

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	4
2	Numerical methods	6
2.1	Multiple shooting	7
2.1.1	Mutiple shooting method for OCP	10
2.2	KKT condition	11
2.3	Quasi Newton method	13
2.3.1	Newton method	14
2.3.2	Quasi Newton method	15
2.4	Sequential quadrative programming	17
3	Optimal control under uncertainty	18
3.1	Classical approach	19
3.2	Training approach	20
4	Numerical solution	22
4.1	Introduction to the rocket car case	23
4.2	Theoretical solution to rocket car case	24
4.3	Apply classical (minmax) approach	29
4.4	Apply training (maxmin) approach	34
4.5	Analysis the numerical results	35
4.5.1	Sensitivity with initial value	35
4.5.2	Shared control $u(t)$ for classical approach	36
4.5.3	Comparison between classical approach and training approach] . .	37
5	Conclusion	38
A	Appendix Runge–Kutta method	39
	Bibliography	41

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:

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

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 are one of the most widespread and successfully used techniques. Direct approaches transform the original infinite optimal control problem into a finite-dimensional nonlinear programming problem (NLP). This NLP is then solved by variants of state-of-the-art numerical optimization methods, and the approach is therefore often sketched as "first discretize, then optimize."

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 the nonlinear programming problem by coupling the control parameterisation with a multiple shooting discretisation of the state variables. The whole interval $[t_0, t_f]$ is, therefore discretized into m subintervals, with $t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f$. In each subinterval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, the control can be approximated by piecewise functions and the state variables are approximated at the shooting nodes τ_i . The initial value $x(\tau_i)$ for the state variables at nodes τ_i must be guessed. Then in each subinterval the state equations must be integrated individually from τ_i to τ_{i+1} . In addition, the continuity conditions (matching conditions) must be satisfied, which require that on each differential node the values $x(\tau_{i+1})$ 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 Newton 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

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

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 $F^*(p)$, as defined in equation 1.3.

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

Parametric optimal control problems are very difficult to solve due to the uncertainty in the parameter p . Since the parameter p can take different values, so does the corresponding solution of $u(t)$, $x(t)$ and $F^*(p)$. 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 modelling 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 details, 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 in state constrained rocket car, with the description of the case and its numerical result given in Chapter 4.

The structure of this paper is as follows. Following this introduction Chapter 1, in next Chapter 2, we focus on 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

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

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, speed 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

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

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

$$g(x(t)) = 0 \text{ or } \leq 0, \quad t \in [t_0, t_f] \quad (\text{path equality or inequality constraints}) \quad (2.2c)$$

$$h(x(t), u(t)) = 0 \text{ or } \leq 0, \quad t \in [t_0, t_f] \quad (\text{mixed control - state constraints}) \quad (2.2d)$$

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

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

$$u^{lower} \leq u(t) \leq u^{upper} \quad (2.2g)$$

$$t \in [t_0, t_f] \quad (2.2h)$$

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

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

2.1 Multiple shooting

The multiple shooting method was initially introduced to solve the boundary value problem (BVP) in differential equation scope ([David Morrison \[1962\]](#) and [Osborne \[1969\]](#)). This method, with some adjustment, is therefore well suited for solving optimal control problems with constraints in differential equations. The text that follows will first explain the main idea how to use multiple shooting to solve BVP in the differential equation scope. Following that, we explain how multiple shooting can be used to solve a general optimal control problem and specifically how it can be used to solve an optimal control problem of the form 2.2.

To illustrate the concept of the shooting method to solve the boundary value problem

(BVP), we use the following example.

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

The analytical solution of above equation is

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

where e is the exponential number. For a given value b , the initial condition $x(t_0) = x_0$ will be determined to satisfy $x(t_f) = b$. As a result, we get the equation $x(t_f) - b = 0$ or $x(t_0)e^{t-t_0} - b = 0$. This derivation is called the shooting method. Generally, the shooting method can be summarized as follows

shooting method

- Step 1, choose an initial value $x(t_0) = x_0$
- Step 2, form a solution of the differential equation from t_0 to t_f
- Step 3, define a cost function for the error at the boundary of each subinterval and evaluate such cost function, and if the cost function value is below a tolerance level, then stop, otherwise continue to Step 4
- Step 4, update the guess for $x(t_0)$ based on some updating schema, go to Step 2

In Step 2, with initial value $x(t_0) = x_0$, numerical integration is used in practice to obtain the solution $x(t), t \in [t_0, t_f]$. The details of the numerical integration method is given in appendix A. In Step 3, the cost function can be defined as $(x(t_f) - b)^2$, and the tolerance level is set as a reasonable small number, e.g. $1e-8$. The tolerance level is set based on a balance of accuracy and computation cost (with respect to time).

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

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

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

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

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

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

The procedure of multiple shooting method can then be summarized as

multiple shooting method

- Step 1, choose multiple shooting nodes $t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f$
- Step 2, choose initial guess for $\hat{y}_k, k = 0, 1, \dots, m - 1$
- Step 3, form solutions of the differential equation in each subinterval $[\tau_k, \tau_{k+1}], k = 0, 1, \dots, m - 1$
- Step 4, define a cost function for the errors at the boundaries of each subinterval. If such cost function value is below a tolerance level, then stop otherwise continue to Step 5
- Step 5, update the guess for $\hat{y}_k, k = 0, 1, \dots, m - 1$ based on some updating schema, go to Step 3

In Step 4, the cost function can be defined as

$$\sum_{k=0}^{m-1} (y(\tau_{k+1}; \hat{y}_k) - \hat{y}_{k+1})^2 \quad (2.6)$$

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

We have summarized the main idea above on how the multiple shooting method can be used to solve a boundary value problem, yet the explanation falls short of mathematical rigor and completeness. In practice, there are many details to be decided when using (multiple) shooting methods. We will discuss these briefly without giving a comprehensive description. In Steps 1 & 2, when choosing the shooting nodes and the initial guess \hat{y}_k , how they are chosen usually depends on the nature of the problem as well as a balance between accuracy and computational cost. For example, the nodes can be equally spaced and the \hat{y}_k can be initialized with the same value, or they can be addressed based on our initial knowledge of the problem. In Step 3, polynomial functions can be used as the approximate solutions, leveraging the fact that Taylor expansion can be used to approximate any continuous function. In Step 4, the cost function is usually defined as the sum of quadratic errors, but it can take other forms as well. In Step 5, the "updating schema" can be defined so that the \hat{y}_k can move in a direction that decreases the cost function. The (quasi) Newton method, for example, can be used as an updating method in Step 5.

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

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

The problem 2.7 is similar to the problem 2.3. In each subinterval, a boundary value problem (BVP) is to be solved, and matching conditions at the boundary of each subinterval are to be enforced so that the final solution is continuous and applicable to the whole interval.

In the examples above, no control variables exist, so only an initial guess \hat{y}_k needs to be introduced for each subinterval. For optimal control problems, piecewise local support functions need to be introduced in the multiple shooting method to address the control variables in each subinterval.

2.1.1 Mutiple shooting method for OCP

The multiple shooting method is well suited for solving optimal control problems of the form as in the equation 2.2, as it can serve as the "first discretize" part of the direct approaches. For OCPs in the form of the equation 2.2, we follow the same idea of how the multiple shooting method can be used to solve boundary value problems. For OCPs, there are both control variables and state variables; therefore, we need to discretize both the control and state variables. For simplicity, we choose the same discretization grid points for the control variables and the state variables. For the control variables in each subinterval, we introduce a 'local support' ansatz function.

Therefore, the first step remains the same, i.e., choosing the suitable time grid to discretize the whole interval $[t_0, t_f]$ into m subintervals $t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f$, with the i -th subinterval as $\mathbb{I}_j = [\tau_j, \tau_{j+1}]$. In the second step, we define a local support function for the control variables. The local support function for the control variables in subinterval \mathbb{I}_j , which is denoted as $u(t) |_{t \in \mathbb{I}_j} = \psi_j(t; w_j), t \in \mathbb{I}_j$ can be of the following type

1. piecewise constant controls

$$u(t) |_{t \in \mathbb{I}_j} = \psi_j(t; w_j) = w_j, t \in \mathbb{I}_j \quad (2.8)$$

2. piecewise linear controls

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

$$w_j = (w_j^l, w_j^r) \quad (2.10)$$

where w_j is an introduced variable of the control support function for subinterval \mathbb{I}_j . For piecewise constant controls, $u(t)$ takes a constant value w_j in the subinterval \mathbb{I}_j . For the 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 . The continuity regarding the control variables can be enforced by additional equality

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

To solve the OCP as in equation 2.2, we first discretize the time horizon $\in [t_0, t_f]$ into multiple subintervals $t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f$, the time grid will be the same for the control and state variables. Then we choose the support function for the control variables as a constant function, i.e. $\psi_j(t; w_j) = w_j, t \in \mathbb{I}_j$. We choose a constant control value in each subinterval for two reasons: first, this is easy to implement; and second, we can still get the correct numerical solution fast for our chosen case. Since the control variable support function is chosen to be a constant value for each subinterval, we, therefore, need to introduce the initial guess for x and u in each subinterval, denoted as $x(\tau_j) = s_j$ and $u_j(t) = w_j, t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$. We can follow the steps

- Step 1, choose multiple shooting nodes $t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f$
- Step 2, choose the initial guess s_j and w_j for each subinterval \mathbb{I}_j , with $x(\tau_j) = s_j, j = 0, 1, \dots, m-1$ and $u_j(t) = w_j, t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 0, 1, \dots, m-1$

- Step 3, we solve the dynamic system using the Runge–Kutta 4 method (see the appendix A), and obtain a solution $x(t; s_j, w_j)$, $t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}]$
- Step 4, we check whether the solution is feasible (i.e., satisfying the constraints in each subinterval). If the solution is feasible. We compute a cost of the error for the subinterval \mathbb{I}_j , which is denoted as F_j . We define the cost function as the sum of F_j over all the subintervals. If the solution is not feasible, we add a penalty term to the cost function for each violation of the constraint.
- Step 5, we compute the cost function value, and if this value is below some tolerance level, stop; otherwise, we update the guesses $s_j, w_j, j = 0, 1, \dots, m-1$ and go to Step 3.

