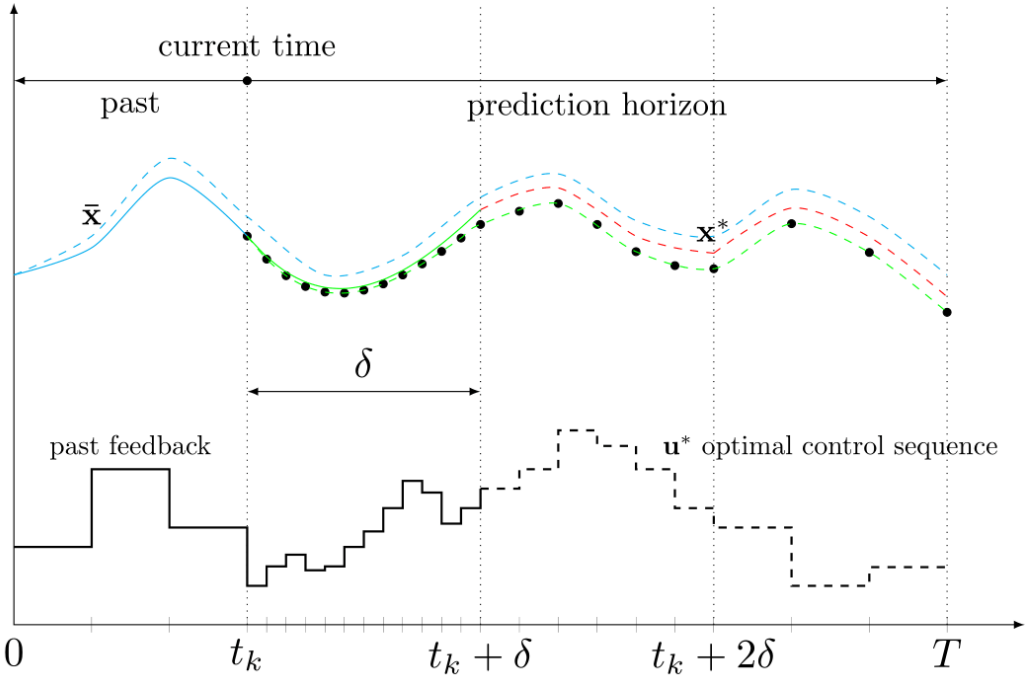


Numerical Methods for Optimal Control and Model Predictive Control

Luís Tiago de Freixo Ramos Paiva
Programa Doutoral em Matemática Aplicada
2014

Orientador
Fernando Arménio da Costa Castro e Fontes
Professor Associado com Agregação
Faculdade de Engenharia da U. Porto

To my family

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Fernando Fontes for the continuous support of my Ph.D. study and research, for his motivation, enthusiasm, knowledge, and friendship. His guidance was immensely valuable during my research and writing of this thesis.

My sincere thanks goes to Prof. Maria do Rosário de Pinho for her assistance along the way, exchanges of knowledge, skills, which helped enrich my Ph.D. experience. Appreciation also goes out to Prof. Margarida Ferreira and Prof. Dalila Fontes for their motivation and encouragement.

I must also acknowledge Prof. Hasnaa Zidani, Prof. Paola Falugi and Prof. Eric Kerrigan for receiving me at their workplaces and for offering me opportunities to acquire knowledge in their groups.

I thank my fellow labmates in the Department of the Electrical and Computer Engineering – Juliana Almeida, Igor Kornienko, Filipa Nogueira, Luís Roque, Sofia Lopes, Amélia Caldeira – and former labmates – Ana Filipa Ribeiro, Haider Biswas, Mário Amorim Lopes, and Rui Calado – for the stimulating discussions, and for all the good times we had in the last three years. A very special thanks goes out to Achille Sassi for our debates about optimal control software and their features. Also I thank my friends in the Mechanical Engineering field: Carlos Veiga Rodrigues, Carlos Silva Santos, José Carlos Ribeiro and Alexandre Silva Lopes for their friendship over the past years.

I would also like to thank my parents and my sister for the support they provided me through my entire life and, in particular, I must acknowledge my wife and best friend, Catarina, for the love, patience and encouragement.

I would like to express my gratitude for the support of the coordinators of the doctorate program, Prof. Sílvio Gama. I would also like to thank the Faculty of Engineering of the University of Porto and the Institute of Systems and Robotics of Porto for the excellent working conditions provided to me.

In conclusion, the support of FCT – Fundação para a Ciência e Tecnologia – under Grants PTDC/EEA-CRO/116014/2009 and PTDC/EEI-AUT/1450/2012 and of the European Union Seventh Framework Programme [FP7-PEOPLE-2010-ITN] under grant agreement n. 64735-SADCO are greatly acknowledged.

Abstract

This thesis addresses, via numerical and optimisation methods, the control of nonlinear systems whose inputs or trajectories are subject to constraints. Nevertheless, we review and apply theoretical results, such as conditions of optimality, to characterise the optimal trajectory and to validate numerical results obtained using our proposed methods.

We overview most used software packages for solving optimal control problems, including numerical solvers which invoke local search methods and interfaces with distinct features. A benchmark involving a differential drive robot with state constraints is presented in order to compare the performances of the solvers.

We propose and develop an optimal control algorithm based on a direct method with adaptive refinement of the time–mesh. When using direct methods to solve nonlinear optimal control, regular time meshes having equidistant spacing are most frequently used. However, in some cases, these meshes cannot cope accurately with nonlinear behaviour and increasing uniformly the number of mesh nodes may lead to a more complex problem, resulting in an incoherent solution. We propose a new adaptive time–mesh refinement algorithm, considering different levels of refinement and several mesh refinement criteria. This technique is applied to solve an open–loop optimal control problem involving nonholonomic vehicles with state constraints, which is characterized by presenting strong nonlinearities and by having discontinuous controls, and a compartmental model for the implementation of a vaccination strategy. This algorithm leads to results with higher accuracy and yet with lower overall

computational time, when compared to results obtained by meshes having equidistant–spacing.

We extend the time–mesh refinement algorithm to be applied to a sequence of optimal control problems in a Model Predictive Control scheme. Model Predictive Control is a technique widely used in industrial control problems that explicitly consider constraints. The receding horizon control strategy can be used in real time applications and it can be implemented for large-scale systems. The proposed algorithm is applied to solve an optimal control problem involving parking manoeuvres. The results are obtained as fast as the ones given by a coarse equidistant–spacing mesh and as accurate as the ones given by a fine equidistant–spacing mesh.

Global Optimisation methods are addressed as well. The accurate solution of optimal control is crucial in many areas of engineering and applied science. Since problems involving nonlinear systems often contain multiple local minima, we study deterministic and heuristic methods which attempt to determine the global solution. A problem involving a car–like system is successfully solved and the global optimum is found.

Resumo

Esta tese tem por base o estudo e desenvolvimento de métodos numéricos e de optimização para o controlo de sistemas não lineares sujeitos a restrições de estado ou controlo. Os principais resultados teóricos, tais como as condições de optimalidade, são analisados e aplicados para caracterizar a trajectória óptima, bem como para validar resultados numéricos obtidos com os métodos propostos.

Algumas das bibliotecas de software disponíveis para resolver problemas de controlo óptimo são apresentadas, entre as quais *solvers* numéricos que implementam métodos de procura local, assim como *interfaces* com características distintas.

Um algoritmo de controlo óptimo com refinamento adaptativo da malha temporal, baseado em métodos directos, é desenvolvido e implementado. Quando são usados métodos directos na resolução problemas de controlo óptimo não linear é usual recorrer–se a uma discretização do domínio temporal numa malha regular e com nós equidistantes. Contudo, em determinados problemas, estas malhas temporais não conseguem representar com precisão o comportamento não linear e aumentar uniformemente o número de nós pode traduzir–se num problema mais complexo, resultando numa solução incorrecta. Assim, um novo algoritmo para um refinamento adaptativo da malha, onde são considerados diferentes níveis de refinamento e critérios de paragem, é proposto. Este algoritmo é aplicado na resolução de problemas de controlo óptimo em malha aberta. Um dos problemas envolve veículos não holonómicos com restrições de estado, caracterizado pela presença de não–linearidades e por ter controlos descontínuos. O algoritmo proposto é também aplicado ao modelo

compartimental para a implementação de uma estratégia de vacinação.

O algoritmo para o refinamento da malha foi estendido com o objectivo de ser aplicado a problemas de controlo óptimo numa estratégia de Controlo Preditivo. O Controlo Preditivo é uma técnica amplamente usada na indústria em problemas de controlo com restrições explícitas. A estratégia de controlo com horizonte deslizante pode ser usada em aplicações em tempo real e em sistemas de larga escala. O método proposto é aplicado na resolução de um problema de controlo óptimo envolvendo manobras de estacionamento de veículos. Os resultados foram obtidos quase tão rápido quanto aqueles que foram calculados usando uma malha lassa com nós equidistantes e com a mesma precisão dos resultados fornecidos por uma malha fina.

Esta tese aborda, também, métodos de Optimização Global. A precisão das soluções de problema de controlo óptimo é crucial em várias áreas da engenharia e das ciências aplicadas. Uma vez que os problemas que envolvem sistemas não lineares contêm, muitas vezes, múltiplos mínimos locais, métodos globais determinísticos e baseados em heurísticas foram estudados. Um problema envolvendo um sistema que modela um veículo foi resolvido com sucesso e o óptimo global foi encontrado.

Contents

List of Tables

List of Figures

List of Acronyms

ACADO Automatic Control And Dynamic Optimisation. 41, 42

AMPL A Modelling Language for Mathematical Programming. 35, 36, 38, 40, 41, 45, 49, 51

BOCOP BOCOP. 42

B&B Branch and Bound. 5, 6, 99, 100

DIDO DIDO. 43

DP Dynamic Programming. 1, 2, 5, 6, 24, 28, 33, 99, 102, 107, 109

FJ Fritz John. 11–13, 15, 20–23

GO Global Optimisation. 5, 6, 98, 99, 107, 109, 110

GOC Global Optimal Control. 6, 98, 107, 109

GPOPS-II Gauss Pseudospectral Optimal Control Software. 47

HJ Hamilton–Jacobi. 6, 44, 106, 107, 109

HJB Hamilton–Jacobi–Bellman. 2, 5, 102

HJE Hamilton–Jacobi Equation. 30, 31

ICLOCS Imperial College London Optimal Control Software. 43, 44, 60

IPOPT Interior–Point Optimiser. 6, 35, 36, 42, 44, 46–53, 60, 68, 77, 108

IS Impulsive System. 110

KKT Karush–Kuhn–Tucker. 13, 15, 23

KNITRO KNITRO. 6, 36, 37, 48–53, 108

MP Maximum Principle. 2, 31

MPC Model Predictive Control. 4–6, 41, 67, 78–84, 87–90, 92, 93, 95, 96, 109, 110

NLO Nonlinear Optimisation. 98

NLP Nonlinear Programming. 3, 6, 8, 10, 32, 34, 35, 37, 39–41, 47, 50, 52, 53, 55, 56, 59, 60, 66, 75, 87, 95, 99, 101, 108

OC Optimal Control. 3, 6, 8, 39, 40, 53, 108

OCP Optimal Control Problem. 1, 2, 4–6, 16, 19, 24, 26, 28, 29, 31, 32, 34, 35, 44, 45, 55, 56, 60, 66, 75, 78, 80–84, 87, 88, 92, 95, 96, 98, 99, 102, 109

PSOPT PSOPT. 46, 47

ROC-HJ Reachability and Optimal Control Software. 44, 45, 105–107

SNOPT Sparse Nonlinear Optimiser. 6, 39, 47

SOCS Sparse Optimal Control Software. 6, 38, 39

TACO Toolkit for AMPL Control Optimisation. 45

WORHP WORHP Optimises Real Huge Problems. 6, 37, 38, 48–53, 108

Nomenclature

$\bar{\varepsilon}$ Levels of refinement

($\mathrm{P_{CL}}$) Car–like System Problem

($\mathrm{P_{CP}}$) Car Parking Manoeuvres Problem

($\mathrm{P_{DD}}$) Differential Drive Robot Problem

($\mathrm{P_{DOC}}$) Discrete Optimal Control Problem

($\mathrm{P_{GO}}$) Global Optimal Control Problem

$\mathcal{N}$ Number of mesh nodes

$\mathcal{S}_k$ Mesh subinterval generated according to some refinement criteria

$\mathcal{S}_{k,i}$ Mesh subinterval of $\mathcal{S}_k$ in the i$^{\mathrm{th}}$ level of refinement

($\mathrm{P_{NLP}}$) Nonlinear Programming Problem

($\mathrm{P_S}$) SEIR Problem

$\left[t_0, t_f\right]$ Time domain

$\varepsilon^{\max}$ Error threshold

$\varepsilon_{\mathbf{x}}$, $\varepsilon_{\mathbf{q}}$ Error on the trajectory/adjoint multipliers

$\mathbb{X}$, $\mathbb{U}$ State/Control domain

Chapter 1

Introduction

"Most things can be improved,
so scientists and engineers optimise."
Lorenz T. Biegler

This thesis addresses, via numerical and optimisation methods, the control of nonlinear systems whose inputs or trajectories are subject to constraints.

Constrained control problems arise naturally and frequently in real applications due to safety reasons, reliability of operation, or physical restrictions that are not described in the dynamic equations not in the control constraints. There are many examples of these state constraints such as minimum altitude or velocity of a plane, maximum temperature or pressure in a chemical reactor, obstacles to be avoided by a vehicle or robot, among others. Nonlinear models are often use to approximate real applications since common hard nonlinearities of a discontinuous nature do not allow linear approximation.

We can solve an Optimal Control Problem (OCP) using Dynamic Programming and Hamilton–Jacobi methods, Indirect Methods or Direct Methods. Dynamic Programming (DP) is a stage wise search method of optimisation problems whose solutions may be viewed as the result of a sequence of decisions. The selection of the

optimal decision in based on the Bellman's Principle of Optimality. In continuous–time problems the DP procedure can be formulated as a partial differential equation known as the Hamilton–Jacobi–Bellman (HJB) equation. The solution of the DP recursion or the HJB partial differential equation is very expensive, except for small dimension problems. An optimal sequence of decisions is obtained if and only if each subsequence is optimal. The analytical solution of the HJB equation is, in general, not possible to achieve and a numerical solution is computationally very hard to compute. Thus, in general, only very low dimensional problems can be solved in reasonable time, which is the main practical limitation on this approach.

Indirect and Direct methods for solving OCP have, also, their advantages and disadvantages [Bet01, BH98, Bie10]. The *Indirect Methods* involve the conditions of optimality, the adjoint equations, a maximisation condition and the boundary conditions, forming a boundary value problem. We can compute the solution via shooting, multiple shooting, or discretisation. The *Direct Methods* directly optimise the objective without formation of the necessary conditions, using control and state parametrisation. When using direct methods, practical methods for solving OCP [Bie10, BH75, JTB04] require Newton-based iterations with a finite set of variables and constraints, which can be achieved by converting the infinite-dimensional problem into a finite-dimensional approximation. The indirect methods only discretise after the optimisation. Therefore, solutions can be achieved with high level of accuracy (significant digits). They are popular in the Aerospace Industry to compute trajectories for rockets/satellites where accurate solutions are needed. However, such analytical solutions are very hard or even impossible to achieve.

Over the past decades, direct methods have become increasingly useful when computing the numerical solution of nonlinear OCP [Bet01]. These methods directly optimise the discretised OCP without using the Maximum Principle (MP) and they are known to provide a robust approach for a wide variety of optimal control problems. To achieve the optimal solution we need a numerical/computational solver. A list of solvers including open–source, freeware and commercial software, working under

different operating systems, is available [Pai13]. Since they are based on local search methods, these solvers compute a local optimal solution, when convergence is achieved. To test them, we consider a minimum time problem involving a differential drive robot system. The selection of an initial guess proved to vital to improve the performances of all solvers.

We can make a better use of solvers if we access them with a proper interface. An interface is a software that provides us the means to communicate with the solver. The Optimal Control (OC) interfaces handle with the discretisation and transcription procedures, while the Nonlinear Programming (NLP) interfaces leave this task to the user. Several interfaces are presented, encompassing both types.

In a direct collocation method, the control and the state are discretised in a set of appropriately chosen mesh, in the time interval. Most frequently, in the discretisation procedure, regular time meshes having equidistant spacing are used. However, in some cases, these meshes are not the most adequate to deal with nonlinear behaviours. One way to improve the accuracy of the results, while maintaining reasonable computational time and memory requirement, is to construct a mesh having different time steps. The best location for the smaller steps sizes is, in general, not known a priori, so the mesh will be refined iteratively.

In this thesis, an adaptive time–mesh refinement algorithm is presented [PF14b]. There are three new main features in this algorithm: (a) we introduce several levels of refinement, obtaining a multi–level time–mesh in a single iteration – such concept allows us to implement the nodes collocation in a faster and clever way; (b) we also introduce different refinement criteria – the relative error of the primal variables, the relative error of the dual variables and a combination of both criteria are used in the refinement procedure; (c) we consider distinct criteria for refining the mesh and for stopping the refinement procedure – the refinement strategy can be driven by the information given by the dual variables and it can be stopped according to the information given by the primal variables. The local error of the adjoint multipliers is chosen as a refinement criterion because they are solution to a linear differential

equation system, easily solved with high accuracy, and because they give sensitivity information. It is still a direct method approach but we can use some information given by necessary conditions, namely, the adjoint differential equation, which is use in the indirect methods context. To decrease the overall computational time, the solution computed in the previous iteration is used as a warm start in the next one, which proved to be of major importance to improve computational efficiency.

In order to validate the results obtained for this problem, we apply the Maximum Principle of Pontryagin. This analysis allows us to characterize the optimal trajectory and control and it also provides us the differential equation system for the multipliers. This last information is needed to estimate the error in the multipliers for one of the refinement criteria.

When using this strategy, an OCP is solved using less nodes in the overall procedure which revert in significant savings in memory and computational cost. In addition, there is no need to decide *a priori* an adequate mesh meeting our accuracy specifications, which is a major advantage of this procedure. The proposed refinement strategy shows more robustness, since it was able to obtain a solution when the traditional approach – starting with a very large number of mesh nodes – failed to do so. With this mesh–refinement strategy, nonlinear OCP solvers can be used in real–time optimization, since approximate solutions can be produced even when the optimizer is interrupted at an early stage.

This technique is applied to solve two different problems, both with pathwise state constraints: a car–like system problem [PF13, PF14a] which involves nonholonomic systems [KM95], and a vaccination strategy problem involving a SEIR model [BPd14, KPP14, NL10] which describes the spreading of an infectious disease. The results show solutions with higher accuracy obtained in a overall computational time that is lower when compared to the ones computed with a mesh having equidistant spacing and to the ones computed with a mesh designed as suggested in [PF13].

A powerful technique to solve OCPs is Model Predictive Control (MPC) which

is also addressed in this thesis. MPC, also referred to as moving horizon control or receding horizon control, is an optimisation based method for feedback control. After MPC appeared in industry, a theoretical basis for this technique has started to emerge [MM90]. MPC has become a preferred control strategy for a large number of processes. The MPC problem is formulated as solving on–line a sequence of finite horizon open–loop OCP subject to system dynamics and constraints involving states and controls. The receding horizon control strategy is especially useful for the control of slow nonlinear systems, such as chemical batch processes, where it is possible to solve, sequentially, open–loop fixed–horizon, optimal control problems on–line [MM90]. MPC can, also, be used in tracking control [GP11]. Much progress has been made on these issues for nonlinear systems [FP12a, GP11] but for practical applications many questions remain, making MPC an interesting topic of research.

We extend the time–mesh refinement algorithm to a sequence of optimal control problems in a MPC scheme. We consider several levels of refinement depending on time, obtaining a multi–level scheme that varies along the time interval. The idea is to establish planning strategies similar to the ones we use in our daily basis routine. The proposed algorithm is applied to solve an optimal control problem involving parking manoeuvres. The results are obtained as fast as those we get with a coarse equidistant–spacing mesh and as accurate as the ones get with a fine equidistant–spacing mesh. Due to the fast response of the algorithm, it can be used to solve real–time optimisation problems.

Global Optimisation (GO) methods are addressed in this thesis as well. Since problems involving nonlinear systems often contain multiple local minima, we study deterministic [HP95b, PR02] and heuristic methods [Wei08] which attempt to determine the global solution. The objective of GO is to find the globally best solution of models when multiple local optima exist. Among the methods for GO, we emphasize Dynamic Programming (DP), the D.C. Programming method, the Lipschitz Optimisation approach and Branch and Bound (B&B) algorithms. DP and HJB methods are used for solving large problems which are divided into smaller problems.

By solving the individual smaller problems, the solution to the larger problem is found. In the nonconvex optimisation context, D.C. Programming plays an important role because of its theoretical aspects as well as its wide range of applications [HT99]. The Lipschitz Optimisation approach to GO has always been attractive since in the cases that we know the Lipschitz constant, global search algorithms can be deployed and convergence theorems easily proved. The Branch and Bound (B&B) technique is a widely used technique to solve several types of difficult optimisation problems. In the B&B methods, the feasible set is relaxed and subsequently partitioned into refined parts – branching – over which lower ad upper bounds of the minimum objective function value can be determined – bounding [HP95a].

We use a Global Optimal Control (GOC) approach, using DP and Hamilton–Jacobi (HJ) methods, which attempt to determine the global solution of a minimum time problem involving a car–like system which has to avoid an obstacle. This problem is successfully solved and the global optimal trajectory is found.

This thesis is organized as follows. In chapter 2, we review some basic concepts and theoretical results in the NLP and OC contexts [BSS06, Bel57, Cla98, Fon99, FH10, Ger12, GSD06, Var72, Vin00]. In chapter 3, we overview available software for solving nonlinear programming problems and optimal control problems, highlighting their features, namely for Interior–Point Optimiser (IPOPT) [WB06], KNITRO [LLC13], WORHP Optimises Real Huge Problems (WORHP) [NBW11], and, also, for Sparse Optimal Control Software (SOCS) and Sparse Nonlinear Optimiser (SNOPT). We do a benchmark in order to compare the performance of the solvers. In chapter 4, we propose an adaptive time–mesh refinement algorithm that is based on block-structured adaptive refinement method. In chapter 5, we provide the principles underlying MPC, its advantages and some computational aspects, as well as an extend algorithm for adaptive time–mesh refinement. In chapter 6, we introduce Global Optimal Control (GOC). GO and GOC methods are described and applied to a nonlinear OCP towards global optimality. In chapter 7, we summarise the conclusions

made along the chapters and we end with some future work proposals.

Chapter 2

Nonlinear Programming and Optimal Control

"I understood that the will could not be improved before the mind had been enlightened."
Johann Heinrich Lambert

In this chapter we review some basic concepts and theoretical results in the NLP and OC contexts [BSS06, Bel57, BH75, Cla13, Cla90, Cla98, Fon99, FH10, Ger12, GSD06, Var72, Vin00].

2.1 Nonlinear Programming

2.1.1 Mathematical Programming Problem

Let us consider a scalar real objective function in n variables, *i.e.*, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$. The Nonlinear Programming (NLP) problem can be written as [Bie10]:

$$\min_{\mathbf{x} \in S} \mathbf{f}(\mathbf{x}) \tag{2.1}$$

where $S \subset \mathbb{R}^n$, which is equivalent to the problem

$$\max_{\mathbf{x} \in S} -\mathbf{f}(\mathbf{x}). \tag{2.2}$$

Let us consider $\mathbf{f}$ and the optimisation problem (2.1), where $S \subset \mathbb{R}^n$ is a nonempty set:

(i) A point $\mathbf{x} \in S$ is called a *feasible solution* to problem (2.1).

(ii) If $\mathbf{x}^* \in S$ and $\mathbf{f}(\mathbf{x}) \geq \mathbf{f}(\mathbf{x}^*)$ for each $\mathbf{x} \in S$, then $\mathbf{x}^*$ is called an *optimal solution*, a *global optimal solution*, or simply a *minimiser* to the problem.

(iii) The collection of optimal solutions is called the set of *alternative optimal solutions*.

(iv) If $\mathbf{x}^* \in S$ and if there exists an $N_\varepsilon(\mathbf{x}^*)$ around $\mathbf{x}^*$ such that $\mathbf{f}(\mathbf{x}) \geq \mathbf{f}(\mathbf{x}^*)$ for each $\mathbf{x} \in S \cap N_\varepsilon(\mathbf{x}^*)$, then $\mathbf{x}^*$ is called a *local optimal solution* or *local minimiser*.

(v) If $\mathbf{x}^* \in S$ and if $\mathbf{f}(\mathbf{x}) > \mathbf{f}(\mathbf{x}^*)$ for each $\mathbf{x} \in S \cap N_\varepsilon(\mathbf{x}^*)$, $\mathbf{x} \neq \mathbf{x}^*$, for some $\varepsilon > 0$, then $\mathbf{x}^*$ is called a *strict local optimal solution* or *strict local minimiser*.

2.1.2 Unconstrained Nonlinear Programming Problem

Let $S \subset \mathbb{R}^n$ be an open set, *e.g.*, $S = \mathbb{R}^n$.

For differentiable functions, there exist conditions to know if a given point $\mathbf{x} \in S$ is a local or global minimiser of a function $\mathbf{f}$.

Theorem 2.1.1. *Suppose that* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ *is differentiable at* $\mathbf{x}^*$. *If* $\mathbf{x}^*$ *is a local minimiser, then* $\nabla \mathbf{f}(\mathbf{x}^*) = 0$.

The above condition is called a *first–order condition* since it uses the gradient vector, which has the first partial derivatives of $\mathbf{f}$ as components. We can also state *necessary conditions* in terms of the Hessian matrix H, which comprises the second derivatives of $\mathbf{f}$. These conditions are called *second–order conditions*.

Theorem 2.1.2 (Necessary Condition)**.** *Suppose that* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ *is twice–differentiable at* $\mathbf{x}^*$*. If* $\mathbf{x}^*$ *is a local minimizer, then* $\nabla \mathbf{f}(\mathbf{x}^*) = 0$ *and* $H(\mathbf{x}^*)$ *is positive semidefinite.*

Since the conditions stated so far are necessary conditions for a local optimal solution, now we present a sufficient condition for a local optimal solution.

Theorem 2.1.3 (Sufficient Condition)**.** *Suppose that* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ *is twice–differentiable at* $\mathbf{x}^*$*. If* $\nabla \mathbf{f}(\mathbf{x}^*) = 0$ *and* $H(\mathbf{x}^*)$ *is positive definite, then* $\mathbf{x}^*$ *is a strict local minimiser.*

When solving an unconstrained problem, we have to minimize a certain function $\mathbf{f}(\mathbf{x})$ without any constraints on the vector $\mathbf{x}$. However, in real applications we have to deal with constrained problems.

2.1.3 Constrained Nonlinear Programming Problem

Let us consider a scalar real objective function in n variables, *i.e.*, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$. The NLP constrained problem can be written as

$$\min_{\mathbf{x} \in S} \mathbf{f}(\mathbf{x}) \tag{2.3}$$

where $S \subset \mathbb{R}^n$ is a closed set. The set S can be defined by the set of points that satisfies certain inequality and equality constraints.

Considering a real objective function in n variables $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$, the NLP constrained problem ($\mathrm{P_{NLP}}$) with inequality and equality constraints can be written as

$$\min_{\mathbf{x} \in \mathbb{X}} \mathbf{f}(\mathbf{x}) \tag{2.4}$$

subject to

(i) a system of inequalities constraints

$$g_i(\mathbf{x}) \leq 0\,, \quad i = 1, \ldots, m \tag{2.5}$$

and

(ii) a system of equalities constraints

$$h_j(\mathbf{x}) = 0\,, \quad j = 1, \ldots, l \tag{2.6}$$

where $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^l$, $\mathbf{x} \in \mathbb{X} \subseteq \mathbb{R}^n$ and $\mathbb{X}$ is a nonempty open set.

The Fritz John (FJ) optimality conditions are geometric necessary conditions that can be written in terms of the gradients of the objective functions, the inequality constraints and the equality constraints.

Theorem 2.1.4 (The FJ Necessary Conditions)**.** *Let $\mathbb{X}$ be a nonempty open set in $\mathbb{R}^n$, and let*

$$\begin{aligned} &\mathbf{f} : \mathbb{R}^n \to \mathbb{R}\,, \\ &g_i : \mathbb{R}^n \to \mathbb{R} \quad \textit{for } i = 1, \ldots, m\,, \\ &h_j : \mathbb{R}^n \to \mathbb{R} \quad \textit{for } j = 1, \ldots, l\,. \end{aligned}$$

Considering the optimisation problem ($\mathrm{P_{NLP}}$)

$$\begin{aligned} \text{Minimise} \quad & \mathbf{f}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) \leq 0\,, \\ & \mathbf{h}(\mathbf{x}) = 0\,, \\ & \mathbf{x} \in \mathbb{X} \end{aligned}$$

let us suppose that

(i) $\mathbf{x}^*$ *is a feasible solution and let* $I = \{i : g_i(\mathbf{x}^*) = 0\}$,

(ii) each g_i *for* $i \notin I$ *is continuous at* $\mathbf{x}^*$,

(iii) $\mathbf{f}$ *and* g_i *for* $i \in I$ *are differentiable at* $\mathbf{x}^*$,

(iv) each $\mathbf{h}$ *is continuously differentiable at* $\mathbf{x}^*$.

