Department of Mathematics and Computer Science

Heidelberg University

Master thesis

in Scientific Computing

submitted by

Yu Xiang

born in November 10, 1989

2023

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 factory production, with constraints on resources available and uncertain market demand. This paper focuses on solving optimal control problems with numerical approaches, particularly with multiple shooting and (quasi) Newton type methods.

In general, optimal control deals with the problem of finding control over the state of 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:

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

Here t is the independent variable (generally speaking, time), usually using t_0 and t_f to represent the initial time and terminal time, respectively. The state variable is $x(t)$, and the control variable is $u(t)$. The objective function, also known as the cost function, is denoted by $F(\cdot)$. The underlying dynamic system is represented by $\dot{x}(t) = f(x(t), u(t))$. $x(t) \in \Omega$ and $u(t) \in \mathbb{U}$, represent the conditions that state variable $x(t)$ and control variable $u(t)$ must meet. The constraints are sometimes expressed as functions of $x(t)$ and $u(t)$ together.

Generally speaking, there are three basic approaches to address optimal control problems: (a) dynamic programming; (b) indirect approaches; and (c) direct approaches (ref Moritz Diehl [2005]). This paper focuses on direct approaches, which transform the original infinite optimal control problem into a finite-dimensional nonlinear programming problem (NLP). This NLP can then solved by variants of numerical optimization methods, and the approach is therefore often sketched as "first discretize, then optimize."

The multiple shooting method can be used in the "first discretize" part of the direct approach. The main idea is to transform the original optimal control problem into a nonlinear programming problem by coupling the control parameterization with a discretization of the state variables. The whole interval $[t_0, t_f]$ is, therefore discretized into m subintervals, with $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$. In each subinterval $\mathbb{I}_j = [\tau_j, \tau_{j+1}]$, the control variables $u(t)$ can be approximated by piecewise functions, and the state variables $x(t)$ are solved numerically with an initial value of $x(\tau_j) = s_j$ at the shooting node τ_j. The initial value $x(\tau_j)$ for the state variables at nodes τ_j must be guessed. Then in each subinterval, the state equations must be integrated individually from τ_j to τ_{j+1}. In addition, the continuity conditions (matching conditions) must be satisfied, which require that on each node the values $x(\tau_{j+1})$ should equal the final value of the preceding subinterval.

With multiple shooting methods applied, the optimal control problem is transformed into the NLP with constraints, which can be solved within Newton type framework.

Newton type methods are iterative methods based on second order derivatives of the cost function (in the unconstrained case) or Lagrange function (in the constrained case). Newton type methods generally, but not always, converges faster than methods using first-order derivatives (e.g. gradient methods), with respect to computational time. 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 employ an approximation to the original Hessian matrix and takes an efficient way to update the approximation matrix. Therefore, quasi Newton method is generally, but not always, faster than Newton method in computational time. For our constrained NLP, we use sequential quadratic programming (SQP), i.e. an iterative method for constrained nonlinear optimization. The SQP methods solve a sequence of optimization subproblems, each of which optimizes a quadratic model of the objective, subject to a linearization of the constraints.

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 parametric optimal control problems and is of the form

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

The difference between equation 1.2 and equation 1.1, is that the solution of equation 1.2 depends on the p from uncertainty set $\mathbb{P}$, the solution here refers to the trajectory of $x(t)$ and $u(t)$ as well as the optimal function value $F^\star(p)$, as defined in equation 1.3.

$$F^\star(p) = min\{F(x(\cdot), u(\cdot), p) \ s.t. \ all \ constraints\}, \tag{1.3}$$

Because of the uncertainty in the parameter p, parametric optimal control problems are extremely difficult to solve. The parameter p, as well as the corresponding solutions $u(t), x(t)$, and $F^\star(p)$, can have different values. Since p is priori unknown, in real life problems, it usually makes sense to solve the parametric optimal control problems in a conservative way, so that a robust solution can be obtained. For example, in the paper Schlöder [2022], when modeling the therapy design of Cerebral Palsy (CP) for the patient, a conservative solution is a desirable result. Two different ways of solving the parametric optimal control problem will be discussed in detail, i.e., the classical approach and the training approach. Both are in the form of a bilevel optimization problem, 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 of a 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 explaining in details how to solve optimal control problems with direct approaches using multiple shooting and quasi Newton method. In Chapter 3, we discuss the approaches for solving parametric 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 as

$$\dot{x}(t) = f(x(t), u(t)),\ x(t_0) = x_0,\ t \in [t_0, t_f] \tag{2.1}$$

We would like to have such a dynamic system run in an optimal way, subject to the constraints 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 an explicit formula. With the example of launching the rocket to the moon, the objective function can be minimizing the fuel used or minimizing the time horizon of the system, subject to the constraints, e.g., gravity, fuel efficiency, velocity limitation, etc. Differential equations are used to express the trajectories of the dynamic system.

In Chapter 1, equation 1.1 provides a basic formulation of an optimal control problem. Here we augment 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

$$\begin{aligned}
\min_{x(\cdot),u(\cdot)}\ & F(x(\cdot), u(\cdot)) = \int_{t_0}^{t_f} L(x(t), u(t))dt + E(x(t_f)) && \text{(2.2a)}\\
s.t.\ & \dot{x}(t) = f(x(t), u(t)),\quad (system\ dynamics) && \text{(2.2b)}\\
& g(x(t)) = 0\ or \le 0,\ t \in [t_0, t_f]\ (path\ equality\ or\ inequality\ constraints) && \text{(2.2c)}\\
& h(x(t), u(t)) = 0\ or \le 0,\ t \in [t_0, t_f]\ (mixed\ control - state\ constraints) && \text{(2.2d)}\\
& x(t_0) = x_0,\quad (initial\ value) && \text{(2.2e)}\\
& r(x(t_f)) \le 0,\quad (terminal\ constraints) && \text{(2.2f)}\\
& u^{lower} \le u(t) \le u^{upper} && \text{(2.2g)}\\
& t \in [t_0, t_f] && \text{(2.2h)}
\end{aligned}$$

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 path constraints, control-state constraints and terminal constraints respectively. The time horizon is $[t_0, t_f]$, with the initial value x_0.

Depending on the nature of the underlying optimal control problems, the mathematical expression can take a modified form of the equation 2.2. Some of the constraints defined in equation 2.2 may not be applicable to certain problems, while others may require the addition of new constraints or the modification of existing constraints. 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 into discussion of possible (minor) modifications 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 approaches, and (c) direct approaches. The optimality principle is used in dynamic programming to recursively compute a feedback control for all time t and all time 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 highlights direct approaches as a widely-used and effective technique for solving optimal control problems. Direct approaches involve converting the infinite optimal control problem into a finite-dimensional NLP, which is then solved using numerical optimization methods. This approach is often summarized as "first discretize, then optimize." One of the primary advantages of direct approaches is their ability to handle inequality constraints, such as the path constraints presented in equation 2.2c. NLP methods are equipped to handle structural changes in active constraints during the optimization process, making direct approaches suitable for addressing inequality constraints and active set changes (ref Moritz Diehl [2005]).

The multiple shooting method is first explained as a technique that can be used in the "first discretize" step of the direct approach. Following that, the KKT conditions, which serve as necessary conditions for optimality in nonlinear programming problems, are explained. These conditions enable a way to verify the optimality of a solution obtained through an optimization algorithm, and inspire the use of the Newton type method in this paper for NLP solving. By solving the NLP, the original OCP of the form 1.1 or 2.2 is consequently solved.

2.1 Multiple shooting

The multiple shooting method was originally developed to address boundary value problems (BVPs) in the context of differential equations (David Morrison [1962] and Osborne [1969]). With some modifications, this method is well-suited to solve optimal control problems with constraints defined by differential equations. The upcoming text will first outline the main concept of using multiple shooting to solve BVPs in the differential equation context. Subsequently, we will describe how the multiple shooting method can be applied to solve a general optimal control problem, with a specific focus on solving the optimal control problem presented in form 2.2.

We will illustrate the shooting method for solving a boundary value problem (BVP) using the following example:

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

The analytical solution to this equation is given by:

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

where e denotes the exponential number. Suppose we want to find the initial condition $x(t_0) = x_0$ that satisfies $x(t_f) = b$ for a given value of b. We can express this condition as $x(t_f) - b = 0$ or equivalently $x(t_0)e^{t_f - t_0} - b = 0$. This procedure is known as the shooting method.

The shooting method can be summarized as follows:

shooting method

1. Choose an initial value $x(t_0) = x_0$.
2. Use numerical integration (see appendix A) to obtain the solution $x(t), t \in [t_0, t_f]$ for the given initial condition $x(t_0) = x_0$.
3. Define a cost function for the error at the boundary and evaluate the cost function. If the cost function value is below a tolerance level (e.g., $1e - 8$), then stop. Otherwise, continue to the next step.
4. Update the guess for $x(t_0)$ based on some updating schema and go back to step 2.

In practice, we use numerical integration (see appendix A) to obtain the solution $x(t), t \in [t_0, t_f]$ in step 2. We define the cost function as $(x(t_f) - b)^2$ and set the tolerance level as a small number to balance accuracy and computation cost.

The multiple shooting method partitions the "shooting" interval into short subintervals and solves a differential equation with boundary value of the form:

$$\dot{y} = f(t, y) \quad y(t_f) = y_f \tag{2.3}$$

Here, y represents the differential variables, $t \in [t_0, t_f]$, and y_f is the boundary value at t_f. The method selects a suitable grid of multiple shooting nodes $\tau_j \in [t_0, t_f]$, where $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$, to create m subintervals covering the entire interval. In each subinterval, the initial guess of the starting value $\hat{y}_j$ is given at τ_j. An initial value problem of the following form is then solved:

$$\begin{aligned} &\dot{y} = f(t, y), \quad t \in [\tau_j, \tau_{j+1}], \quad j = 0, 1, ..., m-1 \\ &y(\tau_j) = \hat{y}_j, \quad j = 0, 1, ..., m-1 \end{aligned} \tag{2.4}$$

The piecewise solution is not necessarily continuous and does not necessarily satisfy the boundary condition $y(t_f) = y_f$. Therefore, additional matching conditions at each subinterval boundary are enforced to ensure continuity.

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

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, select initial guesses for $\hat{y}_j, j = 0, 1, ..., m-1$
- Step 3, form solutions to the differential equation in each subinterval $[\tau_j, \tau_{j+1}], j = 0, 1, ..., m-1$

- Step 4, define a cost function for the sum of the error at the boundary of each subinterval. If such cost function value is below a tolerance level, stop; otherwise, move to Step 5
- Step 5, update the guess for $\hat{y}_k, k = 0, 1, ..., m-1$ using some updating scheme, and go back to Step 3

In Step 4, the cost function can be defined as

$$\sum_{j=0}^{j=m-1} (y(\tau_{j+1}; \hat{y}_j) - \hat{y}_{j+1})^2 \tag{2.6}$$

and the tolerance level is set as a reasonable small number, e.g. $1e-8$.

The main idea of using the multiple shooting method to solve a boundary value problem has been summarized above, but the explanation lacks mathematical rigor and completeness. Many details need to be decided in practice when using shooting methods. For instance, in Steps 1 and 2, the shooting nodes and initial guess $\hat{y}_k$ are usually selected based on the problem's nature and a balance between accuracy and computational cost. In Step 3, polynomial functions can approximate the solutions using Taylor expansion. In Step 4, the cost function typically measures the sum of quadratic errors, but other forms are possible. In Step 5, an updating schema moves $\hat{y}_k$ towards decreasing the cost function, such as the (quasi) Newton method.

The multiple shooting method is a versatile approach that can be applied to various problems. For instance, it can be used to solve the second-order differential system in the form of Equation 2.7.

$$y''(t) = f(t, y(t), y'(t)) \quad y(t_0) = y_0, \quad y(t_f) = y_f, \quad t \in [t_0, t_f] \tag{2.7}$$

This problem 2.7 is similar to the problem 2.3. Within this problem, a boundary value problem (BVP) needs to be solved in each subinterval, and the matching conditions at the boundary of each subinterval need to be enforced to ensure a continuous and valid solution for the entire interval.

The examples discussed above do not consider control variables and only require an initial guess $\hat{y}_j$ for each subinterval. However, in optimal control problems, the multiple shooting method requires the introduction of piecewise local support functions to address control variables. In addition, separate initial guesses for the state and control variables need to be provided.

2.1.1 Mutiple shooting method for OCP

The multiple shooting method is a useful approach for solving optimal control problems of the form described in equation 2.2. This method can serve as the "first discretize" step of the direct approaches. When dealing with optimal control problems, both control variables and state variables need to be discretized. To simplify the process, we choose the same discretization grid points for both variables. Additionally, we introduce a 'local support' ansatz function for the control variables in each subinterval.