In Step 3, the result from solving the dynamic system equation 2.2b might be an infeasible solution, i.e., a solution that does not satisfy the constraints 2.2c, 2.2d, 2.2f and 2.2g. In this case, we can update s_j and w_j iteratively until feasible solutions are found and an optimal solution is reached. In Step 5, we can follow a Newton type method as the updating schema. 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

$$\min_w \sum_{j=1}^m F_j \quad (2.12a)$$

$$s.t. \quad x(\tau_j; s_j, w_j) - s_{j+1} = 0, \quad j = 0, 1, \dots, m-1 \quad (2.12b)$$

$$x(\tau_j) = s_j, \quad u(t) = w_j, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, \dots, m-1 \quad (2.12c)$$

$$g(x(t); s_j, w_j) = 0, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, \dots, m-1 \quad (2.12d)$$

$$h(x(t), w_j; s_j, w_j) \leq 0, \quad t \in \mathbb{I}_j = [\tau_j, \tau_{j+1}], j = 1, 2, \dots, m-1 \quad (2.12e)$$

$$x(t_0) = s_0, \quad (\text{initial value}) \quad (2.12f)$$

$$r(x(t_f)) \leq 0, \quad (\text{terminal constraints}) \quad (2.12g)$$

$$u^{lower} \leq w_j \leq u^{upper}, j = 0, 1, \dots, m-1 \quad (2.12h)$$

$$t_0 = \tau_0 < \tau_1 < \dots < \tau_m = t_f \quad (2.12i)$$

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 2.13

$$\begin{aligned} F_j &= \int_{\tau_j}^{\tau_{j+1}} L(x(t), u(t)) dt && \text{if } j = 0, 1, \dots, m-1 \\ F_j &= \int_{\tau_{j-1}}^{\tau_j} L(x(t), u(t)) dt + E(x(t_f)) && \text{if } j = m \text{ (the last subinterval)} \end{aligned} \quad (2.13)$$

2.2 KKT condition

As stated in Chapter 1, direct approaches transform the original infinite optimal control problem into a finite dimensional nonlinear programming problem (NLP). We have shown

in the previous section 2.1.1 that an OCP of the form 2.2 can be transferred into NLP of the form 2.12. This NLP is then solved by variants of state-of-the-art numerical optimization methods. The KKT condition is the necessary optimality condition for NLP and motivates the methods used to find the solution. Before we explain the KKT condition, we need to introduce several definitions and theorems. We consider a general NLP of the following form

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

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n_g}$ and $h : \mathbb{R}^n \rightarrow \mathbb{R}^{n_h}$ 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 (*Local minimum and local minimizer*) The point $w^* \in \mathbb{R}^n$ is a local minimizer iff $w^* \in \Omega$ and there exists a neighborhood \mathbb{N} (e.g. an open ball around w^*) so that $\forall w \in \Omega \cap \mathbb{N} : f(w) \geq f(w^*)$. The value $f(w^*)$ is a local minimum.

Definition 3 (*Active Constraints and Active Set*) An inequality constraint $h_i(x) \leq 0$ is called active at $x^* \in \Omega$ iff $h_i(x^*) = 0$ and otherwise inactive. The index set $\mathcal{A}(x^*) \subset \{1, \dots, n_h\}$ 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 4 (*LICQ*) The linear independence constraint qualification (LICQ) holds at $x^* \in \Omega$ iff all vectors $\nabla g_i(x^*)$ for $i \in \{1, \dots, n_g\}$ and $\nabla h_i(x^*)$ for $i \in \mathcal{A}(x^*)$ 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:

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

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

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

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

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

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

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

$$\mu_i^* h_i(x^*) = 0, i = 1, \dots, n_h \tag{2.17e}$$

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

Definition 5 (*KKT Point*) We call a triple $(x^*, \lambda^*; \mu^*)$ a "KKT Point" if it satisfies LICQ (Definition 4) and the KKT conditions (Theorem 1).

Definition 6 (*Lagrangian Function*) We define the "Lagrangian function" as

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

Here, we have used the so called "Lagrange multipliers" $\lambda \in \mathbb{R}_g^n$ and $\mu \in \mathbb{R}_h^n$. The last three KKT conditions 2.17c - 2.17e are called the complementarity conditions. This case is called a weakly active constraint. Often we want to exclude this case. On the other hand, an active constraint with $\mu_i^* > 0$ is called strictly active.

Definition 7 *Regard a KKT point $(x^*, \lambda^*; \mu^*)$. We say that strict complementarity holds at this KKT point iff all active constraints are strictly active.*

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

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

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

For an optimization problem, if we can start with an initial guess x_0 and find a updating schema which decreases the objective value while maintaining the Hessian matrix positive, if such updating schema can converge, it will converge to a local minimizer. If the original problem is convex, the local minimizer is, therefore, the global minimizer. Taking advantage of these properties, we can reformulate the NLP as in equation 2.14 as well as the optimal control problem as in equation 2.2, and then re-write in the Lagrangian form 2.18, optimize the (objective) Lagrangian function with quasi Newton method.

2.3 Quasi Newton method

As explained in Section 2.2, KKT condition can be applied to include the constraints into a Lagrangian function, and we can employ an updating schema with an initial guess to solve NLP as in equation 2.14 and optimal control problem as in equation 2.2. Newton and quasi Newton method can be used as the updating method to find the optimal solution. For the sake of simplicity, we focus on explaining the Newton and quasi Newton method for problems without constraints in this section, as we can apply KKT condition to incorporate constraints into a new objective function.

2.3.1 Newton method

A general optimization problem is typically of the form

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

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

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

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^* 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 notation f', f'' are usually expressed as ∇f and H (the Hessian matrix) respectively when x is a vector of variables. In the text that follows, we will use the symbol ∇f and H directly. Therefore, the Talyor expansion can be written as

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

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

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

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

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

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 and $f(x_k)$ converge to x^* and $f(x^*)$, respectively¹. The details of the Newton method is as follows:

¹In another word, the Newton method has converged when the change $\delta(x_k)$ is equal to zero or below a tolerance level, such that the change in the objective function is below a pre-defined tolerance level.

Newton method

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

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

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

2.3.2 Quasi Newton method

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

The first quasi Newton algorithm, i.e. the Davidon–Fletcher–Powell (DFP) method, was proposed by William C. Davidon in 1959 [Davidon \[1959\]](#), which was later popularized by Fletcher and Powell in 1963 [Fletcher R. \[1963\]](#). Some of the most common used quasi-Newton algorithms currently are the symmetric rank-one (SR1) method ([A. R. Conn \[1991\]](#)) and the Broyden–Fletcher–Goldfarb–Shanno(BFGS) method. The family of the quasi Newton algorithms are similar in nature, with most of the difference arising in the part how the approximation Hessian matrix is decided and the updating distance $\delta(x_k)$ is calculated. One of the main advantages of the quasi Newton methods over Newton method is that the approximation Hessian matrix B can be chosen in a way that no matrix needs to be directly inverted. The Hessian approximation B is chosen to satisfy the equation 2.21, 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 the iteration is performed in the BFGS method, as an example illustrating the quasi Newton method. In the BFGS method, instead of computing B_k afresh at every iteration, it has been proposed to update it in a simple manner to account for the curvature measured during the most recent step. To determine

an update scheme for B , we will need to impose additional constraints. One such constraint is the symmetry and positive-definiteness of B , which is to be preserved in each update for $k = 1, 2, 3, \dots$. Another desirable property is that B_{k+1} is sufficiently close to B_k at each update $k + 1$, and such closeness can be measured by the matrix norm, i.e. the quantity $\|B_{k+1} - B_k\|$. We can, therefore, formulate our problem during the $k + 1$ update as

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

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

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

Finally, we get the update formula as follows:

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

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

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

As shown in formula 2.24, 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 .