If $\mathbf{x}^*$ *locally solves problem* (2.4)*, then there exist scalars* u_0 *and* u_i *for* $i \in I$*, and* v_j *for* $j = 1, \ldots, l$*, such that*

$$u_0 \nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}^*)^{\mathrm{T}} + \sum_{j=1}^{l} v_j \nabla h_j(\mathbf{x}^*)^{\mathrm{T}} = 0$$

$$u_0, u_i \geq 0 \quad for\ i \in I$$

$$(u_0, \mathbf{u}_I, \mathbf{v}) \neq (0, 0, 0)$$

where $\mathbf{u}_I$ *and* $\mathbf{v}$ *are vectors with components* u_i*,* $i \in I$*, and* v_j*,* $j = 1, \ldots, l$*, respectively. Furthermore, if* g_i*,* $i \notin I$ *are also differentiable at* $\mathbf{x}^*$*, then the above conditions can be written as*

$$u_0 \nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}^*)^{\mathrm{T}} = 0 \tag{2.7}$$

$$u_i g_i(\mathbf{x}^*) = 0 \quad for\ i = 1, \ldots, m \tag{2.8}$$

$$u_0, u_i \geq 0 \quad for\ i = 1, \ldots, m \tag{2.9}$$

$$(u_0, \mathbf{u}, \mathbf{v}) \neq (0, 0, 0) \tag{2.10}$$

where $\mathbf{u}$ *and* $\mathbf{v}$ *are vectors whose components are* u_i*,* $i = 1, \ldots, m$*, and* v_j*,* $j = 1, \ldots, l$*, respectively.*

2.1.3.1 Fritz John Optimality Conditions

In the FJ conditions, the scalars u_0, u_i for $i = 1, \ldots, m$, and v_i for $i = 1, \ldots, l$, are called the *Lagrange multipliers* associated, respectively, with the objective function, the inequality constraints $g_i(x) \leq 0$, $i = 1, \ldots, m$, and the equality constraints $h_j(x) = 0, j = 1, \ldots, l$.

The condition that $\mathbf{x}^*$ be feasible for the optimisation problem ($\mathrm{P_{NLP}}$) is called the *primal feasibility* [PF] condition. The requirements of (2.7) with (2.9) and (2.10) are called the *dual feasibility* [DF] conditions. The condition (2.8) is called the *complementary slackness* [CS] condition. This condition requires that $u_i = 0$ if the corresponding inequality is nonbinding, *i.e.*if $g_i(\mathbf{x}^*) < 0$, and it allows for $u_i > 0$ only

for those constraints that are binding. Together, the [PF], [DF] and [CS] conditions are called the Fritz John (FJ) *optimality conditions*. Any point $\mathbf{x}^*$ for which there exist Lagrange multipliers $\bar{u_0},\ \bar{u_i},\ i=1,\ldots,m,\ \bar{v_i},\ i=1,\ldots,l$ such that the FJ conditions are satisfied is called an *Fritz John (FJ) point*.

The FJ conditions can also be written in vector form as follows:

$$\begin{aligned} \nabla\mathbf{f}(\mathbf{x}^*)^{\mathrm{T}}u_0+\nabla\mathbf{g}(\mathbf{x}^*)^{\mathrm{T}}\mathbf{u}+\nabla\mathbf{h}(\mathbf{x}^*)^{\mathrm{T}}\mathbf{v}&=0\\ \mathbf{u}^{\mathrm{T}}\mathbf{g}(\mathbf{x}^*)&=0\\ (u_0,\mathbf{u})&\geq(0,0)\\ (u_0,\mathbf{u},\mathbf{v})&\neq(0,0,0) \end{aligned} \tag{2.11}$$

where:

(i) $\nabla\mathbf{g}(\mathbf{x}^*)$ is the $m\times n$ Jacobian matrix whose i$^{\text{th}}$ row is $\nabla g_i(\mathbf{x}^*)$,

(ii) $\nabla\mathbf{h}(\mathbf{x}^*)$ is the $l\times n$ Jacobian matrix whose i$^{\text{th}}$ row is $\nabla h_i(\mathbf{x}^*)$,

(iii) $\mathbf{g}(\mathbf{x}^*)$ is the m vector function whose i$^{\text{th}}$ component is $g_i(\mathbf{x}^*)$, and

(iv) $\mathbf{u}$ and $\mathbf{v}$ are, respectively, an m vector and an l vector, whose elements are the Lagrange multipliers associated with, respectively, the inequality and equality constraints.

2.1.3.2 Karush–Kuhn–Tucker Conditions

From the FJ optimality conditions, it is possible to derive the following Karush–Kuhn–Tucker (KKT) necessary conditions for optimality. Note that when $u_0>0$, we can assume $u_0=1$ without loss of generality, by scaling the dual feasibility conditions.

Theorem 2.1.5 (KKT Necessary Conditions)**.** *Let $\mathbb{X}$ be a nonempty open set in $\mathbb{R}^n$, and let*

$$\mathbf{f}:\mathbb{R}^n\to\mathbb{R},$$

$$g_i : \mathbb{R}^n \to \mathbb{R} \quad \textit{for } i = 1, \ldots, m,$$
$$h_j : \mathbb{R}^n \to \mathbb{R} \quad \textit{for } j = 1, \ldots, l.$$

Let us consider the optimisation problem ($\mathrm{P_{NLP}}$)

$$\begin{aligned} \text{Minimise} \quad & \mathbf{f}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) \leq 0, \\ & \mathbf{h}(\mathbf{x}) = 0, \\ & \mathbf{x} \in \mathbb{X}. \end{aligned}$$

Let $\mathbf{x}^*$ *be a solution, and let* $I = \{i : g_i(\mathbf{x}^*) = 0\}$. *Suppose that*

(i) $\mathbf{f}$ *and* g_i *for* $i \in I$ *are differentiable at* $\mathbf{x}^*$,

(ii) each g_i *for* $i \notin I$ *is continuous at* $\mathbf{x}^*$,

(iii) each h_i *for* $i = 1, \ldots, l$, *is continuously differentiable at* $\mathbf{x}^*$.

In addition, suppose that $\nabla g_i(\mathbf{x}^*)^{\mathrm{T}}$ *for* $i \in I$ *and* $\nabla h_i(\mathbf{x}^*)^{\mathrm{T}}$ *for* $i = 1, \ldots, l$ *are linearly independent. If* $\mathbf{x}^*$ *is a local optimal solution, then there exist unique scalars* u_i *for* $i \in I$,*and* v_i *for* $i = 1, \ldots, l$, *such that*

$$\nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}^*)^{\mathrm{T}} = 0 \tag{2.12}$$
$$u_i \geq 0 \quad \textit{for } i \in I$$

Furthermore, if $g_i, i \notin I$ *are also differentiable at* $\mathbf{x}^*$, *then the above conditions can be written as*

$$\nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}^*)^{\mathrm{T}} = 0$$
$$u_i g_i(\mathbf{x}^*) = 0 \quad \textit{for } i = 1, \ldots, m \tag{2.13}$$
$$u_i \geq 0 \quad \textit{for } i = 1, \ldots, m$$

The KKT conditions can also be written in vector form

$$\begin{aligned} \nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \nabla \mathbf{g}(\mathbf{x}^*)^{\mathrm{T}}\mathbf{u} + \nabla \mathbf{h}(\mathbf{x}^*)^{\mathrm{T}}\mathbf{v} &= 0\,, \\ \mathbf{u}^{\mathrm{T}} g(\mathbf{x}^*) &= 0\,, \\ \mathbf{u} &\geq 0\,, \end{aligned}$$

where

(i) $\nabla \mathbf{g}(\mathbf{x}^*)$ is the $m \times n$ Jacobian matrix whose i$^{\text{th}}$ row is $\nabla g_i(\mathbf{x}^*)$,

(ii) $\nabla \mathbf{h}(\mathbf{x}^*)$ is the $l \times n$ Jacobian matrix whose i$^{\text{th}}$ row is $\nabla h_i(\mathbf{x}^*)$,

(iii) $\mathbf{g}(\mathbf{x}^*)$ is the m vector function whose i$^{\text{th}}$ component is $g_i(\mathbf{x}^*)$, and

(iv) $\mathbf{u}$ and $\mathbf{v}$ are, respectively, an m vector and an l vector, whose elements are the Lagrange multipliers associated with, respectively, the inequality and equality constraints.

The difference between the FJ and KKT conditions is that we can guarantee the $u_0 = 1$ when $\nabla g_i(\mathbf{x}^*)$, for $i \in I$, and $\nabla h_i(\mathbf{x}^*)$ are linearly independent – a condition called constraint qualification.

The following result shows that, under moderate convexity assumptions, the KKT conditions are also sufficient for local optimality.

Theorem 2.1.6 (KKT Sufficient Conditions)**.** *Let $\mathbb{X}$ be a nonempty open set in $\mathbb{R}^n$, and let*

$$\begin{aligned} \mathbf{f} &: \mathbb{R}^n \to \mathbb{R}\,, \\ g_i &: \mathbb{R}^n \to \mathbb{R} \quad \textit{for } i = 1, \ldots, m\,, \\ h_j &: \mathbb{R}^n \to \mathbb{R} \quad \textit{for } j = 1, \ldots, l\,. \end{aligned}$$

Let $\mathbf{x}^*$ be a feasible solution, and let $I = \{i : g_i(\mathbf{x}^*) = 0\}$. Suppose that the KKT conditions hold at $\mathbf{x}^*$, i.e., there exist scalars $\bar{u}_i \geq 0$ for $i \in I$, and $\bar{v}_i$ for $i = 1, \ldots, l$,

such that

$$\nabla \mathbf{f}(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}^*)^{\mathrm{T}} + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}^*)^{\mathrm{T}} = 0 \tag{2.14}$$

$$u_i \geq 0 \quad for\ i \in I$$

Let $J = \{i : \bar{v}_i > 0\}$ *and* $K = \{i : \bar{v}_i < 0\}$. *Furthermore, suppose that*

(i) $\mathbf{f}$ *is pseudoconvex at* $\mathbf{x}^*$,

(ii) g_i *is quasiconvex at* $\mathbf{x}^*$ *for* $i \in I$,

(iii) h_i *is quasiconvex at* $\mathbf{x}^*$ *for* $i \in J$, *and*

(iv) h_i *is quasiconcave at* $\mathbf{x}^*$ *(that is,* $-h_i$ *is quasiconvex at* $\mathbf{x}^*$*) for* $i \in K$.

Then $\mathbf{x}^*$ *is a global optimal solution to problem* ($\mathrm{P_{NLP}}$). *In particular, if these generalised convexity assumptions on the objective and constraint functions are restricted to the domain* $N_\varepsilon(\mathbf{x}^*)$ *for some* $\varepsilon > 0$, *then* $\mathbf{x}^*$ *is a local minimiser for problem* ($\mathrm{P_{NLP}}$).

2.2 Optimal Control

2.2.1 Discrete Optimal Control Problem

Let us consider the following fixed horizon OCP, in discrete form, ($\mathrm{P_{DOC}}$), with input and state constraints

$$\text{Minimise } \mathcal{J}_N\left(\{t_k\}, \{\mathbf{x}_k\}, \{\mathbf{u}_k\}\right) = \mathcal{G}\left(\mathbf{x}_N\right) + \sum_{k=0}^{N-1} \mathcal{L}\left(t_k, \mathbf{x}_k, \mathbf{u}_k\right) \tag{2.15}$$

subject to

(i) the dynamic constraints

$$\mathbf{x}_{k+1} = \mathbf{f}(t_k, \mathbf{x}_k, \mathbf{u}_k) \quad \text{for } k = 0, \ldots, N-1\,, \tag{2.16}$$

(ii) the input constraints

$$\mathbf{u}_k \in \mathbb{U} \subset \mathbb{R}^m \quad \text{for } k = 0, \ldots, N-1\,, \tag{2.17}$$

(iii) the state constraints

$$\mathbf{x}_k \in \mathbb{X} \subset \mathbb{R}^n \quad \text{for } k = 0, \ldots, N\,, \tag{2.18}$$

(iv) and the end–point constraints

$$\mathbf{x}_0 \in \mathbb{X}_0 \subset \mathbb{R}^n \quad \text{and} \quad \mathbf{x}_N \in \mathbb{X}_1 \subset \mathbb{R}^n \tag{2.19}$$

where N is the optimisation horizon, $\{t_k\}$, $\{\mathbf{x}_k\}$ and $\{\mathbf{u}_k\}$ are time, state and control sequences. The functions invoked comprise

(i) the objective function $\mathcal{J}_N(\{t_k\}, \{\mathbf{x}_k\}, \{\mathbf{u}_k\})$,

(ii) the running cost $\mathcal{L} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$,

(iii) the terminal cost $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}$, and

(iv) the dynamic function $\mathbf{f} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$.

Usually, $\mathbb{U}$ is compact, and $\mathbb{X}$, $\mathbb{X}_0$ and $\mathbb{X}_1$ are closed.

In the context of optimisation, a trajectory $\{\mathbf{x}_k\}$ is obtained iteratively considering an initial value $\mathbf{x}_0 \in \mathbb{X}_0$ and a control sequence $\mathbf{u}(\cdot) \in \mathbb{U}$. The constraints provided by the system of equations $\mathbf{f}$ that must be satisfied for all time instants k in the time interval. The state and control sequences that attain the minimum are the optimal sequences or minimisers.

2.2.2 Necessary Conditions of Optimality: Discrete Maximum Principle

The conditions required on the data of the optimisation problem (2.15)–(2.19) assume the following assumptions:

H'1. The function $\mathcal{G}(\mathbf{x})$ is twice–continuously differentiable.

H'2. For every $\mathbf{u} \in \mathbb{U}$, the functions $\mathbf{f}(t, \mathbf{x}, \mathbf{u})$ and $\mathcal{L}(t, \mathbf{x}, \mathbf{u})$ are twice–continuously differentiable with respect to $\mathbf{x}$ and t.

H'3. The terminal constraint function $\mathbf{h}(\mathbf{x})$ is twice–continuously differentiable and satisfies the "constraint qualification" that the Jacobian matrix $\partial \mathbf{h}(\mathbf{x})/\partial \mathbf{x}$ has full row rank for all $\mathbf{x} \in \mathbb{R}^n$.

H'4. The functions $\mathbf{f}(t, \mathbf{x}, \mathbf{u})$ and $\mathcal{L}(t, \mathbf{x}, \mathbf{u})$, and all their first and second partial derivatives with respect to $\mathbf{x}$, are uniformly bounded on $A \times \mathbb{U}$ for any bounded set $A \subset \mathbb{R}^n$.

H'5. The matrix $\partial \mathbf{f}(\cdot, \cdot, \cdot)/\partial \mathbf{x}$ is nonsingular on $\mathbb{R}^n \times \mathbb{U}$.

H'6. The set $\left\{ \begin{bmatrix} \mathbf{f}(t, \mathbf{x}, \mathbf{u}) \\ \mathcal{L}(t, \mathbf{x}, \mathbf{u}) \end{bmatrix} : \mathbf{u} \in \mathbb{U} \right\}$, is convex for all $\mathbf{x} \in \mathbb{R}^n$.

Let us consider the Hamiltonian

$$\mathcal{H}\left(k, \mathbf{x}_k, \eta_k, \mathbf{u}_k, \lambda\right) = \eta_k \cdot \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) - \lambda \mathcal{L}\left(k, \mathbf{x}_k, \mathbf{u}_k\right) \tag{2.20}$$

where λ is a real number and η_k, $k = 0, \ldots, N-1$ are vector in $\mathbb{R}^n$.

The Maximum Principle, in discrete–time, for state constrained problems stated in [GSD06]:

Theorem 2.2.1. *Subject to assumptions H'1–H'6, if the sequences* $\{x_0, \ldots, x_N\}$, $\{u_0, \ldots, u_{N-1}\}$ *are minimisers of problem* ($\mathrm{P_{DOC}}$)*, then there exist a sequence of vectors* $\{\eta_{-1}, \eta_0, \ldots, \eta_{N-1}\}$ *and a real number* λ *such that the following conditions hold:*

(i) Adjoint equations

$$\left(\eta_{k-1}^*\right)^{\mathrm{T}} = \frac{\partial \mathcal{H}\left(\mathbf{x}_k, \eta_k, \mathbf{u}_k, \lambda\right)}{\partial \mathbf{x}_k} \quad \textit{for } k = 0, \ldots, N-1\,. \tag{2.21}$$

(ii) Boundary conditions: There exists a real number $\beta \geq 0$ *and a vector* $\gamma \in \mathbb{R}^l$*, such that*

$$\eta^*_{N-1} = \left[\frac{\partial h}{\partial \mathbf{x}}\right]^{\mathrm{T}} \gamma + \left[\frac{\partial \mathcal{G}}{\partial \mathbf{x}}\right]^{\mathrm{T}} \beta\,, \tag{2.22}$$

$$\lambda^* = \beta \geq 0\,, \tag{2.23}$$

where λ^* *and* η^*_{N-1} *are not simultaneously zero. Moreover, if* $\lambda^* = 0$ *in* (2.23)*, then the vectors* $\left\{\eta^*_{-1}, \ldots, \eta^*_{N-1}\right\}$ *satisfying* (2.21) *and* (2.22) *are all nonzero.*

(iii) Maximisation of the Hamiltonian

$$\mathcal{H}\left(k, \mathbf{x}^*_k, \eta^*_k, \mathbf{u}^*_k, \lambda^*\right) \geq \mathcal{H}\left(k, \mathbf{x}^*_k, \eta^*_k, \mathbf{u}, \lambda^*\right), \tag{2.24}$$

for all $k = 0, \ldots, N-1$ *and all* $\mathbf{u} \in \mathbb{U}$*.*

2.2.3 Relationship Between the Maximum Principle and the Fritz–John and Karush–Kuhn–Tucker conditions

For further discussion, we intend to analyse the relationship between the Maximum Principle and the Fritz–John and Karush–Kuhn–Tucker conditions [GSD06]. Let us consider the following fixed horizon OCP, in discrete form, with input and state constraints

$$\mathrm{P_N}(\bar{\mathbf{x}}): \quad \text{Minimise } \mathcal{J}_N\left(\{k\}, \{\mathbf{x}_k\}, \{\mathbf{u}_k\}\right) \tag{2.25}$$

$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) \quad \text{for } k = 0, \ldots, N-1\,, \tag{2.26}$$

$$\mathbf{x}_0 = \bar{\mathbf{x}}\,, \tag{2.27}$$

$$g_k(\mathbf{u}_k) \leq 0 \quad k = 0, \ldots, N-1\,, \tag{2.28}$$

$$g_N(\mathbf{x}_N) \leq 0\,, \tag{2.29}$$

$$h_N(\mathbf{x}_N) = 0\,, \tag{2.30}$$

where

$$\mathcal{J}_N\left(\{k\}, \{\mathbf{x}_k\}, \{\mathbf{u}_k\}\right) = \mathcal{G}\left(\mathbf{x}_N\right) + \sum_{k=0}^{N-1} \mathcal{L}\left(k, \mathbf{x}_k, \mathbf{u}_k\right) \tag{2.31}$$

and

(i) $\{k\} \triangleq \{0, \ldots, N\}$ is the time sequence,

(ii) $\{\mathbf{x}_k\} \triangleq \{\mathbf{x}_0, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_k \in \mathbb{R}^n$ is the state sequence,

(iii) $\{\mathbf{u}_k\} \triangleq \{\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}\}$, $\mathbf{u}_k \in \mathbb{R}^m$ is the control sequence,

(iv) (2.26)–(2.27) are the state equations,

(v) $g_k : \mathbb{R}^m \to \mathbb{R}^r$, $k = 0, \ldots, N-1$, represent r (elementwise) inequality constraints (compare against (2.17)),

(vi) $g_N : \mathbb{R}^n \to \mathbb{R}^p$ and $h_N : \mathbb{R}^n \to \mathbb{R}^l$ represent, respectively, inequality and equality constraints on the terminal state (compare against (2.18) where only equality constraints on the terminal state were considered).

We will assume that all functions in (2.25)–(2.31) are differentiable functions of their variables and that $\mathbf{f}$ and h_N are continuously differentiable at the optimal solution.

In [GSD06], necessary optimality conditions for the sequences $\{\mathbf{x}_0^*, \ldots, \mathbf{x}_N^*\}$ and $\{\mathbf{u}_0^*, \ldots, \mathbf{u}_{N-1}^*\}$ to be minimisers of the optimisation problem $\mathrm{P_N}(\bar{\mathbf{x}})$ are derived, using the FJ necessary optimality conditions (see Theorem 2.1.4). Note that the FJ conditions are always a necessary condition for optimality under the differentiability assumption, without requiring any constraint qualification.

Defining the vector

$$\mathcal{X} \triangleq \begin{bmatrix} \mathbf{x}_0^{\mathrm{T}} & \ldots & \mathbf{x}_N^{\mathrm{T}} & \mathbf{u}_0^{\mathrm{T}} & \ldots & \mathbf{u}_{N-1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{(N+1)n+Nm} \tag{2.32}$$

we can rewrite the problem (2.25)–(2.31) in the form

$$\begin{aligned} \text{Minimise} \quad & \phi(k, \mathcal{X}) \\ \text{subject to} \quad & g(\mathcal{X}) \leq 0 \\ & h(\mathcal{X}) = 0 \end{aligned} \tag{2.33}$$

where

$$\phi(k,\mathcal{X}) \triangleq \mathcal{G}\left(\mathbf{x}_N\right) + \sum_{k=0}^{N-1} \mathcal{L}\left(k,\mathbf{x}_k,\mathbf{u}_k\right), \tag{2.34}$$

$$g(\mathcal{X}) \triangleq \begin{bmatrix} g_0(\mathbf{u}_0) \\ g_1(\mathbf{u}_1) \\ \vdots \\ g_{N-1}(\mathbf{u}_{N-1}) \\ g_N(\mathbf{x}_N) \end{bmatrix}, \tag{2.35}$$

$$h(\mathcal{X}) \triangleq \begin{bmatrix} \bar{\mathbf{x}} - \mathbf{x}_0 \\ \mathbf{f}(0,\mathbf{x}_0,\mathbf{u}_0) - \mathbf{x}_1 \\ \vdots \\ \mathbf{f}(N-1,\mathbf{x}_{N-1},\mathbf{u}_{N-1}) - \mathbf{x}_N \\ h_N(\mathbf{x}_N) \end{bmatrix}. \tag{2.36}$$

Supposing

$$\mathcal{X}^* = \begin{bmatrix} \mathbf{x}_0^{*\mathrm{T}} & \dots & \mathbf{x}_N^{*\mathrm{T}} & \mathbf{u}_0^{*\mathrm{T}} & \dots & \mathbf{u}_{N-1}^{*\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{2.37}$$

is a minimiser of (2.33), the FJ conditions (see Theorem 2.1.4) hold for problem (2.33) at $\mathcal{X}^*$, that is, there exist a scalar λ^* and vectors $\{\eta_{-1}^*,\dots,\eta_{N-1}^*\}$, γ^* and $\{\nu_0^*,\dots,\nu_N^*\}$ such that

$$\left[\frac{\partial \phi(k,\mathcal{X}^*)}{\partial \mathcal{X}}\right]^{\mathrm{T}} \lambda^* + \left[\frac{\partial h(\mathcal{X}^*)}{\partial \mathcal{X}}\right]^{\mathrm{T}} \begin{bmatrix} \eta_{-1}^* \\ \vdots \\ \eta_{N-1}^* \\ \gamma^* \end{bmatrix} + \left[\frac{\partial g(\mathcal{X}^*)}{\partial \mathcal{X}}\right]^{\mathrm{T}} \begin{bmatrix} \nu_1^* \\ \vdots \\ \nu_N^* \end{bmatrix} = 0, \tag{2.38}$$

$$\begin{bmatrix} \nu_1^* \\ \vdots \\ \nu_N^* \end{bmatrix} g(\mathcal{X}^*) = 0, \tag{2.39}$$

$$(\lambda^*,\nu_0^*,\dots,\nu_N^*) \geq 0, \tag{2.40}$$

$$\left(\lambda^*,\eta_{-1}^*,\dots,\eta_{N-1}^*,\nu_0^*,\dots,\nu_N^*\right) \neq 0, \tag{2.41}$$

where

$$\frac{\partial \phi}{\partial \mathcal{X}} = \begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_0} & \cdots & \dfrac{\partial \mathcal{L}}{\partial \mathbf{x}_{N-1}} & \dfrac{\partial \mathcal{G}}{\partial \mathbf{x}_N} & \dfrac{\partial \mathcal{L}}{\partial \mathbf{u}_0} & \cdots & \dfrac{\partial \mathcal{L}}{\partial \mathbf{u}_{N-1}} \end{bmatrix}, \tag{2.42}$$

$$\frac{\partial h}{\partial \mathcal{X}} = \begin{bmatrix} -I_n & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{\partial \mathbf{f}}{\partial \mathbf{x}_0} & -I_n & 0 & \cdots & 0 & 0 & \frac{\partial \mathbf{f}}{\partial \mathbf{u}_0} & 0 & \cdots & 0 \\ 0 & \frac{\partial \mathbf{f}}{\partial \mathbf{x}_1} & -I_n & \cdots & 0 & 0 & 0 & \frac{\partial \mathbf{f}}{\partial \mathbf{u}_1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{\partial \mathbf{f}}{\partial \mathbf{x}_{N-1}} & -I_n & 0 & 0 & \cdots & \frac{\partial \mathbf{f}}{\partial \mathbf{u}_{N-1}} \\ 0 & 0 & 0 & \cdots & 0 & \frac{\partial h_N}{\partial \mathbf{x}_N} & 0 & 0 & \cdots & 0 \end{bmatrix}, \tag{2.43}$$

$$\frac{\partial g}{\partial \mathcal{X}} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 & \frac{\partial g_0}{\partial \mathbf{u}_0} & 0 & \cdots & 0 \\ \vdots & \ddots & & & \vdots & 0 & \frac{\partial g_1}{\partial \mathbf{u}_1} & \cdots & 0 \\ \vdots & & \ddots & & \vdots & \vdots & & \ddots & \vdots \\ \vdots & & & \ddots & \vdots & 0 & 0 & \cdots & \frac{\partial g_{N-1}}{\partial \mathbf{u}_{N-1}} \\ 0 & \cdots & \cdots & \cdots & \frac{\partial g_N}{\partial \mathbf{x}_N} & 0 & 0 & \cdots & 0 \end{bmatrix}, \tag{2.44}$$

Defining the Hamiltonian

$$\mathcal{H}(k, \mathbf{x}_k, \eta_k, \mathbf{u}_k, \lambda) \triangleq \eta_k \cdot \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) - \lambda \mathcal{L}(k, \mathbf{x}_k, \mathbf{u}_k), \tag{2.45}$$

and writing the FJ conditions (2.38)–(2.41) component–wise, we conclude that in order for the sequences $\{\mathbf{x}_0^*, \ldots, \mathbf{x}_N^*\}$ and $\{\mathbf{u}_0^*, \ldots, \mathbf{u}_{N-1}^*\}$ to be minimisers of (2.25)–(2.31), it is a necessary condition to exist a scalar λ^* and vectors $\{\eta_{-1}^*, \ldots, \eta_{N-1}^*\}$, γ^* and $\{\nu_0^*, \ldots, \nu_N^*\}$, not all zero, such that the following conditions hold:

(i) Adjoint equations:

$$\left(\eta_{k-1}^*\right)^{\mathrm{T}} = \frac{\partial \mathcal{H}(k, \mathbf{x}_k^*, \eta_k, \mathbf{u}_k^*, \lambda)}{\partial \mathbf{x}_k} \quad \text{for } k = 0, \ldots, N-1, \tag{2.46}$$

(ii) Boundary conditions:

$$\eta_{N-1}^* = \left[\frac{\partial h_N(\mathbf{x}_N^*)}{\partial \mathbf{x}_N}\right]^{\mathrm{T}} \gamma^* + \left[\frac{\partial \mathcal{G}(\mathbf{x}_N^*)}{\partial \mathbf{x}_n}\right]^{\mathrm{T}} \lambda^* + \left[\frac{\partial g_N(\mathbf{x}_N^*)}{\partial \mathbf{x}_N}\right]^{\mathrm{T}} \nu^*, \tag{2.47}$$

$$\left(\nu_N^*\right)^{\mathrm{T}} g_N(\mathbf{x}_N^*) = 0, \tag{2.48}$$

$$\lambda^* \geq 0\,, \tag{2.49}$$

$$\nu_N^* \geq 0\,, \tag{2.50}$$

(iii) Hamiltonian conditions:

$$\left[\frac{\partial \mathcal{H}\,(k, \mathbf{x}_k^*, \eta_k^*, \mathbf{u}_k^*, \lambda^*)}{\partial \mathbf{u}_k}\right]^{\mathrm{T}} + \left[\frac{\partial g_k(\mathbf{x}_k^*)}{\partial \mathbf{u}_k}\right]^{\mathrm{T}} \nu^* = 0\,, \tag{2.51}$$

$$(\nu_k^*)^{\mathrm{T}}\, g_k(\mathbf{u}_k^*) = 0\,, \tag{2.52}$$

$$\nu_k^* \geq 0\,, \tag{2.53}$$

for $k = 0, \ldots, N-1$.

Let us consider the following related condition

$$\mathcal{H}\,(k, \mathbf{x}_k^*, \eta_k^*, \mathbf{u}_k^*, \lambda^*) \geq \mathcal{H}\,(k, \mathbf{x}_k^*, \eta_k^*, \mathbf{u}_k, \lambda^*) \quad \text{for all } \mathbf{u}_k \text{ such that } g_k(\mathbf{u}_k) \leq 0\,. \tag{2.54}$$

Notice that the KKT conditions for (2.54) coincide with (2.51)–(2.53) (compare with (2.14)). However, in order to guarantee that (2.54) is a necessary condition for (2.51)–(2.53), and hence for the original problem (2.15)–(2.19), we need additional moderate convexity assumptions. Suppose now that $\mathcal{H}\,(k, \mathbf{x}_k^*, \eta_k^*, \mathbf{u}_k, \lambda^*)$ is pseudoconvex at $\mathbf{u}_k^*$, and the constraint function $g_k(\mathbf{u}_k)$ in (2.54) is quasiconvex at $\mathbf{u}_k^*$. We can then apply the KKT sufficient optimality conditions of Theorem 2.1.6 to conclude that conditions (2.51)–(2.53) imply (2.54).

