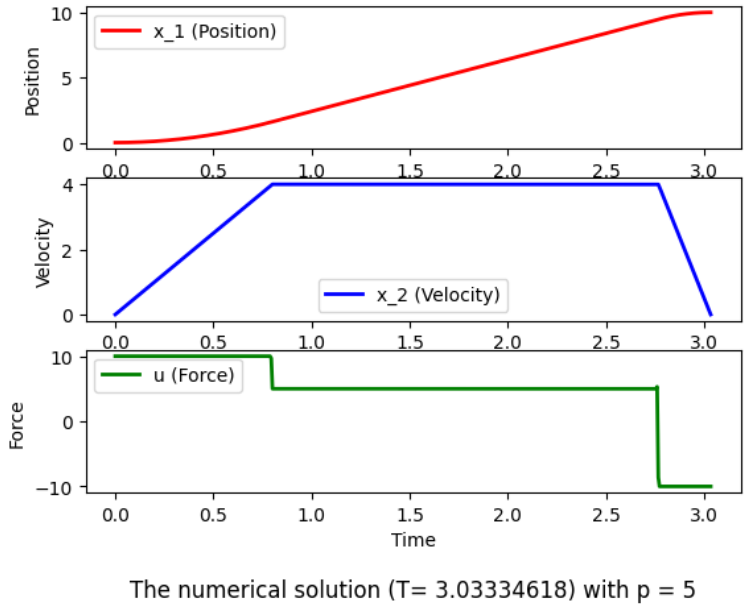


Figure 4.4: Orginal rocket car problem 4.1 solution when p=5

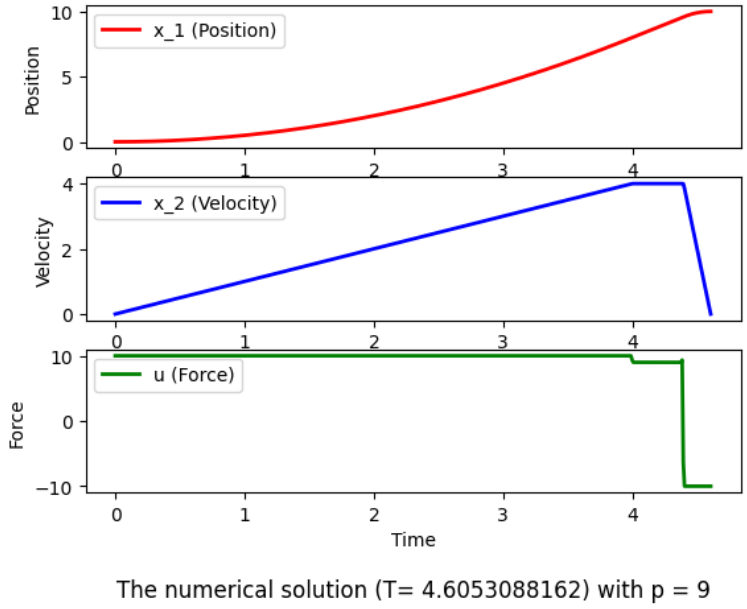


Figure 4.5: Orginal rocket car problem 4.1 solution when p=9

Because of the simplicity of the rocket car case, we can find the theoretical solution of the nominal/original problem for our case 4.1. But for many real life problems, it is very difficult to find a direct solution to the original problem, and for some cases not feasible, due to the uncertainty in the parameter p. That is why in the paper Schlöder [2022], a classical (in the form $minmax$) approach and a training (in the form $maxmin$) approach have been discussed, and both approaches will lead to a conservative solution to the original problem. A conservative solution to the problem in the paper Schlöder [2022], or to general problems, is a acceptable (or desired) result since less risk is taken and the result is more robust. In the next chapter, we discuss, in details, the classical

approach and training approach for the rocket car problem.

For both the classical approach and the training approach, after the discretization of the uncertainty set, the problem to be solved becomes a normal optimal control problem and we can use the mutiple shooting and quasi Newton (in the SQP framework) method to solve. We are levaraging open source package Casadi[3] and Gekko[4] to solve our problem. These two package can let the user choose different nonlinear programming solver and as well as different underlying numerical methods. In our case, we have chosen the mutiple shooting and sequential quadratic programming, together with Runge–Kutta RK4 method. The numerical result from the classical approach and training approach will be discussed in the next two sections.