The first step in this approach remains the same, which is to choose the appropriate time grid to discretize the entire interval $[t_0, t_f]$ into m subintervals $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$, where the j-th subinterval is denoted as $\mathbb{I}_j = [\tau_j, \tau_{j+1}]$. In the second step, we define a local support function for the control variables in each subinterval, which can be denoted as $u(t)\mid_{t\in\mathbb{I}_j} = \psi_j(t; w_j), t \in \mathbb{I}_j$. The local support function can take the following form

1. piecewise constant controls

$$u(t)\mid_{t\in\mathbb{I}_j} = \psi_j(t; w_j) = w_j, t \in \mathbb{I}_j \tag{2.8}$$

2. piecewise linear controls

$$u(t)\mid_{t\in\mathbb{I}_j} = \psi(t; w_j) = w_j^l\left(\frac{\tau_{j+1} - t}{\tau_{j+1} - \tau_j}\right) + w_j^r\left(\frac{t - \tau_j}{\tau_{j+1} - \tau_j}\right), t \in \mathbb{I}_j \tag{2.9}$$

$$w_j = (w_j^l, w_j^r) \tag{2.10}$$

The introduced variable for the control support function in subinterval $\mathbb{I}_j$ is denoted as w_j. For piecewise constant controls, $u(t)$ takes a constant value of w_j within the subinterval $\mathbb{I}_j$. For piecewise linear controls, $u(t)$ takes the value from the linear interpolation between the left boundary w_j^l and right boundary w_j^r in the subinterval $\mathbb{I}_j$. To ensure continuity of the control variables, an additional equality constraint is enforced as follows

$$\psi_j(\tau_{j+1}; w_j) = \psi_{j+1}(\tau_{j+1}; w_{j+1}). \tag{2.11}$$

To solve the OCP described in equation 2.2, we begin by dividing the time horizon $[t_0, t_f]$ into a set of subintervals with common grid points for both the control and state variables. These subintervals are defined by the points $t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f$. We choose the support function for the control variables to be constant over each subinterval, i.e. $\psi_j(t; w_j) = w_j$ for $t \in \mathbb{I}_j$. We do this for two reasons: first, it is easy to implement; and second, we can obtain accurate numerical solutions quickly for our chosen case study. Since the control variables are constant over each subinterval, we need to specify an initial guess for both x and u in each subinterval. We denote these initial guesses as $x(\tau_j) = s_j$ and $u_j(t) = w_j$ for $t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$.

We can the follow the steps

1. Choose multiple shooting nodes $t_0 = \tau_0 < \tau_1 < \cdots < \tau_m = t_f$.

2. Choose initial guesses s_j and w_j for each subinterval $\mathbb{I}_j$, with $x(\tau_j) = s_j$, $j = 0, 1, \ldots, m-1$ and $u_j(t) = w_j$, $t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$, $j = 0, 1, \ldots, m-1$.

3. Solve the dynamic system using a numerical method such as Runge-Kutta 4 (see appendix A) to obtain a solution $x(t; s_j, w_j)$, $t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$.

4. Check if the solution is feasible, i.e., satisfies the constraints in each subinterval. If the solution is feasible, compute a cost of the error for the subinterval $\mathbb{I}_j$, which is denoted as F_j and defined as:

$$\begin{aligned} F_j &= \int_{\tau_j}^{\tau j+1} L(x(t; s_j, w_j), u(t; w_j))dt \quad j = 0, 1, ..., m-2 \\ F_j &= \int_{\tau_j}^{\tau j+1} L(x(t; s_j, w_j), u(t; w_j)) + E(x(t_f)) \quad j = m-1 \end{aligned} \tag{2.12}$$

where L is the running cost function, E is the terminal cost function, and u and x are the control and state variables, respectively. The term $\lambda \cdot \|x(\tau_{j+1}; s_j, w_j) - \psi(\tau_{j+1})\|^2$ is the cost of the error at the bounary of each subinterval, and serves as a penalty term for violatiting the matching condition, and can be added to the cost function F_j. In fact, if the solution is not feasible in a subinterval, a penalty term for each constraint violation in that subinterval should be added to the cost function F_j.

5. Compute the cost function value, which is the sum of F_j over all subintervals:

$$F = \sum_{j=0}^{m-1} F_j$$

If the cost function value is below some tolerance level, stop. Otherwise, update the guesses s_j and w_j and go back to Step 3.

In Step 3, it is possible to obtain an infeasible solution, which means that the computed solution obtained by solving the dynamic system equation 2.2b does not satisfy the constraints 2.2c, 2.2d, 2.2f and 2.2g. To deal with this situation, a penalty term is typically added to the cost function F_j for each constraint violation within that subinterval. Besides the penalty term for the violation of the matching condition, penalty terms can be defined in other forms to handle other constraints. For example, suppose we have a constraint that $g(x(t), u(t)) \leq 0$ for all $t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$, but our computed solution violates this constraint at some point t^*. We can add a penalty term to update the cost function F_j as follows:

$$F_j = F_j + \mu \cdot \sum_{t^*} \max\{0, g(x(t^*), u(t^*))\}$$

where μ (as well as λ for the matching condition) is a positive constant that controls the strength of the penalty. The max function ensures that the penalty term is only added when the constraint is violated. If the constraint is not violated, the penalty term is zero and has no effect on the cost function. This approach encourages the solver to find solutions that satisfy the constraints. There are different ways to define the penalty term, but the one mentioned is commonly used.

In Step 5, a sequential programming approach (a Newton type method, details in Section 2.4) can used as the the updating schema. Multiple shooting combined with Newton type methods can be used for finding the numerical solution of the OCP. In this approach, a sequential programming approach is used to directly solve a subproblem with constraints, without the need to define a penalty term. Within this approach, we can update s_j and w_j iteratively until feasible solutions are found and an 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 and 2.2g. In practice, we do not aim to find the optimal solution for each subinterval; instead, we aim to find the optimal solution for the whole interval, i.e., $min\ \sum_{j=1}^{m} F_j$. With the matching condition added, the original OCP is transferred into NLP in the following form

$$\min_{w} \sum_{j=1}^{m} F_j \tag{2.13a}$$

$$s.t.\ \ x(\tau_{j+1}; s_j, w_j) - s_{j+1} = 0, \quad j = 0, 1, ..., m-1 \tag{2.13b}$$

$$x(\tau_j) = s_j,\ \ u(t) = w_j, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, ..., m-1 \tag{2.13c}$$

$$g(x(t); s_j, w_j) = 0 \textit{ or } \leq 0, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, ..., m-1 \tag{2.13d}$$

$$h(x(t), w_j; s_j, w_j) = 0 \textit{ or } \leq 0, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, ..., m-1 \tag{2.13e}$$

$$x(t_0) = s_0, \quad (\textit{initial value}) \tag{2.13f}$$

$$r(x(t_f)) \leq 0, \quad (\textit{terminal constraints}) \tag{2.13g}$$

$$u^{lower} \leq w_j \leq u^{upper}, j = 0, 1, ..., m-1 \tag{2.13h}$$

$$t_0 = \tau_0 < \tau_1 < ... < \tau_m = t_f \tag{2.13i}$$

The solution $x(t; s_j, w_j)$ comes from solving the dynamic system 2.2b numerically for the subinterval $\mathbb{I}_j = [\tau_j, \tau_{j+1}]$, and F_j is defined as in equation 2.12.

2.2 KKT condition

As mentioned in Chapter 1, direct approaches are used to convert the original infinite optimal control problem into a finite-dimensional nonlinear programming problem (NLP). In the previous section 2.1.1, it was demonstrated that an OCP of the form 2.2 can be transformed into an NLP of the form 2.13. Variants of numerical optimization methods are then used to solve this NLP. The KKT condition serves as the necessary optimality condition for NLP, and it motivates the methods presented in this paper for solving the NLP. Before discussing the KKT condition, several definitions and theorems need to be introduced. We will consider a general NLP in the following form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \\ & h(x) \leq 0 \end{aligned} \tag{2.14}$$

where $f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^{n_g}$ and $h : \mathbb{R}^n \to \mathbb{R}^{n_h}$ are assumed to be twice continuously differentiable.

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

$$\Omega := \{x \in \mathbb{R}^n \mid g(x) = 0,\ h(x) \leq 0\} \tag{2.15}$$

Definition 2 *(Global minimizer) In the optimization problem given by equation 2.14, a global minimizer is a point $x^\star$ in the feasible set Ω, such that $f(x^\star) \leq f(x)$ for all x in the feasible set. In other words, a global minimizer is the point that achieves the minimum value of the objective function $f(x)$ over the entire feasible set.*

Definition 3 *(Local minimum and local minimizer) The point $x^\star \in \mathbb{R}^n$ is a local minimizer iff $x^\star \in \Omega$ and there exists a neighborhood $\mathbb{N}$ (e.g. an open ball around $x^\star$) so that $\forall\ x \in \Omega \cap \mathbb{N} : f(x) \geq f(x^\star)$. The value $f(x^\star)$ is a local minimum.*

Definition 4 *(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 equality could be considered to be always active.

Definition 5 *(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:

$$\tilde{g}(x) = \begin{pmatrix} g(x) \\ h_i(x), \ i \in \mathcal{A}(x^\star) \end{pmatrix} \tag{2.16}$$

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

Theorem 1 *(KKT Conditions) If $x^\star$ is a local minimizer of the problem 2.14 and the LICQ holds at $x^\star$, then there exists so called multiplier vectors $\lambda \in \mathbb{R}^{n_g}$ and $\mu \in \mathbb{R}^{n_h}$ with*

$$\nabla f(x^\star) + \nabla g(x^\star)\lambda^\star + \ \nabla h(x^\star)\mu^\star = 0 \tag{2.17a}$$

$$g(x^\star) = 0 \tag{2.17b}$$

$$h(x^\star) \leq 0 \tag{2.17c}$$

$$\mu^\star \geq 0 \tag{2.17d}$$

$$\mu_i^\star \ h_i(x^\star) = 0, \quad i = 1, ..., n_h \tag{2.17e}$$

The "KKT Conditions" are also known as "First Order Necessary Conditions (FONC)" for constrained optimization, and are thus the equivalent to $\nabla f(x^\star) = 0$ in unconstrained optimization.

Theorem 2 *If a NLP is a convex NLP and there exists a point $x^\star$ at which LICQ holds, then for this conver NLP*
$x^\star$ is a global minimizer $\iff \ \exists \ \lambda, \ \mu$ so that the KKT conditions hold.

A convex optimization problem is an optimization problem in which the objective function is a convex function and the feasible set is a convex set (ref Jorge Nocedal [2006] for more details).

Definition 6 *(KKT Point) We call a triple $(x^\star, \lambda^\star; \mu^\star)$ a "KKT point" if it satisfies LICQ (Definition 5) and the KKT conditions (Theorem 1).*

Definition 7 *Regard a KKT point $(x^\star, \lambda^\star; \mu^\star)$. For $i \in \mathcal{A}(x^\star)$, we say the constraint h_i is weakly active if $u_i^\star = 0$, otherwise if $u_i^\star > 0$, we call it strictly active.*

Definition 8 *(Lagrange Function) We define the "Lagrange function" as*

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^\top g(x) + \mu^\top h(x) \tag{2.18}$$

Here, we have used the "Lagrange multipliers" $\lambda \in \mathbb{R}^{n_g}$ and $\mu \in \mathbb{R}^{n_h}$. The last three KKT conditions 2.17c - 2.17e are called the complementarity conditions. If $h_i(x^\star) = 0$ and also $\mu_i^\star = 0$, this is a weakly active constraint, and often we want to exclude this case. On the other hand, an active constraint with $\mu_i^\star > 0$ is called strictly active.

Definition 9 *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 3 *(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 $\mathbb{Z} \in \mathbb{R}^{n*(n-n_{\tilde{g}})}$ 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 $\mathbb{Z}^\top \nabla_x^2 \mathcal{L}(x^\star, \lambda^\star, \mu^\star)\mathbb{Z} \succeq 0$. (Second Order Necessary Condition (SONC))
- If $\mathbb{Z}^\top \nabla_x^2 \mathcal{L}(x^\star, \lambda^\star, \mu^\star)\mathbb{Z} \succ 0$, 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 (SOSC)).

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

For an optimization problem, if we can start with an initial guess of x_0 and find an updating schema that decreases the objective value while maintaining the Hessian matrix positive, if such an updating schema can converge, it will converge to a local minimizer. If the original problem is convex, the local minimizer is, therefore, the global minimizer. Taking advantage of these properties, with the multiple shooting method applied, the OCP can be transferred into NLP, which can then be solved by the Newton type method.

2.3 Newton type methods

As explained in Section 2.2, KKT condition is the first order necessary optimal condition for constrained optimization, and if the Larange function is twice continuous differentiable and the Second Order Optimality Conditions hold, an updating schema can be applied to an initial guess to find a local minimizer (i.e., a local optimal solution). If such NLP is a convex problem, then the local minimizer is the global minimizer. The Newton and quasi Newton methods can be used as the updating method to find the (local) optimal solution. For the sake of simplicity, in this section, we first explain how the Newton and quasi Newton methods can be applied to problems without constraints. In the section that follows, we expand our explanation on how the Newton type method can be used to solve NLP with constraints.