For the original optimisation problem (2.15)–(2.19), under the above (generalised) convexity assumptions, a necessary condition for the sequences $\{\mathbf{x}_0^*, \ldots, \mathbf{x}_N^*\}$ and $\{\mathbf{u}_0^*, \ldots, \mathbf{u}_{N-1}^*\}$ to be minimisers is that there exist a scalar λ^* and vectors $\{\eta_{-1}^*, \ldots, \eta_{N-1}^*\}$, γ^* and $\{\nu_0^*, \ldots, \nu_N^*\}$, not all zero, such that conditions (2.46)–(2.50) hold, and, furthermore, $\mathbf{u}^*$ minimises the Hamiltonian for $k = 0, \ldots, N-1$.

Moreover, we can apply the KKT necessary optimality conditions to the original problem, that is, we can set $\lambda = 1$ in the FJ conditions (2.38)–(2.41) and in the Hamiltonian (2.45).

2.2.4 Dynamic Programming and Sufficient Conditions for Global Optimum

Dynamic Programming (DP) is a technique which compares the optimal decision with all the other decisions. This global comparison, therefore, leads to optimality conditions which are sufficient. The main advantage of DP, besides the fact that it gives sufficiency conditions, is that DP permits very general problem formulations which do not require differentiability or convexity conditions or even the restriction to a finite–dimensional state space. The only disadvantage of DP is that it can easily give rise to enormous computational requirements.

Let us consider the following discrete OCP

$$\begin{aligned} \text{Minimise} \quad & \mathcal{G}(\mathbf{x}_N) + \sum_{k=0}^{N-1} \mathcal{L}(k, \mathbf{x}_k, \mathbf{u}_k) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) \quad \text{for } k = 0, \ldots, N-1, \\ & \mathbf{u}_k \in \mathbb{U} \subset \mathbb{R}^m \quad \text{for } k = 0, \ldots, N-1, \\ & \mathbf{x}_k \in \mathbb{X} \subset \mathbb{R}^n \quad \text{for } k = 0, \ldots, N, \\ & \mathbf{x}(0) = \mathbf{x}_0 . \end{aligned} \tag{2.55}$$

The main idea underlying DP involves embedding the optimal control problem (2.55), in which the system starts in state $\mathbf{x}_0$ at time 0, into a family of optimal control problems with the same dynamics, objective function, and control constraint as in (2.55) but with different initial states and initial times. More precisely, for each $\mathbf{x} \in \mathbb{X}$ and j between 0 and $N-1$, consider the following problem:

$$\begin{aligned} (\mathrm{P}_{\mathrm{DP}}): \quad \text{Minimise} \quad & \mathcal{G}(\mathbf{x}_N) + \sum_{k=j}^{N-1} \mathcal{L}(k, \mathbf{x}_k, \mathbf{u}_k) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) \quad \text{for } k = j, j+1, \ldots, N-1, \\ & \mathbf{u}_k \in \mathbb{U} \subset \mathbb{R}^m \quad \text{for } k = j, j+1, \ldots, N-1, \\ & \mathbf{x}(j) = \mathbf{x} . \end{aligned}$$

Since the initial time j and initial state $\mathbf{x}$ are the only parameters in the problem above, we use $(\mathrm{P_{DP}})^{j,\mathbf{x}}$ to distinguish between different problems.

The following Lemma is an elementary but crucial observation [Var72].

Lemma 2.2.2. *Suppose $\mathbf{u}_j^*, \ldots, \mathbf{u}_{N-1}^*$ is an optimal control for $(\mathrm{P_{DP}})$ and let $\mathbf{x}_j^* = \mathbf{x}^*, \mathbf{x}_{j+1}^*, \ldots, \mathbf{x}_N^*$ be the corresponding optimal trajectory. Then for any l, $j \le l \le N-1$, $\mathbf{u}_l^*, \ldots, \mathbf{u}_{N-1}^*$ is an optimal control for $(\mathrm{P_{DP}})^{l,\mathbf{x}_l^*}$.*

Let us assume that an optimal solution to $(\mathrm{P_{DP}})^{j,\mathbf{x}}$ exists for all $0 \ge j \ge N-1$, and all $\mathbf{x} \in \mathbb{X}$. Let $\mathcal{V}(j,\mathbf{x})$ be the maximum value of $(\mathrm{P_{DP}})^{j,\mathbf{x}}$. $\mathcal{V}$ is called the *value function*.

Theorem 2.2.3. *Let us define $\mathcal{V}(N,\cdot)$ by $\mathcal{V}(N,\mathbf{x}) = \mathcal{G}(\mathbf{x}_N)$. $\mathcal{V}(j,\mathbf{x})$ satisfies the backward recursion equation*

$$\mathcal{V}(j,\mathbf{x}) = \min\left\{\mathcal{L}(j,\mathbf{x}_j,\mathbf{u}_j) + \mathcal{V}\left(j+1,\mathbf{f}\left(j,\mathbf{x}_j,\mathbf{u}_j\right)\right) | \mathbf{u} \in \mathbb{U}\right\}, \quad 0 \le j \le N-1. \tag{2.56}$$

Corollary 1. *Let $\mathbf{u}_l, \ldots, \mathbf{u}_{N-1}$ be any control for the problem $(\mathrm{P_{DP}})^{j,\mathbf{x}}$ and let $\mathbf{x}_j = \mathbf{x}, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_N$ be the corresponding trajectory. Then*

$$\mathcal{V}(l,\mathbf{x}_l) \le \mathcal{L}\left(l,\mathbf{x}_l,\mathbf{u}_l\right) + \mathcal{V}\left(l+1,\mathbf{f}\left(l,\mathbf{x}_l,\mathbf{u}_l\right)\right), \quad j \le l \le N-1, \tag{2.57}$$

and equality holds for all $j \le l \le N-1$ if and only if the control is optimal for $(\mathrm{P_{DP}})^{j,\mathbf{x}}$.

Corollary 2. *For $j = 0,1,\ldots,N-1$, let $\overline{\mathbf{u}}(j,\cdot) : \mathbb{R}^m \to \mathbb{U}$ be such that*

$$\mathcal{L}\left(j,\mathbf{x}_j,\overline{\mathbf{u}}(j,\mathbf{x}_j)\right) + \mathcal{V}(j+1,\mathbf{f}\left(j,\mathbf{x}_j,\overline{\mathbf{u}}(j,\mathbf{x}_j)\right) = \min\left\{\mathcal{L}\left(j,\mathbf{x}_j,\mathbf{u}_j\right)\right) + \mathcal{V}(j+1,\mathbf{f}(j,\mathbf{x}_j,\mathbf{u}_j) | \mathbf{u} \in \mathbb{U}\right\}.$$

Then $\overline{\mathbf{u}}(j,\cdot)$, $j = 0,1,\ldots,N-1$, is an optimal feedback control, i.e., for any $(j,\mathbf{x}_j)$ the control $\mathbf{u}_j^*, \ldots, \mathbf{u}_{N-1}^*$ defined by $\mathbf{u}_l^* = \overline{\mathbf{u}}(l,\mathbf{x}_l^*)$, $j \le l \le N-1$, where

$$\begin{aligned} \mathbf{x}_{l+1}^* &= \mathbf{f}\left(l,\mathbf{x}_l^*,\mathbf{u}_l^*\right), \quad j \le l \le N-1, \\ \mathbf{x}_l^* &= \mathbf{x}, \end{aligned}$$

is optimal for $(\mathrm{P_{DP}})^{j,\mathbf{x}}$.

2.2.5 Continuous Optimal Control Problem

An OCP [Vin00] has as main goal to find dynamic variables subject to constraints and bounds, which minimises a certain cost function [FH10]. Let us consider the following optimal control problem, in Bolza form, with input and state constraints [Vin00]:

$$\text{Minimise } \mathrm{J}(t,\mathbf{x},\mathbf{u}) = \int_{t_0}^{t_f} \mathrm{L}\left(t,\mathbf{x}(t),\mathbf{u}(t)\right)\mathrm{d}t + \mathrm{G}\left(\mathbf{x}(t_f)\right)$$

subject to

(i) dynamic constraints

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t,\mathbf{x}(t),\mathbf{u}(t)) \quad \text{a.e. } t \in \left[t_0,t_f\right],$$

(ii) input constraints

$$\mathbf{u}(t) \in \mathbb{U} \subset \mathbb{R}^m \quad \text{a.e. } t \in \left[t_0,t_f\right],$$

(iii) pathwise state constraint

$$\mathbf{h}\left(\mathbf{x}(t)\right) \leq 0 \quad \forall t \in \left[t_0,t_f\right],$$

and

(iv) end–point constraints

$$\mathbf{x}(t_0) \in \mathbb{X}_0 \subset \mathbb{R}^n \quad \text{and} \quad \mathbf{x}(t_f) \in \mathbb{X}_1 \subset \mathbb{R}^n,$$

where $\mathbf{x} : \left[t_0,t_f\right] \to \mathbb{R}^n$ is the state, $\mathbf{u} : \left[t_0,t_f\right] \to \mathbb{R}^m$ is the control and $t \in \left[t_0,t_f\right]$ is time. The functions involved comprise the running cost $\mathrm{L} : \left[t_0,t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, the terminal cost $\mathrm{G} : \mathbb{R}^n \to \mathbb{R}$, the dynamic function $\mathbf{f} : \left[t_0,t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and the state constraint $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^k$.

2.2.6 Necessary Conditions of Optimality: Maximum Principle

We use a smooth version of the Maximum Principle for state constrained problems which is valid under the following hypotheses. There exists a scalar $\xi > 0$ such that:

H1. $\mathbf{f}(\cdot, \mathbf{x}, \cdot)$ is $\mathcal{L} \times \mathcal{B}^m$ measurable for fixed $\mathbf{x}$;

H2. $\mathbf{f}(t, \cdot, \mathbf{u})$ is continuously differentiable on $\bar{\mathbf{x}} + \xi\,\mathbb{B}$, $\forall \mathbf{u} \in \mathbb{U}$, a.e. $t \in \left[t_0, t_f\right]$;

H3. There exists $C_u > 0$ such that $||\mathbf{f}(t, \mathbf{x}, \cdot)|| \leq C_u$ for $\bar{\mathbf{x}} + \xi\,\mathbb{B}$, $\forall \mathbf{u} \in \mathbb{U}$, a.e. $t \in \left[t_0, t_f\right]$;

H4. G is continuously differentiable on $\bar{\mathbf{x}} + \xi\,\mathbb{B}$;

H5. $\mathbb{U}$ is compact;

H6. $\mathbf{h}$ is continuously differentiable on $\bar{\mathbf{x}} + \xi\,\mathbb{B}$.

where $\mathbb{B}$ denotes the closed unit ball.

A feasible process $(\mathbf{x}^*, \mathbf{u}^*)$ is a $W^{1,1}$ *local minimizer* if there exists $\delta > 0$ such that $(\mathbf{x}^*, \mathbf{u}^*)$ minimizes $\mathrm{J}(t, \mathbf{x}, \mathbf{u})$ for all feasible processes $(\mathbf{x}, \mathbf{u})$ which satisfy $\| \mathbf{x} - \mathbf{x}^* \|_{W^{1,1}} \leq \delta$ [Vin00].

Let us consider the Hamiltonian

$$\mathrm{H}\left(t, \mathbf{x}(t), \mathbf{p}(t), \mathbf{u}(t)\right) = \mathbf{p}(t) \cdot \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) - \lambda \mathrm{L}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right),$$

the Maximum Principle for state constrained problems in [Vin00, p. 329], and its remark *d*) in page 331.

Theorem 2.2.4. *Let* $(\mathbf{x}^*, \mathbf{u}^*)$ *be a* $W^{1,1}$ *local minimizer of OCP. Assume hypotheses (H1)–(H6) are satisfied. Then, there exist an absolutely continuous function* $\mathbf{p} \in W^{1,1}(\left[t_0, t_f\right]; \mathbb{R}^n)$*, a scalar* $\lambda \geq 0$ *and positive Radon measures* $\mu_i \in C^{\oplus}\left(\left[t_0, t_f\right]\right), i = 1, \ldots, k$ *satisfying*

(i) the nontriviality condition

$$(\mathbf{p}, \mu, \lambda) \neq (0, 0, 0)\,, \tag{NT}$$

(ii) the adjoint system

$$-\dot{\mathbf{p}}(t) = \nabla \mathrm{H}\left(t, \mathbf{x}^*(t), \mathbf{q}(t), \mathbf{u}^*(t)\right), \tag{AS}$$

(iii) the transversality condition

$$\Big(\mathbf{p}(t_0), -\mathbf{q}(t_f)\Big) \in \lambda\, \mathrm{G}_{\mathbf{x}}\Big(\mathbf{x}^*(t_f)\Big) + N_{\mathbb{X}_0 \times \mathbb{X}_1}\Big(\mathbf{x}^*(t_0), \mathbf{x}^*(t_f)\Big)\,, \tag{T}$$

(iv) the Weierstrass condition

$$\mathrm{H}\left(t, \mathbf{x}^*(t), \mathbf{q}(t), \mathbf{u}^*(t)\right) = \max_{\mathbf{u} \in \mathbb{U}} \mathrm{H}\left(t, \mathbf{x}^*(t), \mathbf{q}(t), \mathbf{u}\right), \tag{WC}$$

(v) the complementary slackness condition

$$\operatorname{supp}\{\mu_i\} \subset \{t : h_i\left(\mathbf{x}(t)\right) = 0\}\,, \tag{CS}$$

and $\mathbf{q} : \left[t_0, t_f\right] \to \mathbb{R}^n$ *is a normalized function with bounded total variation defined as*

$$\mathbf{q}(t) = \begin{cases} \mathbf{p}(t) + \displaystyle\int_{[t_0, t)} \sum_{i=1}^{k} \nabla h_i(\mathbf{x}^*(s)) \mathrm{d}\mu_i(s) & t \in [t_0, t) \\ \mathbf{p}(t_f) + \displaystyle\int_{[t_0, t_f]} \sum_{i=1}^{k} \nabla h_i(\mathbf{x}^*(s)) \mathrm{d}\mu_i(s) & t = t_f. \end{cases} \tag{2.58}$$

The conditions of Theorem 2.2.4 will by applied to OCPs in order to characterise the optimal solution and to validate some numerical results.

2.2.7 Hamilton–Jacobi and Sufficient Conditions for Global Optimum

The concept of Dynamic Programming (DP) is based on Bellman's Principle of Optimality [Bel57] which states that "An optimal policy has the property that

whatever the initial state an initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

Applying the Principle of Optimality to continuous time OCPs, we obtain the Hamilton–Jacobi equation, and under some assumptions a very simple deduction of the Maximum Principle is possible [Fon99].

Redefining the *Value Function*, in continuous time, as the infimum cost from a initial pair $(t_0, \mathbf{x}(t_0))$ as

$$V(t_0, \mathbf{x}(t_0)) = \inf_{\substack{\mathbf{x}\in[t_0,t_f] \\ \mathbf{u}\in\mathbb{U}}} \left\{ \int_{t_0}^{t_f} \mathrm{L}\left(s, \mathbf{x}(s), \mathbf{u}(s)\right) \mathrm{d}s + \mathrm{G}\left(\mathbf{x}(t_f)\right) \right\}, \tag{2.59}$$

from the Principle of Optimality, we can see that for any time subinterval $[t, t+\delta] \subset \left[t_0, t_f\right]$, $(\delta > 0)$ we have

$$V(t, \mathbf{x}(t)) = \inf_{\substack{\mathbf{x}\in[t,t+\delta] \\ \mathbf{u}\in\mathbb{U}}} \left\{ \int_{t}^{t+\delta} \mathrm{L}\left(s, \mathbf{x}(s), \mathbf{u}(s)\right) \mathrm{d}s + V\left(t+\delta, \mathbf{x}(t+\delta)\right) \right\}, \tag{2.60}$$

for $\mathbf{x}$ corresponding to $\mathbf{u}$, with the condition

$$V(t_f, \mathbf{x}(t_f)) = \mathrm{G}\left(\mathbf{x}(t_f)\right). \tag{2.61}$$

Assuming the existence of a process $(\mathbf{x}^*, \mathbf{u}^*)$ defined on $[t, t+\delta]$ which is a minimiser for (2.60), we can write

$$-V(t, \mathbf{x}^*(t)) + \int_{t}^{t+\delta} \mathrm{L}\left(s, \mathbf{x}^*(s), \mathbf{u}^*(s)\right) \mathrm{d}s + V\left(t+\delta, \mathbf{x}^*(t+\delta)\right) = 0, \tag{2.62}$$

and for all pairs $(\mathbf{x}^*, \mathbf{u}^*)$

$$-V(t, \mathbf{x}(t)) + \int_{t}^{t+\delta} \mathrm{L}\left(s, \mathbf{x}(s), \mathbf{u}(s)\right) \mathrm{d}s + V\left(t+\delta, \mathbf{x}(t+\delta)\right) \geq 0. \tag{2.63}$$

Let us suppose that $\mathbf{u}^*$ and $\mathbf{u}$ are continuous from the right and, also, that V is continuously differentiable and L is continuous. By adding and subtracting

$V(t+\delta, \mathbf{x}(t))$ to the equations (2.62) and (2.63), dividing by δ, and taking the limit as $\delta \to 0$, we obtain the Hamilton–Jacobi Equation (HJE)

$$V_t(t,\mathbf{x}(t)) + \min_{\mathbf{u}\in\mathbb{U}} \{V_{\mathbf{x}}(t,\mathbf{x}(t)) \cdot \mathbf{f}(t,\mathbf{x}(t),\mathbf{u}) + \mathrm{L}(t,\mathbf{x}(t),\mathbf{u})\} = 0\,, \tag{2.64}$$
$$V(t_f,\mathbf{x}(t_f)) = \mathrm{G}(\mathbf{x}(t_f))\,.$$

This is typically written as

$$V_t(t,\mathbf{x}) - \max_{\mathbf{u}\in\mathbb{U}} \mathrm{H}\,(t,\mathbf{x}(t),-V_{\mathbf{x}}(t,\mathbf{x}),\mathbf{u}) = 0\,, \quad V(t_f,\mathbf{x}) = \mathrm{G}(\mathbf{x})\,,$$

where, as before,

$$\mathrm{H}\,(t,\mathbf{x}(t),\mathbf{p}(t),\mathbf{u}(t)) = \mathbf{p}(t)\cdot\mathbf{f}(t,\mathbf{x}(t),\mathbf{u}(t)) - \mathrm{L}\,(t,\mathbf{x}(t),\mathbf{u}(t))\,,$$

is the (unmaximised) Hamiltonian.

The above analysis relates the HJE and the Value Function. The HJE also features in the following sufficient condition for a pair $(\mathbf{x}^*,\mathbf{u}^*)$ to be a minimiser.

Theorem 2.2.5 (Sufficient Condition)**.** *If there exist a continuously differentiable function V such that the HJE is satisfied and*

$$V_t(t,\mathbf{x}^*(t)) - \mathrm{H}\,(t,\mathbf{x}^*(t),-V_{\mathbf{x}}(t,\mathbf{x}^*),\mathbf{u}^*(t)) = V_t(t,\mathbf{x}^*(t)) - \max_{\mathbf{u}\in\mathbb{U}} \mathrm{H}\,(t,\mathbf{x}^*(t),-V_{\mathbf{x}}(t,\mathbf{x}^*),\mathbf{u})$$

then $(\mathbf{x}^*,\mathbf{u}^*)$ is a local minimiser.

A natural candidate for the function V in Theorem 2.2.5 is the value function, if it is a continuously differentiable function. The main limitation of this result is the fact that a continuously differentiable function V may not exist.

The analytical solution of the HJE is in general not possible to achieve and a numerical solution is computationally very hard to compute. Thus, in general, only very low dimensional problems can be solved in reasonable time, which is the main practical limitation on this approach.

The derivation of the MP can be easily done by assuming that V is C^2 [Dre65]. Let $(\mathbf{x}^*, \mathbf{u}^*)$ be a minimiser satisfying the HJE and let us define

$$\mathbf{p}(t) = -V_x(t, \mathbf{x}^*(t))$$

For an OCP with free terminal state the boundary condition can be written as $V(t_f, \mathbf{x}) = \mathrm{G}(\mathbf{x})$ for all $\mathbf{x}$ in the domain of $\mathrm{G}(\cdot)$. Thus, we have $V_{\mathbf{x}}(t_f, \mathbf{x}^*(t_f)) = \mathrm{G}_{\mathbf{x}}(\mathbf{x}^*(t_f))$. It follows that

$$\mathbf{p}(t_f) = -\mathrm{G}_{\mathbf{x}}(\mathbf{x}^*(t_f))\,.$$

From the HJE, we get

$$\mathbf{p}(t)\cdot\mathbf{f}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) - \mathrm{L}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) = \max_{\mathbf{u}\in\mathbb{U}}\{\mathbf{p}(t)\cdot\mathbf{f}(t,\mathbf{x}^*(t),\mathbf{u}) - \mathrm{L}(t,\mathbf{x}^*(t),\mathbf{u})\}$$

The transversality condition and the maximisation condition are found.

Since the HJE is equal to zero for any $\mathbf{x}$, differentiating with respect to $\mathbf{x}$ we obtain

$$\begin{aligned} V_{t\mathbf{x}}(t,\mathbf{x}^*(t)) + V_{\mathbf{xx}}(t,\mathbf{x}^*(t))\cdot\mathbf{f}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) + \\ + V_{\mathbf{x}}(t,\mathbf{x}^*(t))\cdot\mathbf{f}_{\mathbf{x}}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) + \mathrm{L}_{\mathbf{x}}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) = 0\,. \end{aligned} \tag{2.65}$$

Using (2.65), the derivative of $\mathbf{p}$ is

$$\begin{aligned} \dot{\mathbf{p}}(t) &= -\frac{\mathrm{d}}{\mathrm{d}t}V_{\mathbf{x}}(t,\mathbf{x}^*(t)) \\ &= -\left[V_{t\mathbf{x}}(t,\mathbf{x}^*(t)) + V_{\mathbf{xx}}(t,\mathbf{x}^*(t))\cdot f(t,\mathbf{x}^*(t),\mathbf{u}^*(t))\right] \\ &= -\left[V_{\mathbf{x}}(t,\mathbf{x}^*(t))\cdot\mathbf{f}_{\mathbf{x}}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) + \mathrm{L}_{\mathbf{x}}(t,\mathbf{x}^*(t))\right] \end{aligned}$$

and the Euler–Lagrange equation is

$$-\dot{\mathbf{p}}(t) = \mathbf{p}(t)\cdot\mathbf{f}_{\mathbf{x}}(t,\mathbf{x}^*(t),\mathbf{u}^*(t)) + \mathrm{L}_{\mathbf{x}}(t,\mathbf{x}^*(t),\mathbf{u}^*(t))\,.$$

Chapter 3

Optimisation Software

"The purpose of computing is insight,
not numbers."
Richard Hamming

In this chapter, we review methods for solving an Optimal Control Problem (OCP). Moreover, we are interested in applying direct methods and, for that reason, we introduce three numerical Nonlinear Programming (NLP) solvers and a list of several interfaces used to generate the solver input.

3.1 Introduction

To solve an OCP, we can use Indirect Methods, which involve the Maximum Principle and shooting methods, Dynamic Programming and Hamilton–Jacobi methods, or Direct Methods, which involve discretising and transcribing a OCP into a NLP problem.

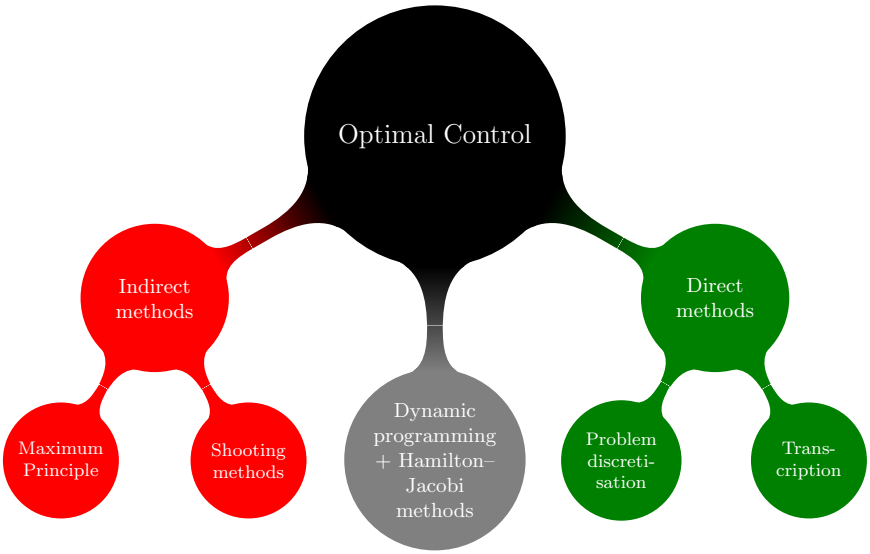


Figure 3.1: Methods for solving an OCP

3.1.1 Dynamic Programming

Dynamic Programming (DP) is a stage wise search method of optimisation problems whose solutions may be viewed as the result of a sequence of decisions. The selection of the optimal decision in based on the Bellman's Principle of Optimality. An optimal sequence of decisions is obtained if and only if each subsequence is optimal. Therefore, if the initial state and decisions from this point are optimal then the remaining decisions must constitute an optimal sequence with respect to the state resulting from the first decision.

For achieving the solution using DP, we should implement the following strategy:

a) the optimal control problem is divided into a certain number of smaller but similar sub–problems;

b) the solution to main problem is rewritten in terms of the solutions for the smaller sub–problems;

c) stage wise solutions start with the smallest sub–problems;

d) solutions of smallest sub–problems are combined to obtain the solutions to sub–

problems of increasing size;

e) the process is continued until we arrive at the solution of the original problem.

It is recommended to construct a table of known results of sub–problems to avoid calculating the same sub–problem twice.

3.1.2 Direct *vs* Indirect Methods

According to [Bet01, BH98, Bie10], Indirect and Direct methods for solving OCP have, each one, their advantages and disadvantages.

The *Indirect Methods* involve the conditions of optimality (Maximum principle), the adjoint equations, a maximisation condition and the boundary conditions, forming a boundary value problem. We can compute the solution via shooting, multiple shooting, or discretisation. Indirect Methods can give us an accurate solution for "special cases" (*e.g.*singular arcs) but they require derivation and implementation of adjoint equations, becoming not robust in general cases.

The *Direct Methods* directly optimise the objective without formation of the necessary conditions, using control and state parametrisation. The Direct Methods can give us a very robust and general approach and some special treatment is needed for "special cases". When using direct methods, practical methods for solving OCP [Bie10, BH75, JTB04] require Newton-based iterations with a finite set of variables and constraints, which can be achieved by converting the infinite-dimensional problem into a finite-dimensional approximation. The *transcription method* has three fundamental steps:

a) converting a dynamic system into a problem with a finite set of variables;

b) solving the finite dimensional problem using a parameter optimisation method (*i.e.*a NLP sub–problem); and

c) assessing accuracy of finite dimensional problem [CdB80, Pin10] and if necessary repeat transcription and optimisation steps.

3.2 Nonlinear Programming Solvers

We are interested in solving OCP via direct methods. After discretising and transcribing an OCP into a NLP, we need a numerical/computational solver to achieve the optimal solution (see Fig. 3.2).

There is list of solvers including open–source, freeware and commercial software, working under different operating systems, is available. In this section, we discuss several widely used NLP solvers, highlighting their features. Since they are based on local search methods, these solvers compute a local optimal solution, when convergence is achieved.

Figure 3.2: Fluxogram illustrating the use of NLP Solvers

3.2.1 IPOPT – Interior Point OPTimiser

"IPOPT is a software package for large–scale nonlinear optimisation. It is designed to find (local) solutions of mathematical optimisation problems." The IPOPT library can be found in http://www.artelys.com/. [WB06]

IPOPT has been developed by Andreas Wächter and Carl Laird and it can be used on Linux, Unix, Mac OS X and Microsoft Windows operating systems. IPOPT is written in C++ and it is released as open source code under the Eclipse Public License (EPL). IPOPT can be used as a library that can be linked to C++, C or Fortran code, as well as a solver executable for the AMPL modelling environment.

The package includes interfaces to CUTEr optimisation testing environment, as well as the MATLAB and R programming environments.

IPOPT implements an interior-point line-search filter method and this approach makes IPOPT particularly suitable for large problems with up to millions of variables and constraints, assuming that the Jacobian matrix of constraint function is sparse, but also small and dense problems can be solved efficiently.

More informations can be found in the IPOPT http://www.coin-or.org/Ipopt/documentation/, where a short IPOPT tutorial [Wä14] is available. Specific instructions to the MATLAB interface can be found at the Matlab interface page.

3.2.2 KNITRO

"KNITRO is an optimisation software library for finding solutions of both continuous (smooth) optimisation models (with or without constraints), as well as discrete optimisation models with integer or binary variables (*i.e.* mixed integer programs). KNITRO is primarily designed for finding local optimal solutions of large-scale, continuous nonlinear problems." The KNITRO library can be found in http://www.artelys.com/.

KNITRO is a software package for solving smooth optimisation problems, with or without constraints [LLC13]. KNITRO has been developed at Ziena Optimisation and it can be used on Linux, Unix, Mac OS X and Microsoft Windows operating systems. KNITRO is written in C, C++, Fortran and Java. KNITRO can be used as a library that can be linked to Fortan, C/C++, Java and Microsoft Excel, as well as a solver executable for Matlab, AMPL, Mathematica, AIMMS, GAMS and MPL modelling environments.

Some of KNITRO key benefits are: (a) it solves complex nonlinear problems since it handles large-scale, complex problems with millions of variables and constraints;

(b) it offers the leading combination of computational efficiency and robustness; (c) it computes high accuracy solutions via the Active Set algorithm; (d) it offers the ability to choose the best algorithm among three options; and (e) it has some flexibility of use.