We have shown that with quasi Newton method, with an initial guess 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 Newton method can be applied to optimization problems with constraints, as well as the OCP problems of the form 2.2.

2.4 Sequential quadratic programming

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

As explained in section 2.1.1, the original OCP with the form 2.2 can be transferred into a NLP of the form 2.12. Within the NLP 2.12, the independent variables are the initial guess of x and u for each subinterval $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$ and the final control value u_m , i.e. $w = (x_0, u_0, x_1, u_1, \dots, x_{m-1}, u_{m-1}, u_m)$. The constraints in formulation 2.12 are applied to the independent variables $w = (x_0, u_0, x_1, u_1, \dots, x_{m-1}, u_{m-1}, u_m)$ piecewise, and they are in the form of equality constraints or inequality constraints. Together with the matching condition 2.12b, then the NLP problem 2.12 can be re-written as the form in NLP 2.14, with the x in NLP 2.14 replaced by $w = (x_0, u_0, x_1, u_1, \dots, x_{m-1}, u_{m-1}, u_m)$.

To solve NLP of the form 2.14, we can use the KKT approach, i.e. including the constraints and the original objective function into a new objective function, defined as a Lagrangian function as 2.18, which we re-write here

$$\mathcal{L}(w, \lambda, \mu) = f(w) + \lambda^\top g(w) + \mu^\top h(w) \quad (2.25)$$

Based on our discussion regarding Newton-style method, to find the minimum of $\mathcal{L}(w, \lambda, \mu)$, we are motivated to solve a linear system as in equation 2.21, where the first order derivatives and Hessian matrix is calculated for the objective function $\mathcal{L}(w, \lambda, \mu)$. Starting with an initial guess of (w_0, λ_0, μ_0) , then we are motivated to update (w, λ, μ) so as to decrease the objective function $\mathcal{L}(\cdot)$ by solving a subproblem of quadratic programming of the following form

$$\min_{\Delta w} \quad \frac{1}{2} \Delta w^\top \nabla_w^2 \mathcal{L}(\cdot) \Delta w + \nabla_w f(w_i)^\top \Delta w \quad (2.26a)$$

$$\text{s.t.} \quad g(w_i) + \nabla g(w_i)^\top \Delta w = 0 \quad (2.26b)$$

$$h(w_i) + \nabla h(w_i)^\top \Delta w \leq 0 \quad (2.26c)$$

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

2.26, we can therefore update the w, λ, μ as

$$w_{k+1} = w_k + \Delta w_k \quad (2.27a)$$

$$\lambda_{k+1} = \lambda_k^{QP} \quad (2.27b)$$

$$\mu_{k+1} = \mu_k^{QP} \quad (2.27c)$$

until convergence.

We have shown that the original OCP of the form 2.2 can be transferred into a NLP of the form 2.12 using multiple shooting method. The NLP 2.12 can then be re-written in the form 2.14, and can be solved using Netwon-style framework, i.e. sequential quadratic programming and quasi Netwon method. The numerical implementation details to a case study of rocket car will be given in Chapter 4.

3 Optimal control under uncertainty

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

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

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

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

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

$$x(t_0) = x_0, \quad (\text{initial value}) \quad (3.1e)$$

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

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

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

$$p \in \mathbb{P} \quad (3.1i)$$

$$x = x(, p^*) \text{ if } p = p^* \quad (3.1j)$$

$$t \in [t_0, t_f] \quad (3.1k)$$

In other words, the state variables x depend not only on the system dynamics 3.1b, the control variables u , but also 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 again

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

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 details in the following sections, i.e. the classical approach and the training approach.

3.1 Classical approach

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

$$\min_{x \in \mathbb{R}^n} \max_{p \in \mathbb{P}} f(x, p) \tag{3.3}$$

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

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

transformed into another form, as following

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

In the classical approach, the set of feasible controllable variables are given by $u(\cdot)$, which yield feasible trajectories $x(\cdot, p)$ for all $p \in \mathbb{P}$. The value of the objective function in the lower level does not depend on p and $x(\cdot, p)$. In other words, in this approach, the dynamic system has no prior knowledge about the value of the parameter p and gets no feedback during the process and has to decide the control strategy in advance. In some OCP under uncertainty, it is not possible to find a unified control $u(\cdot)$ which have feasible 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 theoretical solution can be derived, then the original bilevel problem turns into a single level problem, which can be solved with numerical methods discussed in Chapter 2. In paper [Vasile \[2014\]](#), a general algorithm have been proposed for solving robust optimization or *minmax* problem. Basically, the idea is to break the bilevel problem into two parts and iteratively solving these two parts one after another.

$$p^* = \arg \max_{p \in \mathbb{P}} F(x^*(t; p), u^*(t)) \tag{3.5}$$

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

$$x^*(t; p), u^*(t) = \arg \min_{x(\cdot, p^*), u(\cdot)} F(x(t; p^*), u(t)) \tag{3.6}$$

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

3.2 Training approach

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

Approach", and it can be written as

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

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

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

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

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

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

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

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

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

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 condition¹ is satisfied, then the iteration stops. Otherwise, a step d_k from x_k is constructed such that $\|d_k\| \leq \Delta_k$ holds,

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

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

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

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

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

at the new interpolation points

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

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

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

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

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

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

4 Numerical solution

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

4.1 Introduction to the rocket car case

The description of the rocket car case is mostly coming from the paper [Schlöder \[2022\]](#), with content either verbatim or in a modified form. We consider the rocket car case with state constraints, i.e. the one-dimensional movement of a mass point under the influence of some constant acceleration/deceleration, e.g. modeling head-wind or sliding friction, which can accelerate and decelerate in order to reach a desired position. The mass of the car is normalized to 1 unit¹ and the constant acceleration/deceleration enters the model, suffering from uncertainty in form of an unknown parameter $p \in \mathbb{P} \subset \mathbb{R}$, with the uncertainty set \mathbb{P} convex and compact. We consider a problem in which the rocket car shall reach a final feasible position and velocity in a minimum time:

$$\min_{T, u(\cdot), x(\cdot; p)} T \tag{4.1a}$$

$$s.t. \quad x = (x_1, x_2), \tag{4.1b}$$

$$\dot{x} = T \begin{pmatrix} x_2(t; p) \\ u(t) - p \end{pmatrix}, \quad t \in [0, 1], \tag{4.1c}$$

$$x(0, p) = 0, \tag{4.1d}$$

$$x_1(1; p) \geq 10, \tag{4.1e}$$

$$x_2(t; p) \leq 4, \quad t \in [0, 1], \tag{4.1f}$$

$$x_2(1; p) \leq 0, \tag{4.1g}$$

$$T \geq 0, \tag{4.1h}$$

$$u(t) \in [-10, 10], \quad t \in [0, 1]. \tag{4.1i}$$

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 $u : [0, 1] \rightarrow \mathbb{R}$ in equation 4.1 represents the acceleration/deceleration value. The condition 4.1d, i.e. $x(0, p) = 0$, indicates that at starting time $t = 0$, both the position and velocity of the car is 0. The condition 4.1e, i.e. $x_1(1; p) \geq 10$, indicates that the

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

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

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

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

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

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

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

$$\dot{x} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = T \begin{pmatrix} x_2(t; p) \\ u(t) - p \end{pmatrix} = \begin{pmatrix} T x_2(t; p) \\ T(u(t) - p) \end{pmatrix} \quad (4.4a)$$

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau} \frac{\partial \tau}{\partial t} \\ \frac{\partial x_2}{\partial \tau} \frac{\partial \tau}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial x_1}{\partial \tau} T \\ \frac{\partial x_2}{\partial \tau} T \end{pmatrix} = \begin{pmatrix} T x_2(t; p) \\ T(u(t) - p) \end{pmatrix} \quad (4.4b)$$

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

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

4.2 Theoretical solution to rocket car case

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

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

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

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

In words, we accelerate as strongly as possible (the acceleration value $u^*(t) = 10$) until the velocity $x_2^*(t; p) = 4$, and then keep the velocity $x_2^*(t; p)$ constant for a certain period of time², and eventually decelerate as strongly as possible until the velocity reaches the state $x_2(1; p) \leq 0$ and $x_1(1; p) \geq 10$. The first moment that these two conditions have been reached, is the moment that we find the optimal/smallest T .

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

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

Because the objective is to minimize the time between the starting state and ending state, then it is optimal to run at the maximum allowable speed whenever possible, i.e with $x_2 = 4$ as in the constraint 4.1f. Otherwise, time is wasted if a car is running at a speed that is less than the maximum allowable speed, i.e. $x_2 < 4$. Since the starting state is at position 0 and speed 0, to reach the maximum allowable speed as soon as possible, the car should be accelerated as strongly as possible at the beginning until the speed reaches $x_2 = 4$. Then the car should be running at this maximum speed for a certain period. After this period, the car should be decelerated as strongly as possible until the speed decreases to 0, and exactly at the moment that the speed reaches 0, the position should reach 10. It is this moment that the optimal/smallest T is achieved. How long the period should be running at maximum speed, depends on the acceleration/deceleration distance, which in turn depends on starting/ending position (in our case, it is 0 and 10 respectively) and starting/ending speed (in our case, both are 0).