4.3 Apply classical (minmax) approach

We have explained in Section 3.1 about the classical appraoch. Within this section, we show the numerical results of applying the classical approach for the chosen rocket car case.

First we discretize the whole uncertainty set $\mathbb{P}$ into multiple points in an increasing order $[p_0, p_1, ..., p_n] \subset \mathbb{P}$ so that $p_i, i = 0, 1, ..., n$ can approximately representing the whole set $\mathbb{P}$ when n is large enough. Here $p_0 = p_l$ and $p_n = p_u$, with $[p_l, p_u] = [0, 9]$, as shown in equation 4.2. We solve the *minmax* problem of the following form

$$\min_{\epsilon, u(\cdot), x(\cdot)} \max_{p_i \in \mathbb{P}} \epsilon \tag{4.7a}$$

$$s.t.\ T_i \leq \epsilon \tag{4.7b}$$

$$x = (x_1, x_2), \tag{4.7c}$$

$$\dot{x} = T_i \begin{pmatrix} x_2(t; p_i) \\ u(t) - p_i \end{pmatrix}, \qquad \forall\ p_i,\ t \in [0, 1], \tag{4.7d}$$

$$x(0, p_i) = 0, \tag{4.7e}$$

$$x_1(1; p_i) \geq 10 \qquad \forall\ p_i, \tag{4.7f}$$

$$x_2(t; p_i) \leq 4, \qquad \forall\ p_i, t \in [0, 1] \tag{4.7g}$$

$$x_2(1; p_i) \leq 0, \qquad \forall\ p_i, \tag{4.7h}$$

$$T \geq 0, \tag{4.7i}$$

$$u(t) \in [-10, 10], \qquad t \in [0, 1]. \tag{4.7j}$$

for all p_i where $p_i \in [p_0, p_1, ..., p_n] \subset \mathbb{P}$. Since both $u(\cdot)$ and $x(\cdot)$ are functions of t, we also needs to discretize these two varaibles in the time horizon (i.e. 500 subintevals in the numerical implementation). In this classical approach, the driver has no prior knowledge about the value of the parameter p and gets no feedback during the process and has to set up the driving strategy in advance. The realization of the trajectory of $u(\cdot)$ and $x(\cdot)$ for each p_i, therefore, may not be an optimal solution compared to the case when knowing the p_i value in advance.

[3]Please refer the website https://web.casadi.org for more details

[4]Please refer the website https://gekko.readthedocs.io/en/latest/ for more information

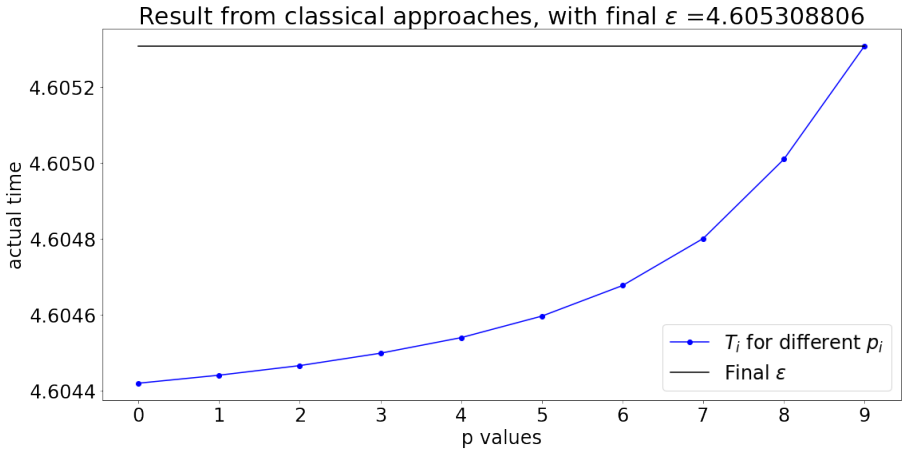


Figure 4.6: Final ϵ and T_i for different p_i values

For each $(p_i, x(t;p_i), (t;p_i))$, we get one T_i, which we try to get maximum value, and meanwhile over the $T_i, i = 0, 1, ..., n$, we would like to minimize ϵ ($\epsilon = \underset{i}{max}\ T_i$). In the end, we reach a worst ϵ over all $T_i, i = 0, 1, ..., n$, and for each i, we get a trajectory $(p_i, x(t;p_i), (t;p_i))$.