The KNITRO key features are: (a) efficient and robust solution on large scale problems; (b) one active-set and two interior-point/barrier algorithms; (c) two algorithms for mixed-integer nonlinear optimisation; (d) parallel multi–start feature for global optimisation; (e) heuristics, cutting planes, branching rules for MINLP; (f) ability to run multiple algorithms concurrently; (g) fast infeasibility detection; and (h) automatic computation of approximate first and second derivatives.

Problems classes solved by KNITRO solves problems involving em general NLP problems, including non-convex; systems of nonlinear equations; linear problems; quadratic problems, both convex and non-convex; least squares problems/regression, both linear and nonlinear; mathematical programs with complementary constraints; and mixed–integer nonlinear problems.

More information can be found in the KNITRO manual [LLC13] available in its web–page. Specific KNITRO/Matlab interface documentation can be found on the Matlab Optimisation Toolbox – web–page.

3.2.3 WORHP – WORHP Optimises Really Huge Problems

"WORHP Optimises Real Huge Problems (WORHP) is a software library for mathematical nonlinear optimisation, suitable for solving problems with thousands or even millions of variables and constraints." The WORHP library can be found in http://www.worhp.de/.

WORHP has been developed under the direction of Christof Büskens with Matthias Gerdts, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows operating systems. WORHP is written in Fortran and C. WORHP offers a

total of nine interfaces: 3 for Fortran, 3 for C/C++, Matlab, ASTOS and AMPL, for different programming languages and communication paradigms.

WORHP options involve: (a) different finite difference derivative approximations, (b) several BFGS and sparse BFGS strategies, (c) extended optimality and termination criteria, (d) numerical inaccuracy validation, (e) lowpass-filter termination checks, (f) different feasibility strategies, (g) miscellaneous recovery strategies, (h) strategies for automatic scaling, (i) workspace management system, (j) automatic Hessian structure approximation, (k) multiple relaxation variables, (l) warm–start capability for QPSOL, (m) hot–start capability for WORHP, (n) Lagrange multiplier initialization, (o) interfaces to following linear algebra solvers (SuperLU, MA57, MA86, PARDISO, MUMPS, WSMP), (p) modularization using Unified Solver Interface, (q) advanced, basic and simple interfaces, (r) cross-language and cross-platform capability, (s) check routine for structural errors in matrices, (t) detailed termination output routine, (u) process monitoring, and (v) stage history.

Documentation is available on WORHP web–page, namely a tutorial [wor12], an User Manual [wor13] and applications [NBW11].

3.2.4 Other Commercial Packages

SOCS – Sparse Optimal Control Software

"The Sparse Optimal Control Family, developed by The Boeing Company, contains two advanced software packages, available separately or together."
The Sparse Optimal Control Software (SOCS) library can be found in Boeing web–page.

Sparse Optimal Control Software (SOCS) is general-purpose software for solving optimal control problems [BH97, Tec]. Applications include trajectory optimisation, chemical process control and machine tool path definition.

SOCS has been developed at The Boeing Company and it is supported on most UNIX and Windows operating systems. This software is supported on most major platforms with at least 14 decimal digits of precision, which on most systems means double precision. SOCS and all lower-level support routines are written in ANSI-Standard FORTRAN 77. SOCS can be used as a library that can be linked to Fortran 77.

SNOPT – Sparse Nonlinear OPTimiser

Sparse Nonlinear Optimiser (SNOPT) is a software package for solving large-scale optimisation problems (linear and nonlinear programs) [GMS08]. It is especially effective for nonlinear problems whose functions and gradients are expensive to evaluate. The functions should be smooth but need not be convex. The SNOPT library can be found in http://www.sbsi-sol-optimize.com/.

SNOPT has been developed by Philip Gill, Walter Murray and Michael Saunders, and it is intended for any machine with a reasonable amount of memory and a FORTRAN compiler. SNOPT is implemented in FORTRAN 77 and distributed as source code. SNOPT may be called from a driver program, typically in Fortran, C or MATLAB.

3.3 Interfaces

An interface is a software that provides us the means to communicate with the solver. Using the interface, we are able to prepare all data associated to the problem we are solving as input to the solver.

Interfaces can be catalogued in two groups: OC interfaces or NLP interfaces. The first ones handle with the discretisation and transcription procedures, while the other ones leave this task to the user (see Figure 3.3). In this section, several interfaces

are presented, encompassing OC interfaces and NLP interfaces, open–source, freeware and commercial software, working under different operating systems.

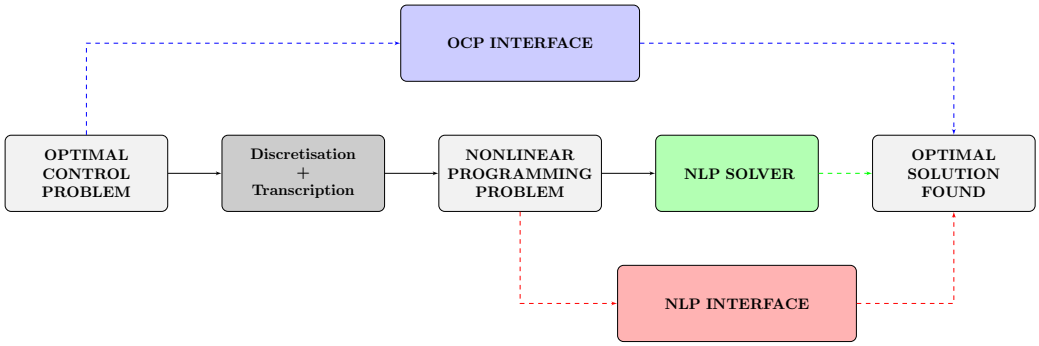


Figure 3.3: Fluxogram illustrating the use of Interfaces

3.3.1 AMPL – A Modelling Language for Mathematical Programming

"A Modelling Language for Mathematical Programming (AMPL) is a comprehensive and powerful algebraic modelling language for linear and nonlinear optimisation problems, in discrete or continuous variables." The AMPL library can be found in the official site.

AMPL has been developed by Robert Fourer, David Gay and Brian Kernighan at Bell Laboratories, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows. AMPL is written in C++ and it is released as open source code under the Eclipse Public License (EPL).

AMPL [FGK02] integrates a modelling language for describing optimisation data, variables, objectives, and constraints, a command language for browsing models and analysing results, and a scripting language for gathering and manipulating data and for implementing iterative optimisation schemes.

AMPL provides a general and natural syntax for arithmetic, logical, and conditional expressions, and familiar conventions for summations and other iterated

operators. This software enables NLP features such as initial primal and dual values, user-defined functions, fast automatic differentiation, and automatic elimination of "defined" variables. AMPL has tools for automatic handling of linear and convex quadratic problems in continuous and integer variables, and it promotes the use of sets and set operators.

Further documentation can be found in [FGK02, FGK90].

3.3.2 ACADO – Toolkit for Automatic Control and Dynamic Optimisation

"Automatic Control And Dynamic Optimisation (ACADO) is a software environment and algorithm collection for automatic control and dynamic optimisation. It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control, state and parameter estimation and robust optimisation." The ACADO library can be found in the official site [HFD11a, HFD11b].

ACADO has been developed under the direction of Moritz Diehl, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows. ACADO is implemented as self-contained C++ code and comes along with user–friendly MATLAB interface. The object–oriented design allows for convenient coupling of existing optimisation packages and for extending it with user–written optimisation routines. ACADO is released under the LGP License.

Not only we can solve standard optimal control problems with ACADO, but it also offers systematic and advanced tools for solving general optimal control problems with multiple and conflicting objectives. The ACADO Code Generation tool can automatically generate Gauss-Newton real–time iteration algorithms for fast nonlinear MPC applications. This software provides also tools to solve state and parameter estimation problems.

On the ACADO documentation web–page is available its User's Manual [HFVQ13], the User's Manual for Matlab [AHF11] and a ACADO Introductory Talk.

3.3.3 BOCOP – The optimal control solver

"The BOCOP project aims to develop an open-source toolbox for solving optimal control problems." The BOCOP library can be found in the official site.

BOCOP has been developed V. Grelard, P. Martinon and F. Bonnans at Inria-Saclay, and it is available for linux precompiled packages (Mac and Windows versions are still under testing). The core files for BOCOP are written in C++ and released under the Eclipse Public License. User supplied functions can be written in plain C, and do not require advanced programming skills.

The BOCOP project aims to develop an open-source toolbox for solving optimal control problems, with collaborations involving industrial and academic partners. It is developed since 2010 in the framework of the Inria–Saclay initiative for an open source optimal control toolbox.

BOCOP can be used in command line mode, especially for experienced users, however, it is recommend using the GUI, at least for the first steps. It provide visualization scripts for Matlab and Scilab.

BOCOP currently uses IPOPT (with MUMPS as linear solver) for solving the nonlinear programming problem resulting from the direct transcription of the optimal control problem. BOCOP relies on ADOL-C (with ColPack for the sparsity) to compute derivatives of the objective and constraints by automatic differentiation.

Further information can be found in BOCOP user's guide [BGG$^+$14] and run the examples available on its official web–page.

3.3.4 DIDO – Automatic Control And Dynamic Optimisation

"DIDO, the leading optimal control software, powers users by offering the easiest and direct solutions to the most complex problems." The DIDO software can be found in the official site.

DIDO has been developed at Elissar Global, and it can only be used on Microsoft Windows. DIDO requires MATLAB and it is released under academic and commercial licenses.

DIDO foundation is pseudospectral theory and is the only pseudospectral solution with mathematically proven convergence properties. DIDO is a MATLAB program for solving hybrid optimal control problems. The general–purpose program is named after DIDO, the legendary founder and first queen of Carthage who is famous in mathematics for her remarkable solution to a constrained optimal control problem even before the invention of calculus.

This generality allows for (a) fairly complex interior point constraints, (b) pre–defined segments, (c) differentially-flat segments, (d) Transition conditions, (e) mid–manoeuvre changes in dynamics, (f) multi-dynamical systems, (g) mid–manoeuvre changes in the cost function, (h) switches, and (i) discrete events.

3.3.5 ICLOCS – Imperial College London Optimal Control Software

"The code allows users to define and solve optimal control problems with general path and boundary constraints and free or fixed final time. It is also possible to include constant design parameters as unknowns." The Imperial College London Optimal Control Software (ICLOCS) software can be found in the official site.

ICLOCS has been developed by Paola Falugi, Eric Kerrigan and Eugene van Wyk, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows. ICLOCS is

implemented in MATLAB and it is released as open source code under the BSD License.

ICLOCS starts by transcribing the OCP to a static optimisation problem by either direct multiple shooting or direct collocation methods. The direct multiple shooting formulation requires the solution of initial value problems that can be determined using the an open–source sensitivity solver. The direct collocation formulations discretise the system dynamics using implicit Runge–Kutta method and can also be used to incorporate discrete–time problems. Once the OCP has been transcribed, it can be solved with a selection of nonlinear constrained optimisation algorithms given by IPOPT or MATLAB's `fmincon` solver. The derivatives of the ODE right-hand side, cost and constraint functions are also required for the optimisation and they can be either estimated numerically or supplied analytically.

Further documentation can be found in ICLOCS User's Guide [FKvW10] and the user can learn about ICLOCS by testing several default examples.

3.3.6 ROC-HJ – Reachability and Optimal Control Software

The software Reachability and Optimal Control Software (ROC-HJ) implements a set of numerical methods for solving some HJs equations arising in optimal control theory. The library also offers some useful tools for analysing the numerical solutions and it provides the designing the optimal control laws along with the corresponding optimal trajectories [BDZ13a]. The library can be used for a large class of deterministic control problems including: reachability analysis, path planning, collision avoidance, infinite horizon control problems, minimum time problems, Mayer or Bolza type problem, state-constrained control problems, differential games, and exit time problems.

ROC-HJ has been developed by Olivier Bokanowski, Anna Désilles, and Hasnaa Zidani, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows.

ROC-HJ is a C++ MPI/OpenMP library for solving $d-$dimensional Hamilton–Jacobi–Bellman equations by finite difference methods, or semi–lagrangian methods.

ROC-HJ implements finite difference methods – the discretisation with respect to the time variable is performed by Euler scheme or Runge-Kutta method (RK2 or RK3), and the discretisation in space is based on upwind finite difference, Lax-Freidrich method, or ENO2 – and semi–lagrangian methods – the user may choose an integration scheme for the characteristics such as explicit Euler scheme, RK2, RK3, adaptive method using a number of intermediary time steps. For the interpolation method, the code uses only the bilinear scheme.

ROC-HJ has its own editor. Further documentation can be found in [BDZ13b].

3.3.7 TACO – Toolkit for AMPL Control Optimisation

"Toolkit for AMPL Control Optimisation (TACO) is the Toolkit for AMPL Control Optimisation. It defines some add-ons to the AMPL modeling language that allow the elegant formulation of ODE/DAE optimal control problems in AMPL." The TACO toolkit can be found in the official site.

TACO has been developed by Christian Kirches and Sven Leyffer, and it can be used on Linux, Unix, Mac OS X and Microsoft Windows. TACO is written in C and it is freeware.

TACO reads AMPL files and detects the structure of the OCP. This toolkit is designed to facilitate the coupling of existing optimal control software packages to AMPL.

Further documentation can be found in [KL13].

3.3.8 Pseudospectral Methods in Optimal Control

PSOPT

PSOPT is an open source optimal control software package written in C++ that uses direct collocation methods, including pseudospectral and local discretizations, available in the office site. Pseudospectral methods solve optimal control problems by approximating the time-dependent variables using global polynomials, such as Legendre or Chebyshev functions. Local discretisation methods approximate the time dependent functions using local splines, and can be seen as implementations of implicit Runge-Kutta integrators. With both global and local methods, differential equations, continuous constraints and integrals associated with the problem are discretised over a grid of nodes. Sparse nonlinear programming is then used to find local optimal solutions.

PSOPT is able to deal with problems with the distinct characteristics: (a) single or multiphase problems; (b) continuous time nonlinear dynamics; (c) nonlinear path constraints; (d) general event constraints; (e) integral constraints; (f) interior point constraints; (g) bounds on controls and state variables; (h) general cost function with Lagrange and Mayer terms; (i) linear or nonlinear linkages between phases; (j) fixed or free initial phase time; (k) fixed or free final phase time; (l) optimisation of static parameters; and (m) optimal parameter estimation given sampled observations.

Among other features, the implementation allows: (a) choice between Legendre, Chebyshev, central differences, trapezoidal or Hermite-Simpson discretisation; (b) large scale nonlinear programming using IPOPT and (optionally) SNOPT; (c) estimation of the discretisation error; (d) automatic scaling; (e) automatic differentiation using the ADOL-C library; (f) numerical differentiation by using sparse finite differences; (g) automatic identification of the sparsity of the derivative matrices; (h) DAE formulation, so that differential and algebraic constraints can be implemented in the same C++ function; (i) easy to use interface to GNUplot to produce graphical

output, including 2D plots, 3D curves and surfaces, and polar plots; and (j) automatic generation of LaTeX code to produce a table that summarizes the mesh refinement process.

Full details on PSOPT and its features can be found in its documentation [Bec11].

GPOPS-II - Gauss Pseudospectral OPtimal Control Software

Gauss Pseudospectral Optimal Control Software (GPOPS-II) is a general purpose optimal control software available in the office site. GPOPS-II is a new open-source MATLAB optimal control software that implements a brand new hp-adaptive Legendre-Gauss-Radau quadrature integral pseudospectral method for solving general nonlinear optimal control problems. Using GPOPS-II, the optimal control problem is transcribed to a nonlinear programming problem (NLP). The NLP is then solved using either the solver SNOPT or the solver IPOPT.

Among other features, GPOPS-II (a) allows for an extremely general formulation of the optimal control problem, (b) allows for inclusion of integral constraints and highly general boundary conditions, (c) complete first and second sparse finite-differencing of optimal control problem to compute all derivatives required by the NLP solver., (d) provides Gaussian quadrature integration methods for rapid convergence, (e) enables the inclusion of the NLP solver SNOPT (for Academic Users) and IPOPT (for Not-for-Profit and Commercial Users), and (f) has no third-party products other than MATLAB are required.

More information about the methodology used in GPOPS-II can be found in [PR13, PR14].

3.4 Solvers Benchmark

3.4.1 Differential Drive Robot

To test the solvers of section 3.2 – IPOPT, KNITRO and WORHP – we consider a minimum time problem involving a differential drive robot system. The geometry of such vehicle is presented in Fig. 3.4 where (x, y) is the position of mid–point of the axle connecting the both wheels and ψ is the heading angle.

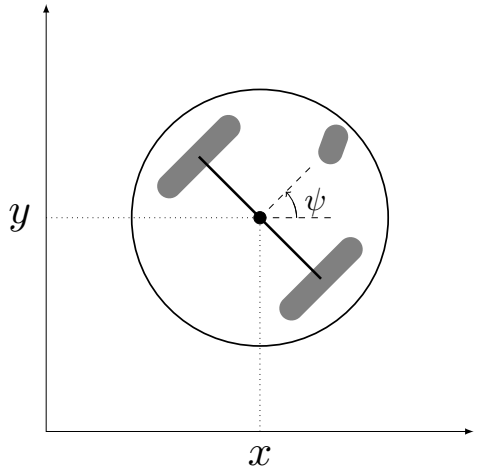


Figure 3.4: Differential drive robot geometry

The movement of a differential drive robot is based on two separately driven wheels placed on either side of the robot body. There is no need for an additional steering motion since it can change its direction just by varying the relative rate of rotation of its wheels.

Aiming minimum time, the differential drive robot ($\mathrm{P_{DD}}$) can be stated as:

$$\text{Minimise } t_f \tag{3.1}$$

subject to

(i) dynamic constraints

$$\dot{x}(t) = (u_1(t) + u_2(t))\cos(\psi(t)) \qquad \text{a.e. } t \in \left[t_0, t_f\right]$$

$$\begin{aligned} \dot{y}(t) &= (u_1(t)+u_2(t))\sin(\psi(t)) && \text{a.e. } t \in \left[t_0, t_f\right] \\ \dot{\psi}(t) &= u_1(t)-u_2(t) && \text{a.e. } t \in \left[t_0, t_f\right], \end{aligned} \tag{3.2}$$

where $\mathbf{x}(t) = (x(t), y(t), \psi(t))$ is the state and $\mathbf{u}(t) = (u_1(t), u_2(t))$ is the control – $u_1(t)$ and $u_2(t)$ are the speed of each wheel in $\left[\text{ms}^{-1}\right]$,

(ii) input constraints

$$\begin{aligned} 0 \leq u_1(t) \leq 1 && \text{a.e. } t \in \left[t_0, t_f\right] \\ 0 \leq u_2(t) \leq 1 && \text{a.e. } t \in \left[t_0, t_f\right], \end{aligned}$$

where $u_1(t)$ and $u_2(t)$ are the speed of each wheel in $\left[\text{ms}^{-1}\right]$.

(iii) end–point constraints

$$\mathbf{x}(0) = \mathbf{x}_0 = (x_0, y_0, \psi_0) = (0,0,0) \tag{3.3}$$

$$\mathbf{x}(t_f) \in \mathbb{X}_1 = \left\{(x,y,\psi) : \left(x-x_f\right)^2 + \left(y-y_f\right)^2 + \left(\psi-\psi_f\right)^2 \leq r^2\right\}, \tag{3.4}$$

where $r^2 = 0.1$ and $\mathbf{x}_f = (x_f, y_f, \psi_f) = (10,0,0)$ is a user–defined target point, and

(iv) pathwise state constraint

$$h\left(\mathbf{x}(t)\right) = (\bar{y} - y(t)) - k\left(\bar{x} - x(t)\right)^2 \leq 0\,, \quad \forall t \in \left[t_0, t_f\right], \tag{3.5}$$

where $(\bar{x}, \bar{y}) = (5,1)$ and $k = 10$.

The goal is to drive this differential drive robot from $\mathbf{x}_0$ to some point near $\mathbf{x}_f$ according to the terminal condition (3.4) without violating the state constraint (3.5).

3.4.2 Numerical Results

The differential drive robot problem (P_{DD}) was written in the AMPL interface and it was solved using IPOPT, KNITRO and WORHP solvers with the same set of

options, namely with the same acceptable tolerance. We computed the solutions in a computer with a Intel™ Core© i7-4770K CPU @3.50 GHz.

In Fig. 3.5, the xy trajectory obtained using IPOPT is shown. The optimal trajectories for state variables and controls associated to (P_{DD}) are shown in Fig. 3.6.

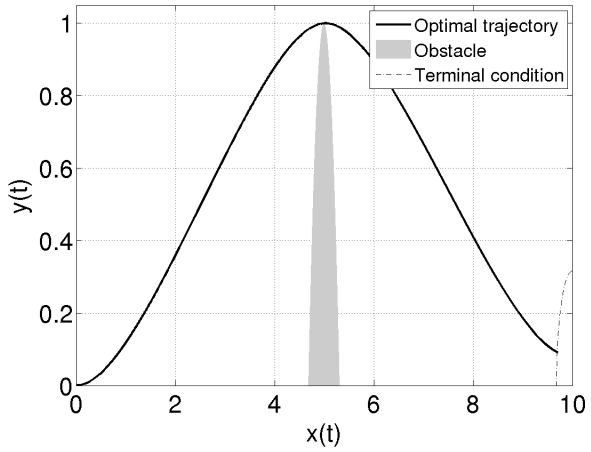


Figure 3.5: xy trajectory for (P_{DD})

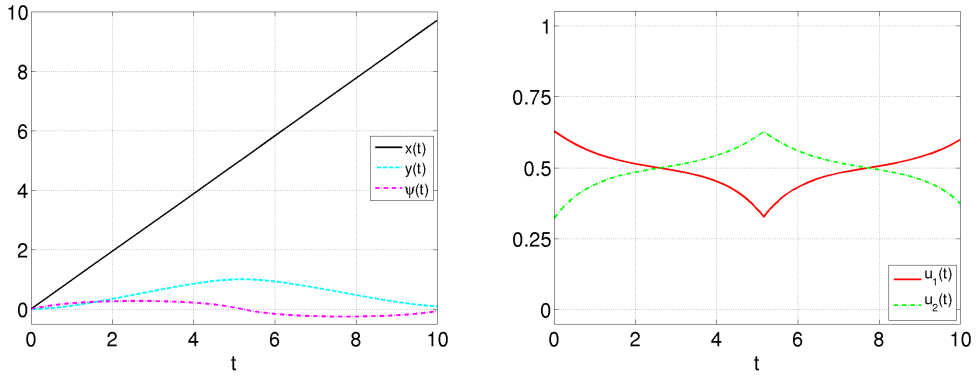
As it can be seen in Fig. 3.5, the differential drive robot successfully goes round the obstacle and it stops inside the circle defined by (3.4).

The numerical results concerning all solvers are shown in Table 3.1, which shows information about the number of nodes, the initial guess for the objective function, the number of iterations needed to solve the NLP problem, the objective functional, and the CPU times spent for the solver computations and NLP evaluations.

Table 3.1: Comparing results for (P_{DD}) without an initial guess

Solver	N_j	Initial guess	I_j	Objective	CPU time (s)	
					Solver	NLP eval
IPOPT	1000	0	2899	9.4986378544	64.106	13.585
KNITRO	1000	0	710	9.4986463551	16.970	2.534
WORHP	1000	0	(288)	local infeasibility	(14.566)	(4.864)

Considering an equidistant–spacing mesh with 1000 node points, IPOPT and



(a) Optimal Trajectories (b) Control

Figure 3.6: Optimal solution for (P_{DD})

KNITRO converge to a local optimal solution. WORHP was inefficient, not converging to a valid solution. According to Table 3.1, IPOPT and KNITRO solvers establish the minimum time around 9.4986s. When solving (P_{DD}), KNITRO is the fastest solver to compute a solution, taking 25% of the CPU time spent by IPOPT.

By default, when using AMPL, the value of the objective function is zero–initialised. However, we can improve the performance of the solvers by setting-up an initial guess. In problem (P_{DP}) we can compute an estimate for the travelling distance. With that information and considering the maximum speed for the differential drive robot, we can underestimate the minimum time need for the robot to arrive to the target area.

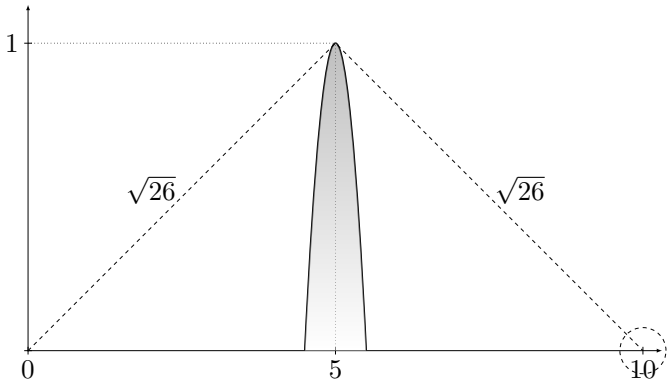


Figure 3.7: Estimate for the travelling distance (P_{DD})

According to Fig. 3.7, the estimate for the travelling distance is $2\sqrt{26} \approx 10.198\,\mathrm{m}$ and, since the maximum speed in $1\,\mathrm{ms}^{-1}$, we get an estimate of 10.198 s for the minimum time.

Initialising the value of the objective function to 10.198 s and considering the same mesh, we solve, once more, the problem ($\mathrm{P_{DD}}$). According to Table 3.2, all three solvers establish the minimum time around 9.49863 s. This value is lower than the initial guess 10.198 s because the differential drive robot immediately stops when it enters in the target area, defined by (3.4), before it reaches $\mathbf{x}(t_f)$.

Table 3.2: Comparing results for ($\mathrm{P_{DD}}$) with an initial guess

Solver	N_j	Initial guess	I_j	Objective	CPU time (s)	
					Solver	NLP eval
IPOPT	1000	10.198	1986	9.4986378544	31.825	7.890
KNITRO	1000	10.198	515	9.4986351465	13.483	1.889
WORHP	1000	10.198	60	9.4986352207	9.260	0.197

We notice that IPOPT provides the same result and compute it twice faster than the previous one. KNITRO computes a lower solution than the one given without setting-up the initial guess and it takes about 80% of the CPU time. With this initial guess, WORHP is able to provide a solution and it do it in just 9.457 seconds, being the fastest solver tested. When solving ($\mathrm{P_{DD}}$), WORHP is 4.2× faster than IPOPT and it is 1.6× faster than KNITRO.

3.5 Final Remarks

With respect to interfaces, there are a lot of options we can chose from, involving open–source, freeware and commercial software, working under different operating systems. The choice of an interface should be made taking into account the number of solvers it can connect and the level of programming expertise of the user.

Among open–source, freeware and commercial software, there are also a big list of OC and NLP solvers useful for solving optimal control problems.

The minimum time problem (P_{DD}) involving a differential drive robot system was successfully solved using IPOPT and KNITRO, and also by WORHP when an initial guess was provided. The selection of an initial guess proved to vital to improve the performances of all solvers. The three solvers computed similar results but, still, they can compared in terms of CPU time spent. In that matter, WORHP was the fastest one.

Chapter 4

Time–Mesh Refinement for Optimal Control

"The whole of science is nothing more than
a refinement of everyday thinking."
Albert Einstein

In this chapter we propose a time–mesh refinement algorithm that is based on block-structured adaptive refinement method. These results were submitted to the Discrete and Continuous Dynamical Systems journal, a publication of the American Institute of Mathematical Sciences [PF14b]. The strategy proposed was reported in [PF13], [PF14a], and its last improved version in [PF14b]. It is based on block-structured adaptive refinement method which became popular within fluid mechanics since multi–grid algorithms can be used for time and space domains.

4.1 Introduction

In a direct collocation method, the control and the state are discretised in a set of appropriately chosen mesh, in the time interval. Then, the continuous–time

Optimal Control Problem (OCP) is transcribed into a finite–dimensional Nonlinear Programming (NLP) which can be solved using widely available software packages [Pai13]. Most frequently, in the discretisation procedure, regular time meshes having equidistant spacing are used. However, in some cases, these meshes are not the most adequate to deal with nonlinear behaviours. One way to improve the accuracy of the results, while maintaining reasonable computational time and memory requirement, is to construct a mesh having different time steps. The best location for the smaller steps sizes is, in general, not known a priori, so the mesh will be refined iteratively.

In a mesh–refinement procedure the problem is solved, typically, in an initial coarse uniform mesh in order to capture the basic structure of the solution and of error. Then, this initial mesh is repeatedly refined according to a chosen strategy until some stopping criteria is attained.

Several mesh refinement methods employing direct collocation methods have been described in the recent years. In [Bet01] and [BH98] a mesh refinement procedure is developed for changing the discretisation in order to improve the accuracy of the approximation involving an integer programming technique. In [BBCH00] its shown that there can be order reduction for Implicit Runge–Kutta methods that can be utilized in direct transcription trajectory optimisation by modifying a currently used mesh refinement strategy. In [ZT11] a density function is used to generate a fixed-order mesh on which the problem is solved. In [PHR14] an approach based on varying the order of the approximation in each mesh interval and using the exponential convergence rate of a Gaussian quadrature collocation method is reported. In [PF13], a adaptive mesh refinement strategy based on block–structured refinement method for solving continuous–time nonlinear OCP is presented. It is a purely direct method approach and the convergence is achieved by increasing the number of nodes and by selecting their placement according the refinement criterion. In this algorithm, just one refinement criterion can be defined and it coincides with the stopping criterion.

4.2 Adaptive Mesh Refinement Algorithm

In this approach, we are concerned in solving the problem, in a first step, on a coarse mesh and, according to a decision based on some refinement criteria, the mesh is refined locally or entirely, using different levels of refinement. The discretised problem is then solved on the new refined mesh using information from the coarser mesh solution of the previous step. According to this strategy, we do not need to remove nodes.