2.3.1 Newton method

A general optimization problem typically takes the form of

$$\begin{aligned} min \quad & f(x) \\ s.t. \quad & x \in \Omega \end{aligned} \tag{2.19}$$

Here $x \in \Omega$ represents the constraints for which x must satisfy, it may be in the form of $\{g(x) = 0,\ h(x) \le 0\}$ as in the equation 2.14, i.e. the feasible set $\Omega = \underset{x}{arg}\ \{g(x) = 0,\ h(x) \le 0\}$. In this section, we only focus on the problems without constraints, i.e., the problems of the form

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.20}$$

The problem 2.20 can be solved via Newton's method, by constructing a sequence $\{x_k\}$ from an initial guess (starting point) x_0 that converges towards a minimizer $x^\star$ of $f(x)$, 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 notations 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 symbols ∇f and H directly. Therefore, the Talyor expansion can be written as

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

The next iteration, 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

$$\nabla f(x_k) + H(x_k)\delta(x_k) = 0 \tag{2.21}$$

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 converge to the local minimizer $x^\star$. The steps of the Newton method are as follows

Newton method

1. Choose an initial point x_0 and set $k = 0$
2. Compute the gradient $\nabla f(x_k)$ and the Hessian matrix $H(x_k)$
3. Solve the linear system $H(x_k)\delta(x_k) = -\nabla f(x_k)$ for the search direction $\delta(x_k)$
4. Choose a step size α_k by line search along the direction $\delta(x_k)$, such that $f(x_k + \alpha_k \delta(x_k)) < f(x_k)$

5. Update the current iterate: $x_{k+1} = x_k + \alpha_k \delta(x_k)$
6. Check whether the stopping criterion is satisfied. If the stopping criterion is satisfied, stop and output x_{k+1} as the solution. Otherwise, set $k = k + 1$ and go to step 2

Note that in Step 3, we compute $\delta(x_k)$ by solving the linear system with $H(x_k)$, and in Step 4, we choose a step size α_k that satisfies the Armijo condition along the direction of $\delta(x_k)$. The Armijo condition can be written as:

$$f(x_k + \alpha_k \delta(x_k)) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T \delta(x_k)$$

The Armijo parameter c_1 is a small constant (typically $c_1 = 10^{-4}$ or $c_1 = 10^{-5}$) that determines the amount of decrease in the objective function that is required to accept a step. Larger values of c_1 require a larger decrease in the objective function for the step to be accepted, making the method more conservative. Finally, in Step 5, we update the current iterate by taking a step in the direction of $\delta(x_k)$ with step size α_k. The step size α_k is a positive scalar that determines the size of the step taken in the descent direction. The Armijo condition ensures that the step size is chosen such that it results in sufficient decrease in the objective function.

In Step 6, the Newton method is said to have converged if either of the following conditions is satisfied as the stopping criterion:

1. The Newton step $\delta(x_k)$ is considered sufficiently small, i.e., $|\delta(x_k)|^2 \leq \epsilon$.
2. The norm of the change in the objective function value $|f(x_{k+1}) - f(x_k)|$ is sufficiently small, either $\frac{|f(x_{k+1}) - f(x_k)|}{|f(x_k)|} \leq \epsilon$ or $|f(x_{k+1}) - f(x_k)| \leq \epsilon$. Typically, the relative difference is preferred as it measures the change in the objective function relative to its current value, which is more informative and less dependent on the magnitude of the function.

where ϵ is a user-defined tolerance level. Note that these conditions are not mutually exclusive, and one can use a combination of both to check for convergence. If the stopping criterion is satisfied, the algorithm is terminated and the current iterate x_{k+1} is returned as the approximate solution.

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 local minimum, a saddle point or may not converge at all. In other words, because of the sensitivity of initialization, the Newton method may fail to find the local solution. Secondly, the Newton method can be computationally expensive (with respect to time), with the second-order derivatives, aka the Hessian matrix $H(\cdot)$ and its inverse being very expensive to compute. It may also happen that the Hessian matrix is not positive definite, therefore, the Newton method cannot be used at all for solving the optimization problem. Due to these limitations of the Newton method, the quasi Newton method is, therefore, usually preferred for solving optimal control or general optimization problems.

2.3.2 Quasi Newton method

We have stated that one limitation or the downside of the Newton method is that it 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 as 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 commonly 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 quasi Newton algorithms is similar in nature, with most of the differences arising in 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 the Newton method is that the approximation Hessian matrix B can be chosen in such a way that no matrix needs to be directly inverted. The Hessian approximation B is chosen to satisfy the equation 2.21, with the approximation matrix B replacing the original Hessian matrix H, i.e.

$$\nabla f(x_k) + B_k \delta(x_k) = 0 \tag{2.22}$$

In the text that follows, we explain how 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 need to impose additional constraints. One such constraint is the symmetry and positive-definiteness of B, which are to be preserved in each update for $k = 1, 2, 3, \ldots$. 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

$$\begin{aligned} &\underset{B_{k+1}}{min}\ \|B_{k+1} - B_k\| \\ &s.t.\ \ B_{k+1} = B_{k+1}^T,\ \ B_{k+1}\delta(x_k) = y_k \end{aligned} \tag{2.23}$$

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 2.23 directly is not trivial, but it has been proved (ref Jorge Nocedal [2006]) that problem 2.23 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

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

Given that B_k is positive definite, by enforcing the curvature condition $\delta(x_k)^Ty_k > 0$, then the update B_{k+1} will also be positive definite. Finally, we get the update formula as follows:

$$B_{k+1} = B_k + \frac{y_ky_k^T}{y_k^T\delta(x_k)} - \frac{B_k\delta(x_k)\delta(x_k)^TB_k}{\delta(x_k)^TB_k\delta(x_k)} \tag{2.24}$$

With the starting matrix B_1 initialized as positive definite, and the curvature condition ($\delta(x_k)^Ty_k > 0$) and closeness condition satisfied in each update step, then B_k will remain positive definite for all $k = 1, 2, 3, ...$ Instead of updating B_k directly, 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 be 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

$$B_{k+1}^{-1} = (I - \frac{\delta(x_k)y_k^T}{y_k^T\delta(x_k)})B_k^{-1}(I - \frac{y_k\delta(x_k)^T}{y_k^T\delta(x_k)}) + \frac{\delta(x_k)\delta(x_k)^T}{y_k^T\delta(x_k)} \tag{2.25}$$

As shown in the formula 2.25, 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 based on the true Hessian matrix $H(x_0)$, calculated with the starting point x_0. If necessary, we can adjust $H(x_0)$ slightly to ensure the starting matrix is positive definite.

We have shown that with the quasi Newton method, with an initial guess of x_0, we can iteratively update x_k until convergence, so that an optimal solution to the unconstrained optimization problem as in 2.20 can be obtained. In the next section, we explain how the quasi Netwon method can be applied to NLP with constraints, i.e. the problem 2.13.

2.4 Sequential quadratic programming

As explained in Section 2.1.1, the original OCP 2.2 can be transferred into a NLP 2.13. Such a NLP 2.13 can then be solved with the sequential quadratic programming (SQP) method.

For the NLP 2.13, the independent variables are the initial guess for x and u in each subinterval, denoted as $x(\tau_j) = s_j$ and $u_j(t) = w_j, t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$. The constraints in the NLP 2.13 are applied to the independent variables $s_j, w_j, j = 0, 1, ..., m-1$. Together with the matching condition 2.13b, the NPL 2.13 is in fact with the form as in the NLP 2.14, with the x in NLP 2.14 replaced by $(s_j, w_j), j = 0, 1, ..., m-1$.

SQP is one of the most successful methods for the numerical solution of constrained NLP. For the NLP 2.14, SQP is an iterative procedure that models the NLP for a given iterate $x_k, k = 1, 2, 3, ...$, by a quadratic programming (QP) subproblem, solves that QP subproblem, and then uses the solution to construct a new iterate x_{k+1}. This is done so that the sequence $x_k, k = 1, 2, 3, ...$ converges to a local minimum $x^\star$ of the NLP 2.14 as $k \to \infty$. In this sense, the NLP resembles the Newton and quasi Newton methods for the numerical solution of the NLP without constriants. However, the presence of constraints makes both the analysis and the implementation of SQP methods more complicated. A major advantage of SQP is that the iterates x_k need not to be feasible points, since the computation of feasible points in case of nonlinear constraint functions may be as difficult as the solution of the NLP itself.

To solve NLP of the form 2.14, we need to use the KKT conditions and introduce the Lagrange function as 2.18, which we re-rewrite here

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^\top g(x) + \ \mu^\top h(x) \tag{2.26}$$

The QP subproblems that have to be solved in each iteration step should reflect the local properties of the NLP with respect to the current iteration x_k. Therefore, a natural idea is to replace the

- objective functional f by its local quadratic approximation
  $f(x_k + \delta_x) \approx f(x_k) + \nabla f(x_k)^T \delta(x_k) + \frac{1}{2}\delta(x_k)^T H(x_k)\delta(x_k)$
- constraint functions g and h by their local affine approximations
  $g(x_k + \delta_x) \approx \ g(x_k) + \nabla g(x_k)\delta(x_k)$
  $h(x_k + \delta_x) \approx \ h(x_k) + \nabla h(x_k)\delta(x_k)$

where the definitions of $\nabla f(x_k), \delta(x_k), H(x_k)$ and the notation ∇ are given in Section 2.3.1. This leads to the following form of the QP subproblem

$$\min_{\delta(x_k)} \quad \frac{1}{2}\delta(x_k)^\top \nabla^2 \mathcal{L}(\cdot)\delta(x_k) + f(x_k)^T \delta(x_k) + f(x_k) \tag{2.27a}$$

$$\text{s.t.} \quad g(x_k) + \nabla g(x_k)^T \delta(x_k) = 0 \tag{2.27b}$$

$$h(x_k) + \nabla h(x_k)^T \delta(x_k) \leq 0 \tag{2.27c}$$

where $\nabla^2 \mathcal{L}(\cdot)$ is the Hessian matrix of function $\mathcal{L}(\cdot)$, whose updating can follow the quasi Newton method. Note that the term $f(x_k)$ in the expression above may be left out for the minimization problem, since it is constant under the $\min\limits_{\delta(x_k)}$ operator.

By solving such a subproblem, we can obtain an update for $x_{k+1}, \lambda_{k+1}, \mu_{k+1}$. We can iteratively solve the subproblems until convergence. The steps of the SQP algorithm for NLP with constraints can be summarized as follows.

1. Initialization: Choose an initial feasible point x_0 and set $k = 0$. Set the initial Hessian approximation B_0 and the convergence tolerance ϵ.

2. Solve quadratic subproblem: Solve the quadratic subproblem to obtain the search direction $\delta(x_k)$ by minimizing the quadratic model function $q_k(\delta(x_k))$, subject to the linearized constraints:

$$\min_{\delta(x_k)} q_k(\delta(x_k)) = f(x_k) + \nabla f(x_k)^T \delta(x_k) + \frac{1}{2}\delta(x_k)^T B_k \delta(x_k)$$

   subject to:

$$g(x_k) + \nabla g(x_k)^T \delta(x_k) = 0$$
$$h(x_k) + \nabla h(x_k)^T \delta(x_k) \leq 0$$

   This can be done using a QP solver. The term $f(x_k)$ can be left out.

3. Compute step size: Compute the step size α_k by backtracking line search, using the Armijo-Goldstein condition:

$$f(x_k + \alpha_k \delta(x_k)) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T \delta(x_k)$$

   where $0 < c_1 < 1$ is a constant.

4. Update solution, i.e. update the iterate as $x_{k+1} = x_k + \alpha_k \delta(x_k)$, and update the Lagrange multipliers as $\lambda_{k+1} = \lambda_k + \alpha_k \nabla g(x_k)$ and $\mu_{k+1} = \mu_k + \alpha_k \nabla h(x_k)$.

5. Check convergence: Check for convergence by computing the updating distance: If $\delta(x_k) < \epsilon$, then stop. Otherwise, go to Step 6.

6. Update Hessian approximation: Update the Hessian approximation by using a quasi-Newton update, following the updating equation 2.24 or 2.25. Then, set $k = k + 1$ and go back to Step 2.

The quadratic subproblem in Step 2 of the Sequential Quadratic Programming (SQP) method is typically solved using a quadratic programming (QP) solver. The QP solver is used to find the search direction $\delta(x_k)$ that minimizes the quadratic function $q_k(\delta(x_k))$ subject to the linear equality constraint $g(x_k) + \nabla g(x_k)^T \delta(x_k) = 0$ and the linear inequality constraint $h(x_k) + \nabla h(x_k)^T \delta(x_k) \leq 0$.

There are various QP solvers available that can be used to solve the quadratic subproblem. These solvers typically use a variety of algorithms such as interior-point methods, active-set methods, or augmented Lagrangian methods. The choice of QP solver may depend on factors such as the size of the problem, the structure of the constraints, and the desired accuracy of the solution.

We have shown that the NLP 2.13 can be solved using Newton-type methods, such as Sequential Quadratic Programming (SQP) approach with quasi-Newton as updating schema. Therefore, the method discussed in this Chapter can be used to solve OCP of the form 2.14. The numerical implementation details for a case study of a rocket car using this approach will be given in Chapter 4.