In the actual implementation, we have take $n = 40$, i.e. in total 40 p_i points ranging the whole uncertainty set $\mathbb{P} = [0, 9]$. Nevertheless, we only show the result for $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ for the sake of readability. The final time ϵ and T_i is shown in Figure 4.6, where $\epsilon = \underset{i}{max}\ T_i, i = 0, 1, 2, ..., 9$, and ϵ reaches the maximum value 4.6053, when $p_i = 9$. The differences in T_i arise mainly because of the numerical errors, in our implementation, the number of subinterval is chosen as 500 and the numerical error tolerance level is set $1e-6$. When we increase the number of subintervals and decrease the tolerance level, the T_i for $p_i \neq= 9$ should converge to $T_i, p_i = 9$.

The trajectories $(p_i, x(t;p_i), (t;p_i))$ for $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ are shown in Figure 4.7, Figure 4.8 and Figure 4.9 respectively. In all the three Figures 4.7, 4.8 and 4.9, the standardized time $t \in [0, 1]$ is used as x-axis. In Figure 4.7, the $u(t)$ acceleration/deceleration Force is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$; in Figure 4.8, the position $(x_1(t))$ is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$; and in Figure 4.9, the velocity/speed $(x_2(t))$ is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

We have explained that the trajectories of different p_i from the classical approach may not be optimal, compared to the case when solving the OCP 4.1 directly with the known p_i. In the following, we compare the trajectories of the result from the classical approach for $p_i = 0$, $p_i = 5$, and $p_i = 9$ to the cases when solving the OCP 4.1 directly with $p_i = 0$, $p_i = 5$, and $p_i = 9$ known in advance. The results from solving OCP 4.1 directly with $p_i = 0$, $p_i = 5$, and $p_i = 9$ known in advance, are already shown in Figures 4.3, 4.4 and 4.5.

In Figure 4.10, Figure 4.11 and Figure 4.12, we show the comparisons for $p_i = 0$, $p_i = 5$, and $p_i = 9$ between results of solving OCP 4.1 directly and results applying classical approach to problem 4.7. Obviously, we can see from from Figure 4.10 and Figure 4.11, the solution from classical apporoach is more conservative compared to solving OCP 4.1 directly for $p = 0$ and $p = 5$ respectively. When $p = 9$, there is only one line in Figure 4.12 for the position $x_1(t)$, force $u(t)$ and velocity $x_2(t)$. This confirms that indeed, when $p = 9$, the worst case happens, and the corresponding $\epsilon = 4.6053$ is equal to result of solving the 4.1 directly with $p = 9$. For other $p_i \neq 9$, we can always find an feasible

solution with a corresponding T_i, which will always be less than or equal to 4.6053.