4.2.1 Adaptive Mesh Refinement

The adaptive mesh refinement process starts by discretising the time interval $\left[t_0, t_f\right]$ in a coarse mesh, π_0, containing N_0 equidistant nodes. After being transcribed into a NLP problem, the OCP is solved in this coarse mesh to catch the main structure of the problem. Then, the mesh is progressively refined. According to some refinement criteria, the mesh is divided in K mesh intervals

$$\mathcal{S}_k = [\tau_{k-1}, \tau_k[, \quad k = 1, \ldots, K-1 \qquad \text{and} \qquad \mathcal{S}_K = [\tau_{K-1}, \tau_K]$$

where $(\tau_0, \ldots, \tau_K)$ coincide with nodes. These mesh intervals $\mathcal{S}_k$ form a partition of the time interval, that is,

$$\bigcup_{k=1}^{K} \mathcal{S}_k = \left[t_0, t_f\right] \quad \text{and} \quad \bigcap_{k=1}^{K} \mathcal{S}_k = \emptyset\,,$$

while the mesh nodes have the property that

$$\tau_0 < \tau_1 < \tau_2 < \ldots < \tau_K\,.$$

According to the algorithm reported in [PF13], the subintervals $\mathcal{S}_k$ that verify the refinement criteria were refined by adding a fixed number $\mathcal{N}$ of equidistant nodes between each two mesh points in such way that the refined mesh π_{j+1} will contain the nodes of the prior one π_j. This property is an important feature in block–structured

schemes. The procedure was repeated until the stopping criterion was achieved. We also considered a more conservative approach by refining the neighbours of $\mathcal{S}_k$, *i.e.*, $\mathcal{S}_{k-1}$ and $\mathcal{S}_{k+1}$. The resulting mesh using this strategy is denoted further ahead by π_R.

Now, we present an improved version of this algorithm by introducing different levels of refinement in a single iteration. After selecting the intervals $\mathcal{S}_k$ that verify the refinement criteria, they are divided into smaller subintervals according to the user–defined levels of refinement

$$\bar{\varepsilon} = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m] \, .$$

For example, the higher levels can be defined as multiple powers of 10 of the first level $\bar{\varepsilon} = \left[1, 10, 10^2, \ldots, 10^m\right] \varepsilon_1$.

A subinterval $\mathcal{S}_{k,i}$ is at the i^{th} level of refinement if

$$S_{k,i} = \{t \in S_k \ : \ \varepsilon(t) \in [\varepsilon_i, \varepsilon_{i+1}[\} \tag{4.1}$$

for $i = 1, \ldots, m$, and it will be refined by adding $\mathcal{N}^i$ of equidistant nodes between each two mesh points. This procedure adds more node points to the subintervals in higher levels of refinement, corresponding to higher errors, and it adds less node points to those in lower refinement levels.

By defining several levels of refinement, we get a multi–level time–mesh in a single iteration as shown in Fig. 4.1. The resulting mesh using this strategy is denoted further ahead by π_{ML}.

4.2.2 Refinement and Stopping Criteria

In order to proceed with the mesh refinement strategy, we have to define some refinement criteria and a stopping criterion. We consider three refinement criteria:

1. the estimate of the relative error of the trajectory (primal variables) ($\varepsilon_{\mathbf{x}}$)

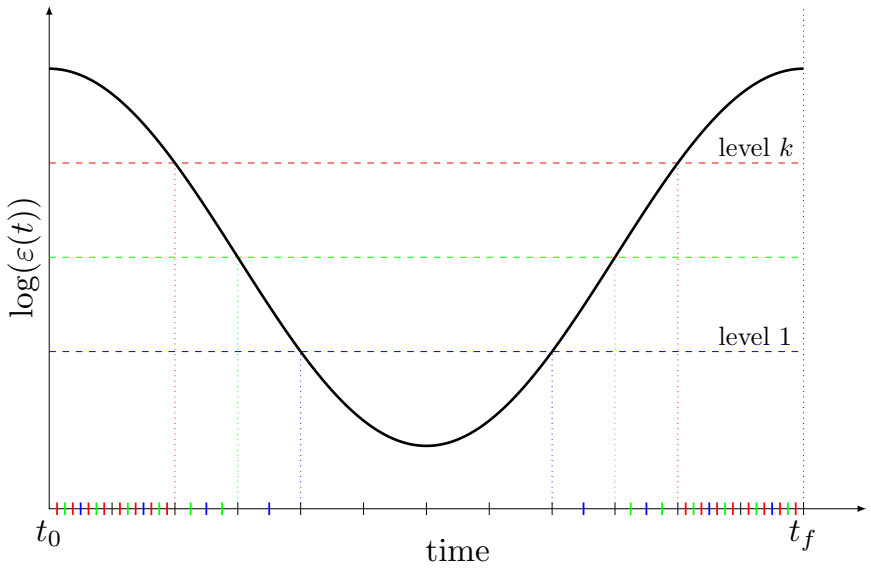


Figure 4.1: Illustration of the time–mesh refinement strategy

2. the estimate of the relative error of the adjoint multipliers (dual variables) ($\varepsilon_{\mathbf{q}}$)

3. a combination of both criteria

and we consider a threshold for the relative error of the trajectory as the stopping criterion.

For the first refinement criterion, the relative error estimate is, at each time, the difference between the obtained state trajectory and an higher order approximation of the solution of the dynamic differential equation. In this case, the solution is given by piecewise cubic polynomials using Hermite interpolation and, then, it is integrated using the Romberg quadrature. At each refinement iteration, the local error ($\varepsilon_{\mathbf{x}}$) is computed and this information is taken into account when deciding if the refinement procedure should continue.

In the second case, we consider the multipliers $\mathbf{q}_{\text{MP}}$ which are solution of the differential equation system (AS), (T) and (2.58) given by the Maximum Principle 2.2.4, and we also consider the multipliers $\mathbf{q}_{\text{KKT}}$ obtained by applying the Kuhn–Tucker conditions to nonlinear optimisation problem which results from the transcription of the optimal control. The relative error estimate is, at each

time, the difference between the multipliers $\mathbf{q}_{\mathrm{KKT}}$ computed by the numerical solver and $\mathbf{q}_{\mathrm{MP}}$ computed by integrating numerically the adjoint equation given by the Maximum Principle. This criterion is chosen because these multipliers give sensitivity information. Furthermore, $\mathbf{q}_{\mathrm{MP}}$ are solution to a linear differential equation system, which can be easily solved in a faster way and with higher accuracy. At each refinement step, the local error of the multipliers ($\varepsilon_{\mathbf{q}}$) is evaluated

$$\varepsilon_{\mathbf{q}} = ||\mathbf{q}_{\mathrm{MP}} - \mathbf{q}_{\mathrm{KKT}}||$$

and the procedure selects which time intervals should be further refined.

In the last case, we use both refinement criteria simultaneously and the procedure will continue until the stopping criterion is satisfied.

As stopping criterion, we consider the L^{∞} norm of the relative error of the primal variables ($\varepsilon_{\mathbf{x}}$). Even when using the relative error of the adjoint multipliers ($\varepsilon_{\mathbf{q}}$) as refinement criteria, we still need to estimate the error on the trajectory since it is the stopping criterion. However, we do not need to compute $\varepsilon_{\mathbf{x}}(t)$ for all t – as in the case of the refinement criteria – but just an estimate of $\left|\left|\varepsilon_{\mathbf{x}}^{(j)}\right|\right|_{\infty}$ which is much faster to obtain.

4.2.3 Warm Start

Since the proposed procedure increases the number of nodes, more computational time would be expected. To decrease the CPU time, when going from a coarse mesh to a refined one progressively, the previous solution is used as a warm start for the next iteration. To create this warm start, the solution obtained in the coarse mesh is projected in the refined one using the cubic Hermite interpolation. This action proved to be vital in the decreasing of the overall computational time. In particular, we notice that the number of iterations of the NLP solver remains within the same order of magnitude when we considerably increase the number of nodes.

4.2.4 Algorithm Implementation

The overall procedure is described in Algorithm 1. To test the algorithm, the proposed procedure was implement in MATLAB R2008a combined with the Imperial College London Optimal Control Software (ICLOCS) – version 0.1b [FKvW10]. ICLOCS is an optimal control interface and it uses the Interior–Point Optimiser (IPOPT) solver, which is an open-source software package for large-scale nonlinear optimisation [WB06]. The problems are solved in a computer with a Intel™ Core© 2 CPU 6600@2.40 GHz.

```
Data: Cost functions, dynamics, constraints, initial/terminal boundaries,
      parameters, refinement and stopping criteria $\left(\varepsilon_{\mathbf{x}}^{\max}$ and $\varepsilon_{\mathbf{q}}^{\max}\right)$
Result: candidates to optimal solution, controls
initialization;
select a time–mesh;
discretize and transcribe the OCP;
solve the NLP;
estimate the discretisation error – $\varepsilon_{\mathbf{x}}^{(0)}$;
estimate the error on the multipliers – $\varepsilon_{\mathbf{q}}^{(0)}$;
while stopping criteria not met do
    select the mesh subintervals $\mathcal{S}_{j,i}$ to be refined ;
    apply the discretisation scheme according to the multi–level refinement criteria;
    transcribe the OCP ;
    create warm start;
    solve the NLP;
    estimate the discretisation error – $\varepsilon_{\mathbf{x}}^{(j)}$;
    estimate the error on the multipliers – $\varepsilon_{\mathbf{q}}^{(j)}$;
end
```

Algorithm 1: Adaptive time–mesh refinement algorithm considering both refinement criteria

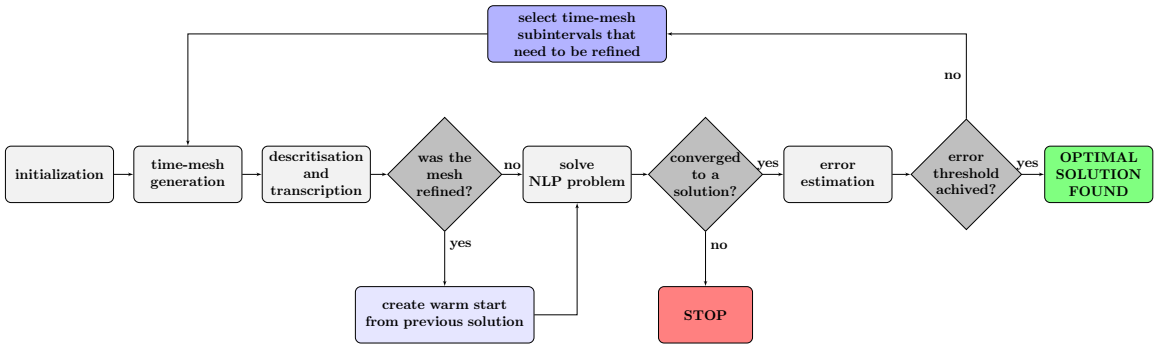


Figure 4.2: Adaptive time–mesh refinement diagram

4.3 Application

To test and to validate the proposed algorithm, a problem involving nonholonomic systems [KM95, PF13] and another one regarding the SEIR model [BPd14, KPP14, NL10] are solved considering, in both cases, a pathwise state constraint.

Both problems, ($\mathrm{P_{CL}}$) and ($\mathrm{P_S}$), are solved considering four meshes:

a) π_{ML} obtained by the multi–level time–mesh refinement strategy considering $\mathcal{N} = 4$;

b) π_{R} obtained by the (single–level) time–mesh refinement strategy considering $\mathcal{N} = 4$;

c) π_{F} considering equidistant–spacing with the lowest time step of π_{ML};

d) π_{S} considering equidistant–spacing with the same number of nodes of π_{ML}.

4.3.1 Car–like System

A car–like system that moves in a plane generally has three degrees of freedom: translation along the two axes in the plane and rotation about the axis perpendicular to the plane. Nevertheless, these vehicles cannot move freely in all three degrees of motion due to their steering constraints. These kind of models are highly nonlinear and, for that reason, it is expected that a refined mesh having non equidistant spacing is more suitable. Such problems belong to the class of nonholonomic systems [KM95].

A system is *nonholonomic* if the velocity set $\mathbf{f}(\mathbf{x}, \mathbb{U})$ does not contain a neighbourhood of the origin.

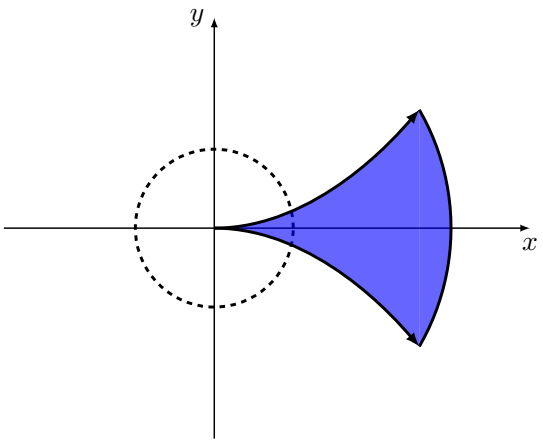


Figure 4.3: Nonholonomic system characterisation: Speed profile

In Fig. 4.4, the geometry of a car–like system is given. For a given time t, $(x(t), y(t))$ is the position of mid–point of the axle connecting the rear wheels, $\psi(t)$ is the yaw angle, $\delta(t)$ is the steering angle and l is the wheelbase of the vehicle, *i.e.*, the distance between its front and rear wheels. We consider also the curvature $c(t)$ which relates to the steering angle δ and the minimum turning radius by

$$c(t) = \frac{\tan(\delta(t))}{l} \quad \text{and} \quad R_{\min} = \frac{1}{|c_{\max}|}\,.$$

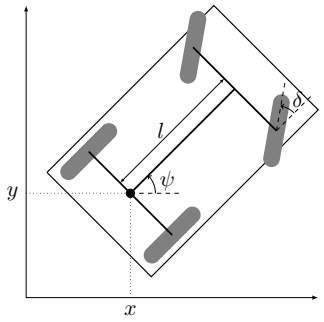


Figure 4.4: Car–like system geometry

4.3.1.1 Problem Statement

Let us consider $t \in [0,10]$, in seconds, $\mathbf{x}(t) = (x(t), y(t), \psi(t))$ and $\mathbf{u}(t) = (u(t), c(t))$. Aiming minimum energy, the car–like system problem ($\mathrm{P_{CL}}$) can be stated as:

$$\text{Minimize} \int_0^{10} u^2(t)\mathrm{dt} \tag{4.2}$$

subject to

- the dynamic constraints

$$\begin{aligned} \dot{x}(t) &= u(t)\cos(\psi(t)) && \text{a.e. } t \in [0,10] \\ \dot{y}(t) &= u(t)\sin(\psi(t)) && \text{a.e. } t \in [0,10] \\ \dot{\psi}(t) &= u(t)\,c(t) && \text{a.e. } t \in [0,10]\,, \end{aligned} \tag{4.3}$$

  where $u(t)$ is the speed and $c(t)$ is the curvature,

- the input constraints

$$\begin{aligned} 0 \leq u(t) \leq 1 && \text{a.e. } t \in [0,10] \\ -0.7 \leq c(t) \leq 0.7 && \text{a.e. } t \in [0,10]\,, \end{aligned}$$

- the end–point constraints

$$\mathbf{x}(0) = \mathbf{x}_0 = (x_0, y_0, \psi_0) = (0,0,0) \tag{4.4}$$

$$\mathbf{x}(10) \in \mathbb{X}_1 = \left\{ (x,y,\psi) : \left(x - x_f\right)^2 + \left(y - y_f\right)^2 + \left(\psi - \psi_f\right)^2 \leq r^2 \right\}, \tag{4.5}$$

  where $r^2 = 0.1$ and $\mathbf{x}_f = (x_f, y_f, \psi_f) = (10,0,0)$ is a user–defined target point, and

- the pathwise state constraint

$$g\left(\mathbf{x}(t)\right) = (\bar{y} - y(t)) - k\left(\bar{x} - x(t)\right)^2 \leq 0\,, \quad \forall t \in [0,10]\,, \tag{4.6}$$

  where $(\bar{x}, \bar{y}) = (5,1)$ and $k = 10$.

The goal is to drive this car–like system with minimum energy from $\mathbf{x}_0$ to some point near $\mathbf{x}_f$ according to the terminal condition (4.5) while overcoming the state constraint (4.6).

4.3.1.2 Numerical Results

For this problem (P_{CL}), we consider

$$\begin{aligned}\varepsilon_{\mathbf{x}}^{\max} &= 5 \times 10^{-5} \\ \varepsilon_{\mathbf{q}}^{\max} &= 5 \times 10^{-4} \\ \bar{\varepsilon}_{\mathbf{x}} &= \left[1, 10, 10^2, 10^3, 10^4, 10^5\right] \varepsilon_{\mathbf{x}}^{\max} \\ \bar{\varepsilon}_{\mathbf{q}} &= \left[1, 10, 10^2, 10^3, 10^4\right] \varepsilon_{\mathbf{q}}^{\max}\end{aligned}$$

where $\varepsilon_{\mathbf{x}}^{\max}$ is used in the stopping criterion and the vectors $\bar{\varepsilon}_{\mathbf{x}}$ and $\bar{\varepsilon}_{\mathbf{q}}$ are used in the refinement criteria.

As it can be seen in Fig. 4.5, the car–like system successfully overcomes the obstacle and it stops when the terminal condition (4.5) is satisfied.

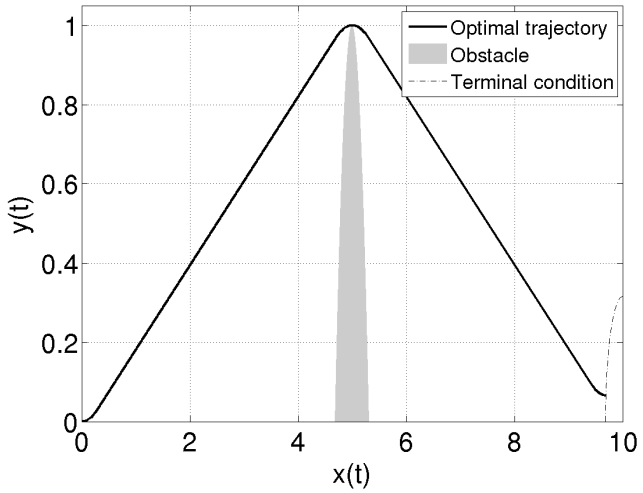
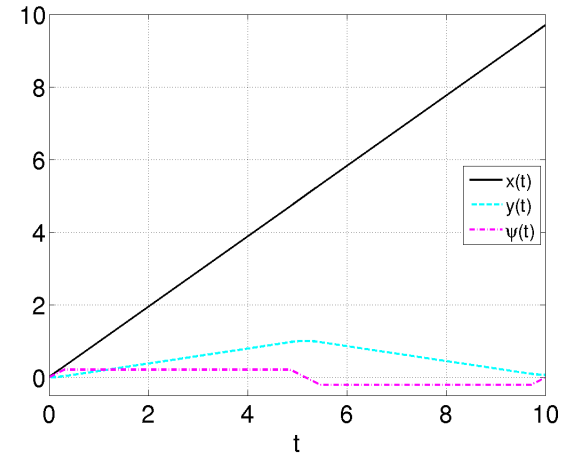


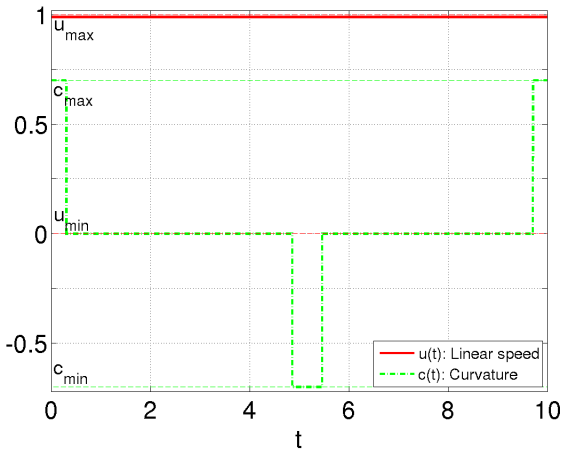
Figure 4.5: Optimal trajectory for (P_{CL})

According to Fig. 4.6b, where the controls associated to (P_{CL}) are shown, the constraint for the curvature $c(\cdot)$ becomes active at the start, in the middle and at the end of the trajectory.

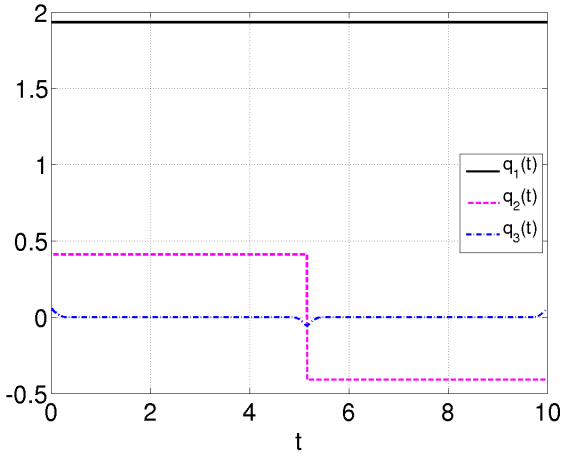
The local errors of the trajectory for all meshes are shown in Fig. 4.7a using a logarithmic scale. The subintervals that need refinement are at the start, in the middle



(a) Optimal solution

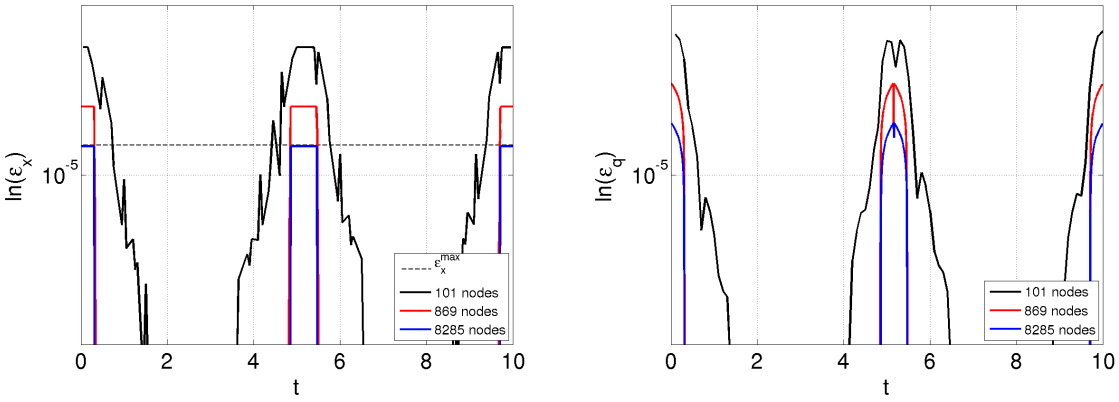


(b) Optimal control



(c) Adjoint multipliers

Figure 4.6: Numerical results of (P_{CL}) using π_{ML}



(a) Error on the trajectory $\left(\varepsilon_{\mathbf{x}}^{(j)}\right)$ (b) Error on the $\mathbf{q}$ multipliers $\left(\varepsilon_{\mathbf{q}}^{(j)}\right)$

Figure 4.7: Local error for ($\mathrm{P_{CL}}$) using all meshes

and at the end of the time interval, since the local errors are greater than the user–specified threshold, coinciding with the subintervals where the curvature is nonzero and the constraint for the curvature becomes active. We also notice that there are different subintervals belonging to different levels of refinement which indicates that the procedure to generate π_{ML} is quite distinct from the one used to generate π_{R}.

The adjoint multipliers are presented in Fig. 4.6c and the local error associated to them is shown in Fig. 4.7b. Comparing Fig. 4.7a with Fig. 4.7b, we see that the errors in the trajectory and in the adjoint multiplier have a similar structure along time, further validating the use of the adjoint multiplier in the refinement criteria.

The numerical results concerning the four meshes are shown in Table 4.1, which shows information about the number of nodes, the smallest time step, the number of iterations needed to solve the NLP problem, the objective functional, the maximum absolute local errors of the trajectory and the $\mathbf{q}$ multipliers, and the CPU times for solving the OCP problem and for computing the local error as well.

According to Table 4.1, the mesh π_{ML} has only 32.4% of the nodes of π_{F}, nevertheless both meshes have maximum absolute local errors of the same order of magnitude. Since the procedure to obtain π_{ML} uses a warm start at each refinement iteration, the OCP can be solved three consecutive times and it is still much faster than

Table 4.1: Comparing results for the Car–like system problem (P_{CL})

π_j	N_j	Δt_j	I_j	Objective	$\left\|\varepsilon_{\mathbf{x}}^{(j)}\right\|_\infty$	$\left\|\varepsilon_{\mathbf{q}}^{(j)}\right\|_\infty$	CPU time (s) Solver	$\varepsilon_{\mathbf{x}}$	$\varepsilon_{\mathbf{q}}$
π_0	101	$1/100$	25	**9.7**700173	1.029E^{-2}	2.443E^{-2}	1.859	0.638	0.004
π_1	869	$1/25600$	34	**9.7805**247	4.096E^{-4}	1.444E^{-3}	8.895	6.345	0.007
π_2	8285	$1/25600$	50	**9.78053**98	4.714E^{-5}	1.684E^{-4}	186.870	102.973	0.018
π_{ML}	8285	$1/25600$	109	**9.78053**98	4.714E^{-5}	1.684E^{-4}	197.624	109.956	0.029
π_0	101	$1/100$	25	**9.7**700173	1.029E^{-2}	2.443E^{-2}	1.859	0.638	0.004
π_1	182	$1/400$	21	**9.7**798518	2.572E^{-3}	8.180E^{-3}	2.806	1.626	0.004
π_2	577	$1/1600$	31	**9.780**4932	6.407E^{-4}	3.061E^{-3}	7.938	5.494	0.007
π_3	2193	$1/6400$	38	**9.7805**272	1.598E^{-4}	7.837E^{-4}	35.156	23.129	0.013
π_4	8707	$1/25600$	31	**9.78053**92	3.996E^{-5}	1.971E^{-4}	162.731	118.277	0.021
π_R	8707	$1/25600$	153	**9.78053**92	3.996E^{-5}	1.971E^{-4}	210.490	149.164	0.049
π_F	25601	$1/25600$	406	**9.78053**77	3.996E^{-5}	–	4840.185	773.192	–
π_S	8285	$1/8284$	80	**9.7805**577	1.236E^{-4}	–	333.428	113.912	–

to solve this problem with the mesh π_F. In fact, computing the solution using π_{ML} takes only 4% of the time needed to get a solution using π_F, causing significant savings in memory and computational cost. The use of multi–level refinement algorithm in real time optimisation problems, such as MPC, has benefits since it is possible to obtain a solution very quickly even if the procedure is interrupted in an early stage. According to Table 4.1, if the procedure is interrupted after 12 seconds, a solution with local error lower than 4.096×10^{-4} is obtained.

The mesh π_S has the same number of nodes of π_{ML} but considering equidistant spacing. The analysis of the solution obtained using this mesh allows us to verify

the importance of nodes collocation, *i.e.*, having a mesh with non–equidistant spacing nodes produces a solution with higher accuracy than the one obtain using a mesh with equidistant spacing nodes. Moreover, the CPU time spent to compute solution using π_{ML} is 31,3% lower than the one spent to obtain a solution using π_{S}, emphasizing the relevance of using meshes with non–equidistant spacing nodes.

When comparing both refined meshes, we notice that the process to compute the mesh π_{ML} having several refinement levels in a single iteration took only 2 refinement iterations, producing a mesh that has 95% of the nodes of π_{R}. The solution is obtained 6% faster when compared to the CPU time spent to compute the solution using π_{R}. We could expect to be even faster but, as shown in the Table 4.1, the second refinement iteration takes 50 IPOPT iterations to converge to the optimal solution. This event occurs because the solution of the previous refinement iteration, which is used to create a warm start, has only about 10% of the nodes, having less information about the structure of the solution.

In terms of CPU time, we can see that it is much faster to compute $\varepsilon_{\mathbf{q}}$ than $\varepsilon_{\mathbf{x}}$. In the all procedures, the use of $\varepsilon_{\mathbf{q}}$ in the refinement criterion reduces the computational time, making the refinement algorithm faster.

With respect to IPOPT and according to Table 4.1, even if the number of nodes is increasing fast at each refinement step, the number of IPOPT iterations are of the same order of magnitude. This is another advantage of the mesh refinement strategy.

4.3.1.3 Characterisation of the Solution using the Necessary Conditions of Optimality

In this section we establish necessary conditions of optimality for the car–like system problem ($\mathrm{P_{CL}}$) considering the pathwise state constraint (4.6). Then, we verify that the solution obtained numerically satisfies the necessary conditions. Similar analysis to the second problem ($\mathrm{P_S}$) involving the SEIR model is reported in [BPd14].

Considering the problem ($\mathrm{P_{CL}}$), let us recall

$$
\begin{aligned}
&\mathrm{L}\,(t,\mathbf{x},\mathbf{u}) = u^2\,,\\
&\mathrm{G}\left(\mathbf{x}(t_f)\right) = 0\,,\\
&\mathbf{f}(t,\mathbf{x},\mathbf{u}) = [u\cos(\psi),\, u\sin(\psi),\, u\,c]\,,\\
&g\,(\mathbf{x}) = (1-y) - 10\,(5-x)^2\,,\\
&\mathbb{U} = [0,1]\times[-0.7,0.7]\,,\\
&\mathbf{x}(t_0) = (0,0,0)\,,\\
&\mathbb{X}_1 = \left\{(x,y,\psi) : \left(x-x_f\right)^2 + \left(y-y_f\right)^2 + \left(\psi-\psi_f\right)^2 \leq r^2\right\}.
\end{aligned}
$$

The assumptions (H1)–(H6) presented in section 2.2 are satisfied.