3 Optimal control under uncertainty

Besides the form 2.2, some OCP may have uncertain parameters whose value is a 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 parametric optimization problem, and the formulation 1.2 can be augmented with mathematical details as well, leading to an OCP under uncertainty of the following form

$$\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & F(x(\cdot),u(\cdot),p) = \int_{t_0}^{t_f} L(x(\cdot),u(\cdot),p)dt + E(x(t_f),p) && (3.1a)\\
s.t. \quad & \dot{x}(t) = f(x(t),u(t),p), \quad (system\ dynamics) && (3.1b)\\
& g(x(t),p) = 0\ or \leq 0,\ t \in [t_0,t_f]\ (path\ equality\ or\ inequality\ constraints) && (3.1c)\\
& h(x(t),u(t),p) = 0\ or \leq 0,\ t \in [t_0,t_f]\ (mixed\ control - state\ constraints) && (3.1d)\\
& x(t_0) = x_0, \quad (initial\ value) && (3.1e)\\
& r(x(t_f),p) \leq 0, \quad (terminal\ constraints) && (3.1f)\\
& u^{lower} \leq u(t) \leq u^{upper} && (3.1g)\\
& p \in \mathbb{P} && (3.1h)\\
& t \in [t_0,t_f] && (3.1i)
\end{aligned}$$

In other words, the state $x(t)$ depends not only on the system dynamics 3.1b and the control $u(t)$, but also on an uncertain parameter p. Parametric 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 parametric 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 parametric optimal control problem for our subsequent discussion, which we show here

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

In the paper Schlöder [2022], multiple methods of solving the parametric optimal control problem have been discussed. The main idea of solving the parametric optimal control problem 3.2 in a conservative way is to transform the problem into another form. Two different ways of solving the parametric optimal control problem will be discussed in detail 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 feasibility under constraints as well as the objective function value. The aim of robust optimization is to robustify or immunize a solution against uncertainty in terms of feasibility and optimality. In this paper, 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 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 dominant 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 the "classical approach." In this paper, we are particularly interested in the robustness of OCP under uncertainty. In our chosen case, the uncertainty appears in the form of an uncertain parameter that enters the differential equations.

For the original problem 3.2, the parameter p lies in an uncertainty set $\mathbb{P}$, we can firstly reach one objective, i.e., obtain one solution with respect to one particular p, say $p^{\star}$, i.e., solve a lower level problem. We continue solving many lower level problems with different p values, and after identifying these solutions, we identify the worst possible solution. Based on the results of the lower level, we can continue to find the best solution with respect to x, i.e., solving an upper level problem. "Worst-case treatment planning by bilevel optimal control" is a bilevel optimization problem, which is an optimization problem in which another optimization problem enters the constraints. Mathematically, the problem 3.2 is transformed into another form, as follows

$$\begin{aligned}\underset{x(\cdot),u(\cdot)}{min}\ \underset{p\in\mathbb{P}}{max}\ \ & F(x(\cdot),u(\cdot),p)\\ s.t.\ \ & \dot{x}(t)=f(x(t),u(t),p)\\ & x(t)\in\Omega\\ & u(t)\in\mathbb{U}\\ & p\in\mathbb{P}\\ & t\in[t_0,t_f]\end{aligned} \tag{3.4}$$

The set of feasible controllable variables is given by $u(\cdot)\in\mathbb{U}$ in the classical approach, which yields feasible trajectories $x(cdot)$ for $pinmathbbP$.The value of the objective function at the lower level does not depend on p and $x(\cdot)$. 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 some OCP, under conditions of uncertainty, it is not possible to find a unified control $u(\cdot)$ that has feasable solutions for all $p\in\ \mathbb{P}$. In this case, we may need to find the pair for $x(\cdot;p),u(\cdot;p)$ for different p, and among the pairs, we find the worst solution.

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 a theoretical solution can be derived. The original bilevel problem then turns into a single-level problem, which can be solved with the numerical methods discussed in Chapter 2. In paper Vasile [2014], a general algorithm has been proposed for solving the robust optimization or *minmax* problem. Basically, the idea is to break the bilevel problem into two parts and solve them collaboratively, one after another.

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

where $x^\star(\cdot), u^\star(\cdot)$ comes from the solution of

$$x^\star(\cdot), u^\star(\cdot) = \underset{x(\cdot), u(\cdot)}{arg\ min}\ \ \{F(x(\cdot), u(\cdot), p^\star)\ s.t.\ all\ constraints\} \tag{3.6}$$

Such an 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 of $F(\cdot)$ will be obtained given the realized parameter p. The paper Schlöder [2022] calls this approach "worst case modeling training approach", and it can be written as

$$\begin{aligned} &\max_{p_i \in \mathbb{P}} F_i \\ &\text{s.t. } \ F_i = \min_{x(\cdot), u(\cdot)} \ F(x(\cdot), u(\cdot), p_i) \\ &\qquad \dot{x}(t) = f(x(t), u(t), p_i) \\ &\qquad x(t) \in \Omega \\ &\qquad u(t) \in \mathbb{U} \\ &\qquad t \in [t_0, t_f] \end{aligned} \tag{3.7}$$

Due to the *maxmix* notation, this approach to solving the bilevel problem can also be called the *maxmin* approach. The method used in the paper Schlöder [2022] for the training approach is a gradient free method; more precisely, a model based derivative free optimization (DFO) approach for box-constrained optimization problems, i.e. the BOBYQA algorithm, is used. It can be used to solve problems of the form

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

The name BOBYQA is an acronym for "Bound Optimization By Quadratic Approximation", and is used to solve the 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 lower-level parametric OCP of the training approach 3.7 is solved with a direct DFO approach, and the resulting (finite dimensional) solution is viewed as a dependent variable. Furthermore, the uncertain 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 detail in the paper Powell [2009], and we reiterate the main idea in the text that follows.

The method of BOBYQA is iterative, with 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 being 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, and 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 $\|d_k\| \leq \Delta_k$ holds, $x = x_k + d_k$ 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, say y_t, 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, with 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 the BOBYQA algorithm require 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 $Q(\cdot)$.

To use BOBYQA algorithm to solve the given bilevel optimization problem 3.7 (i.e. the training approach), the following steps can be taken:

[1]Typically, a termination condition is satisfied when the objective value cannot be improved further after some iterations. If the termination condition of the BOBYQA algorithm, please refer to the paper Powell [2009] for more details.

BOBYQA algorithm for the training approach

1. Define the objective function for the upper level optimization problem as the maximum of the lower level objective function over the set of parameters p_i:

$$\max_{p_i \in \mathbb{P}} F_i = \max_{p_i \in \mathbb{P}} \min_{x(\cdot),u(\cdot)} F(x(\cdot), u(\cdot), p_i) \tag{3.14}$$

2. Choose an initial set of parameter values p_i
3. For each parameter value p_i, solve the lower level optimization problem by using BOBYQA algorithm to find the minimum of the objective function $F(x(\cdot), u(\cdot), p_i)$ over the variables $x(\cdot)$ and $u(\cdot)$. The BOBYQA algorithm requires specifying the lower and upper bounds for the variables $x(\cdot)$ and $u(\cdot)$
4. Evaluate the objective function F_i at the solution obtained in step 3 for each parameter value p_i
5. Update the parameter values by choosing the parameter value that gives the highest value of F_i as the next set of parameter values

$$p \leftarrow \arg\max_{p_i \in \mathbb{P}} F_i \tag{3.15}$$

6. Repeat steps 3-5 until convergence is achieved, that is, until the change in the objective function values F_i between iterations is below a specified tolerance level
7. The final set of parameter values obtained in step 6 gives the solution to the bilevel optimization problem

Note the BOBYQA algorithm has certain limitations and strong assumptions that need to be considered. Firstly, the algorithm assumes that the uncertainty set is of moderate size and box-shaped. Secondly, it assumes that there is only one local extremum, i.e., the lower level problem has a unique solution for each $p \in \mathbb{P}$. In practice, this assumption may not hold, and the algorithm may not converge to the local optimum. Although the BOBYQA algorithm is a gradient-free method with respect to the objective function $F(\cdot)$, it still relies on the gradient of the approximation function $Q(\cdot)$ while updating the iteration. This can result in numerical errors and computational costs while calculating the gradients of the approximation function $Q(\cdot)$ and updating them in each iteration.

This paper, instead, utilizes the gradient of the objective function $F(\cdot)$ directly, with some approximation applied as well. We have used the multiple shooting and Newton type 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 represent the whole set $\mathbb{P}$ when n is large enough. Then we take the *max* over all the lower level solutions (i.e., one T_i corresponding to one p_i).

We discuss in detail in the following Chapter 4 how the numerical methods discussed in Chapter 2 can be used to solve the bilevel problem in the classical and training approaches, using a case study (i.e., a state constrained rocket car) as an example.

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 problem is based on the paper Schlöder [2022]. The problem considers the one-dimensional movement of a mass point, representing the rocket car, under the influence of some constant acceleration/deceleration. The acceleration/deceleration is subject to uncertainty in the form of an unknown parameter $p \in \mathbb{P}$, with $\mathbb{P}$ being a convex and compact set. The goal is for the rocket car to reach a final feasible position and velocity in minimum time. The mass of the car is normalized to 1 unit, and the constant acceleration/deceleration enters the model. The problem is formulated as

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

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. $x(0) = 0$, indicates that at starting time $t = 0$, both the position and velocity of the car is 0. The condition 4.1e, i.e. $x_1(1) \geq 10$, indicates that the position of the car at $t = 1$ must be greater than or equal to 10. The condition 4.1f, i.e. $x_2(t) \leq 4$, indicates that the velocity of the car is always smaller or equal to 4 across the whole period. The condition 4.1g, i.e. $x_2(1) \leq 0$, 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 are the lower and upper boundary of the parameter p, respectively.

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

$$\dot{x} = \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = T \begin{pmatrix} x_2(t) \\ u(t) - p \end{pmatrix} = \begin{pmatrix} Tx_2(t) \\ T(u(t) - p) \end{pmatrix} \tag{4.4a}$$

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau}\frac{\partial \tau}{\partial t} \\ \frac{\partial x_2}{\partial \tau}\frac{\partial \tau}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau}T \\ \frac{\partial x_2}{\partial \tau}T \end{pmatrix} = \begin{pmatrix} Tx_2(t) \\ T(u(t) - p) \end{pmatrix} \tag{4.4b}$$

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

The equation $\frac{\partial x_1}{\partial \tau} = x_2(t)$ means the change in the position in real time is proportional to the velocity at that moment. The equation $\frac{\partial x_2}{\partial \tau} = u(t) - p$ means the change in velocity 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 selected the rocket car case for two primary reasons. Firstly, this case is easy to comprehend. Secondly, a theoretical solution exists, which is beneficial in evaluating the outcome of numerical methods. The optimization problem described in equation 4.1 has a singular global solution, and no additional local solution is present. The optimal solution based on the uncertain parameter p can be obtained by:

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

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

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

The optimal solution involves accelerating the car as strongly as possible until the velocity reaches a value of 4, at which point the car should maintain a constant velocity for a period of time. The constant velocity is maintained by canceling out the inherent deceleration caused by factors such as friction or headwind. Finally, the car should decelerate as strongly as possible until the velocity reaches zero, subject to the constraints that the velocity must be non-negative and the position of the car must be at least 10 units away from the starting point. The optimal solution is obtained by finding the smallest time T that satisfies these conditions.

The rocket car problem is an example of a typical bang-bang control problem, where the optimal solution is achieved by taking extreme values of acceleration, velocity, and deceleration whenever possible. A more formal and theoretical proof of the general bang-bang control problem requires a deep understanding of concepts such as calculus of variations, Hamilton-Jacobi-Bellman equation, and the Pontryagin Maximum Principle. The paper Konstantin Palagachev [2016] provides a proof and explanation of the solution to the bang-bang problem within a bilevel Optimal Control Problem. However, for readers who want to learn more about these theories, we recommend referring to papers such as McShane [1989], Gamkrelidze [1999], and Bertsekas [2005].

The theoretical solution to our rocket car problem, represented by equations 4.5 and 4.6, can be derived by solving the underlying partial differential equation 4.1c directly while considering the constraints. Appendix B of Schlöder [2022] provides the proof for this approach. Although the proof is detailed, the underlying idea is simple and straightforward. In this context, we explain the concept briefly and demonstrate with an example that the theoretical solution obtained through formulations 4.5 and 4.6 is accurate. For illustration purposes, we use a fixed parameter value of $p = 0$.

Since the aim is to minimize the time taken to reach the ending state from the starting state, the optimal strategy is to operate the rocket car at maximum allowable velocity as much as possible. The constraint 4.1f indicates that x_2 must equal 4 to achieve this velocity. If the rocket car runs at a velocity lower than the maximum allowable velocity, i.e., $x_2 = 4$, time is wasted. Given that the starting state has a position of 0 and a velocity of 0, the rocket car should be accelerated as strongly as possible at the beginning until the velocity reaches $x_2 = 4$, to reach the maximum allowable velocity as soon as possible. Once it reaches this velocity, the car should continue running at the maximum velocity for a specific duration. After this time, the car should be decelerated as strongly as possible until the velocity drops to 0, and the car's position should reach 10 exactly when the velocity reaches 0. This moment is when the optimal/smallest value of T is attained. The length of time for which the car should run at maximum velocity is determined by the acceleration/deceleration distance, which is dependent on the starting/ending position (0 and 10, respectively) and the starting/ending velocity (both are 0 in our case).

This is a qualitative explanation of why the solution from formulations 4.5 and 4.6 is correct. We will now demonstrate that, with $p = 0$, the optimal time is indeed $T = 2.9$. At the beginning, $\tau_0 = 0$, and we should accelerate as much as possible until the real time $\tau_1 = 0.4$ (i.e., $t = \frac{\tau}{T} = 0.137931$). We obtain $\tau_1 = 0.4$ 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 velocity reaches 4, and it maintains this velocity $x_2 = 4$ until $\tau_2 = 2.5$. From $\tau = 2.5$, the car starts to decelerate as strongly as possible, and at $T = \tau_3 = 2.9$, the velocity reaches 0, and the total distance traveled is exactly 10. Then all the constraints 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 proof by contradiction method. In the optimal strategy above, there are 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 from the one we have described above but leads to less time while satisfying all the constraints. First, we assume the alternative strategy does not accelerate as strongly as possible before τ_1. This means that, within this alternative strategy, the car's velocity is always less than or equal to the velocity 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. Therefore, it will take more time for the car to cover the remaining distance in the alternative strategy. This alternative strategy, therefore, cannot take less time compared to the optimal strategy. Similarly, we can prove that any modification to the optimal strategy will lead to a bigger T value. Therefore, using the example of $p = 0$, we have shown that 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 a proof can be found in paper Schlöder [2022].