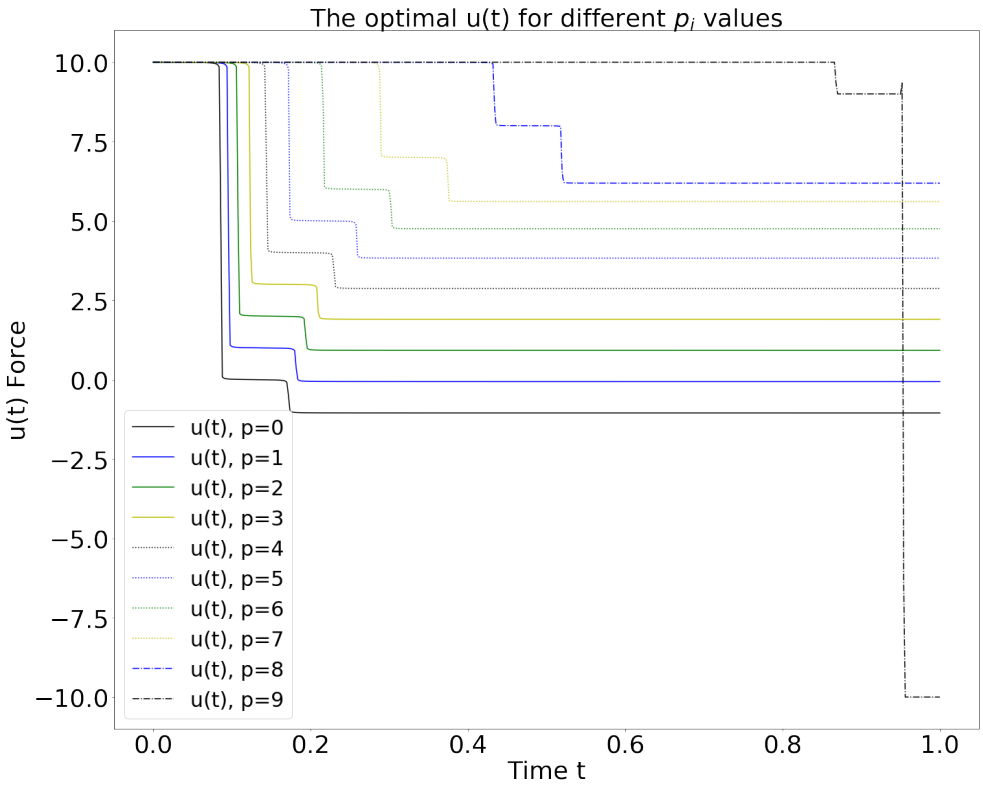


Figure 4.7: u(t) Force for different p_i values

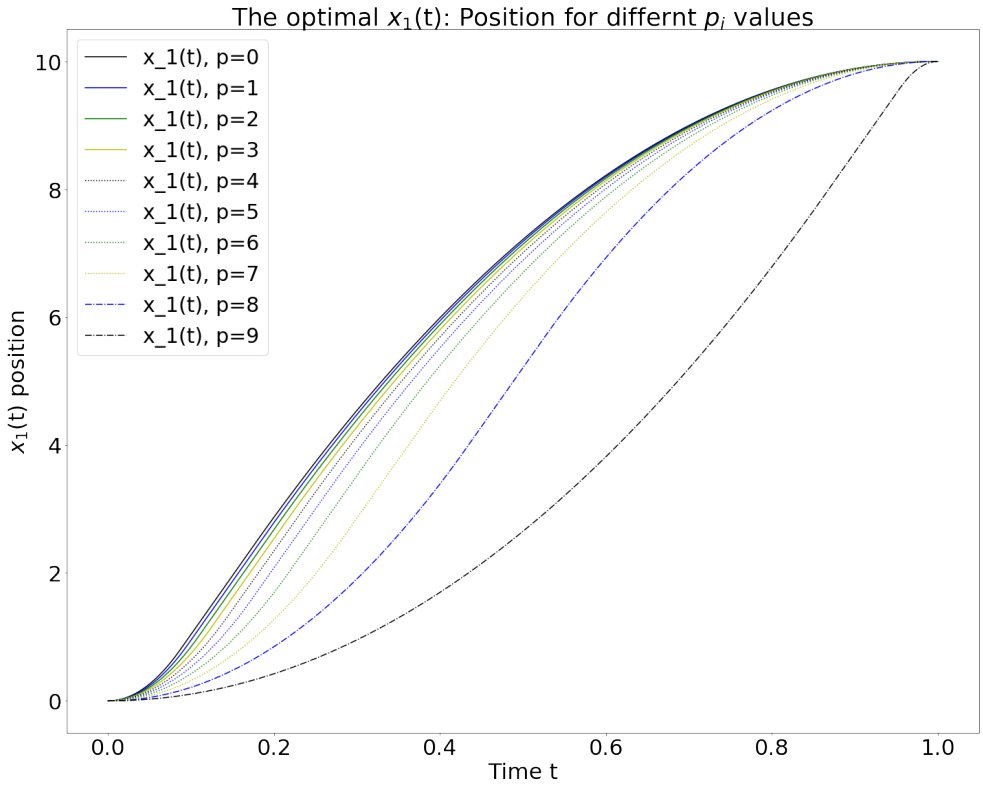


Figure 4.8: $x_1(t)$ Position for different p_i values

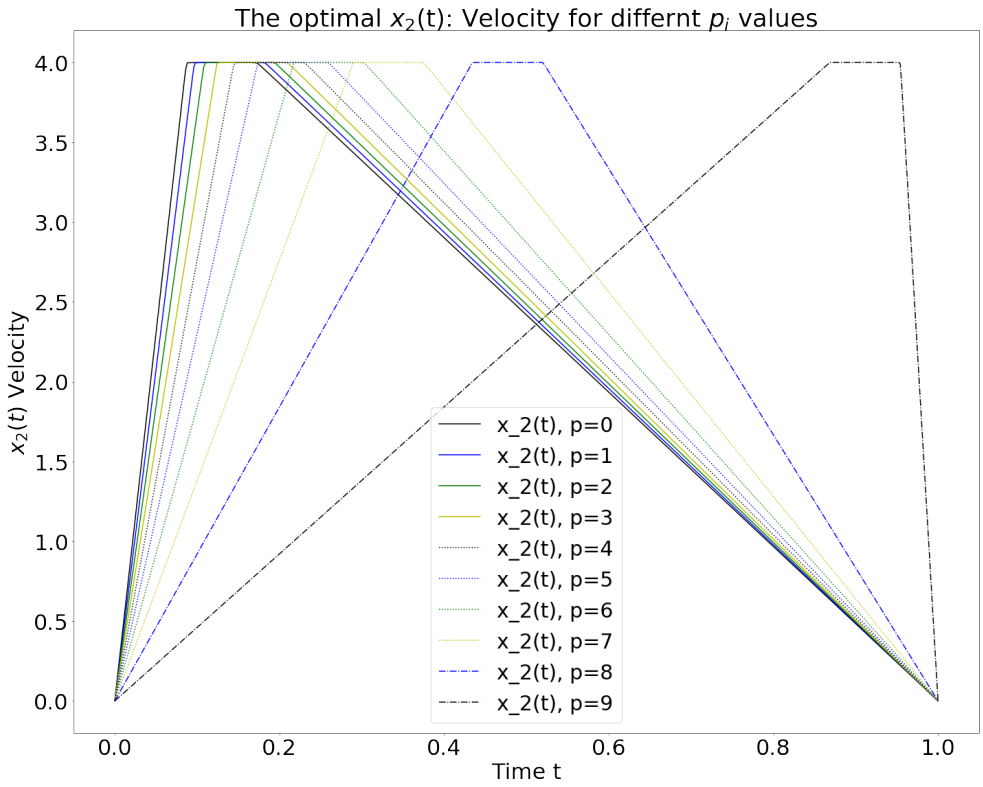