A nontrivial choice of multipliers can be made if there exists a continuous feedback $\mathbf{u} = \eta(t,\xi)$ such that

$$h_t(t,\xi) + h_{\mathbf{x}}(t,\xi)\cdot\mathbf{f}(t,\xi,\eta(t,\xi)) < -\delta' \tag{4.7}$$

for some positive δ', wherever (t,ξ) is close to the graph of $\mathbf{x}^*(\cdot)$ and ξ is near the state constraint boundary [RV99]. Moreover, in this case the Maximum Principle can be written with $\lambda = 1$.

Let us recall that $\mathbf{x}(t) = (x(t),\, y(t),\, \psi(t))$, $\mathbf{u}(t) = (u(t),\, c(t))$ and

$$h\,(t,\mathbf{x}(t)) = (\bar{y} - y(t)) - k\,(\bar{x} - x(t))^2\,.$$

Considering the pathwise state constraint (4.6), from (4.7) we may write

$$h_{\mathbf{x}}(t,\xi)\cdot\mathbf{f}(t,\xi,\eta(t,\xi)) = -2k\,(\bar{x}-\xi_1)\,\eta_1(t,\xi)\,\cos(\xi_3) - \eta_1(t,\xi)\,\sin(\xi_3) < -\delta' \tag{4.8}$$

where $\xi = (\xi_1,\xi_2,\xi_3)$ and $\eta = (\eta_1,\eta_2)$. For a ξ in a neighbourhood of $\mathbf{x}^*(\cdot)$ we can choose η sufficiently large satisfying the equation (4.8). Thus, when the trajectory is in a neighbourhood of the boundary, there exists a control that drives the car–like system away from the state constraint boundary.

Proposition 1. *Consider problem* ($\mathrm{P_{CL}}$). *A local minimizer* $(\mathbf{x}^*, \mathbf{u}^*)$ *satisfies*

1. $(\mathbf{p}, \mu, \lambda) \neq (0,0,0)$,
2. $\begin{cases} p_1(t) = p_1(t_0) \in \mathbb{R}, & t \in \left[t_0, t_f\right] \\ p_2(t) = p_2(t_0) \in \mathbb{R}, & t \in \left[t_0, t_f\right] \end{cases}$,
3. $-\mathbf{q}(t_f) \in N_{\mathbb{X}_1}\left(x_f^*, y_f^*, \psi_f^*\right) = \alpha\left(2\left(x_f^* - x_f\right), 2\left(y_f^* - y_f\right), 2\left(\psi_f^* - \psi_f\right)\right)$ *where* $\alpha > 0$ *and* $(x_f^*, y_f^*, \psi_f^*) = \left(x^*(t_f), y^*(t_f), \psi^*(t_f)\right)$,
4. $\mathrm{H}\left(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*\right) = \max\limits_{u,c \in \mathbb{U}} q_1 u \cos(\psi^*) + q_2 u \sin(\psi^*) + q_3 uc - u^2$,
5. $\operatorname{supp}\{\mu\} \subset I\left(\mathbf{x}^*\right) = \left\{t \in \left[t_0, t_f\right] : (\bar{y} - y(t)) - k\left(\bar{x} - x(t)\right)^2 = 0\right\}$.

Proof. The nontriviality condition (NT) is ensured with $\lambda = 1$.

Considering $\mathbf{p}(t) = (p_1(t), p_2(t), p_3(t))$ and $\mathbf{q}(t) = (q_1(t), q_2(t), q_3(t))$, we define the Hamiltonian as

$$\mathrm{H}(t, \mathbf{x}, \mathbf{q}, \mathbf{u}) = q_1 u \cos(\psi) + q_2 u \sin(\psi) + q_3 uc - u^2 . \tag{4.9}$$

From the adjoint system

$$\begin{aligned} -\dot{p}_1(t) &= \mathrm{H}_x(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = 0 && \text{a.e. } t \in \left[t_0, t_f\right] \\ -\dot{p}_2(t) &= \mathrm{H}_y(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = 0 && \text{a.e. } t \in \left[t_0, t_f\right] \\ -\dot{p}_3(t) &= \mathrm{H}_\psi(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = -q_1 u^* \sin(\psi^*) + q_2 u^* \cos(\psi^*) && \text{a.e. } t \in \left[t_0, t_f\right] . \end{aligned} \tag{4.10}$$

Since $\mathbf{p}(\cdot)$ is an absolutely continuous function, we get

$$\begin{aligned} p_1(t) = p_1(t_0) \in \mathbb{R}, \quad t \in \left[t_0, t_f\right] \\ p_2(t) = p_2(t_0) \in \mathbb{R}, \quad t \in \left[t_0, t_f\right] \end{aligned}$$

concluding that $q_1(\cdot)$ and $q_2(\cdot)$ may change their value just when $t \in \{t \in \left[t_0, t_f\right] : h\left(\mathbf{x}^*(t)\right) = 0\}$. In fact, according to Fig. 4.6c, $q_1(t)$ is a constant function and $q_2(t)$ changes its value just in one time instant when the trajectory hits the state constraint.

Considering $\mathrm{G}(\cdot)=0$, the end–point constraints (4.4) and (4.5), the transversality condition (T) reads

$$-\mathbf{q}(t_f)\in N_{\mathbb{X}_1}\left(x_f^*,y_f^*,\psi_f^*\right)=\alpha\left(2\left(x_f^*-x_f\right),2\left(y_f^*-y_f\right),2\left(\psi_f^*-\psi_f\right)\right)$$

where $\alpha>0$ and $(x_f^*,y_f^*,\psi_f^*)=\left(x^*(t_f),y^*(t_f),\psi^*(t_f)\right)$, and according to the Weierstrass condition (WC)

$$\mathrm{H}\left(t,\mathbf{x}^*,\mathbf{q},\mathbf{u}^*\right)=\max_{u,c\in\mathbb{U}} q_1 u\cos(\psi^*)+q_2 u\sin(\psi^*)+q_3 uc-u^2\,. \tag{4.11}$$

Recalling the pathwise state constraint (4.6), from (CS) we obtain

$$\operatorname{supp}\{\mu\}\subset I\left(\mathbf{x}^*\right)=\left\{t\in\left[t_0,t_f\right]:(\bar{y}-y(t))-k\left(\bar{x}-x(t)\right)^2=0\right\} \tag{4.12}$$

for some $(\bar{x},\bar{y})\in\mathbb{R}^2$ and $k\in\mathbb{R}$. Moreover,

$$\nabla g\left(\mathbf{x}^*(t)\right)=(2k(\bar{x}-x^*(t)),-1,0) \tag{4.13}$$

and

$$q_1(t)=p_1(t)+2k\int\limits_{[t_0,t)}(\bar{x}-x^*(t))\mathrm{d}\mu(s) \tag{4.14}$$

$$q_2(t)=p_2(t)-\int\limits_{[t_0,t)}\mathrm{d}\mu(s) \tag{4.15}$$

$$q_3(t)=p_3(t)\,, \tag{4.16}$$

implying $q_3(\cdot)$ is an absolutely continuous function. □

Applying the Maximum Principle we obtain a set of conditions characterising the optimal solution which are in agreement with the numerical results (cf.Fig. 4.6c)

Proposition 2. *Let* $(\mathbf{x}^*,\mathbf{u}^*)$ *be a local minimizer to the problem* ($\mathrm{P_{CL}}$). *Applying the Maximum Principle 2.2.4, we conclude that*

1. *if* $u^*,c^*\in\operatorname{int}\mathbb{U}$ *then*

$$\begin{cases} u^*=\dfrac{q_1}{2}\cos(\psi^*)+\dfrac{q_2}{2}\sin(\psi^*) \\ q_3=0 \end{cases}, \tag{4.17}$$

2. if $c^* = c_{\min}$*, then* $q_3 \leq 0$*, and*

3. if $c^* = c_{\max}$*, then* $q_3 \geq 0$*.*

Proof. Let us first consider the case when $\mathbf{u} \in \operatorname{int} \mathbb{U}$. In this case, the Hamiltonian, stated in proposition 1, is maximized when

$$\mathrm{H}_{\mathbf{u}}(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = 0\,.$$

When $\mathbf{u}^* \in \operatorname{int} \mathbb{U}$ we get

$$\begin{cases} \mathrm{H}_u(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = q_1 \cos(\psi^*) + q_2 \sin(\psi^*) + q_3 c^* - 2u^* = 0 \\ \mathrm{H}_c(t, \mathbf{x}^*, \mathbf{q}, \mathbf{u}^*) = q_3 u^* = 0 \end{cases}$$

implying

$$\begin{cases} u^* = \dfrac{q_1}{2} \cos(\psi^*) + \dfrac{q_2}{2} \sin(\psi^*) \\ q_3 = 0 \end{cases}.$$

When $c^* = c_{\min}$, the Weierstrass condition (WC) reads

$$q_3 u^* \left(c_{\min} - c\right) \geq 0\,.$$

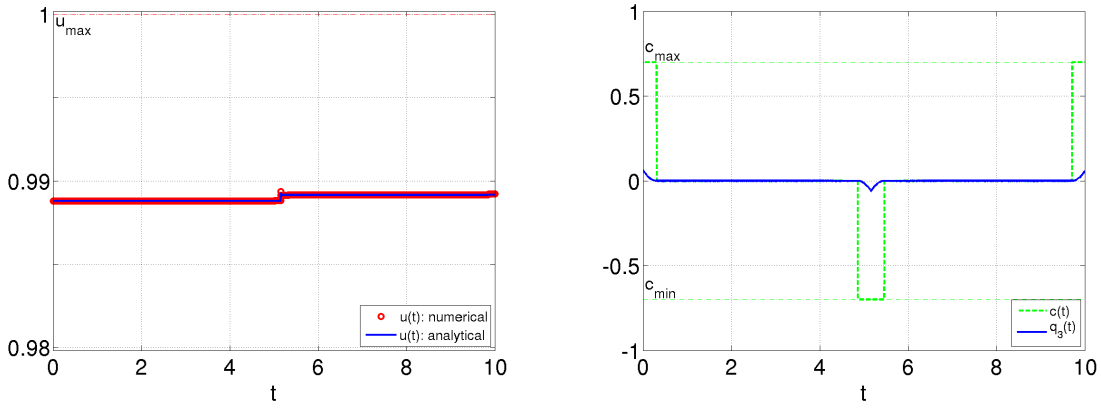
Since $u^* \geq 0$ and $c \geq c_{\min}$, we conclude $q_3 \leq 0$ when $c^* = c_{\min}$. Furthermore, it can be shown that $q_3 \geq 0$ when $c^* = c_{\max}$. □

Remark 1. *We use the numerical results for* ψ^**,* c^* *and* $\mathbf{q}$ *to evaluate* u^* *and we compare it against the* $\hat{u}^*$ *given by the numerical procedure. According to Fig. 4.8a, they coincide.*

In Fig. 4.8b, we provide the graphics of c^* *and* q_3 *obtained numerically and we can verify relationships of the proposition 2.*

4.3.2 The SEIR Model

The SEIR model is a compartmental model that describes the spreading of an infectious disease among a population (N) by dividing it into four different



(a) $u^*(t)$: analytical *vs* numerical (b) Relationship between $c^*(t)$ and $q_3(t)$

Figure 4.8: Solution characterisation for the problem ($\mathrm{P_{CL}}$)

compartments: susceptible (S), exposed but not yet infectious (E), infectious (I) and recovered (R) [BPd14, KPP14, NL10]. SEIR models can represent many human infectious diseases such as measles, pox, flu or dengue. According to [BPd14], we can add to the dynamical system given in [NL10] an extra variable (W), which stands for the number of vaccinated people and which is governed by the differential equation $\dot{W}(t) = u(t)S(t)$. Then, the ordinary differential equation governing $\dot{R} = gI(t) - dR(t) + u(t)S(t)$ can be replaced with the one for $\dot{N}$.

4.3.2.1 Problem Statement

We consider $t \in [0, 20]$, in years, $\mathbf{x}(t) = (S(t), E(t), I(t), N(t), W(t))$ and $\mathbf{u}(t) = u(t)$. This problem ($\mathrm{P_S}$) can be stated as:

$$\text{Minimize} \int_0^{20} 0.1\, I(t) + u^2(t) \mathrm{d}t \tag{4.18}$$

subject to

(i) the dynamic constraints

$$\dot{S}(t) = bN(t) - dS(t) - cS(t)I(t) - u(t)S(t) \qquad \text{a.e. } t \in [0, 20]$$

$$\begin{aligned}
\dot{E}(t) &= cS(t)I(t) - (e+d)E(t) && \text{a.e. } t \in [0,20] \\
\dot{I}(t) &= eE(t) - (g+a+d)I(t) && \text{a.e. } t \in [0,20] \\
\dot{N}(t) &= (b-d)N(t) - aI(t) && \text{a.e. } t \in [0,20] \\
\dot{W}(t) &= u(t)S(t) && \text{a.e. } t \in [0,20]\,,
\end{aligned} \tag{4.19}$$

where $u(t)$ represents the percentage of susceptible individuals being vaccinated per unit of time,

(ii) the input constraints

$$0 \leq u(t) \leq 1\,, \quad \text{a.e. } t \in [0,20] \tag{4.20}$$

(iii) the end–point constraints

$$\mathbf{x}(t_0) = (S_0, E_0, I_0, N_0, W_0)\,, \tag{4.21}$$

(iv) the state constraint $h\,(t, \mathbf{x}(t)) = S(t) - 1100 \leq 0\,,\ \forall t \in [0,20]$.

This problem is nonlinear, thus an adaptive mesh is expected to be more adequate.

4.3.2.2 Numerical Results

For problem ($\mathrm{P_S}$), we consider

$$\begin{aligned}
\varepsilon_{\mathbf{x}}^{\max} &= 5 \times 10^{-5} \\
\varepsilon_{\mathbf{q}}^{\max} &= 5 \times 10^{-4} \\
\bar{\varepsilon}_{\mathbf{x}} &= \left[1, 5, 10, 50, 10^2\right] \varepsilon_{\mathbf{x}}^{\max} \\
\bar{\varepsilon}_{\mathbf{q}} &= \left[1, 5, 10, 50, 10^2\right] \varepsilon_{\mathbf{q}}^{\max}
\end{aligned}$$

The optimal trajectory is shown in Fig. 4.9a and 4.9b and the control is presented in Fig. 4.9c. The local errors of the trajectory for all meshes are shown in Fig. 4.9d using the logarithmic scale. Regarding the latter figure, we see that the time domain needs to be entirely refined in the first step and there are different subintervals belonging to different levels of refinement.

Table 4.2: Parameters with their clinically approved values [NL10].

Parameters	Definition of Parameters	Clinical values
b	natural birth rate	0.525
d	natural death rate	0.5
c	incidence coefficient	0.001
e	exposed to infectious rate	0.5
g	recovery rate	0.1
a	disease induced death rate	0.2
S_0	initial susceptible population	1000
E_0	initial exposed population	100
I_0	initial infected population	50
R_0	initial recovered population	15
N_0	initial population	1165
W_0	initial vaccinated Population	0

The numerical results concerning the four meshes are shown in Table 4.3, which, as before, shows information about the number of nodes, the smallest time step, the number of iterations needed to solve the NLP problem, the objective functional, the maximum absolute local error of the trajectory and the CPU times for solving the OCP problem and for computing the local error as well.

According to Table 4.3, the mesh π_{ML} has 85% of the nodes of π_{F} and computing the solution using π_{ML} takes only 30% of the CPU time needed to get a solution using π_{F}. Nevertheless, we get solution with the same accuracy.

The solution obtained using the mesh π_{S}, which has the same number of nodes of π_{ML} but with equidistant spacing, has less accuracy than the one computed on π_{ML}. Moreover, computing the solution on π_{S} took 3 times the CPU time spent to get the

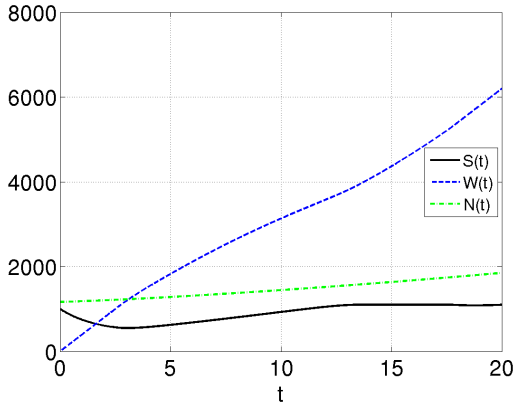
Table 4.3: Comparing results for the SEIR problem (P_S)

π_j	N_j	Δt_j	I_j	Objective	$\left\|\varepsilon_{\mathbf{x}}^{(j)}\right\|_\infty$	$\left\|\varepsilon_{\mathbf{q}}^{(j)}\right\|_\infty$	CPU time (s) Solver	$\varepsilon_{\mathbf{x}}$	$\varepsilon_{\mathbf{q}}$
π_0	101	$1/100$	56	**25**.65774542	1.246E^{-3}	1.246E^{-2}	4.121	0.907	0.007
π_1	1589	$1/6400$	24	**25.5**8223462	1.285E^{-4}	1.285E^{-4}	21.877	14.611	0.012
π_2	5498	$1/6400$	23	**25.5787**9504	4.191E^{-5}	4.491E^{-5}	64.236	76.057	0.018
π_{ML}	5498	$1/6400$	103	**25.5787**9504	4.191E^{-5}	4.491E^{-5}	90.234	91.575	0.037
π_0	101	$1/100$	56	**25**.65774542	1.246E^{-3}	1.246E^{-2}	4.121	0.907	0.007
π_1	401	$1/400$	18	**25.5**9608912	3.115E^{-4}	4.894E^{-3}	4.672	3.423	0.009
π_2	1601	$1/1600$	21	**25.5**8021428	7.786E^{-5}	1.261E^{-3}	16.850	16.439	0.014
π_3	5789	$1/6400$	25	**25.5787**5577	3.998E^{-5}	3.379E^{-5}	85.464	83.978	0.022
π_{R}	5789	$1/6400$	120	**25.5787**5577	3.998E^{-5}	3.379E^{-5}	111.117	204.747	0.052
π_{F}	6401	$1/6400$	70	**25.5787**1473	3.646E^{-5}	–	292.155	99.496	–
π_{S}	5498	$1/5497$	66	**25.5784**340705	2.266E^{-4}	–	254.806	87.912	–

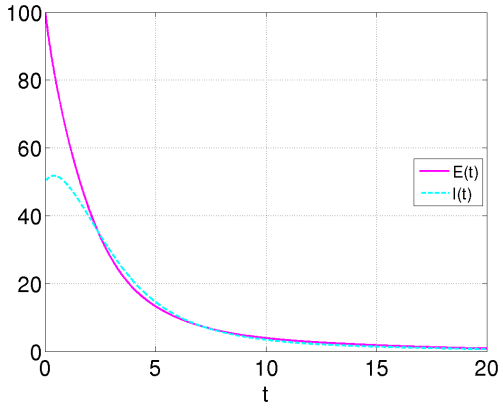
solution using π_{ML}.

When comparing both refined meshes, we notice that the process to compute the mesh π_{ML} having several refinement levels in a single iteration took only 2 refinement iterations, producing a mesh that has 95% of the nodes of π_{R}. The use of the refinement levels brings a significant improvement in the overall computing time, since the solution is obtained 19% faster when compared to the time spent to compute the solution using π_{R}.

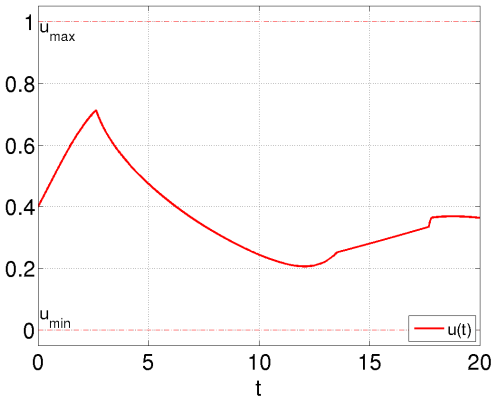
With respect to CPU time, once again, we see that it is much faster to compute $\varepsilon_{\mathbf{q}}$ than $\varepsilon_{\mathbf{x}}$. Also in this application, the use of $\varepsilon_{\mathbf{q}}$ as refinement criterion reduces the



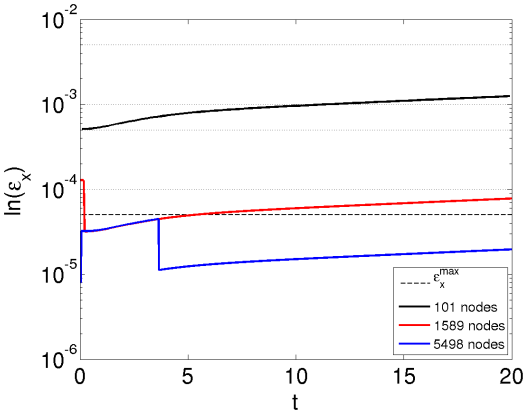
(a) Optimal trajectory: $S(t)$, $W(t)$, $N(t)$



(b) Optimal trajectory: $E(t)$, $I(t)$



(c) Vaccination strategy: $u(t)$



(d) Error on the trajectory: $\left(\varepsilon_{\mathbf{x}}^{(j)}\right)$

Figure 4.9: Results for the problem (P_S)

computational time, making the refinement algorithm quicker.

In terms of IPOPT and according to Table 4.3, the number of IPOPT iterations are of the same order of magnitude at each refinement step, even if the number of nodes is increasing.

Among other results, the characterisation of the solution for this problem (P_S), using the necessary conditions of optimality and considering state constraints as well as mixed constraints, is reported in [BPd14] and [Bis13].

4.4 Final Remarks

We develop an adaptive mesh refinement algorithm providing local mesh resolution refinement only where it is required. In the end, the OCP is solved using an adapted mesh which has less nodes in the overall procedure, yet with higher concentration of nodes in time subintervals where the trajectory shows higher nonlinear behaviour. This procedure showed significant savings in memory and computational cost when compared to equidistant meshes.

When using this strategy, where the mesh is progressively refined to catch special features of the problem, there is no need to define *a priori* the most appropriately mesh, which is another advantage of this procedure. According to the proposed algorithm, we do not need to remove nodes, nevertheless this algorithm can be extended to a version that includes to coarsen the mesh. This feature would be of relevance in the context of MPC in which sequences of similar optimal control problems are solved.

Due to the fast response of the algorithm, it can be used to solve real–time optimisation problems, in particular, in model predictive control. The use of adaptive mesh refinement algorithm in real time optimisation problems, such as MPC, has additional benefits since it is possible to quickly obtain a solution even if the procedure is interrupted at an early stage.

The applications presented in this chapter demonstrate the advantage of the proposed adaptive mesh strategy, which leads to results with greater accuracy and with lower overall computational time when compared to other commonly used approaches.

Chapter 5

Time–Mesh Refinement for Model Predictive Control

"Prediction is very difficult,
especially if it's about the future."
Niels Bohr

More than 15 years after Model Predictive Control (MPC) appeared in industry, known to be an effective way to deal with multivariable constrained control problems [RRTP76], a theoretical basis for this technique has started to emerge [MM90]. MPC has become a preferred control strategy for a large number of processes and the main reasons are the ability to handle constraints in an optimal way and the flexible formulation in the time domain [ABQ+99].

Some questions regarding the feasibility of the on–line optimisation, stability and performance are largely understood for systems described by linear models. Much progress has been made on these issues also for nonlinear systems [FP12a, GP11] but for practical applications many questions remain, making MPC an interesting topic of research.

In this chapter, we provide the principles underlying MPC, its advantages and

some computational aspects. We introduce the generic MPC algorithm and, by adapting the algorithm 1 described on Chapter 4, we propose an adaptive time–mesh refinement algorithm in the MPC context. To our knowledge this is the first time that the time–mesh is adapted while using the MPC technique to solve an OCP.

5.1 Introduction

Model Predictive Control (MPC), also referred to as moving horizon control or receding horizon control, is an optimisation based method for the feedback control. The first term points out the use of model based predictions, while the second one highlights the moving horizon idea [Fon01, Raw00].

The idea of MPC – linear or nonlinear – is to use a model of the process in order to predict the system and optimise its future behaviour. Regarding the Linear MPC, it involves MPC schemes in which linear models are used to predict the system dynamics, even though the dynamics of the closed–loop system is nonlinear. Linear MPC approaches have found successful applications, especially in the process industries, in a very wide range from chemicals to aerospace industries. An overview of commercially available MPC technology can be found in [QB03].

Nevertheless, researchers and industries have, in general, to deal with nonlinear systems. Among other reasons, these nonlinear systems arise from [FA03]:

- higher product quality specifications,
- increasing productivity demands,
- tighter environmental regulations,
- demanding economical considerations,

requiring the process industry to operate systems closer to the boundary of the admissible operating region. In these cases, linear models are often inappropriate

to describe the process dynamics and nonlinear models emerge, motivating the use of Nonlinear MPC. Nonlinear MPC has become an attractive feedback strategy and its primary applications are stabilization and tracking problems [GP11].

The receding horizon control strategy is especially useful for the control of slow nonlinear systems, such as chemical batch processes, where it is possible to solve, sequentially, open–loop fixed–horizon optimal control problems on–line [MM90]. When applying this control strategy, the current control action is obtained by solving, at each sampling instant, a finite horizon open–loop optimal control problem, using the current state of the plant as the initial state. The optimisation procedure gives us an optimal control sequence and the first control in this sequence is applied to the plant [CB04, FA03, Fon03, MRRS00].

The nonlinear MPC is a technique that can be use in real time applications and it can be implemented for large-scale systems. It guarantees on feasibility and stability [Fon01], and robustness [MM93, FM03, RLL$^+$09].

5.2 Principle of Model Predictive Control

If there were no disturbances and no model–plant mismatch, and if the optimisation problem could be solved for infinite horizons, then we could apply the input function found at time $t = 0$ to the system for all times $t \geq 0$. However, this is not possible in general. Due to disturbances and model–plant mismatch, the real behaviour of the system is different from the predicted one. In this case, we can implement a MPC methodology in order to update the trajectory and to correct the behaviour of the system.

The MPC problem is formulated as solving on–line a sequence of finite horizon open–loop OCP subject to system dynamics and constraints involving states and controls. Figure 5.1 shows the basic principle of MPC.

Let us suppose that we have a controlled process whose state $\mathbf{x}(\cdot)$ is measured at discrete time instants t_i, $i = 0, 1, 2, \ldots$. Since it is a controlled process, we can alter the future behaviour of the state of the system by selecting a certain control input $\mathbf{u}(\cdot)$. Based on measurements obtained at the time instant t_k, the controller predicts the future input such that a predetermined open–loop performance objective functional is optimised. Then, the open–loop control is implemented until the next measurement becomes available. Using the new measurement at the time instant $t_k + \delta_k$, where δ_k is the sampling time step, the whole procedure – prediction and optimisation – is repeated to find a new input function with the control and prediction horizon moving forward [Fon01, MHL99]. The sampling step of the MPC procedure is often considered to be fixed, *i.e.*, the measurement takes place every $\delta_k = \delta$ sampling time–units.

MPC can, also, be used in tracking control. In this case, the main purpose is to determine the control inputs $\mathbf{u}$ such that $\mathbf{x}$ follows a given reference $\mathbf{x}_{ref}$ as good as possible. Thus, if the current state $\mathbf{x}(t_k)$ is close to the reference then we want to preserve it there, otherwise if the current state is aside from the reference then we want to control the system towards the reference $\mathbf{x}_{ref}(t_k)$ [GP11].

5.3 Mathematical Formulation of Nonlinear Model Predictive Control

Let us consider the following OCP:

$$\text{Minimise} \quad \int_{t_0}^{t_f} \mathrm{L}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \mathrm{d}t + \mathrm{G}\left(\mathbf{x}(t_f)\right) \tag{5.1}$$

$$\text{subject to} \quad \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \qquad \text{a.e. } t \in \left[t_0, t_f\right], \tag{5.2}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0\,, \tag{5.3}$$

$$\mathbf{x}(t_f) \in \mathbb{X}_1 \subset \mathbb{R}^n\,, \tag{5.4}$$

$$\mathbf{x}(t) \in \mathbb{X} \subset \mathbb{R}^n \qquad \text{a.e. } t \in \left[t_0, t_f\right], \tag{5.5}$$

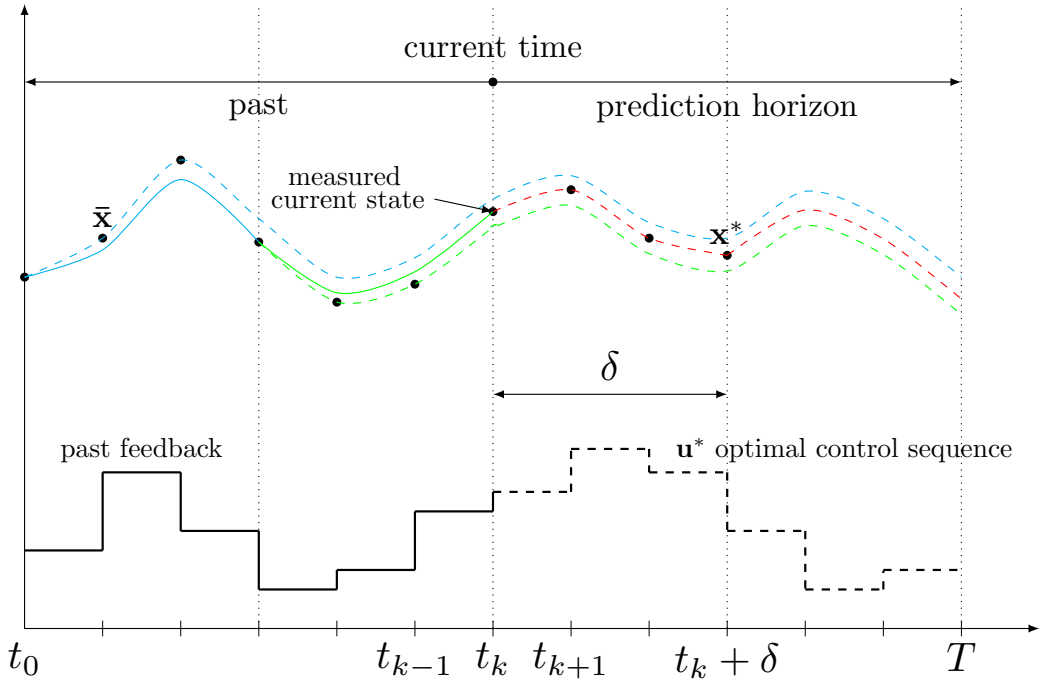


Figure 5.1: Principle of model predictive control

$$\mathbf{u}(t) \in \mathbb{U} \subset \mathbb{R}^m \qquad \text{a.e. } t \in \left[t_0, t_f\right], \tag{5.6}$$

where $\mathbf{x} : \left[t_0, t_f\right] \to \mathbb{R}^n$ is the state, $\mathbf{u} : \left[t_0, t_f\right] \to \mathbb{R}^m$ is the control and $t \in \left[t_0, t_f\right]$ is time. As before, the functions involved comprise the running cost $\mathrm{L} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, the terminal cost $\mathrm{G} : \mathbb{R}^n \to \mathbb{R}$ and the system dynamics $\mathbf{f} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$.