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

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

assume there is another strategy that differs the one we have described above, yet leading to a less time while satisfying all the constraints. First, we assume the alternative strategy does not accelerate as strong as possible before τ_1 , this means within this alternative strategy, the speed of the car is always less or equal to the speed of our optimal strategy before $\tau_1 = 0.4$. This means that before $\tau_1 = 0.4$, the travel distance in the alternative strategy is less than our optimal strategy, and therefore, it will takes more time for car to cover the remaining distance in the alternative strategy. This alternative strategy, therefore, can not take less time compared to the optimal strategy. Similarly, we can prove that any modication to the optimal strategy will lead a bigger T value. Therefore, by using the example of $p = 0$, we have shown the solution 4.5 and 4.6 is indeed optimal with respect to different p for all the p in the uncertainty set $\mathbb{P} = [p_l, p_u] = [0, 9]$. The details of such proof can be found in paper [Schlöder \[2022\]](#).

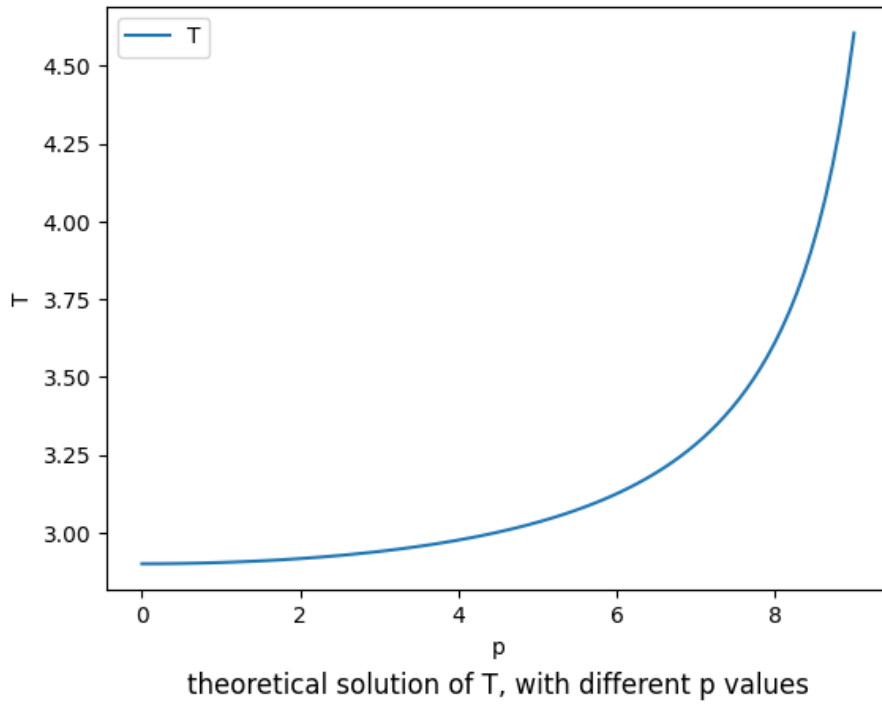


Figure 4.1: The theoretical optimal T values against p values in the uncertainty set, from the formulation 4.5

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

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

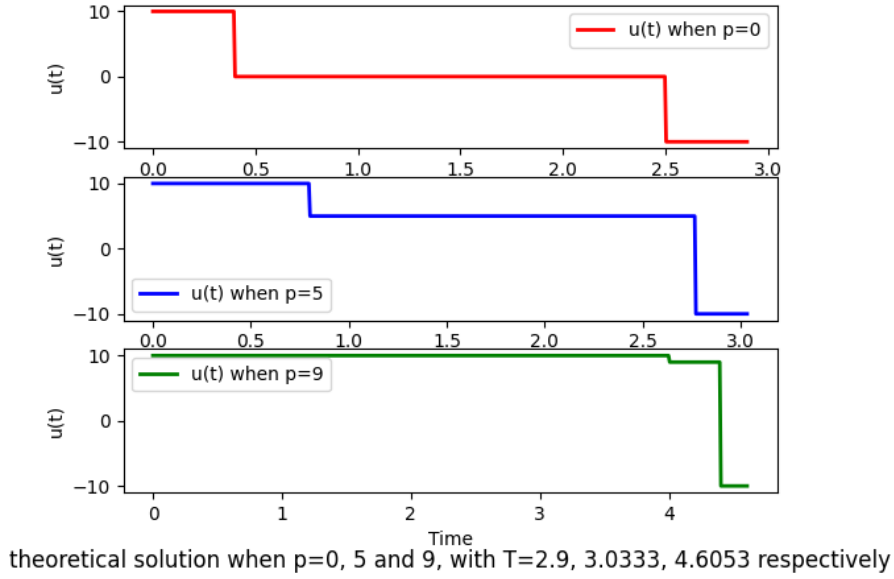
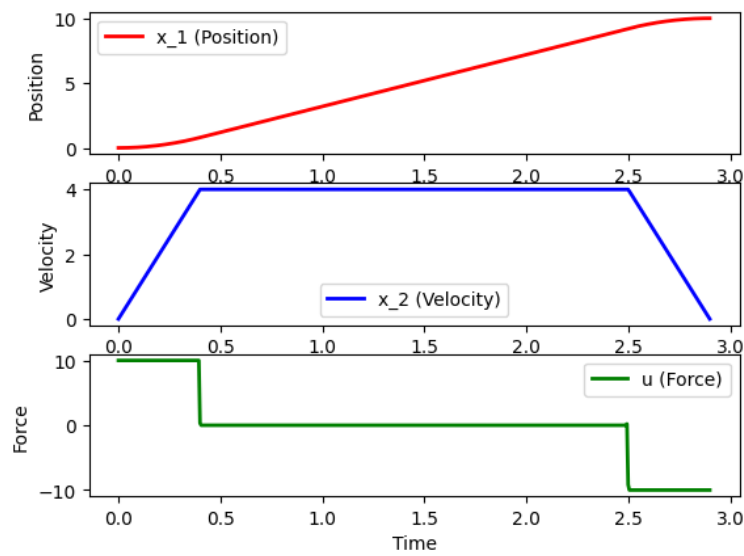


Figure 4.2: The theoretical $u(t)$ values when $p=0$, $p=5$ and $p=9$, from the formulation 4.6

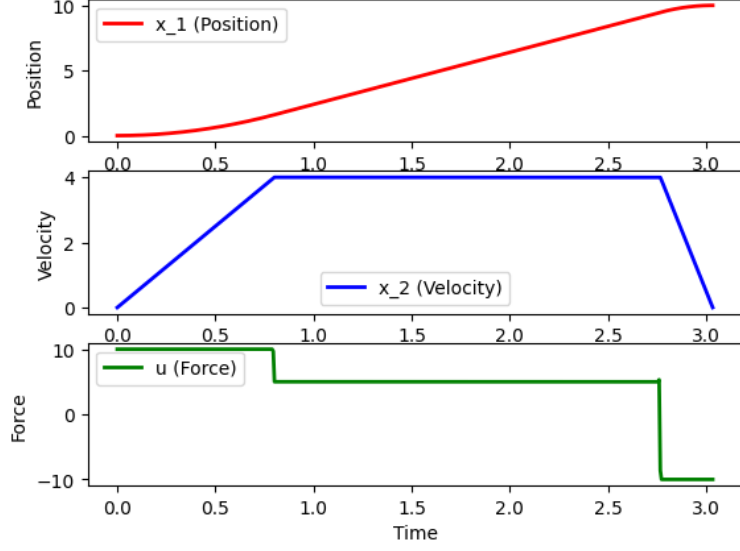
respectively.