Figure 4.9: $x_2(t)$: Velocity for different p_i values

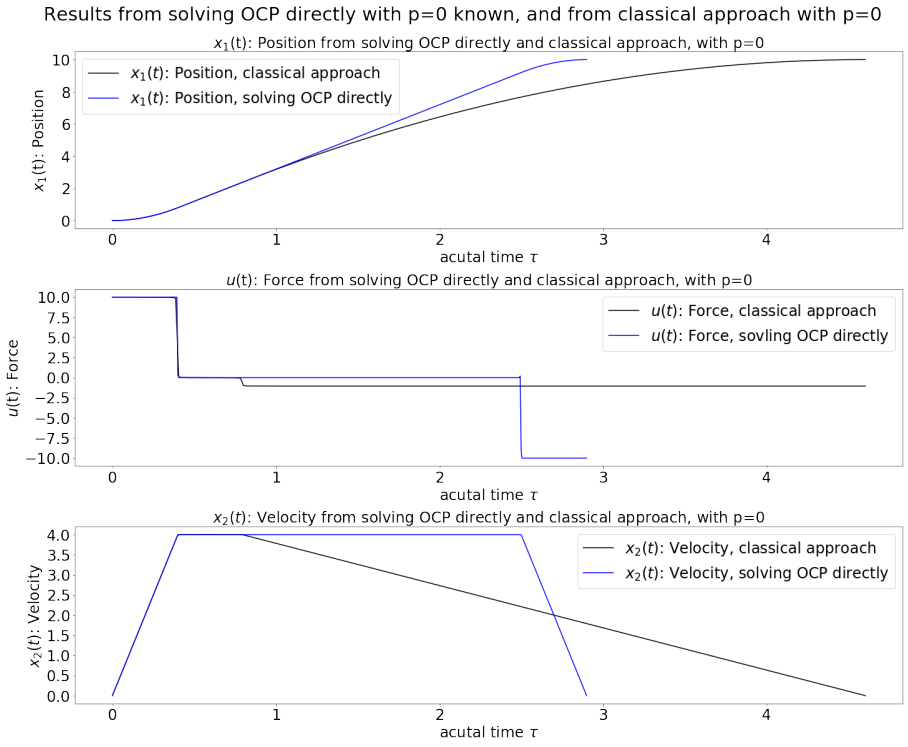


Figure 4.10: Compare the results from solving OCP 4.1 with $p = 0$ and the result of applying the classical approach to problem 4.7 with $p = 0$