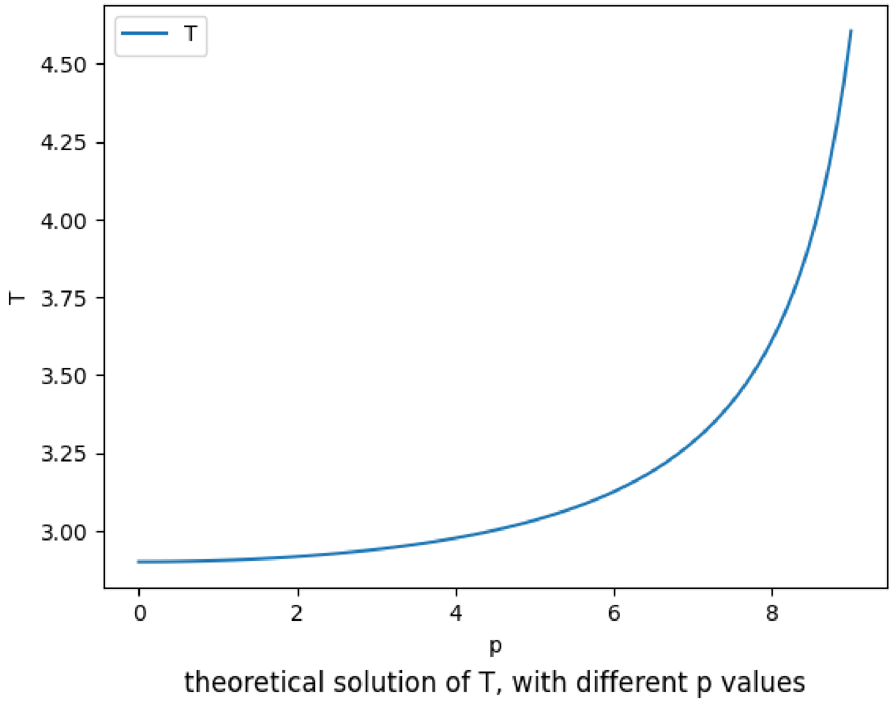

Figure 4.1: The theoretical optimal T values against p values in the uncertainty set, from the formulation 4.5

We can also verify the accuracy of the theoretical solution through numerical means. To start, we showcase the behavior of the theoretical solution for T as p varies over the entire uncertainty set $\mathbb{P} = [0, 9]$, as displayed in Figure 4.1. Next, we remove the uncertainty by presenting the solution to the original problem 4.1 for several selected p_i values, where $p_i \in [p_0, p_1, ..., p_n] \subset \mathbb{P} = [0, 9]$. To illustrate, we have chosen $p = 0$, $p = 5$, and $p = 9$ as examples. Figures 4.2 demonstrate the theoretical T values and corresponding $u(t)$ values for these three p values.

When p is fixed, the OCP under uncertainty reduces to a normal OCP problem, which can be solved directly with the numerical methods discussed in Chapter 2, without the need for bilevel optimization. We discretize $[0, 1]$ into 500 subintervals $\mathbb{I}j = [\tau j - 1, \tau_j]$, with the discretization points as $0 = t_0 < t_1 < \cdots < t_m = t_f = 1$. Using an error tolerance level of $1e - 6$ and the Runge-Kutta RK4 method, the results obtained from solving the normal OCP with $p = 0$, $p = 5$ and $p = 9$ are shown in Figure 4.3, Figure 4.4 and Figure 4.5 respectively.

These three figures 4.3, 4.4, and 4.5 demonstrate that the results obtained from $p = 0$,

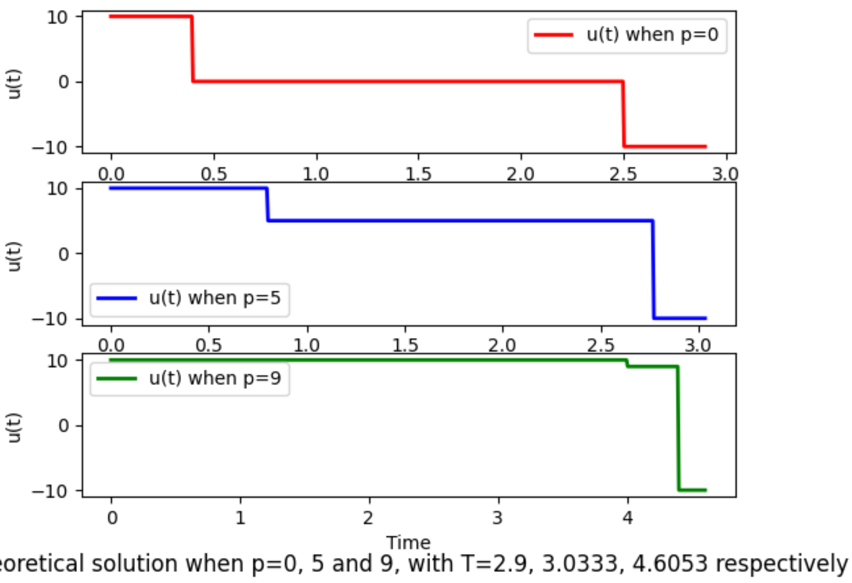

Figure 4.2: The theoretical u(t) values when p=0, p=5 and p=9, from the formulation 4.6

$p = 5$, and $p = 9$ are consistent with the theoretical results presented in formulations 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 selected can solve the rocket car problem for a fixed p value. These methods can, therefore, be applied to solve general OCPs.

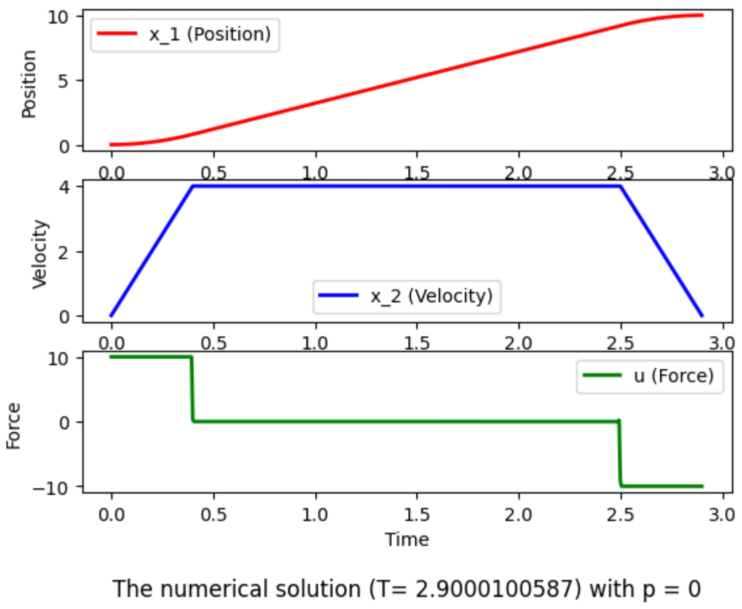

Figure 4.3: Orginal rocket car problem 4.1 solution when p=0

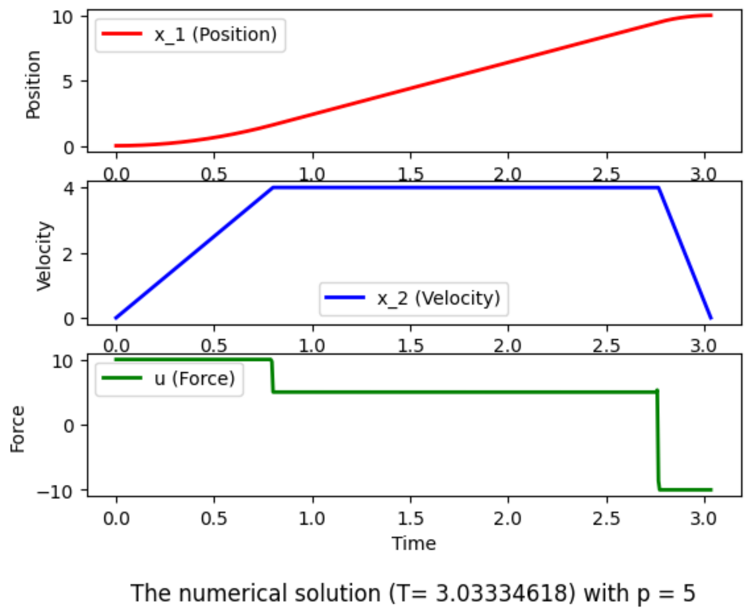

Figure 4.4: Orginal rocket car problem 4.1 solution when p=5

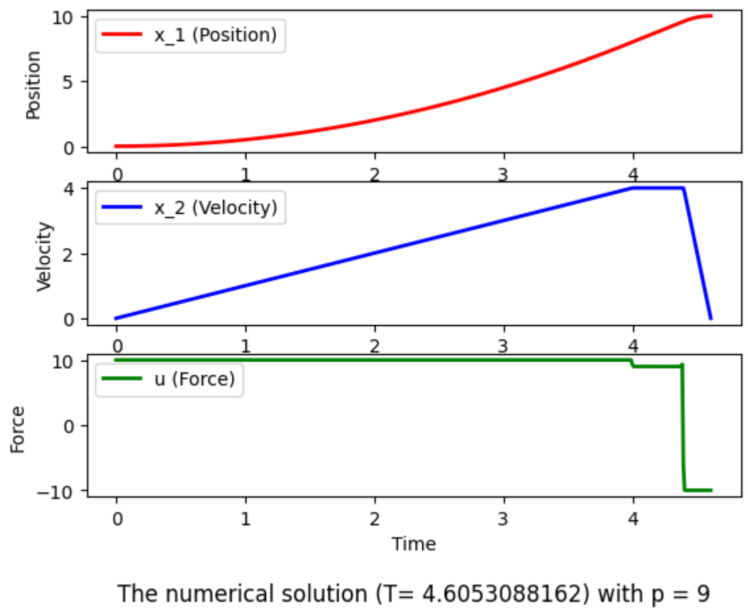

Figure 4.5: Orginal rocket car problem 4.1 solution when p=9

The rocket car case provides a simple problem for which we can obtain the theoretical solution to the original problem. However, in many real-life scenarios, finding a direct solution to the original problem is challenging and may not be feasible due to uncertainty in the parameter p. To address this challenge, the paper Schlöder [2022] proposes both classical and training approaches, which yield conservative solutions to the original problem. A conservative solution is desirable as it reduces risk and improves robustness.

After discretizing the uncertainty set, the problem becomes a normal optimal control problem that can be solved using the multiple shooting and quasi-Newton (in the SQP framework) method. We utilize the open-source packages Casadi[1] and Gekko[2] for solving the problem. These packages allow users to choose different nonlinear programming solvers and underlying numerical methods. For our problem, we have used the multiple shooting and sequential quadratic programming method with the underlying numerical integration using the Runge-Kutta RK4 method. We discretize $t \in [0,1]$ into 500 subintervals $\mathbb{I}_j = [\tau_j, \tau_{j+1}]$, where $0 = t_0 < t_1 < ... < t_m = t_f = 1$, and set the error tolerance level to $1e-6$. The numerical results from the classical and training approaches will be presented in the next two sections.

4.3 Apply classical (minmax) approach

In Section 3.1, we explained the classical approach, and in this section, we present the numerical results of applying the classical approach to the chosen rocket car case.

First, we discretize the entire uncertainty set $\mathbb{P}$ into multiple points in increasing order $[p_0, p_1, \dots, p_n] \subset \mathbb{P}$ so that $p_i, i = 0, 1, \dots, n$ can approximate 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 in the following form

$$
\begin{aligned}
&\underset{\epsilon, u(\cdot), x(\cdot)}{min} \quad \underset{p_i \in \mathbb{P}}{max}\ \epsilon && && (4.7a)\\
&\qquad s.t. \quad T_i \leq \epsilon && && (4.7b)\\
&\qquad\qquad x = (x_1, x_2), && && (4.7c)\\
&\qquad\qquad \dot{x} = T_i \begin{pmatrix} x_2(t) \\ u(t) - p_i \end{pmatrix}, && \forall\ p_i,\ t \in [0,1], && (4.7d)\\
&\qquad\qquad x(0) = 0, && && (4.7e)\\
&\qquad\qquad x_1(1) \geq 10 && && (4.7f)\\
&\qquad\qquad x_2(t) \leq 4,\ t \in [0,1] && && (4.7g)\\
&\qquad\qquad x_2(1) \leq 0, && && (4.7h)\\
&\qquad\qquad T \geq 0, && && (4.7i)\\
&\qquad\qquad u(t) \in [-10, 10],\ t \in [0,1]. && && (4.7j)
\end{aligned}
$$

for all p_i where $p_i \in [p_0, p_1, ..., p_n] \subset \mathbb{P}$. Both $u(\cdot)$ and $x(\cdot)$ are functions of t and need to be discretized in the time horizon (i.e., 500 subintervals). In this classical approach, the driver has no prior knowledge about the value of the parameter p and receives no feedback during the process, so they must set up the driving strategy in advance. Consequently, the realization of the trajectory of $u(\cdot)$ and $x(\cdot)$ for each p_i may not be an optimal solution compared to the case when knowing the value of p_i in advance.