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



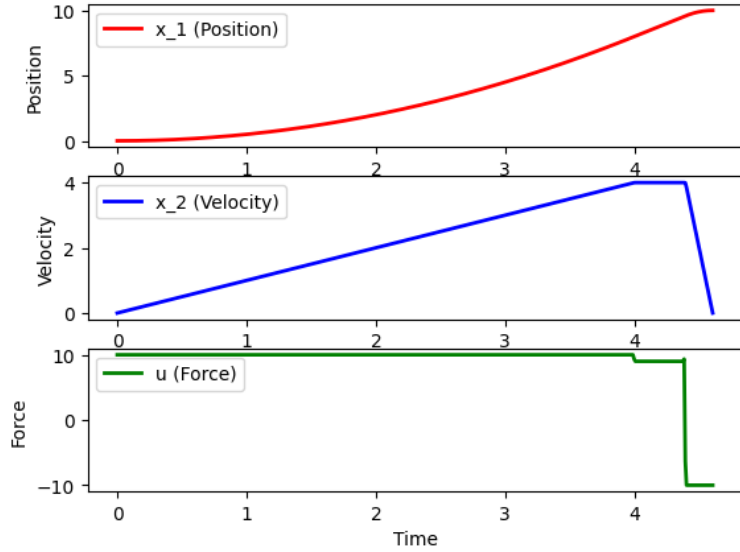
The numerical solution ($T= 2.9000100587$) with $p = 0$

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



The numerical solution ($T= 3.03334618$) with $p = 5$

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



The numerical solution ($T= 4.6053088162$) with $p = 9$

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

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

For both the classical approach and the training approach, after the discretization of the uncertainty set, the problem to be solved becomes a normal optimal control problem and we can use the multiple shooting and quasi Newton (in the SQP framework) method to solve. We are leveraging open source package Casadi³ and Gekko⁴ to solve our problem. These two packages can let the user choose different nonlinear programming solver and as well as different underlying numerical methods. In our case, we have chosen the multiple shooting and sequential quadratic programming, together with Runge–Kutta RK4 method. We have also discretized $t \in [0, 1]$ into 500 subintervals $\mathbb{I}_j = [\tau_{j-1}, \tau_j]$, i.e. the discretization points as $0 = t_0 \leq t_1 < \dots < t_m = t_f = 1$, and an error tolerance level as $1e - 6$. The numerical result from the classical approach and training approach will be discussed in the next two sections.

4.3 Apply classical (minmax) approach

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

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

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

$$s.t. \quad T_i \leq \epsilon \quad (4.7b)$$

$$x = (x_1, x_2), \quad (4.7c)$$

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

$$x(0, p_i) = 0, \quad (4.7e)$$

$$x_1(1; p_i) \geq 10 \quad \forall p_i, \quad (4.7f)$$

$$x_2(t; p_i) \leq 4, \quad \forall p_i, t \in [0, 1] \quad (4.7g)$$

$$x_2(1; p_i) \leq 0, \quad \forall p_i, \quad (4.7h)$$

$$T \geq 0, \quad (4.7i)$$

$$u(t) \in [-10, 10], \quad t \in [0, 1]. \quad (4.7j)$$

for all p_i where $p_i \in [p_0, p_1, \dots, p_n] \subset \mathbb{P}$. Since both $u(\cdot)$ and $x(\cdot)$ are functions of t , we also need to discretize these two variables 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 gets no feedback during the process and has to set up the driving strategy in advance. The realization of the trajectory of $u(\cdot)$ and $x(\cdot)$ for each p_i , therefore, may not be an optimal solution compared to the case when knowing the p_i value in advance.

³Please refer the website <https://web.casadi.org> for more details

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

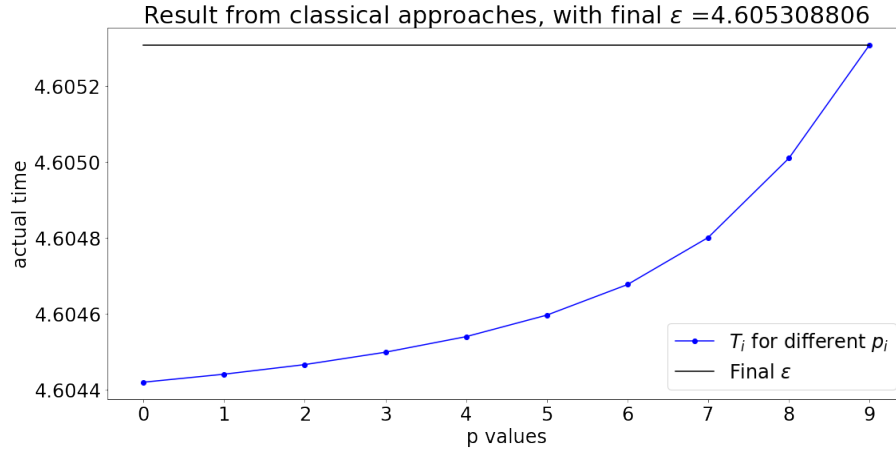


Figure 4.6: Final ϵ and T_i for different p_i values

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

In the actual implementation, we have take $n = 40$, i.e. in total 40 p_i points ranging the whole uncertainty set $\mathbb{P} = [0, 9]$. Nevertheless, we only show the result for $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ for the sake of readability. The final time ϵ and T_i is shown in Figure 4.6, where $\epsilon = \max_i T_i, i = 0, 1, 2, \dots, 9$, and ϵ reaches the maximum value 4.6053, when $p_i = 9$. The differences in T_i arise mainly because of the numerical errors, in our implementation, the number of subinterval is chosen as 500 and the numerical error tolerance level is set $1e - 6$. When we increase the number of subintervals and decrease the tolerance level, the T_i of $p_i \neq 9$ should converge to T_i 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 shown in Figure 4.7, Figure 4.8 and Figure 4.9 respectively. In all the three Figures 4.7, 4.8 and 4.9, the standardized time $t \in [0, 1]$ is used as x-axis. In Figure 4.7, the $u(t)$ acceleration/deceleration Force is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$; in Figure 4.8, the position $(x_1(t))$ is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$; and in Figure 4.9, the velocity/speed $(x_2(t))$ is shown for each $p_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