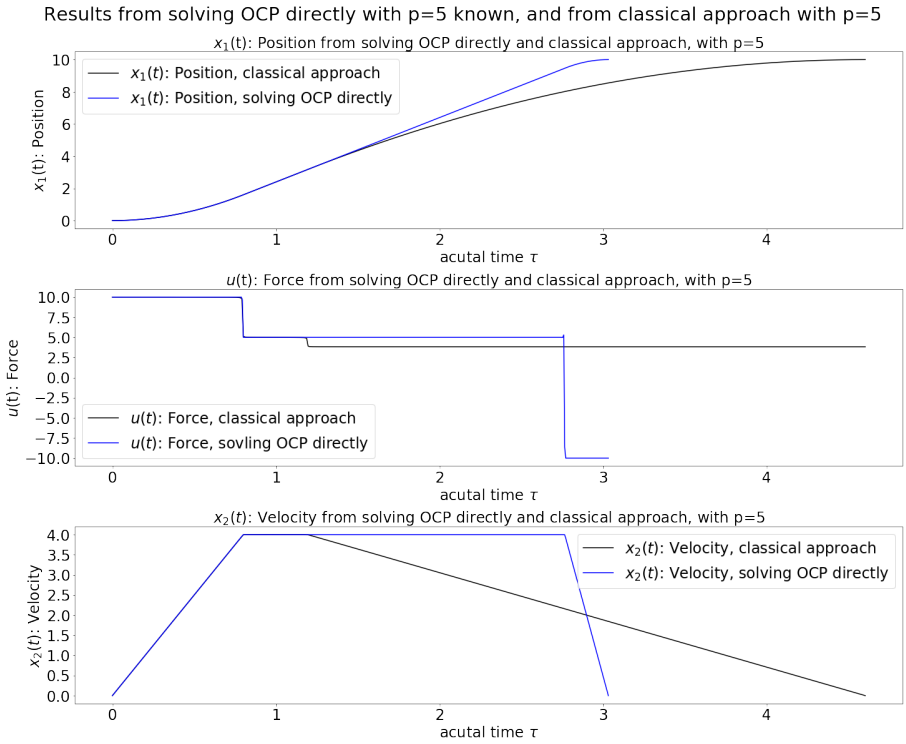


Figure 4.11: Compare the results from solving OCP 4.1 with $p = 5$ and the result of applying the classical approach to problem 4.7 with $p = 5$