[1]Refer to https://web.casadi.org for more details
[2]Refer to https://gekko.readthedocs.io/en/latest/ for more details

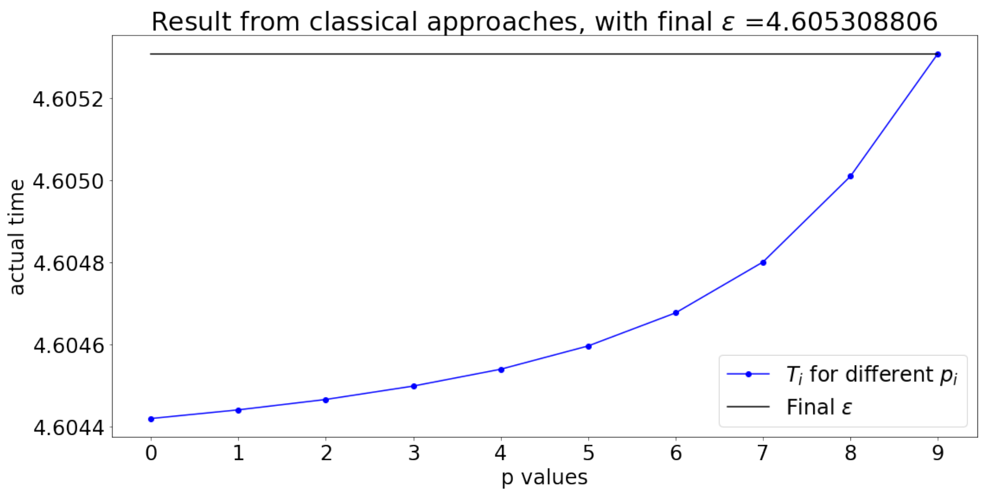

Figure 4.6: Final ϵ and T_i for different p_i values

For each p_i in the set $[p_0, p_1, ..., p_n] \subset \mathbb{P}$, we obtain a corresponding value of T_i, which we maximize over $T_i, i = 0, 1, ..., n$. We aim to minimize ϵ ($\epsilon = \underset{i}{max}\ T_i$). Ultimately, we obtain a worst-case ϵ value over all T_i for $i = 0, 1, ..., n$, and for each i, we get a trajectory $(p_i, x(t; p_i), (t; p_i))$.

In our implementation, we set $n = 40$, covering the entire uncertainty set $\mathbb{P} = [0, 9]$. However, for the sake of readability, we only show results for $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Figure 4.6 displays the final time ϵ and T_i, where $\epsilon = \max_i T_i$ and the maximum value is achieved when $p_i = 9$, with ϵ reaching a value of 4.6053. Differences in T_i mainly arise due to numerical errors, with our implementation using 500 subintervals and a numerical error tolerance of $1e - 6$. Increasing the number of subintervals and decreasing the tolerance level should result in T_i values for $p_i \neq 9$ converging to those of $p_i = 9$.

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 depicted in Figures 4.7, 4.8, and 4.9. In all three figures, the standardized time $t \in [0, 1]$ is used as the x-axis. Figure 4.7 shows the $u(t)$ acceleration/deceleration force for each p_i value, while Figure 4.8 displays the position $x_1(t)$ and Figure 4.9 displays the velocity $x_2(t)$ for each p_i.

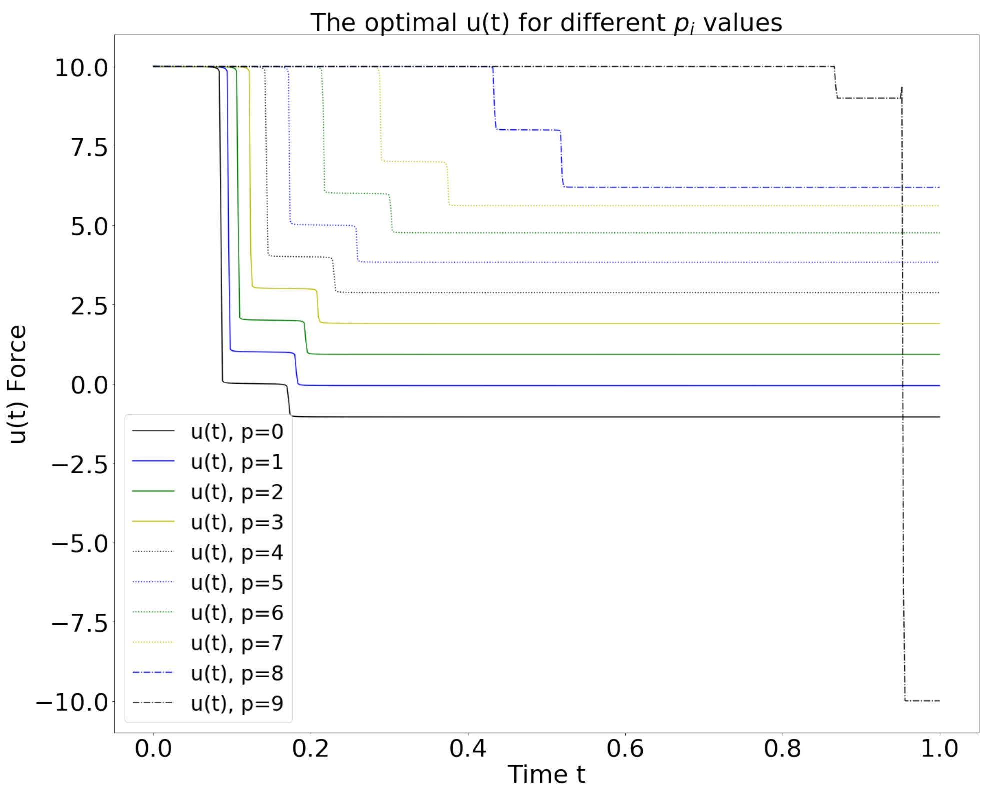

Figure 4.7: u(t) Force for different p_i values

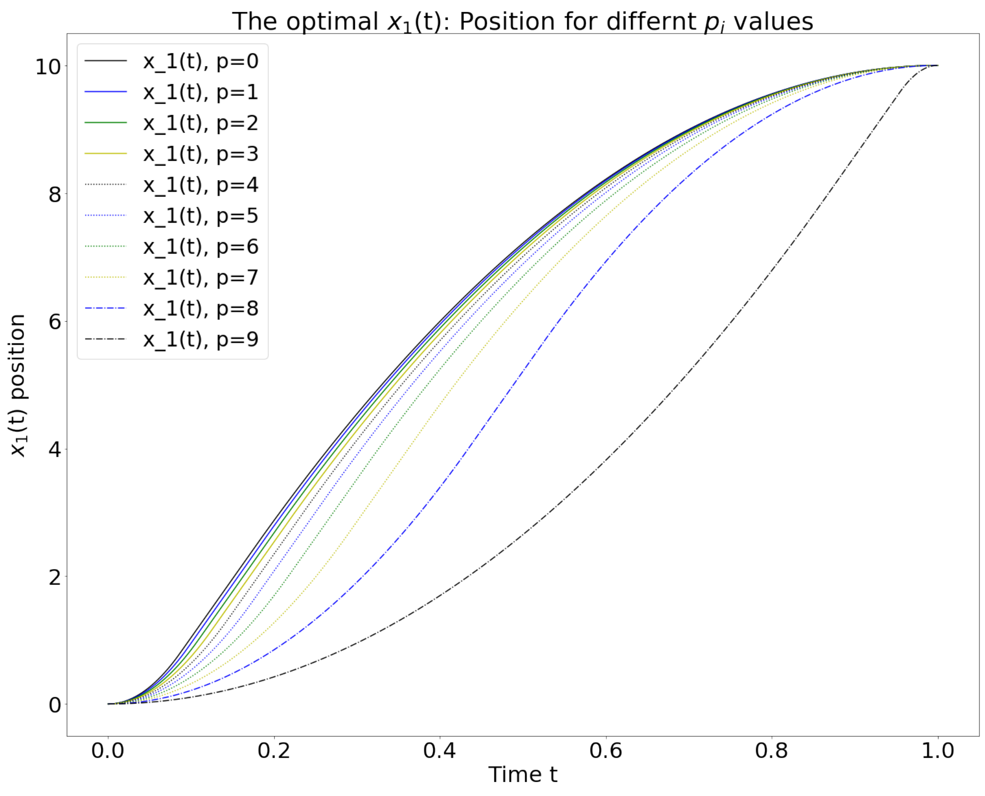

Figure 4.8: $x_1(t)$ Position for different p_i values

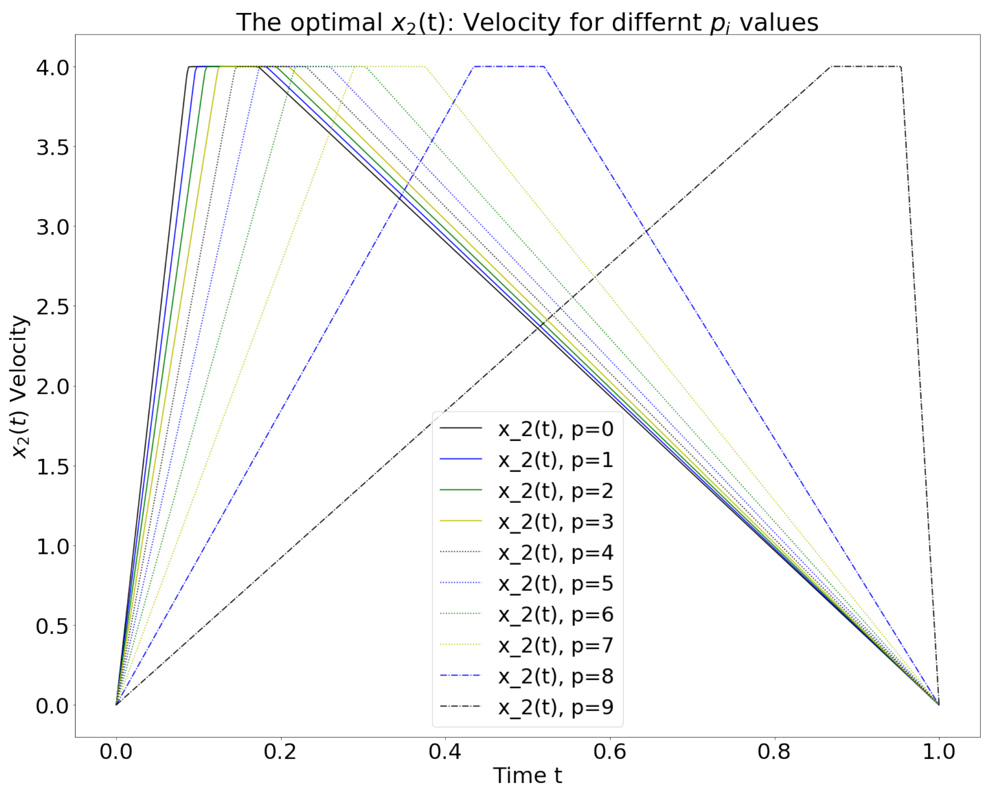

Figure 4.9: $x_2(t)$: Velocity for different p_i values

We have previously discussed that trajectories for different values of p_i obtained using the classical approach may not be optimal compared to the solutions obtained by directly solving the OCP 4.1 with known values of p_i. In this section, we compare the trajectories obtained from the classical approach with the results of solving OCP 4.1 directly with $p_i = 0$, $p_i = 5$, and $p_i = 9$. Figures 4.3, 4.4, and 4.5 already show the results of solving OCP 4.1 directly with $p_i = 0$, $p_i = 5$, and $p_i = 9$.

In Figures 4.10, 4.11, and 4.12, we present a comparison between the solutions obtained using the classical approach and the solutions obtained by directly solving OCP 4.1 for $p_i = 0$, $p_i = 5$, and $p_i = 9$. From Figures 4.10 and 4.11, it is clear that the classical approach leads to more conservative solutions than directly solving OCP 4.1 for $p_i = 0$ and $p_i = 5$, respectively. However, when $p_i = 9$, as shown in Figure 4.12, the classical approach yields the worst-case scenario, and the corresponding $\epsilon = 4.6053$ is equal to the result of directly solving OCP 4.1 with $p_i = 9$. For any other value of $p_i \neq 9$, we can always find a feasible solution with a corresponding T_i, which will be less than or equal to 4.6053.