Considering the sampling step $\delta > 0$, the prediction horizon T and a sequence $\{t_i\}_{i \geq 0}$, the MPC algorithm can be implemented in four steps:

1. Measure state of the plant $\mathbf{x}_{t_k}$;

2. Determine $\bar{\mathbf{u}} : [t_k, t_k + T] \to \mathbb{R}^m$ solution to the OCP:

$$\text{Minimise} \quad \int_{t_k}^{t_k+T} \mathrm{L}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \mathrm{d}t + \mathrm{G}\left(\mathbf{x}(t_k + T)\right) \tag{5.7}$$

$$\text{subject to} \quad \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \quad \text{a.e. } t \in [t_k, t_k + T], \tag{5.8}$$

$$\mathbf{x}(t_k) = \mathbf{x}_{t_k}, \tag{5.9}$$

$$\mathbf{x}(t_k + T) \in S \subset \mathbb{R}^n, \tag{5.10}$$

$$\mathbf{x}(t) \in \mathbb{X} \subset \mathbb{R}^n \quad \text{a.e. } t \in [t_k, t_k + T]\,, \tag{5.11}$$

$$\mathbf{u}(t) \in \mathbb{U} \subset \mathbb{R}^m \quad \text{a.e. } t \in [t_k, t_k + T]\,, \tag{5.12}$$

3. Apply the control $\mathbf{u}^*(t) := \bar{\mathbf{u}}(t)$ to the plant in the interval $t \in [t_k, t_k + \delta]$, disregarding the remaining control $\bar{\mathbf{u}}(t), t > t_k + \delta$;

4. Repeat this procedure for the next sampling time instant $t_k + \delta$.

5.4 Extension of the Time–Mesh Refinement Algorithm

We extend the adaptive time–mesh refinement algorithm 1 described on section 4.2 in order to allow different refinement levels according to some partition of the time domain. This extension is of relevance in the MPC context.

5.4.1 Motivation

In MPC context, the prediction can be interpreted in the sense of planning. When we make plans to the future, we establish planning strategies depending on the prediction horizon. When we think about planning our – professional or private – schedule, we do a detail plan for one day (hourly planning), we have a pretty good idea of what we will do the following week (daily planning), and we have some clouded thoughts about what we will do until next year (monthly planning).

Let us consider a time interval $t \in \left[t_0, t_f\right]$, a sampling step $\delta > 0$ and a prediction horizon T. When applying the MPC procedure to solve an OCP, at each time instant t_k we compute the solution in $[t_k, t_k + T]$ but we just implement the open–loop control until $t_k + \delta$. Therefore, taking under consideration the planning strategy discussed above, it would be an improvement if we have an adaptive time–mesh able to cope this feature, *i.e.*, a time–mesh that is highly refined in the lower limit of the time

interval $[t_k, t_k + T]$ and it is coarser in the upper limit of the same interval. Then, we would implement the control on the time interval $[t_k, t_k + \delta]$ computed with high accuracy in a refined mesh. For the remaining time interval we have an estimate of the solution.

Following the described strategy, we obtain an adaptive time–mesh refinement algorithm which generate meshes with higher concentration of node points in the beginning of the interval $[t_k, t_k + T]$ and less concentration of node points in the end of the same interval, enforcing the idea of having more nodes point where they are needed and keeping a low overall number of node points. This is an important issue because we want to increase the accuracy of the solution without compromising low CPU times.

5.4.2 Time–Mesh Refinement Algorithm

As in section 4.2, the time interval is divided in K intervals

$$\mathcal{S}_k = [\tau_{k-1}, \tau_k[, \quad k = 1, \ldots, K-1 \qquad \text{and} \qquad \mathcal{S}_K = [\tau_{K-1}, \tau_K]$$

where $(\tau_0, \ldots, \tau_K)$ coincide with nodes.

We also recall the concept of level of refinement. The intervals $\mathcal{S}_k$ that verify the refinement criteria are divided into smaller subintervals according to the user–defined levels of refinement

$$\bar{\varepsilon} = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m] \, .$$

A subinterval $\mathcal{S}_{k,i}$ is at the i^{th} level of refinement if

$$S_{k,i} = \{t \in S_k \ : \ \varepsilon(t) \in [\varepsilon_i, \varepsilon_{i+1}[\}$$

for $i = 1, \ldots, m$, and it will be refined by adding $\mathcal{N}^i$ of equidistant nodes between each two mesh points.

In this extension, we also consider a time–dependent stopping criterion for the mesh refinement algorithm with different levels $\bar{\varepsilon}_{\mathbf{x}}(t)$. Instead of having a fixed

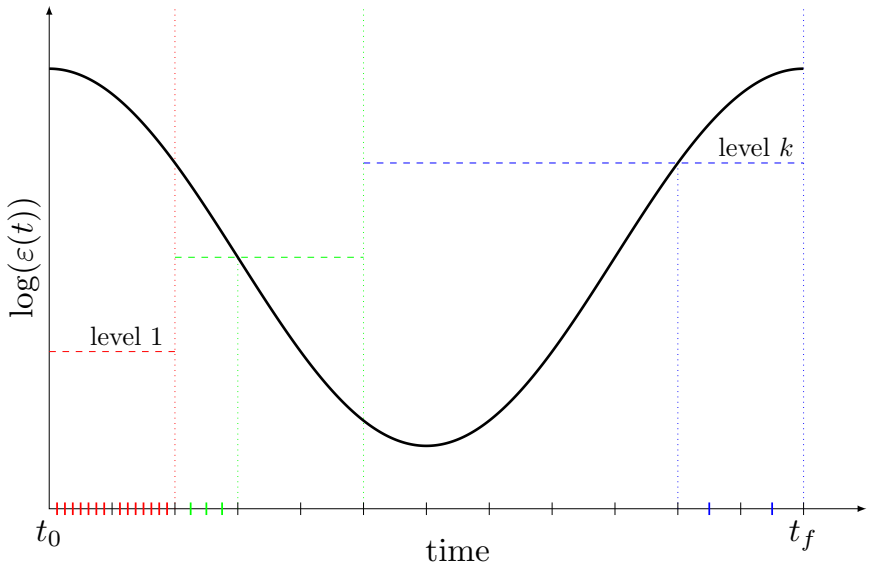


Figure 5.2: Illustration of the extended time–mesh refinement strategy

stopping criterion $\varepsilon_{\mathbf{x}}^{\max}$, now we have a time–dependent $\bar{\varepsilon}_{\mathbf{x}}(t)$ stopping criterion which sets different levels of accuracy for the solution, along the time domain. For example, the time–dependent levels of refinement can be defined as

$$\bar{\varepsilon}_{\mathbf{x}}(t) = \begin{cases} \varepsilon_{\mathbf{x}}^{\max}, & t \in [t_k, t_k + \beta_1 T] \\ \alpha_1 \varepsilon_{\mathbf{x}}^{\max}, & t \in]t_k + \beta_1 T, t_k + \beta_2 T] \\ \alpha_2 \varepsilon_{\mathbf{x}}^{\max}, & t \in]t_k + \beta_2 T, t_k + \beta_3 T] \\ \dots & \\ \alpha_j \varepsilon_{\mathbf{x}}^{\max}, & t \in]t_k + \beta_j T, t_k + T] \end{cases}$$

where $1 < \alpha_1 < \alpha_2 < \ldots < \alpha_j \leq \varepsilon_{\mathbf{x}}^{\max}$ and $0 < \beta_1 < \beta_2 < \ldots < \beta_j < 1$.

This procedure adds more node points to the subintervals that are in lower levels of the stopping criterion for the refinement procedure, corresponding to time instants close to the initial time. By defining the levels of refinement in this way, we get a more accurate solution in time subintervals close to the current time as illustrated in Fig. 5.2.

5.4.3 Refinement Criteria

In order to proceed with the mesh refinement strategy, we have to define some refinement criteria and a stopping criterion. We consider three refinement criteria:

1. the estimate of the relative error of the trajectory (primal variables) ($\varepsilon_{\mathbf{x}}$),
2. the estimate of the relative error of the adjoint multipliers (dual variables) ($\varepsilon_{\mathbf{q}}$),
3. a combination of both criteria.

5.4.4 Warm Start

Since we are solving a sequence of open–loop OCPs, to decrease the CPU time, when going from the problem in $[t_k, t_k + T]$ to the one in $[t_k + \delta, t_k + T + \delta]$, the solution of the previous previous is used as a warm start for the problem. To create this warm start, the solution obtained in $[t_k, t_k + T]$ is projected in the new mesh in $[t_k + \delta, t_k + T + \delta]$ using the cubic Hermite interpolation. This action proved to be vital in the decreasing of the overall computational time. In particular, we notice that the number of iterations of the NLP solver remains within the same order of magnitude along the procedure.

5.4.5 Model Predictive Control coupled with the Extended Algorithm

We start the MPC procedure in the typical way but considering an adaptive mesh refinement strategy. We descritise the time interval $\left[t_0, t_f\right]$ using the algorithm proposed in Chapter 4 and we solve our OCP in open–loop. Then, we implement the control in the first sampling interval. When starting the next MPC step, we apply the time–mesh refinement strategy in order to find the best mesh suited to the solve

the OCP in the second sampling interval. We repeat this technique until the MPC procedure ends.

In the MPC context, we develop the following algorithm:

1. Measure state of the plant $\mathbf{x}_{t_k}$;

2. (a) Select the intervals $S_{k,j}$ to be refined according to the time–dependent levels of refinement $\bar{\varepsilon}_{\mathbf{x}}(t)$;

   (b) Determine $\bar{\mathbf{u}} : [t_k, t_k + T] \to \mathbb{R}^m$ solution to the OCP:

$$\text{Minimise} \quad \int_{t_k}^{t_k+T} \mathrm{L}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \mathrm{d}t + \mathrm{G}\left(\mathbf{x}(t_k + T)\right) \tag{5.13}$$

$$\text{subject to} \quad \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \quad \text{a.e. } t \in [t_k, t_k + T] \,, \tag{5.14}$$

$$\mathbf{x}(t_k) = \mathbf{x}_{t_k} \,, \tag{5.15}$$

$$\mathbf{x}(t_k + T) \in S \subset \mathbb{R}^n \,, \tag{5.16}$$

$$\mathbf{x}(t) \in \mathbb{X} \subset \mathbb{R}^n \quad \text{a.e. } t \in [t_k, t_k + T] \,, \tag{5.17}$$

$$\mathbf{u}(t) \in \mathbb{U} \subset \mathbb{R}^m \quad \text{a.e. } t \in [t_k, t_k + T] \,, \tag{5.18}$$

3. Apply the control $\mathbf{u}^*(t) := \bar{\mathbf{u}}(t)$ to the plant in the interval $t \in [t_k, t_k + \delta]$, discarding the remaining control $\bar{\mathbf{u}}(t), t > t_k + \delta$;

4. Repeat this procedure for the next sampling time instant $t_k + \delta$.

This MPC algorithm is illustrated in Fig. 5.3.

5.4.6 Algorithm Implementation

To test the algorithm, the proposed procedure was implement in MATLAB R2014a combined with the Imperial College London Optimal Control Software – ICLOCS – version 0.1b [FKvW10]. ICLOCS is an optimal control interface and it uses the IPOPT – Interior Point OPTimizer – solver, which is an open-source software

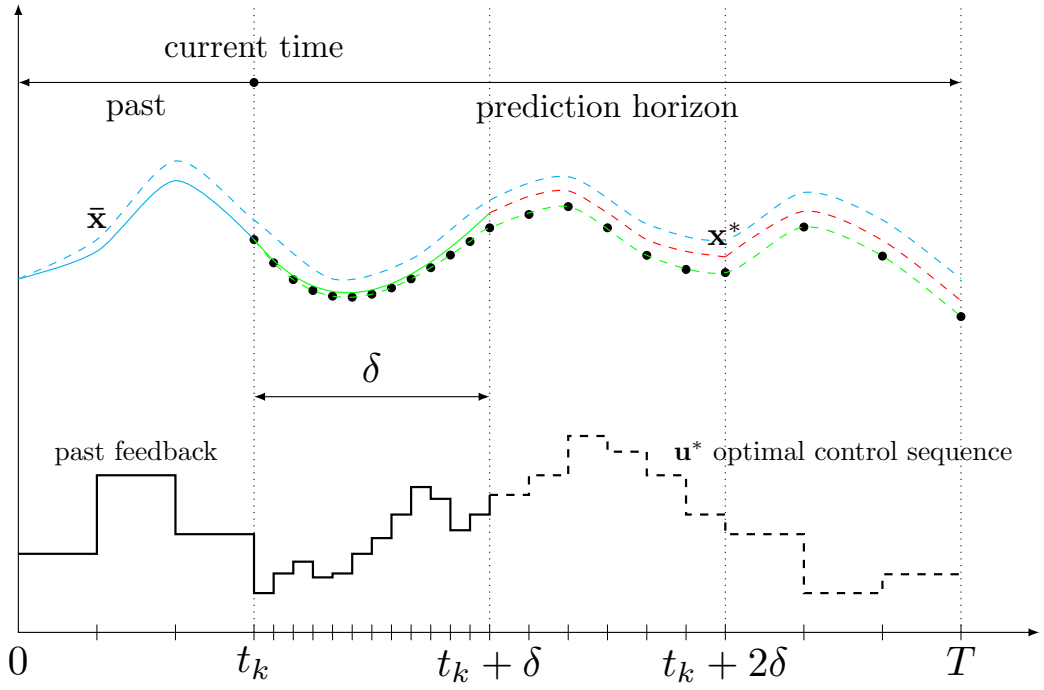


Figure 5.3: Time–mesh refinement algorithm for MPC

package for large-scale nonlinear optimisation [WB06]. The problems are solved in a computer with a Intel™ Core© i7-4770K CPU @3.50 GHz.

5.5 Application

5.5.1 Parking Manoeuvres

In order to apply our MPC strategy, let us consider, once again, the car–like system problem with $t \in [0, 20]$, in seconds, $\mathbf{x}(t) = (x(t), y(t), \psi(t))$ and $\mathbf{u}(t) = (u(t), c(t))$. Aiming minimum energy, this problem ($\mathrm{P_{CP}}$) [PF14c] can be stated as:

$$\text{Minimize} \int_0^{20} u^2(t)\mathrm{dt} \tag{5.19}$$

subject to

(i) the dynamic constraints

$$\begin{aligned} \dot{x}(t) &= u(t)\cos(\psi(t)) && \text{a.e. } t \in [0,20] \\ \dot{y}(t) &= u(t)\sin(\psi(t)) && \text{a.e. } t \in [0,20] \\ \dot{\psi}(t) &= u(t)\,c(t) && \text{a.e. } t \in [0,20] \end{aligned} \tag{5.20}$$

where $u(t)$ is the speed and $c(t)$ is the curvature,

(ii) the input constraints

$$\begin{aligned} -1 \le u(t) \le 1 \qquad & \forall t \in [0,20] \\ -0.7 \le c(t) \le 0.7 \qquad & \forall t \in [0,20] \end{aligned}$$

(iii) the end–point constraints

$$\mathbf{x}(0) = \mathbf{x}_0 = (x_0, y_0, \psi_0) = (1.5, 3.5, \pi/2) \tag{5.21}$$

$$\mathbf{x}(20) \in \mathbb{X}_1 = \left\{ (x,y,\psi) : \left(x - x_f\right)^2 + \left(y - y_f\right)^2 + \left(\psi - \psi_f\right)^2 \le r^2 \right\} \tag{5.22}$$

where $r^2 = 0.1$ and $\mathbf{x}_f = (x_f, y_f, \psi_f) = (0,4,0)$ is a user–defined target point, and

(iv) the pathwise state constraint

$$\begin{cases} -M \le y(t) \le M & \text{if } x(t) \in [x_0, x^*] \\ -b\,(t, \mathbf{x}(t)) \le y(t) \le b\,(t, \mathbf{x}(t)) & \text{if } x(t) \in [x^*, x^\star] \\ -m \le y(t) \le m & \text{if } x(t) \in [x^\star, x_f] \end{cases} \tag{5.23}$$

where

$$b\,(t, \mathbf{x}(t)) = y^\star - \sqrt{\rho^2 - (x(t) - x^\star)^2}\,, \quad \rho = |x^\star - x^*|\,.$$

In order to apply the MPC algorithm, we start by introducing some perturbations on the system dynamics test–plant:

$$\begin{cases} \dot{x}(t) = u(t)\,(1+\delta_u)\cos(\psi(t)) \\ \dot{y}(t) = u(t)\,(1+\delta_u)\sin(\psi(t)) \\ \dot{\psi}(t) = u(t)\,(1+\delta_u)\,c(t)\,(1+\delta_c) \end{cases} . \tag{5.24}$$

where δ_u and δ_c are perturbations associated to the controls u and c respectively.

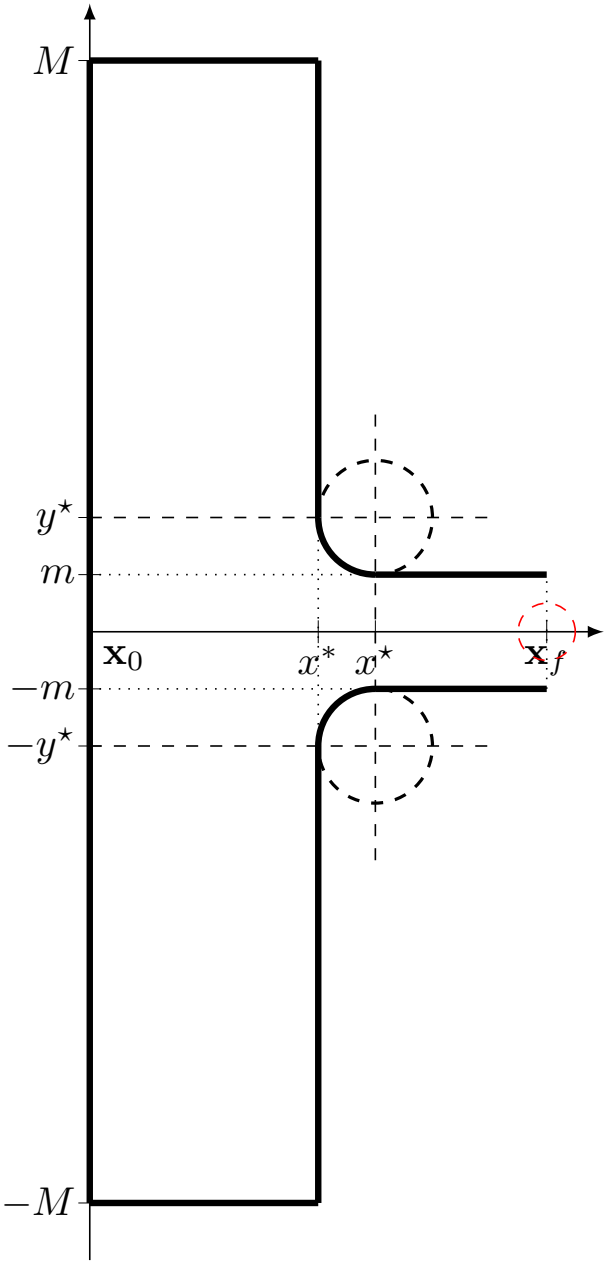


Figure 5.4: Pathwise state constraints (5.27) for (P_{CP})

5.5.2 Numerical Results

To test and to validate the proposed algorithm, a problem involving parking manoeuvres is solved considering pathwise state constraints.

We choose this application because MPC can overcome nonholonomy challenges since it involves planning, not just reactive control; it can generate required nonlinear, discontinuous feedback; and it is known that sampled–data MPC framework combines well with sampling-feedbacks [Fon03]. MPC can, also, overcome constraints challenges since it is known to be a (if not the main) technique to deal appropriately with constraints and MPC simply dealt with them within the optimisation.

Considering the end–point constraints

$$\mathbf{x}(0) = (1.5, 3.5, \pi/2) \tag{5.25}$$

$$\mathbf{x}(10) \in \mathbb{X}_1 = \left\{(x, y, \psi) : x^2 + (y-4)^2 + \psi^2 \leq 0.1\right\} \tag{5.26}$$

the pathwise state constraint

$$\begin{cases} -4 \leq y(t) \leq 4 & \text{if } x(t) \in [0,2] \\ -b\,(t, \mathbf{x}(t)) \leq y(t) \leq b\,(t, \mathbf{x}(t)) & \text{if } x(t) \in [2,3] \\ -0.5 \leq y(t) \leq 0.5 & \text{if } x(t) \in [3,4] \end{cases} \tag{5.27}$$

where

$$\rho = 1\,, \quad b\,(t, \mathbf{x}(t)) = -1.5 - \sqrt{1 - (x(t) - 3)^2}\,.$$

We consider $\delta = 2$ s which means that we will solve a sequence of 10 open–loop OCPs and we define $\delta_u = \delta_c = 0.1$. We also set

$$\varepsilon_{\mathbf{x}}^{\max} = 5 \times 10^{-5}$$

and

$$\bar{\varepsilon}_{\mathbf{x}}(t) = \begin{cases} \varepsilon_{\mathbf{x}}^{\max}, & t \in [t_k, t_k + 0.1T] & \rightarrow & \text{short–term planning} \\ 10 \times \varepsilon_{\mathbf{x}}^{\max}, & t \in\,]t_k + 0.1T, t_k + 0.3T] & \rightarrow & \text{mid–term planning} \\ 10^3 \times \varepsilon_{\mathbf{x}}^{\max}, & t \in\,]t_k + 0.3T, t_k + T] & \rightarrow & \text{long–term planning} \end{cases}$$

This problem is solved considering three meshes:

a) π_{ML} obtained by the multi–level time–mesh refinement strategy with MPC considering $\mathcal{N} = 4$;

b) π_{F} considering equidistant–spacing with the lowest time step of π_{ML} with MPC;

c) π_{C} considering equidistant–spacing with the greatest time step of π_{ML}.

As it can be seen in Fig. 5.5, considering the mesh π_{ML}, the car–like system successfully stops when the terminal condition (4.5) is satisfied without violating any constraint. The sequence of solutions given by each sampling step on MPC is shown in Fig. 5.6. The predictions are plotted with a dashed line , while the implemented controls are plotted with a solid line. Each segment is drawn with a different color representing different MPC sampling times.

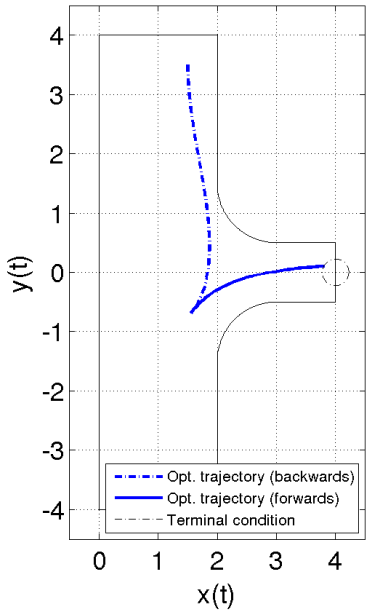


Figure 5.5: Optimal trajectory for ($\mathrm{P_{CP}}$) using MPC

The numerical results concerning the three meshes are shown in Table 5.1, which shows information about the number of nodes, the smallest time step, the number of

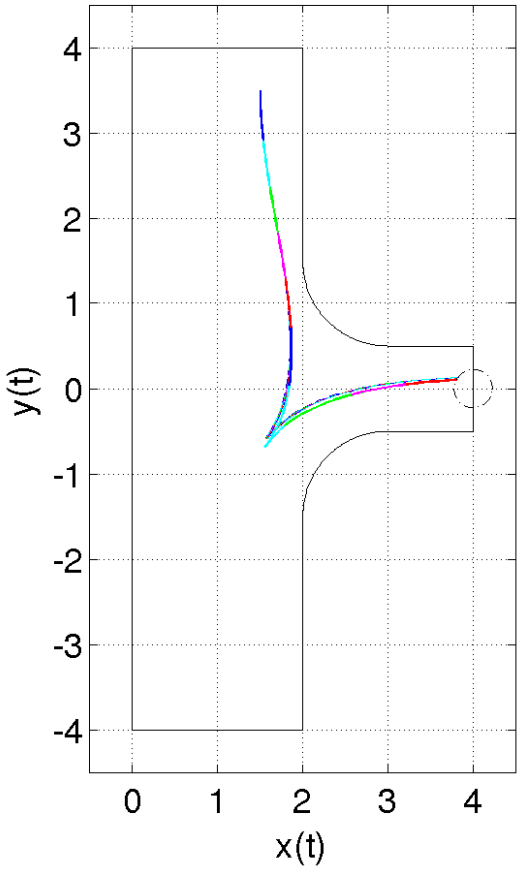


Figure 5.6: Sequence of optimal trajectories for (P_{CP})

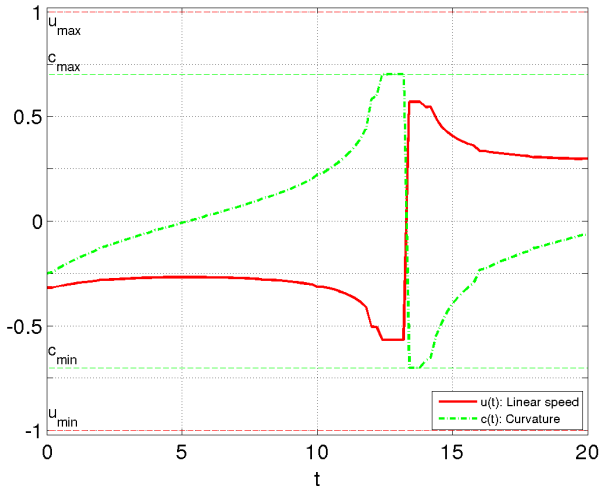


Figure 5.7: Optimal control for (P_{CP})

iterations needed to solve the NLP problem, the maximum absolute local errors of the trajectory, and the CPU times for solving the OCP problem and for computing the local error as well.

Table 5.1: Comparing MPC results for the problem (P_{CP})

π_j	N_j	Δt_j	I_j	$\left\|\left\|\varepsilon_{\mathbf{x}}^{(j)}\right\|\right\|_\infty$	CPU time (s)	
					Solver	$\varepsilon_{\mathbf{x}}$
π_{ML}	365	$1/3200$	304|13|13|13|13|10|16|5|5|5	4.169E^{-5}	11.448	5.231
π_F	3201	$1/3200$	371|34|22|20|18|9|8|7|7|7	3.730E^{-5}	53.493	31.239
π_C	201	$1/200$	233|81|13|11|6|6|6|5|5|5	1.261E^{-3}	8.667	1.960

According to Table 5.1, the mesh π_{ML} has only 11.4% of the nodes of π_F, nevertheless both meshes have maximum absolute local error of the same order of magnitude. Computing the solution using π_{ML} takes less than 20% of the time needed to get a solution using π_F, resulting in significant savings in memory and computational cost.

The mesh π_C is the initial coarse mesh considering equidistant spacing. Without applying our refinement strategy, the MPC produces a solution with lower accuracy, 1.261E^{-3}, when compared against the solution obtained via refinement procedure, 4.169E^{-5}. Moreover, the CPU time spent to compute solution using π_{ML} is, as expected, 50% higher than the one spent to obtain a solution using π_C, however it is a good trade–off since the accuracy of the solution increases by two orders of magnitude.

Fig. 5.8 shows solutions for different initial conditions. In all tests, the procedure gives the optimal solution which is computed spending a few seconds.

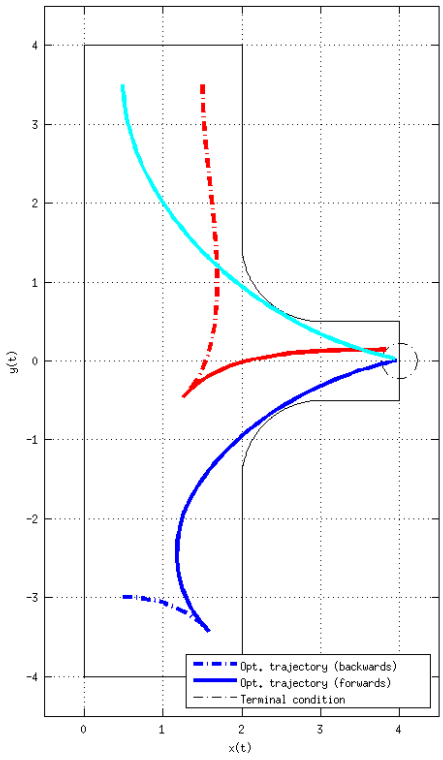


Figure 5.8: Optimal trajectories for (P_{CP}) considering different initial conditions

5.6 Final Remarks

MPC states for a robust technique to solve OCPs, specially if we are leading with disturbances in the model or if we have model–plant mismatch. In these cases, the real behaviour of the system is different from the predicted one and the MPC methodology provides an update of the optimal trajectory based on the measurements obtained at sampling time instants.