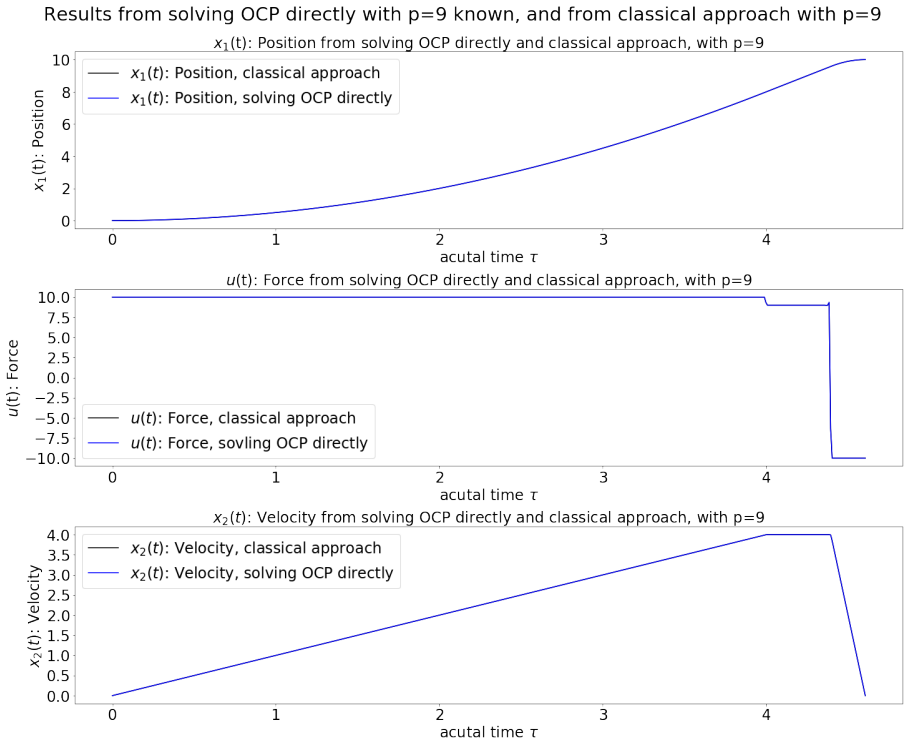


Figure 4.12: Compare the results from solving OCP 4.1 with $p = 9$ and the result of applying the classical approach to problem 4.7 with $p = 9$

4.4 Apply training (maxmin) approach

Contrast to the classical approach, in the training approach it is assumed that the driver of the rocket car is able to perform optimally for every p because of a preceding training period. We discretize the uncertainty set $\mathbb{P}$ the same as in the classical approach, i.e. we take the discretized points $p_0, p_1, ..., p_n$ which cover the whole uncertainty set $\mathbb{P}$. For each $p_i \in [p_0, p_1, ..., p_n] \subset \mathbb{P}$, we solve the problem

$$\max_{p_i \in \mathbb{P}} \min_{T, u(\cdot), x(\cdot)} \quad T \tag{4.8a}$$

$$s.t. \quad x = (x_1, x_2), \tag{4.8b}$$

$$\dot{x} = T \begin{pmatrix} x_2(t; p_i) \\ u(t) - p_i \end{pmatrix}, \qquad t \in [0, 1], \tag{4.8c}$$

$$x(0, p) = 0, \tag{4.8d}$$

$$x_1(1; p_i) \geq 10, \tag{4.8e}$$

$$x_2(t; p_i) \leq 4, \qquad t \in [0, 1], \tag{4.8f}$$

$$x_2(1; p_i) \leq 0, \tag{4.8g}$$

$$T \geq 0, \tag{4.8h}$$

$$u(t) \in [-10, 10], \qquad t \in [0, 1]. \tag{4.8i}$$

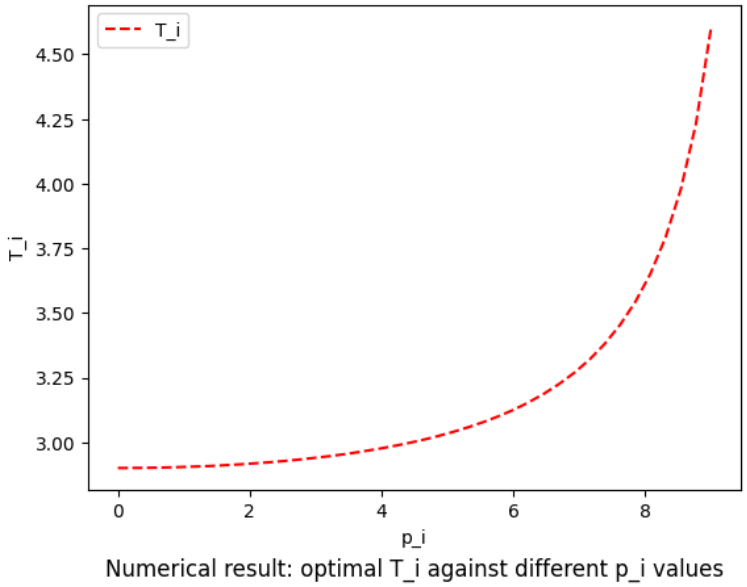


Figure 4.13: Numerical result: different T_i values against different p_i values, with $p_i = 9$ and $T_i = 4.6053$ the worst case

With a given p_i value, the lower level optimization problem turns out to be a normal OCP. One p_i value will correspond to one optimal T_i values. Among all the optimal T_i, we would like to find the worst case, i.e. the largest among all T_i values. Similar to the discretization of the uncertainty set in the classical approach, here we also take $n = 40$, i.e. in total 40 p_i points ranging the whole uncertainty set $\mathbb{P} = [0, 9]$. The result of T_i against p_i is shown in Figure 4.13. Figure 4.13 is almost indentical to Figure 4.1, which confirms

again that our numerical result with different p is consistent with the theoretical result. Furthermore, the worst case over all p_i is when p_i takes the biggest value, i.e. $p_i = 9$, and in this case $T_i = 4.6053$. We already shown in Figure 4.5 the solution to original problem when $p_i = 9$. The worst case, therefore, have the same solution as shown in Figure 4.5, with $T = 4.6053$, which we do repeat here in Figure 4.14.