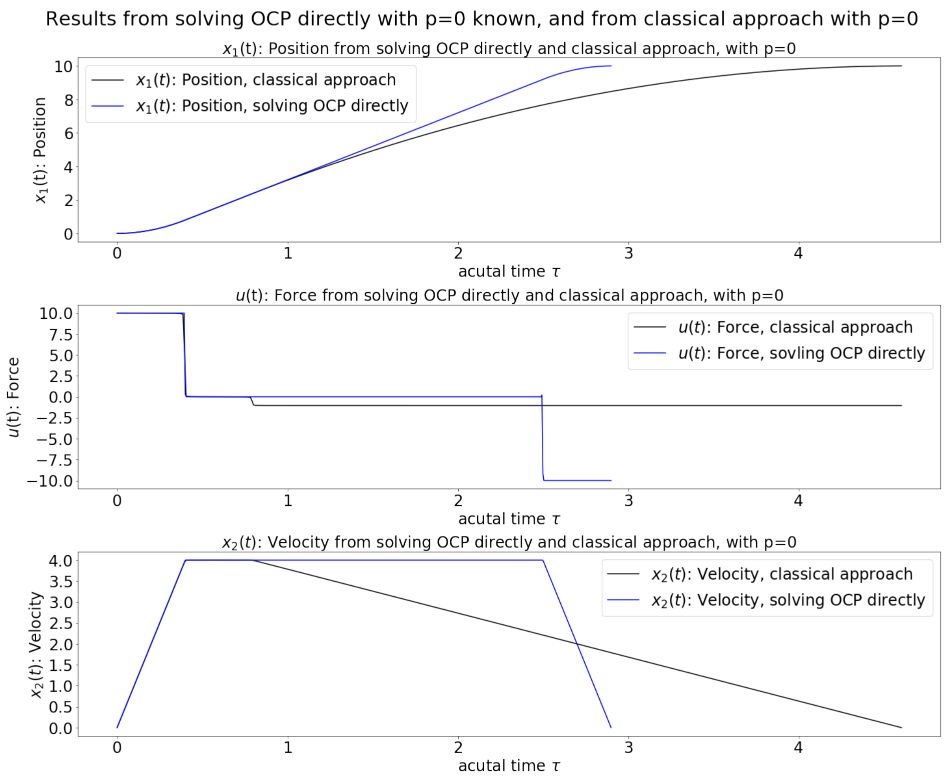

Figure 4.10: Compare the results from solving OCP 4.1 with $p = 0$ and the result of applying the classical approach to problem 4.7 with $p = 0$

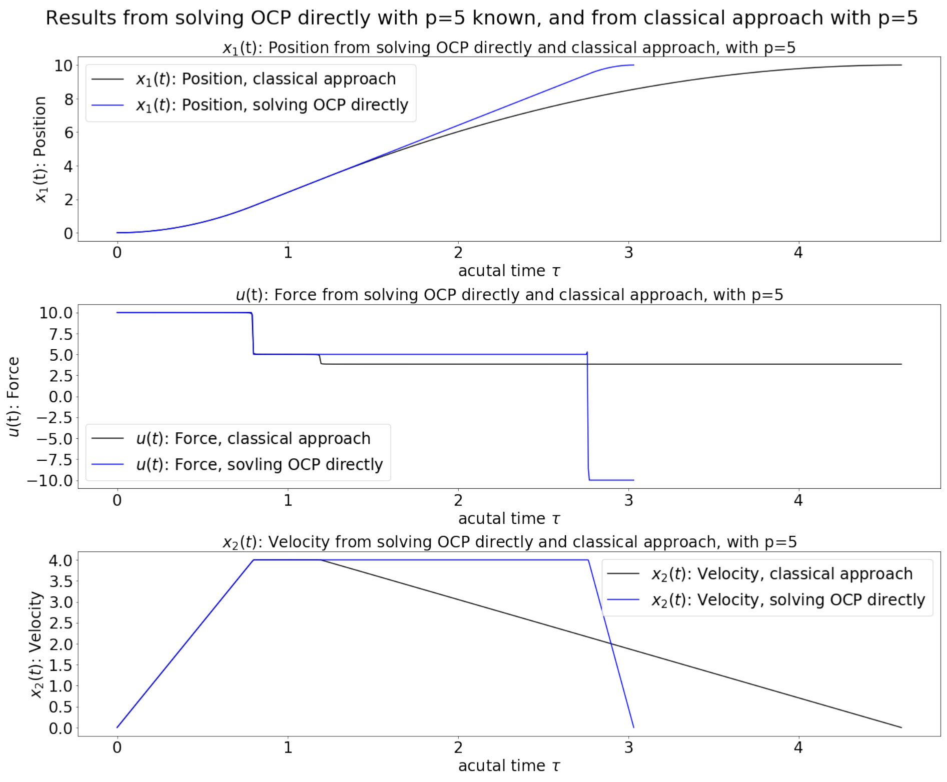

Figure 4.11: Compare the results from solving OCP 4.1 with $p = 5$ and the result of applying the classical approach to problem 4.7 with $p = 5$

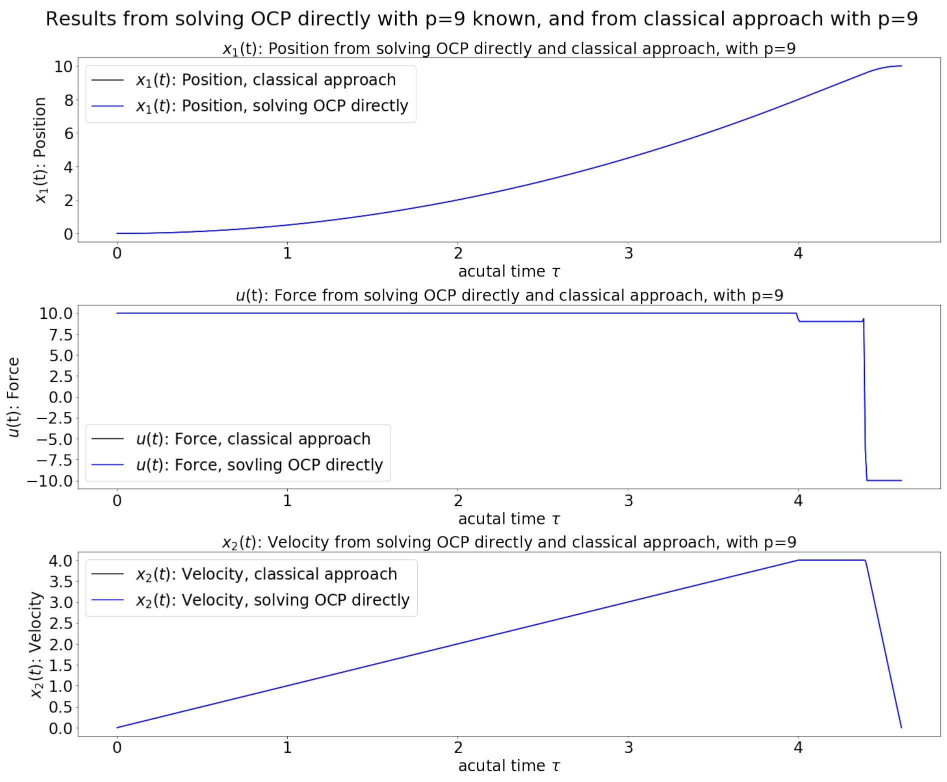

Figure 4.12: Compare the results from solving OCP 4.1 with $p = 9$ and the result of applying the classical approach to problem 4.7 with $p = 9$

4.4 Apply training (maxmin) approach

In contrast to the classical approach, the training approach assumes that the driver of the rocket car can perform optimally for any value of p, as a result of prior training. To discretize the uncertainty set $\mathbb{P}$, we use the same method as in the classical approach by selecting discrete points $p_0, p_1, ..., p_n$ that cover the entire uncertainty set $\mathbb{P}$. For each p_i in the interval $[p_0, p_1, ..., p_n] \subset \mathbb{P}$, we solve the corresponding problem.

$$\begin{aligned}
&\max_{p_i \in \mathbb{P}} \min_{T,u(\cdot),x(\cdot)} \quad T && (4.8a)\\
&\qquad s.t. \quad x = (x_1, x_2), && (4.8b)\\
&\qquad\qquad \dot{x} = T \begin{pmatrix} x_2(t) \\ u(t) - p_i \end{pmatrix}, \quad t \in [0,1], && (4.8c)\\
&\qquad\qquad x(0) = 0, && (4.8d)\\
&\qquad\qquad x_1(1) \geq 10, && (4.8e)\\
&\qquad\qquad x_2(t) \leq 4, \quad t \in [0,1], && (4.8f)\\
&\qquad\qquad x_2(1) \leq 0, && (4.8g)\\
&\qquad\qquad T \geq 0, && (4.8h)\\
&\qquad\qquad u(t) \in [-10, 10], \quad t \in [0,1]. && (4.8i)
\end{aligned}$$

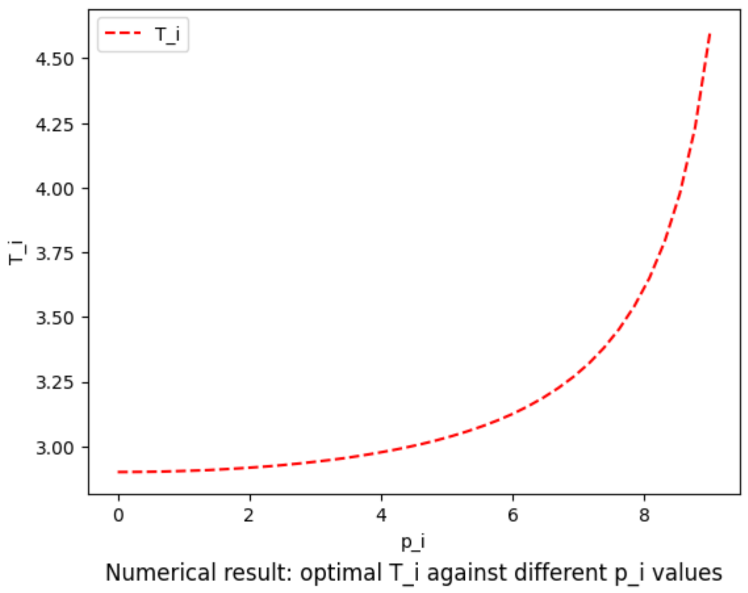

Figure 4.13: Numerical result: different T_i values against different p_i values, with $p_i = 9$ and $T_i = 4.6053$ the worst case

After setting a value for p_i, we can solve a normal optimal control problem (OCP). Each p_i value corresponds to one optimal T_i value. Our goal is to find the worst-case scenario among all the optimal T_i values, i.e., the largest among them. 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 resulting T_i values for each p_i are shown in Figure 4.13. This figure is almost identical to Figure 4.1, indicating that our numerical results with different p values are consistent with the theoretical results. The worst-case scenario occurs when p_i takes its maximum value of 9, and in this case, $T_i = 4.6053$. We have already shown the solution to the original problem for $p_i = 9$ in Figure 4.5. Therefore, the worst-case scenario has the same solution as shown in Figure 4.5 with $T = 4.6053$, which we repeat in Figure 4.14.

4.5 Analysis the numerical results

In addition to the numerical results discussed above, this section includes additional numerical analysis and a discussion of all the numerical results.

4.5.1 Sensitivity with initial value

In this section, we demonstrate the stability of our numerical implementation with respect to different initial values $u(t_0) \in [0, 10]$. As previously explained, the rocket car should accelerate as fast as possible at the beginning, i.e., $u(t_0)$ should take the maximum allowable value of 10 in order to obtain the optimal/smallest T. With different $u(t_0) \in [0, 10]$, the control variables $u(t)$ will move towards 10 during the acceleration stage, as shown in the first subplot of Figure 4.15. We have used only 100 subintervals for $[0, 1]$ in this plot. As demonstrated in the second subplot of Figure 4.15, when the initial value of $u(t_0)$ deviates from the ideal initial value $u(t_0) = 10$, it takes the longest time T, compared to other initial values. As the number of subintervals increases, the

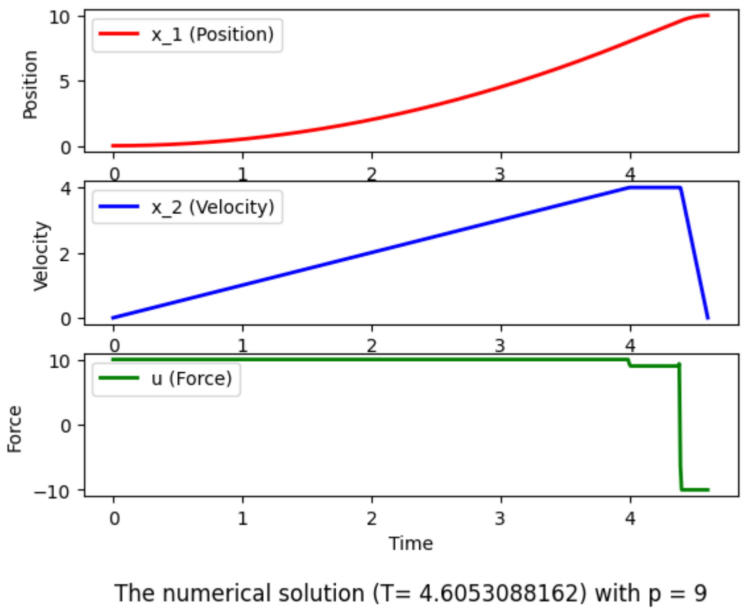

Figure 4.14: Training approach to problem 4.8, final result, worst case happens when $p = 9$

T associated with initial values other than $u(t_0) = 10$ converges to $T^{\star}(u(t_0) = 10)$. The results of this section confirm the background knowledge discussed in this paper and the theoretical solution.