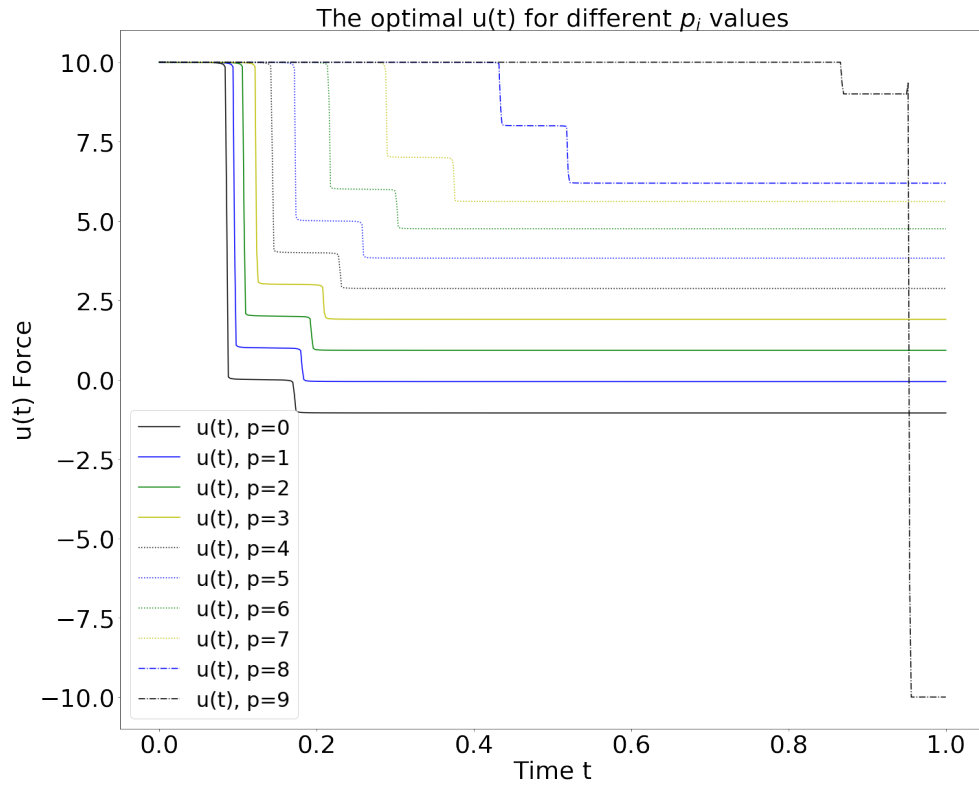


Figure 4.7: $u(t)$ Force for different p_i values

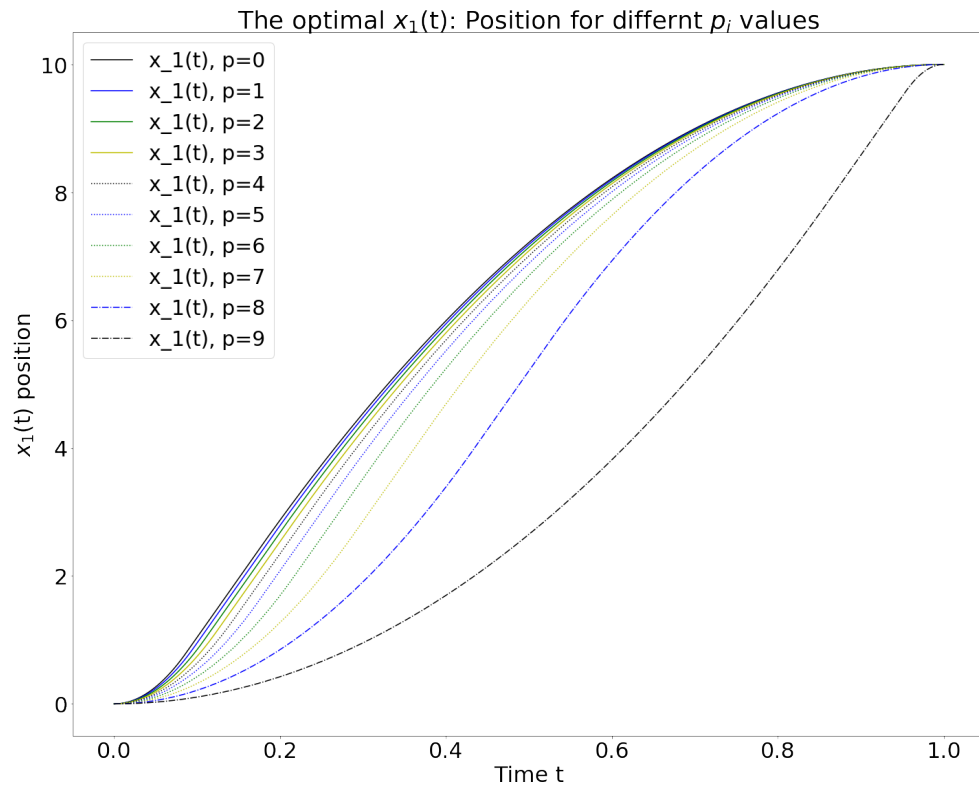


Figure 4.8: $x_1(t)$ Position for different p_i values

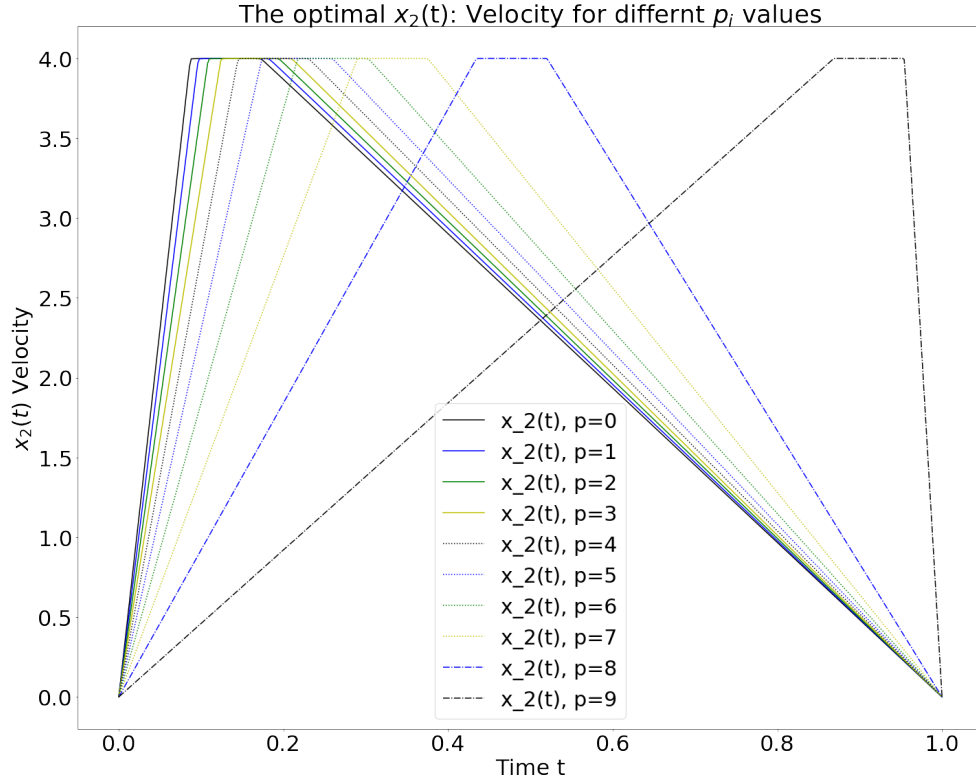


Figure 4.9: $x_2(t)$: Velocity for different p_i values

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

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

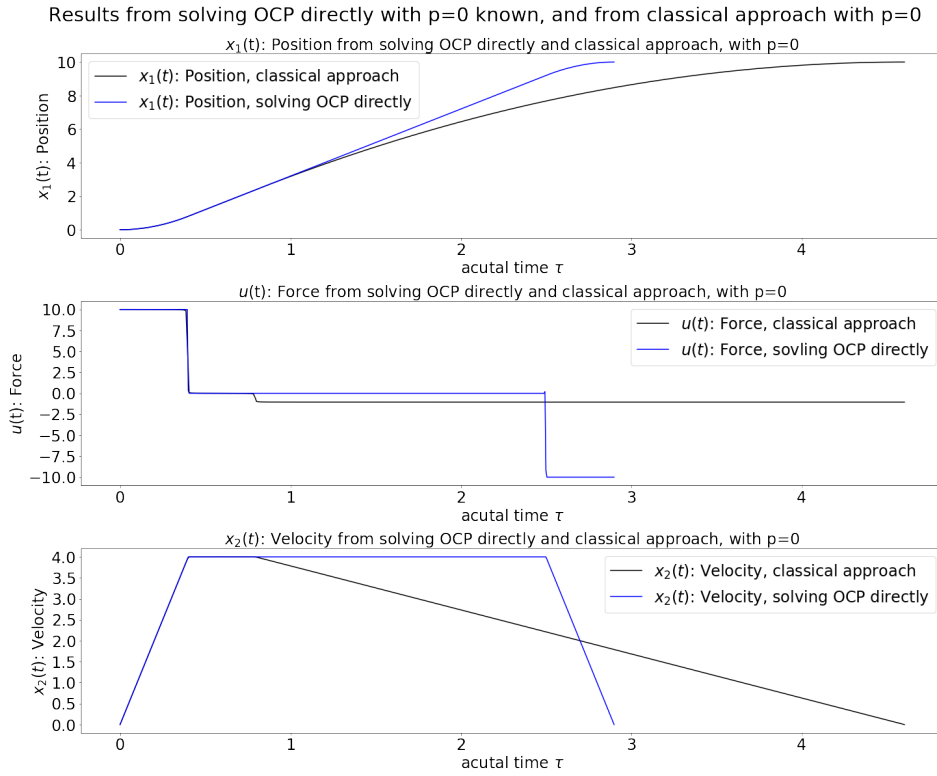


Figure 4.10: Compare the results from solving OCP 4.1 with $p = 0$ and the result of applying the classical approach to problem 4.7 with $p = 0$

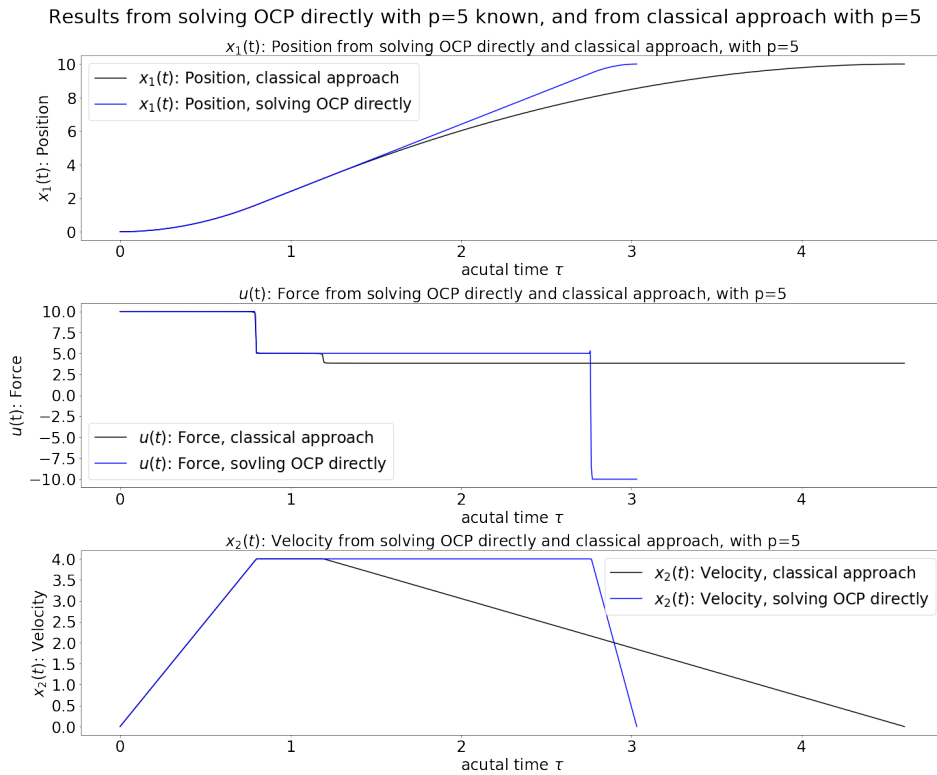


Figure 4.11: Compare the results from solving OCP 4.1 with $p = 5$ and the result of applying the classical approach to problem 4.7 with $p = 5$