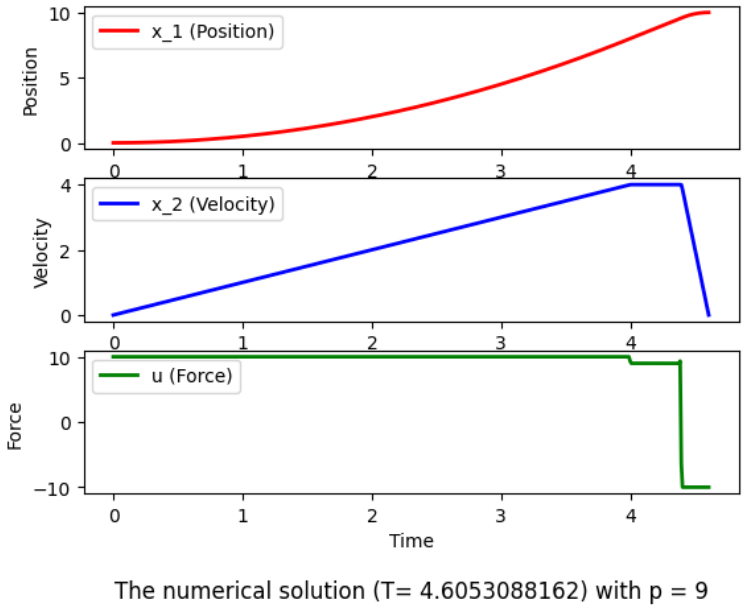


Figure 4.14: Training approach to problem 4.8, final result, worst case happens when $p = 9$

4.5 Analysis the numerical results

As show above, both the classical approach and training approach leads to an identical solution with final time as $T = 4.6053$, which is conservative with respective to oiriginal OCP 4.1. The time $T = 4.6053$ is conservative because regardless how the uncertain parameter p takes the value in the uncertainty set $\mathbb{P}$, we can always find a feasible solution that leads to a final time that is less than or equal to $T = 4.6053$. Neverthless, the classical approach and the training approach do not necessary lead to an identical solution for a general OCP under uncertainty. It happens that both approaches lead to the same solution to our chosen case purely because of the coincidence, and the worst case happens when $p = 9$ at the right boundary point.

5 Conclusion

Bibliography

Ph. L. Toint A. R. Conn, N. I. M. Gould. Convergence of quasi-newton matrices generated by the symmetric rank one update. 1991.

Dimitri P Bertsekas. Dynamic programming and optimal control. 2005.

John F. Zancanaro David Morrison, James D. Riley. Multiple shooting method for two-point boundary value problems. 1962.

William C. Davidon. Variable metric method for minimization. 1959.

Powell M.J.D. Fletcher R. A rapid convergent decent method for minimization. 1963.

R. V. Gamkrelidze. Discovery of the maximum principle. 1999.

Stephen J. Wright Jorge Nocedal. Numerical optimization. 2006.

Matthias Gerdts Konstantin Palagachev. Exploitation of the value function in a bilevel optimal control problem. 2016.

E. J. McShane. The calculus of variations from the beginning through optimal control theory. 1989.

Holger Diedam Pierre-Brice Wieber Moritz Diehl, Hans Georg Bock. Fast direct multiple shooting algorithms for optimal robot control. *Fast Motions in Biomechanics and Robotics*, 2005.

Michael J. D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Technical Report, Department of Applied Mathematics and Theoretical Physics*, 2009.

Matthias Schlöder. Numerical methods for optimal control of constrained biomechanical multi-body systems appearing in therapy design of cerebral palsy. 2022.

Massimiliano Vasile. On the solution of min-max problems in robust optimization. 2014.

Max A. Woodbury. Inverting modified matrices. 1950.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den (Datum) .....

Declaration:

I hereby confirm that I wrote this work independently and did not use any sources other than those indicated.

Heidelberg, (Date) .....