4.5.2 Shared control $u(t)$ for classical approach

In the classical approach, a feasible solution $u(:, p), x(:, p)$ can be found for each p when solving the lower level part. However, if we force a unified $u(t)$ for all p in the uncertainty set, we cannot find a feasible $u(t)$ that satisfies all p values if the uncertainty set is too large. If we narrow down the uncertainty set, for example, by setting it to $\mathbb{P} = [0, 1]$, we can still find a feasible solution with shared control states $u(t)$ for all the parameters within the uncertainty set, as shown in Figure 4.16. The trajectories for both $x(t)$ and the shared $u(t)$ at the boundary parameters $p = 0$ and $p = 1$ satisfy the constraints, as shown in the subplots for "x_1 : Position" and "x_2 : Velocity". However, if we increase the uncertainty set, we fail to find a shared $u(t)$ that leads to feasible solutions for all p in $\mathbb{P} = [0, 9]$. This is consistent with the theoretical analysis discussed in the next section, Section 4.5.3, which shows that the classical approach will not yield a better solution than the training approach.

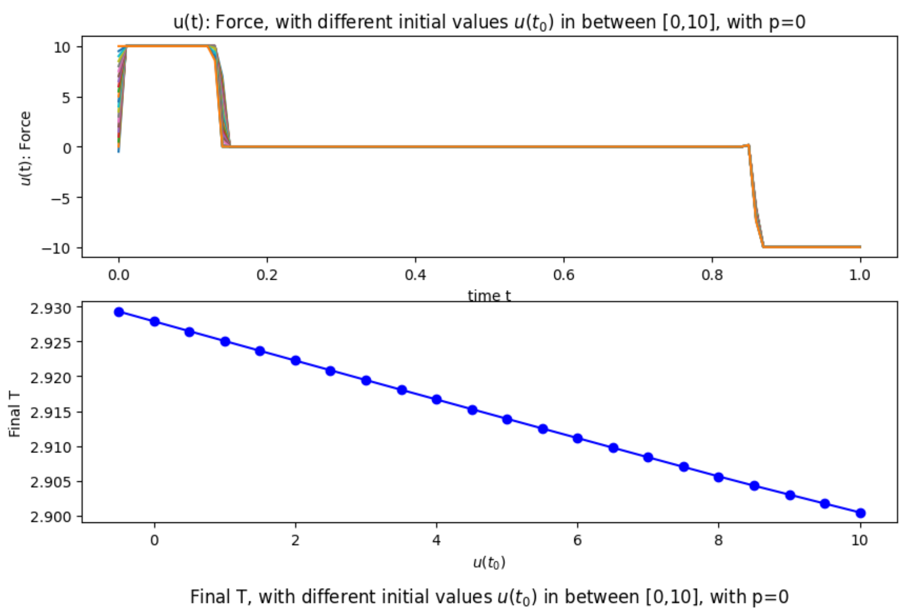

Figure 4.15: Results of solving OCP directly, with $p = 0$ and different initial values $u(t_0)$ in the set [0,10].

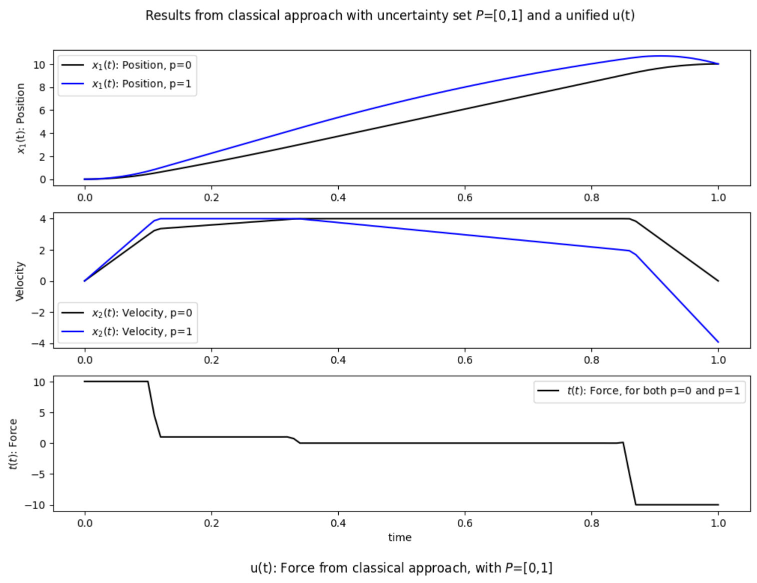

Figure 4.16: Results from classical approach with a unified $u(t)$ with uncertainty set $\mathbb{P} = [0, 1]$, with $T = 2.9495636746$ when $p = 0$, and $T = 3.9240282505$ when $p = 1$.

4.5.3 Comparison between classical approach and training approach

In theory, the classical approach is expected to result in a solution that is inferior to the training approach. This is because in the training approach, the value of p is known in advance, and the worst-case solution is obtained by maximizing over all $p \in \mathbb{P}$. However, in the classical approach, the value of p is a priori unknown, and we must find solutions that yield parameter-dependent variables for all possible realizations of the uncertain parameters. Then, among these solutions, we need to find the optimal one. We won't go into the proof here, but we refer to Schlöder [2022] for details. The conclusion of this theoretical result is shown directly in Theorem 4. In addition, we present an example to support this idea.

Theorem 4 *Let $\Omega_x \subset \mathbb{R}^{n_x}$ and $\Omega_p \subset \mathbb{R}^{n_p}$ be compact subsets and $f : \Omega_x * \Omega_p \ \to \mathbb{R}$ a continuous function. Then we have*

$$\underset{p\in\Omega_p}{max}\ \underset{x\in\Omega_x}{min}\ f(x,p) \leq \underset{x\in\Omega_x}{min}\ \ \underset{p\in\Omega_p}{max}\ f(x,p) \tag{4.9}$$

The optimal objective function value of the *max min* problem in the training approach overestimates that of the *min max* problem in the classical approach[3]. Examples can be found easily where the gap between the two is greater than zero. For instance, suppose we take $\Omega_x = [-5, 5]$ and $\Omega_p = [-1, 1]$, and consider the function

$$f : \Omega_x * \Omega_p \ \to \mathbb{R},\ (x,p) \to (x-p)^2 + p$$

Then

$$\underset{p\in\Omega_p}{max}\ \underset{x\in\Omega_x}{min}\ f(x,p) = 1 \leq \frac{5}{4} = \underset{x\in\Omega_x}{min}\ \ \underset{p\in\Omega_p}{max}\ f(x,p)$$

As shown above, both the classical and training approaches lead to an identical solution with a final time of $T = 4.6053$, which is conservative with respect to the original OCP 4.1. The time $T = 4.6053$ is conservative regardless of how the uncertain parameter p takes a 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 $T = 4.6053$. Nevertheless, the classical and training approaches do not necessarily lead to an identical solution for a general OCP under uncertainty. In general, the classical approach should lead to a worse solution compared to the training approach. In our case, it just happens that both approaches lead to the same solution, and the worst-case scenario occurs when $p = 9$ at the right boundary point.

5 Conclusion

In this paper, we present a theoretical explanation of how direct approaches can be employed to solve optimal control problems. Among direct approaches, we utilize the multiple shooting method to discretize the whole interval into subintervals. To simplify the process, we choose a constant value for the control variables $u_j(t)\ |_{\mathbb{I}j=[\tau_j,\tau j+1]}$ and an initial guess of $x(\tau_j)$ for each subinterval. This allows us to introduce new variables $(w_j, s_j), s_j, j = 0, 1, \ldots, m-1$ for each subinterval. We define w_j and s_j as $u_j(t) = w_j$ and $x(\tau_j) = s_j$, respectively, where $t \in \mathbb{I}j = [\tau_j, \tau j + 1]$ and $j = 0, 1, \ldots, m-1$. By doing so, we transform the original optimal control problem into a nonlinear programming problem with constraints, which can be solved using a Newton-type method.

[3]The proof that *max min* (training) approach over estimates that of *min max* approach is given in paper Schlöder [2022]

For an unconstrained optimization method, a Newton or quasi-Newton method can be directly used to iteratively obtain a local optimal solution until convergence. If the resulting nonlinear programming problem is convex, the local optimal solution is also the global optimal solution. In the case of a constrained optimization problem, we can define a Lagrange function that incorporates the constraints into the objective function. The KKT conditions are necessary conditions for optimality in a constrained optimization problem. By using the KKT conditions with the Lagrange function, we can solve the nonlinear programming problem in a Newton-type framework, specifically using the sequential quadratic programming method.

The sequential quadratic programming method is an iterative approach that utilizes a quadratic approximation of the Lagrange function to update the decision variables and Lagrange multipliers. At each iteration, a quadratic programming subproblem is solved to obtain the search direction. The search direction is then used to update the decision variables and Lagrange multipliers, and the process is repeated until convergence. By solving the derived nonlinear programming problem, we can obtain a solution to the original optimal control problem.

In real-life scenarios, some optimal control problems (OCPs) may involve uncertainty in the form of unknown parameters p that belong to the uncertainty set $\mathbb{P}$. Such problems can be particularly challenging to solve due to the presence of uncertainty. In such cases, we aim to obtain a conservative solution that is worse than the optimal solution for a specific p value, regardless of how the unknown parameter takes values within the uncertainty set $\mathbb{P}$. There are two approaches to obtain a conservative solution: the classical approach and the training approach.

This paper demonstrates the application of the numerical methods discussed above with a case study involving a state-constrained rocket car. When the unknown parameter p takes a fixed value in the uncertainty set $\mathbb{P}$, the OCP under uncertainty reduces to a normal OCP problem that can be solved using multiple shooting and a Newton-type method. The results are presented in figures such as Figure 4.3, 4.4, and 4.5. We have also performed a sensitivity analysis with respect to the initial value, as shown in Figure 4.15. The results confirm our analysis that the numerical methods used are stable and robust.

We have continued our numerical implementation with both the classical and training approach, which lead to a conservative solution. In our specific case, both approaches provide the same conservative solution. This result is not contradictory to our previous analysis, where we showed that the classical approach generally provides a worse solution compared to the training approach. Specifically, if we force the same control trajectory $u(t)$ for all parameters $p \in \mathbb{P}$, no feasible $u(t)$ can be found as explained in section 4.5.2. This indicates that the optimal T is infinity.

Based on these findings, we can conclude that our numerical results are consistent with our theoretical knowledge of the numerical methods discussed in this paper. Furthermore, we can apply these methods to more general optimal control problems, including those with uncertainty.

A Appendix Numerical Integration and Runge Kutta method

In this appendix, we demonstrate how to utilize numerical integration methods to solve a differential equation in the following format:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t_0) = x_0, \ t \in [t_0, t_f] \tag{A.1}$$

Equation A.1 represents a typical initial value problem (IVP), and its solution can be obtained using numerical methods. Analytical solutions for Equation A.1 may exist; however, for most real-life problems, finding an analytical solution is either impossible or extremely difficult. In such cases, numerical methods are the only practical way to find the solution. Given an initial value x_0, the solution to Equation A.1 is given by:

$$\begin{aligned} x(t) - x_0 &= \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau, \quad t \in [t_0, t_f] \\ x(t) &= x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau, \quad t \in [t_0, t_f] \end{aligned} \tag{A.2}$$

The integral in Equation A.2 can be approximated using numerical methods. One such method is the Euler method, also known as the forward Euler method, which 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 Euler method approximates Equation A.2 as follows

$$x(t) = x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau \ \approx \ x_0 + t f(x(t_0), u(t_0)) \tag{A.3}$$

The Euler method is an explicit and first-order method for approximating integrals. Another widely used method is the trapezoidal rule, which is an implicit second-order method. Equation A.2 can be approximated by the trapezoidal rule as shown in equation A.4.

$$x(t) = x_0 + \int_{t_0}^{t} f(x(\tau), u(\tau)) d\tau \ \approx \ x_0 + \frac{t}{2}[f(x(t_0), u(t_0)) + f(x(t), u(t))] \tag{A.4}$$

Note that the solution $x(t)$ becomes an input in the approximation in equation A.4, which is why this method is classified as an implicit method. The trapezoidal rule belongs to the family of Runge-Kutta methods, which are a family of implicit and explicit iterative methods with various orders of derivatives used. Among them, the most widely used is the Runge-Kutta RK4, which we describe in the following text. Let an initial value problem be specified as follows

$$\dot{y} = f(t, y), \quad y(t_0) = y_0 \tag{A.5}$$

We would like to obtain approximating an unknown function y of time t, where the rate of change of y is a function of t and y itself. The initial time is t_0, and the corresponding initial value of y is y_0. The function f, as well as the initial conditions t_0 and y_0, are given. To approximate the solution, a step size $h > 0$ is chosen, and a definition is made as follows

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

$$t_{n+1} = t_n + h \tag{A.7}$$

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

$$k_1 = f(t_n, y_n), \tag{A.8}$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \tag{A.9}$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \tag{A.10}$$

$$k_4 = f\left(t_n + h, y_n + hk_3\right). \tag{A.11}$$

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.

RK4 is a fourth-order method, which means that the error between the exact solution and the approximation is proportional to h^4, where h is the step size. As a result, RK4 is more accurate than lower-order methods such as Euler's method or the trapezoidal rule. However, RK4 is also more computationally expensive, as it requires four evaluations of the function f per time step.

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 Osborne. On shooting methods for boundary value problems. journal of mathematical analysis and applications. 1969.

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) . . . .