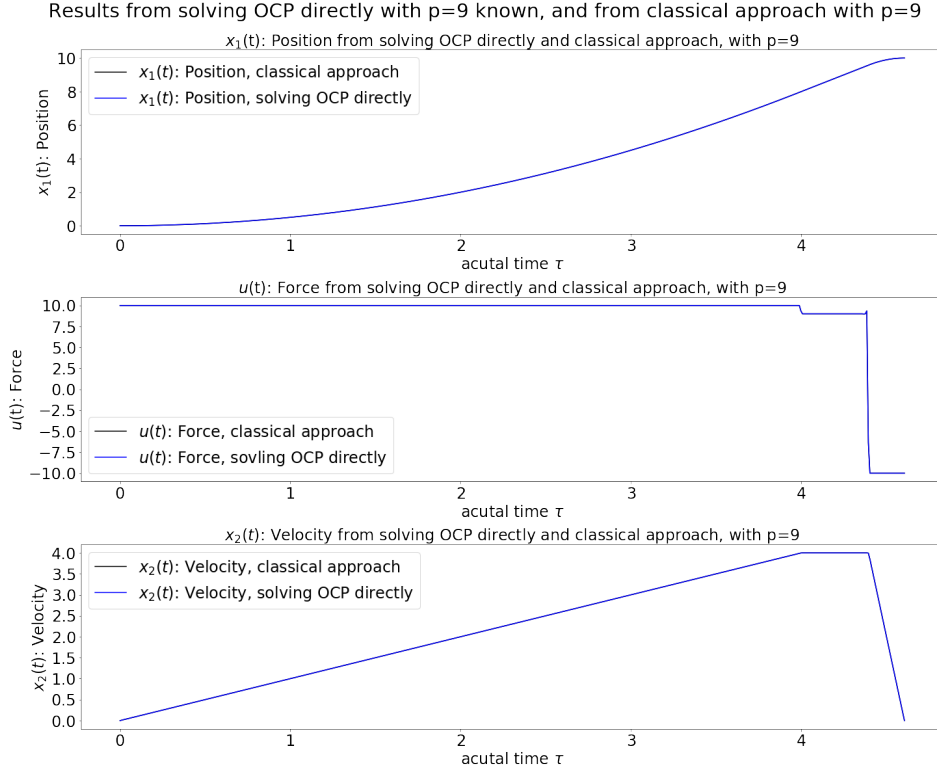


Figure 4.12: Compare the results from solving OCP 4.1 with $p = 9$ and the result of applying the classical approach to problem 4.7 with $p = 9$

4.4 Apply training (maxmin) approach

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

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

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

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

$$x(0, p) = 0, \quad (4.8d)$$

$$x_1(1; p_i) \geq 10, \quad (4.8e)$$

$$x_2(t; p_i) \leq 4, \quad t \in [0, 1], \quad (4.8f)$$

$$x_2(1; p_i) \leq 0, \quad (4.8g)$$

$$T \geq 0, \quad (4.8h)$$

$$u(t) \in [-10, 10], \quad t \in [0, 1]. \quad (4.8i)$$

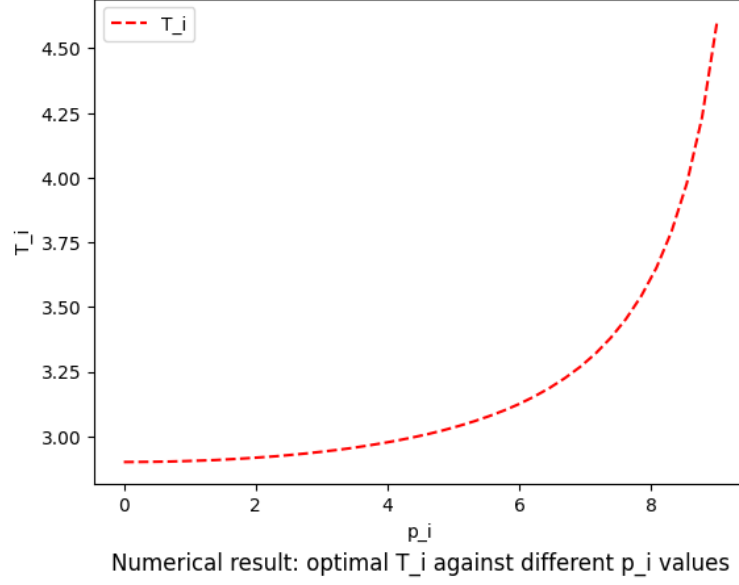


Figure 4.13: Numerical result: different T_i values against different p_i values, with $p_i = 9$ and $T_i = 4.6053$ the worst case

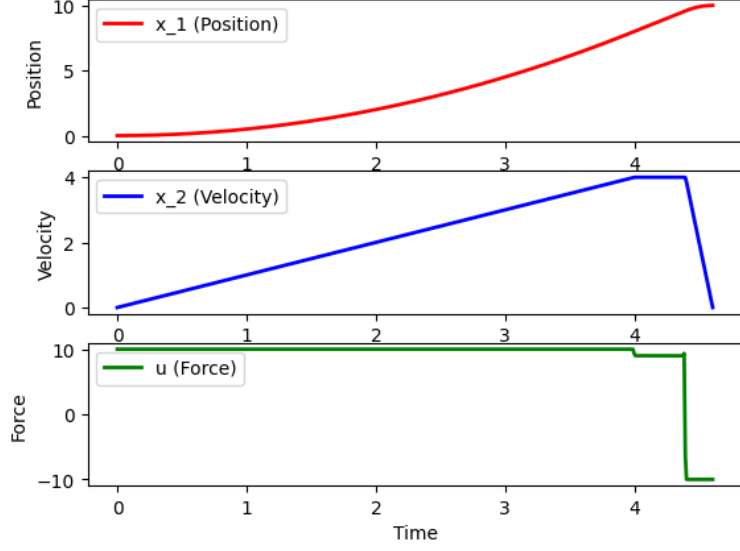
With a given p_i value, the lower level optimization problem turns out to be a normal OCP. One p_i value will correspond to one optimal T_i values. Among all the optimal T_i , we would like to find the worst case, i.e. the largest among all T_i values. Similar to the discretization of the uncertainty set in the classical approach, here we also take $n = 40$, i.e. in total 40 p_i points ranging the whole uncertainty set $\mathbb{P} = [0, 9]$. The result of T_i against p_i is shown in Figure 4.13. Figure 4.13 is almost identical to Figure 4.1, which confirms again that our numerical result with different p is consistent with the theoretical result. Furthermore, the worst case over all p_i is when p_i takes the biggest value, i.e. $p_i = 9$, and in this case $T_i = 4.6053$. We already shown in Figure 4.5 the solution to original problem when $p_i = 9$. The worst case, therefore, have the same solution as shown in Figure 4.5, with $T = 4.6053$, which we do repeat here in Figure 4.14.

4.5 Analysis the numerical results

Besides, the numerical results discussed above, in this section we perform some additional numerical analysis and discuss the numerical results altogether.

4.5.1 Sensitivity with initial value

In this part, we show that our numerical implementation is stable with different initial values $u(t_0) \in [0, 10]$. As explained before, the car should accelerate as fast as possible at the beginning, i.e. $u(t_0)$ should take maximum allowable value 10 at the beginning in order to have the optimal/smallest T . With different $u(t_0) \in [0, 10]$, the control variables $u(t)$ moves towards 10 at the acceleration stage, as shown in the first subplot of Figure 4.15. In this plot, we have chosen only 100 subintervals for $[0, 1]$. As shown in the second subplot of Figure 4.15, when the initial value of $u(t_0)$ deviates with the ideal initial value $u(t_0) = 10$, it takes most time T , compared to other initial values. When the number of subintervals is increased, then the T with initial value other than $u(t_0) = 10$



The numerical solution ($T = 4.6053088162$) with $p = 9$

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

will converge to the $T^*(u(t_0) = 10)$. The result in this section again confirms with 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, when finding the solution of the lower level part for each p , we can find a feasible solution $u(:, p), x(:, p)$ for each p . However, if we force a unified $u(t)$ for all p in the uncertainty set, we cannot find a feasible $u(t)$ for all the p values if the uncertainty set is too big. When we narrow the uncertainty set, for example, if we narrow the uncertainty set to $\mathbb{P} = [0, 1]$, we can still find feasible solution, with shared control states (i.e. $u(t)$) for all the parameters within this uncertainty set $\mathbb{P} = [0, 1]$, as shown in Figure 4.16. In Figure 4.16, we have plot the x and the unified/shared $u(t)$ for the parameter at the boundary $p = 0$ and $p = 1$. Clearly, as shown in the subplot for x_1 : Position and x_2 : Velocity, both trajectories satisfy the constraints with the unified $u(t)$ in the third subplot. However, if we increase the uncertainty set, we fail to find a shared $u(t)$ which leads to feasible solutions for all p in $\mathbb{P} = [0, 9]$. This is consistent with the theoretical analysis discussed in the next section 4.5.3, that the classical approach will lead to a solution that is not better than the training approach.

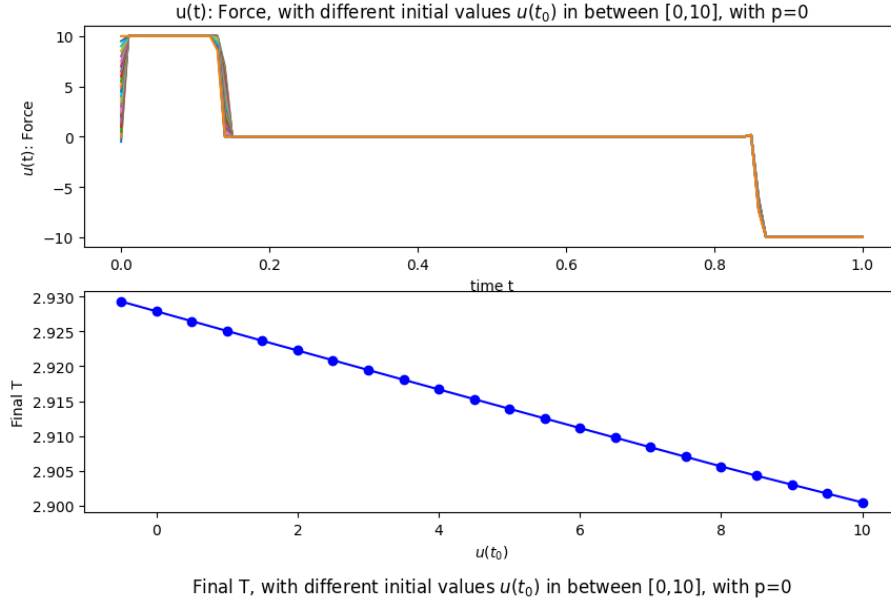


Figure 4.15: Results of solving OCP directly, with $p = 0$ and different initial values $u(t_0)$ in the set $[0,10]$.

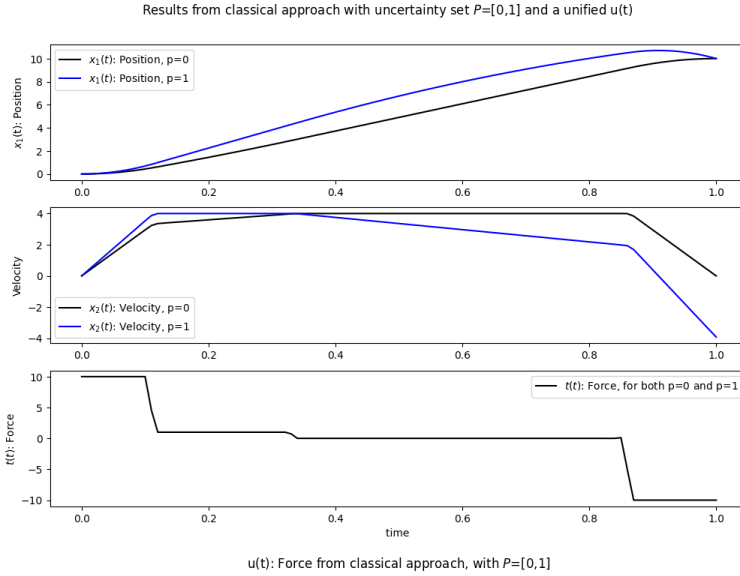


Figure 4.16: Results from classical approach with a unified $u(t)$ with uncertainty set $\mathbb{P} = [0,1]$, with $T = 2.9495636746$ when $p = 0$, and $T = 3.9240282505$ when $p = 1$.

4.5.3 Comparison between classical approach and training approach]

In theory, the classical approach will lead to a solution that is worse than the training approach. Within training approach, the p value is known beforehand and worst solution is obtained by maximizing over all $p \in \mathbb{P}$. While in the classical approach the p value is priori unknown, and we need to find solutions that yield parameter-dependent variables for all possible realizations of the uncertain parameters, and among them find the optimal

solution. We do not dive deep into the proof (ref [Schlöder \[2022\]](#) for details), here we show the conclusion in Theorem 3 directly and one example to support the idea.

Theorem 3 *Let $\Omega_x \subset \mathbb{R}^{n_x}$ and $\Omega_p \subset \mathbb{R}^{n_p}$ be compact subsets and $f : \Omega_x * \Omega_p \rightarrow \mathbb{R}$ a continuous function. Then we have*

$$\max_{p \in \Omega_p} \min_{x \in \Omega_x} f(x, p) \leq \min_{x \in \Omega_x} \max_{p \in \Omega_p} f(x, p) \quad (4.9)$$

The optimal objective function value of the *max min* problem (training approach) over estimates the one of the *min max* problem (classical approach). It is easy to find examples in which the gap is indeed greater than zero. For example, let $\Omega_x = [-5, 5]$, $\Omega_p = [-1, 1]$ and consider the function

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

Then⁵

$$\max_{p \in \Omega_p} \min_{x \in \Omega_x} f(x, p) = 1 \leq \frac{5}{4} = \min_{x \in \Omega_x} \max_{p \in \Omega_p} f(x, p)$$

As show above, both the classical approach and training approach leads to an identical solution with final time as $T = 4.6053$, which is conservative with respect to original OCP 4.1. The time $T = 4.6053$ is conservative, regardless 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 approach and the training approach do not necessary lead to an identical solution for a general OCP under uncertainty. In general, classical approach should lead to a worse solution compared to training approach. In our case, it just happens that both approaches lead to the same solution, and the worst case happens when $p = 9$ at the right boundary point.

5 Conclusion

In this paper, we have explained in theory how direct approaches can be used for solving optimal control problems. Within direct approaches, multiple shooting method is used to discretize the whole interval into subintervals, with the objective functions and constraints also discretized and applied piecewise for each subinterval. Matching condition is enforced at the boundary of each subinterval. With numerical methods such as Runge–Kutta method applied to the underlying dynamics system (i.e. differential equation) in each subinterval, the original OCP is turned into a piecewise nonlinear optimisation problem. We can, therefore, apply the KKT condition to incorporate the constraints and the objective function into a new Lagrangian function, whose optimal solution can be found via its derivatives. Newton and quasi Newton is used and the Lagrangian function optimal is found via the sequential quadratic approach.

⁵The proof that *max min* (training) approach over estimates that of *min max* approach is given in paper [Schlöder \[2022\]](#)

In real life, some OCPs may have uncertainty, in the form of unknown parameters p in the uncertainty set \mathbb{P} . Such problems is even harder to solve due to the uncertainty. In this case, we would like to have a conservative solution, which is worse than the optimal solution for a particular p value regardless how the unknown parameter take values in the uncertainty set \mathbb{P} . There are two approaches to reach the conservative solution, the classical approach and the training approach.

For the numerical methods discussed in this paper, their application is demonstrated with a case study in state constrained rocket car. When the unknown parameter p takes a fixed value in the uncertainty set \mathbb{P} , the OCP under uncertainty degenerates into a normal OCP problem and the solution can be obtained via mutiple shooting and quasi Newton style method. The result have been shown in e.g. Figure 4.3, 4.4 and 4.5. We have also perform a sensitivity analysis with respect to to the initial value, as shown in Figure 4.15. The results confirms with our analysis that the numerical method we are using are stable and roubust.

We further continue the numerical implementation with classical approach and training approach, and both leads to a conservative solution. For our particular case, the conservative solutions from both approach are the same. This is not contracdictory to our analysis that classical approach leads to a worse solution compared to traning approach. Actually, if we force the same control trajectory $u(t)$ for all the parameters $p \in \mathbb{P}$, no fesiable $u(t)$ can be found as explained in section 4.5.2, indicating the optimal T is infinity.

Therefore, we can conclude that our numerical results are consistent with our theoretical knowledge in the numerical methods discussed in this paper. These method can, therefore, be applied to more general optimal control problems and optimal control problems under uncertainty.

A Appendix Runge–Kutta method

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

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

If the initial value x_0 for A.1 is known, then equation A.1 becomes an initial value problem (IVP) and the solution can be found via numerical method. If analytical solutions exist for equation A.1, then we can use the analytical solution directly. Nevertheless, for most real-life problems, the analytical solution either does not exist or is very difficult to find, and numerical method is the only practical way to find the solution. Given the initial value x_0 , equation A.1 will have solution as

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

The integral in equation A.2 can be approximated by numerical methods, and one simple approximation to the integral in equation A.2 can be obtained via Euler method.

The Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. Using a step size equal to 1, the approximation to equation A.2 via Euler method is of the form

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

Euler method is a first order and explicit method for approximating integrals, another widely used method is the trapezoidal rule, which is an implicit second-order method. The trapezoidal rule can approximate equation A.2 as

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

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

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

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

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

$$t_{n+1} = t_n + h \quad (\text{A.7})$$

for $n = 0, 1, 2, 3, \dots$, using

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

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

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

$$k_4 = f(t_n + h, y_n + hk_3). \quad (\text{A.11})$$

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 .

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. Zangwill 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 Gerds 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)