We develop an extended adaptive time–mesh refinement algorithm providing local mesh resolution refinement only where it is required. In this extension, we consider a time–dependent stopping criterion for the mesh refinement algorithm with different levels $\bar{\varepsilon}(t)$. In the end, the OCP is solved using MPC with an adapted mesh which has less nodes in the overall procedure, yet having maximum absolute local error of

the same order of magnitude when compared against a refined mesh with equidistant–spacing.

Due to the fast response of the algorithm, it can be used to solve real–time optimisation problems. The application presented in this chapter demonstrate the advantage of the proposed adaptive mesh strategy, which leads to results obtained as fast as the ones given by a coarse equidistant–spacing mesh and as accurate as the ones given by a fine equidistant–spacing mesh.

Chapter 6

Global Optimal Control

"If there is a problem you cannot solve, then there is an easier problem you can solve: find it!"

George Pólya

The accurate solution of optimal control problems is crucial in many areas of engineering and applied science. Problems involving systems which are described by nonlinear differential equations often contain multiple local minima. For these cases, methods which attempt to determine the global solution exist [EF00, Flo99, HP95a, Kro93, QB03, RL92].

In this chapter, Global Optimal Control (GOC) methods are introduced to address the nonlinear OCP towards global optimality.

6.1 Introduction

Optimisation is often based on highly nonlinear descriptive models. Nonlinear Optimisation (NLO) models – based on a highly nonlinear system description – frequently possess multiple optima, thus finding the best possible solution requires a global scope search approach. The objective of Global Optimisation (GO) is to find

the globally best solution of models when multiple local optima exist.

There several methods to search for the global optimum which can be divided in two groups: Exact Methods [HP95b, PR02] and Heuristic Methods [Wei08].

In the Exact Methods context, we can find several methods such as (a) Dynamic Programming; (b) Branch and Bound Algorithms; (c) D.C. Programming; (d) Lipschitz Optimisation. The first one is a method for OCP and the remain ones are methods for NLP.

With respect to Heuristic GO Methods, we have available strategies such as (a) Approximate Convex Underestimation; (b) Continuation Methods; (c) Simple Globalise Local Search Methods Population-Based Strategies – Ant Colony Optimization, Genetic Algorithms and Particle Swarm Optimisation; (d) Sequential Improvement of Local Optima – Simulated Annealing and (e) Tabu Search, among others. Exact Methods have provable theoretical global convergence properties, as opposed to Heuristic GO Methods which have not. However, some Heuristic GO Methods can be modified to gain global convergence features.

6.2 Global Exact Methods Overview

Among the exact methods, we emphasize B&B algorithms, the D.C. Programming method, the Lipschitz Optimisation approach, and Dynamic Programming (DP).

6.2.1 Global Methods for Nonlinear Programming Problems

Let us consider the following problem

$$\begin{aligned} \text{Minimise} \quad & \mathbf{f}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathbb{X} \\ & \mathbf{g}(\mathbf{x}) \leq 0 \end{aligned} \tag{6.1}$$

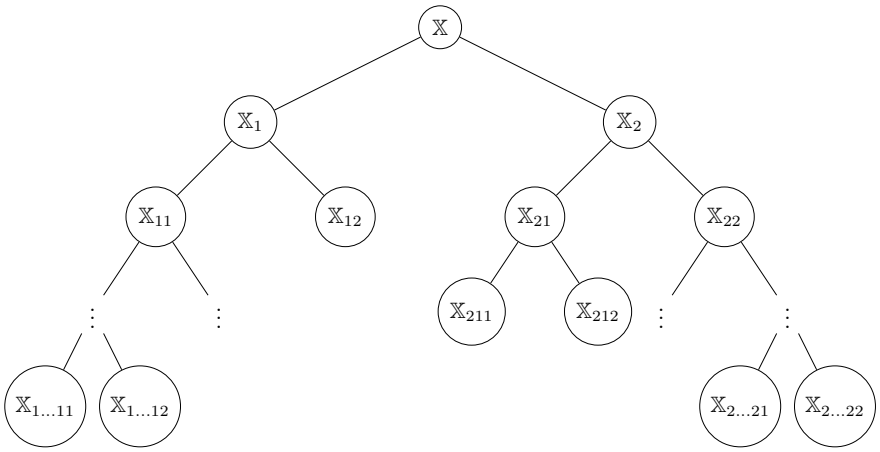


Figure 6.1: Illustration of the B&B method

$$\mathbf{h}(\mathbf{x}) = 0$$

where $\mathbb{X} \subset \mathbb{R}^n$ is a closed convex set, $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$ and $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^n$.

Branch and Bound Algorithm

The Branch and Bound (B&B) technique is a widely used procedure to solve several types of difficult optimisation problems.

In the B&B methods, the feasible set is relaxed and subsequently partitioned into refined parts – branching – over which lower and upper bounds of the minimum objective function value can be determined – bounding [HP95a]. Parts of the feasible set with lower bounds exceeding the best upper bound found at a certain stage of the algorithm are deleted from further consideration – pruning – since this parts of the domain do not contain the optimum.

B&B methods are often visualised by a search tree where the root node represents the initial relaxation of the feasible set and the remain ones correspond to successively generated partition sets. If a certain partition set is obtained by a direct partition of the previous one, the two corresponding nodes are connected by an arc, as it can be seen in Fig. 6.1.

D.C. Programming

In the nonconvex optimisation context, D.C. Programming plays an important role because of its theoretical aspects as well as its wide range of applications [HT99]. A function is called D.C. if it can be represented as the difference of two convex functions. NLP problems involving D.C. functions are called D.C. programming problems.
Let us consider the problem (6.1) and that $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are D.C. functions. An interesting feature of D.C. Programming is that any problem of the form (6.1) can be reduced to a canonical problem of minimising a linear function over the intersection of a convex set with the complement of an open set.

Lipschitz Optimisation

The Lipschitz Optimisation approach to global Optimisation has always been attractive [Pin96]. Let us consider, once again, the problem (6.1) and that $\mathbf{f}(\cdot)$ is a Lipschitz function. Knowing the Lipschitz constant, *i.e.*, a bound on the rate of change of the objective function, global search algorithms can be developed and convergence theorems easily proved. Since Lipschitz Optimisation methods are deterministic, there is no need for multiple runs. These methods also have few parameters to be specified, besides the Lipschitz constant, thus the need for parameter finite–tuning is minimised. This type of methods can place bounds on how far they are from the optimum function value, and hence can use stopping criteria that are more meaningful than a simple iteration limit.

6.2.2 Global Methods for Optimal Control Problems

Let us consider the following optimal control problem, in Bolza form, with input and state constraints:

$$\text{Minimise} \quad \mathrm{J}(\mathbf{x},\mathbf{u}) = \int_{t_0}^{t_f} \mathrm{L}\left(t,\mathbf{x}(t),\mathbf{u}(t)\right)\mathrm{d}t + \mathrm{G}\left(\mathbf{x}(t_f)\right)$$

$$
\begin{aligned}
\text{subject to } \ & \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \quad \text{a.e. } t \in \left[t_0, t_f\right], \\
& \mathbf{u}(t) \in \mathbb{U} \subset \mathbb{R}^m \quad \text{a.e. } t \in \left[t_0, t_f\right], \\
& \mathbf{h}\left(\mathbf{x}(t)\right) \leq 0 \quad \forall t \in \left[t_0, t_f\right], \\
& \mathbf{x}(t_0) \in \mathbb{X}_0 \subset \mathbb{R}^n \quad \text{and} \quad \mathbf{x}(t_f) \in \mathbb{X}_1 \subset \mathbb{R}^n,
\end{aligned}
$$

where

$$
\begin{aligned}
&\mathbf{x} : \left[t_0, t_f\right] \to \mathbb{R}^n, \\
&\mathbf{u} : \left[t_0, t_f\right] \to \mathbb{R}^m, \\
&\mathrm{L} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}, \\
&\mathrm{G} : \mathbb{R}^n \to \mathbb{R}, \\
&\mathbf{f} : \left[t_0, t_f\right] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n, \\
&\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^k.
\end{aligned}
$$

Dynamic Programming and Hamilton–Jacobi Methods

As seen before, Dynamic Programming (DP) is a stage wise search method of optimisation problems whose solutions may be viewed as the result of a sequence of decisions. The selection of the optimal decision in based on the Bellman's Principle of Optimality. According to Fig. 6.2, illustrating a top–down view of DP, we can use a recursive procedure to solve an OCP.

For some $t \in \left[t_0, t_f\right[$, in a given finite horizon $t_f > 0$, let us consider the following initial value problem

$$
\begin{aligned}
&\dot{\mathbf{x}}(s) = \mathbf{f}(s, \mathbf{x}(s), \mathbf{u}(s)) \quad \text{a.e. } s \in \left(t, t_f\right) \\
&\mathbf{x}(t) = \mathbf{x}_t \\
&\mathbf{u}(s) \in \mathbb{U}.
\end{aligned}
\tag{6.2}
$$

Since the control problem is in presence of state constraints, a state–space constrained HJB equation has been associated to the value function of (6.2) which takes the form [ABZ13]:

$$\begin{cases} V_t(t,\mathbf{x}) - \mathrm{H}(t,\mathbf{x}(t), -V_{\mathbf{x}}(t,\mathbf{x}), \mathbf{u}(t)) = 0\,, \quad \mathbf{x} \in \mathbb{X}, \quad t \in \left[t_0, t_f\right] \\ V(t_f, \mathbf{x}(t_f)) = \mathrm{G}(\mathbf{x}(t_f)) \end{cases}$$

where

$$\mathrm{H}(t,\mathbf{x},\mathbf{p}) = \max_{\mathbf{u}\in\mathbb{U}} (\mathbf{p}\cdot\mathbf{f}(t,\mathbf{x},\mathbf{u}) - \mathrm{L}(t,\mathbf{x},\mathbf{u}))\,.$$

Let the set of all feasible trajectories starting in $\mathbf{x}$ at time t be denoted as:

$$S_{\left[t,t_f\right]}(\mathbf{x}) = \left\{\mathbf{x} \in W^{1,1} : \mathbf{x} \text{ satisfies (6.2)}\right\}. \tag{6.3}$$

Let us also consider a non–empty closed set $K \in \mathbb{R}^n$ which is the set $K = \{\mathbf{x} : \mathbf{h}(\mathbf{x}) \leq 0\}$. Therefore, a trajectory $\mathbf{y} \in S_{\left[t,t_f\right]}(\mathbf{x})$ is admissible, on the time interval $\left(t, t_f\right)$, if $\mathbf{y}(s) \in K$, for all $s \in \left(t, t_f\right)$.

The problem of *backward reachable sets* from a closed target $\mathbb{X}_1 \in \mathbb{R}^n$ consists in characterizing, for every $t \in \left[t_0, t_f\right]$, the set of all initial positions from which it is possible to find an admissible trajectory that reaches the target $\mathbb{X}_1$ at time t_f while lying in the set K on $\left[t, t_f\right]$:

$$R(t) = \left\{\mathbf{x}_t \in \mathbb{X} : \exists \mathbf{x} \in S_{\left[t,t_f\right]}(\mathbf{x}_t) \text{ such that } \mathbf{x}(t_f) \in \mathbb{X}_1, \text{ and } \mathbf{x}(s) \in K \text{ for } s \in \left[t, t_f\right]\right\}. \tag{6.4}$$

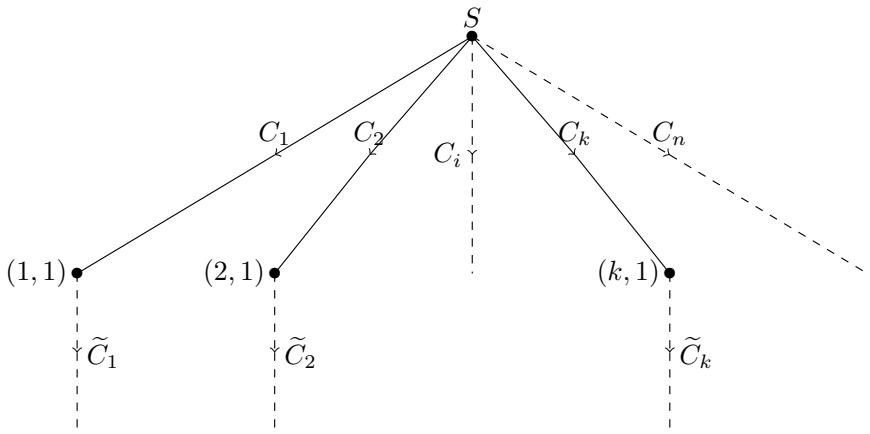


Figure 6.2: Illustration of the Dynamic Programming procedure

It is known that the backward reachable set can be seen as a level–set for the value function [MB05]. In our problem

$$R(t) = \left\{ \mathbf{x} : V(\mathbf{x}) \leq t_f - t \right\} \tag{6.5}$$

6.3 Application

6.3.1 Problem Statement

Let us consider the time t, in seconds, $\mathbf{x}(t) = (x(t), y(t), \psi(t))$ and $\mathbf{u}(t) = (u(t), c(t))$. Aiming minimum time, the car–like system problem ($\mathrm{P_{GO}}$) can be stated as:

$$\text{Minimise } t_f \tag{6.6}$$

subject to

(i) dynamic constraints

$$\begin{aligned} \dot{x}(t) &= u(t)\cos(\psi(t)) && \text{a.e. } t \in \left[t_0, t_f\right] \\ \dot{y}(t) &= u(t)\sin(\psi(t)) && \text{a.e. } t \in \left[t_0, t_f\right] \\ \dot{\psi}(t) &= u(t)\, c(t) && \text{a.e. } t \in \left[t_0, t_f\right], \end{aligned} \tag{6.7}$$

where $u(t)$ is the speed and $c(t)$ is the curvature,

(ii) input constraints

$$\begin{aligned} -1 \leq u(t) \leq 1 && \text{a.e. } t \in \left[t_0, t_f\right] \\ -0.7 \leq c(t) \leq 0.7 && \text{a.e. } t \in \left[t_0, t_f\right], \end{aligned}$$

(iii) end–point constraints

$$\mathbf{x}(t_0) = \mathbf{x}_0 = (x_0, y_0, \psi_0) \tag{6.8}$$

$$\mathbf{x}(t_f) \in \mathbb{X}_1 = \left\{ (x, y, \psi) : \left(x - x_f\right)^2 + \left(y - y_f\right)^2 + \left(\psi - \psi_f\right)^2 \leq r^2 \right\}, \tag{6.9}$$

where $r \in \mathbb{R}$ and $\mathbf{x}_f = (x_f, y_f, \psi_f)$ is a user–defined target point, and

(iv) the state constraint

$$y \leq \bar{y} - b \quad \vee \quad y \geq \bar{y} + b, \quad \text{if } x \in [\bar{x} - a, \bar{x} + a], \quad \forall t \in \left[t_0, t_f\right], \tag{6.10}$$

where $(a, b) \in \mathbb{R}^2$ is half the width (horizontal) and half the length (vertical), respectively, of a rectangle centred in $(\bar{x}, \bar{y}) \in \mathbb{R}^2$.

The goal is to drive this car–like system from $\mathbf{x}_0$ to some point near $\mathbf{x}_f$ according to the terminal condition (6.9) while avoiding the obstacle (6.10).

6.3.2 Numerical Results

Let us consider the end–point constraints

$$\mathbf{x}(0) = (0, 0, 0)$$
$$\mathbf{x}(t_f) \in \mathbb{X}_1 = \left\{(x, y, \psi) : (x - 10)^2 + y^2 + \psi^2 \leq 0.5\right\},$$

and the state constraint

$$y \leq -1.999 \quad \vee \quad y \geq 2.001, \quad \text{if } x \in [4.9, 5.1]$$

We define $(a, b) = (0.1, 2)$ and we choose $0 < \bar{y} \ll 1$ because we want a vertical rectangle, almost symmetrical with respect to the horizontal axis, with a very small perturbation in its center $(\bar{x}, \bar{y}) = (5, 0.001)$. This perturbation cause the rectangle to be slightly unsymmetrical with respect to the horizontal axis and by doing this we know that there are two sub–optimal solutions and there is only only global minimum.

The car–like system problem ($\mathrm{P_{GO}}$) was written in C++ and it was solved using ROC-HJ in a computer with a Intel™ Core© i7-4770K CPU @3.50 GHz.

The discretisation with respect to the time variable is performed by the 2^{nd} order Runge-Kutta method. The discretisation in space is based on upwind finite difference method. We consider $(x, y) \in [0, 10] \times [-5, 5]$ and $\psi \in [0, 2\pi]$. Then, we generate a equidistant–spacing space mesh $200 \times 200 \times 50$. With this information, the ROC-HJ

software generate a time–mesh with 692 nodes and it takes 13.5762s of CPU time to find the backward reachable set. After the solution of the HJ equation is computed, and considering $\mathbf{x}(0) = (0, 0, 0)$, the ROC-HJ software finds the optimal trajectory in 1.1265s.

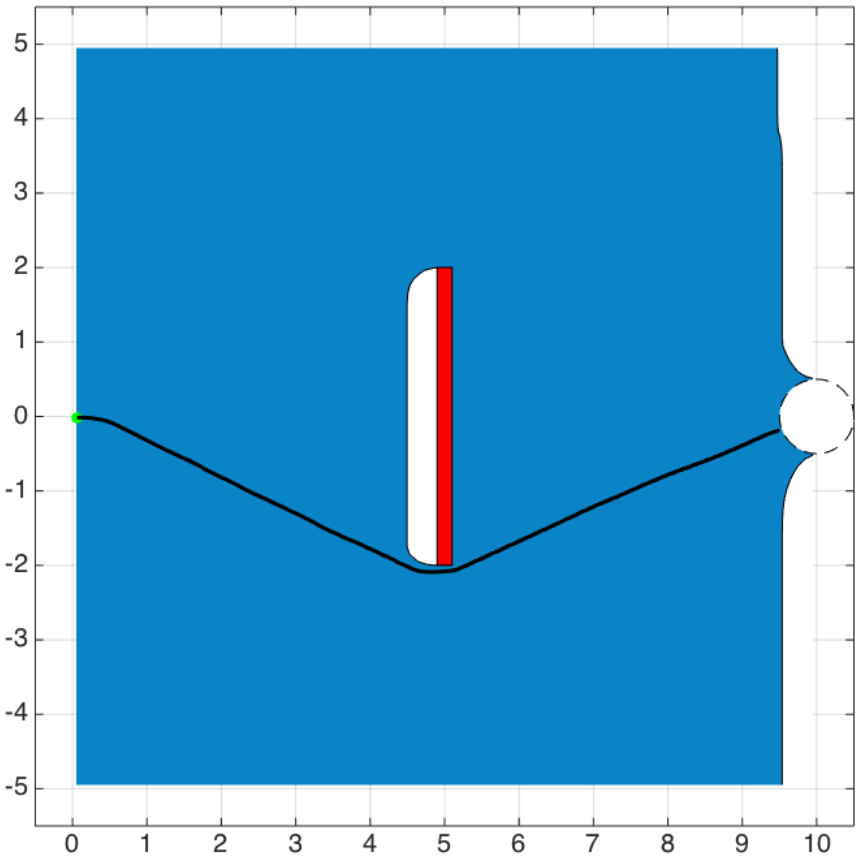


Figure 6.3: Optimal trajectory and reachable set using ROC-HJ for (P_{GO})

As it can be seen in Fig. 6.3, the car–like system successfully avoids the obstacle and it stops when the terminal condition (6.9) is satisfied. In Fig. 6.3, the blue area corresponds to the backward reachable set and the red rectangle coincides with the obstacle. The minimum time needed to reach the target area is 10.9284268708s.

6.4 Final Remarks

Nonlinear models exist in many applications, *e.g.*, in advanced engineering design, biotechnology, data analysis, environmental management, financial planning, process control, risk management, scientific modelling, and others. Their solution often requires a global search approach.

There are GO methods and GOC methods which can be divided in two main categories: Exact methods and Heuristic Methods. Among the described methods, we use DP and HJ methods. We solve an application involving a car–like system which has to avoid an obstacle. ROC-HJ The application presented in this chapter demonstrates the advantage of DP and of GOC methods where the global optimum is achieved.

Global Optimal Control (GOC) is a subject of growing practical interest as indicated by recent software implementations and by an increasing range of applications. In spite of remarkable progress, GOC remains a field of extreme numerical challenges, in particular, in practical attempts to handle complex and sizeable problems within an acceptable time frame.

Chapter 7

Conclusion

"Mathematics is the science which draws necessary conclusions."

Benjamin Peirce

7.1 Contributions

Summarising all the conclusions made along the chapters, we conclude that there are a lot of interfaces we can chose from, involving open–source, freeware and commercial software, working under different operating systems. The choice of an interface should be made taking into account the number of solvers it can connect and the level of programming expertise of the user. Among open–source, freeware and commercial software, there are also a big list of OC and NLP solvers useful for solving optimal control problems. The minimum time problem (P_{DD}) involving a differential drive robot system was successfully solved using the IPOPT, KNITRO and WORHP solvers. The three solvers provided similar results with errors of the same order of magnitude.

With respect to adaptive mesh refinement strategy, we develop a new algorithm

providing local mesh resolution only where it is required. In the end, the OCPs are solved using an adapted mesh which has less nodes in the overall procedure, yet with higher concentration of nodes in time subintervals where the trajectory shows nonlinear behaviour. Therefore, this procedure is characterised by having significant savings in memory and computational cost. When using this strategy, where the mesh is progressively refined to catch special features of the problem, there is no need to define *a priori* the most appropriately mesh, which is another advantage of this procedure. In addition, the algorithm using the proposed refinement strategy showed more robustness, since it was able to obtain a solution when the traditional approach by starting with a very fine mesh failed to do it. The applications presented, ($\mathrm{P_{CL}}$) and ($\mathrm{P_S}$), demonstrate the advantage of the proposed adaptive mesh strategy, which leads to results with greater accuracy and with lower overall computational time when compared to other common used approaches.

We develop an extended adaptive time–mesh refinement algorithm providing local mesh resolution refinement only where it is required. In this extension, we consider a time–dependent stopping criterion for the mesh refinement algorithm with different levels $\bar{\varepsilon}^{\max}(t)$. In the end, the OCP is solved using MPC with an adapted mesh which has less nodes in the overall procedure, yet having maximum absolute local error of the same order of magnitude when compared against a refined mesh with equidistant–spacing. Due to the fast response of the algorithm, it is extended be use to solve real time optimization problems, in particular, in MPC.

Global Optimisation (GO) is a subject of growing practical interest as indicated by recent software implementations and by an increasing range of applications. In spite of remarkable progress, GO remains a field of extreme numerical challenges, in particular, in practical attempts to handle complex and sizeable problems within an acceptable time frame. There are several GO methods which can be divided in two main categories: Exact methods and Heuristic Methods. Among the described methods, we use DP and HJ methods. We implement a Global Optimal Control (GOC) problem which attempt to determine the global solution of a car–like system

which has to avoid an obstacle. This problem is successfully solved and the global optimal trajectory is found.

7.2 Future Work

We intend to develop future work in the following main directions:

Adaptive Time–Mesh Refinement: The proposed adaptive time–mesh refinement algorithm adds nodes to the initial coarse mesh according to some refinement criteria. One of our goals is to improve this algorithm by allowing it to remove/disregard nodes as well.

Model Predictive Control: The sampling step of the MPC procedure is often considered to be fixed, but we indent to develop MPC strategies where we can vary the sampling step along the MPC prediction;

Impulsive Dynamical Systems: We will apply our time–mesh refinement algorithm to compute the solution of Impulsive System (IS). IS [AKP10, FP12a, FP12b, Fra09] are systems in which the state trajectories can have discontinuities (jumps, sudden changes) in response to (impulsive) controls, which are often an instantaneous action with high impact on the results and the timing of this action is often crucial. Since these discontinuity instants might not be known in advance and they might depend on a chosen control action, we believe that our adaptive time–mesh algorithm is a good tool to compute with high accuracy the timing of the impulsive control.

GO: Since it is a subject of growing practical interest and the range of applications is increasing, we intend to explore other tools for solving global optimal control problems. We also will develop adaptive mesh strategies and other numerical algorithms in this field.

Appendix A

Background

Let us consider a point $\mathbf{x} \in \mathbb{R}^n$, *i.e.*, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. We define an ε–neighbourhood around $\mathbf{x}$ as the set

$$N_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : ||\mathbf{y} - \mathbf{x}|| < \varepsilon\} \ , \quad \text{for } \varepsilon > 0 \, , \tag{A.1}$$

where $||\cdot||$ denotes the Euclidean norm of a vector in $\mathbb{R}^n$.

Let S be an arbitrary set in $\mathbb{R}^n$. A point $\mathbf{x}$ is said to be in the *closure* of S, denoted by $\mathrm{cl}\, S$, if

$$S \cap N_\varepsilon(\mathbf{x}) \neq \emptyset \, , \quad \forall \varepsilon > 0 \, . \tag{A.2}$$

A point $\mathbf{x} \in S$ is in the interior of S, denoted by $\mathrm{int}\, S$, if $N_\varepsilon(\mathbf{x}) \subset S$ for some $\varepsilon > 0$. If $S = \mathrm{cl}\, S$, then S is called *closed*. If $S = \mathrm{int}\, S$, then S is called *open*.

A point $\mathbf{x}$ is in the boundary of S, denoted by ∂S, if $N_\varepsilon(\mathbf{x})$ contains at least one point in S and one point not in S for every $\varepsilon > 0$. Hence, a set S is closed if and only if it contains all its boundary points. Moreover, $\mathrm{cl}\, S \equiv S \cup \partial S$ is the smallest closed set containing S. Similarly, a set S is open if and only if it does not contain any of its boundary points. The only sets in $\mathbb{R}^n$ that are both open and closed are the empty set and $\mathbb{R}^n$ itself.

Definition A.0.1 (Convex Set)**.** *A set $S \subset \mathbb{R}^n$ is* convex *if the line segment joining*

any two points of the set also belongs to the set. In other words, if

$$\mathbf{x},\, \mathbf{y} \in S \;\Rightarrow\; \alpha\,\mathbf{x} + (1-\alpha)\mathbf{y} \in S\,, \quad \forall \alpha \in [0,1]\,. \tag{A.3}$$

Definition A.0.2 (Convex Cone)**.** *A nonempty set $C \in \mathbb{R}^n$ is called a* cone with vertex zero *if*

$$\mathbf{x} \in C \;\Rightarrow\; \alpha\,\mathbf{x} \in C\,, \quad \forall \alpha \geq 0\,. \tag{A.4}$$

In addition, if C is convex, then C is called a convex cone*.*

Definition A.0.3 (Convex Function)**.** *Let us consider a nonempty convex set $S \in \mathbb{R}^n$ and a function $f : S \to \mathbb{R}$.*

(i) f is convex on S if

$$f\left(\alpha\,\mathbf{x} + (1-\alpha)\mathbf{y}\right) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) \tag{A.5}$$

for each $\mathbf{x}, \mathbf{y} \in S$ and for each $\alpha \in]0,1[$;

(ii) f is strictly convex *on S if*

$$f\left(\alpha\,\mathbf{x} + (1-\alpha)\mathbf{y}\right) < \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) \tag{A.6}$$

for each distinct $\mathbf{x}, \mathbf{y} \in S$ and for each $\alpha \in]0,1[$.

Furthermore, the function f is (strictly) concave *on S if $-f$ is (strictly) convex on S.*

Convex functions have the following useful properties:

(i) Let $f_1, f_2, \ldots, f_k : \mathbb{R}^n \to \mathbb{R}$ be convex functions. Then

- $f(x) = \sum_{i=1}^{k} \alpha_i f_i(x)$, where $\alpha_j > 0$ for $i = 1, \ldots, k$, is a convex function, and
- $f(x) = \max\{f_1(x), f_2(x), \ldots, f_k(x)\}$ is a convex function.

(ii) Suppose that $g : \mathbb{R}^n \to \mathbb{R}$ is a concave function. Let us consider $S = \{x : g(x) > 0\}$, and let us define $f : S \to \mathbb{R}$ as $f(x) = 1/g(x)$. Then f is convex over S.

(iii) Let $g : \mathbb{R} \to \mathbb{R}$ be a non–decreasing, univariate, convex function, and let $h : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, the function $f : \mathbb{R}^n \to \mathbb{R}$ defined as $f(x) = g(h(x))$ is a convex function.

(iv) Let $g : \mathbb{R}^m \to \mathbb{R}$ be a convex function, and let $h : \mathbb{R}^n \to \mathbb{R}^m$ be an affine function of the form $h(x) = Ax + b$, where A is an $m \times n$ matrix, and b is an $m \times 1$ vector. Then, the function $f : \mathbb{R}^n \to \mathbb{R}$ defined as $f(x) = g(h(x))$ is a convex function.

Definition A.0.4 (Quasiconvex Function)**.** *Let $f : S \to \mathbb{R}$, where S is a nonempty convex set in $\mathbb{R}^n$. The function f is* quasiconvex *if, for each $x_1, x_2 \in S$, the following inequality holds:*

$$f(\alpha\, x_1 + (1-\alpha)x_2) \leq \max\{f(x_1), f(x_2)\} \quad \forall \alpha \in]0,1[. \tag{A.7}$$

The function f is quasiconcave *if $-f$ is quasiconvex.*

Let S be a set in $\mathbb{R}^n$ with a nonempty interior and let $f : S \to \mathbb{R}$:

(i) $\mathbf{f}$ is said to be *differentiable* at $\bar{\mathbf{x}} \in \operatorname{int} S$ if there exists a vector $\nabla \mathbf{f}(\bar{\mathbf{x}})^{\mathrm{T}} \in \mathbb{R}^n$, the gradient vector, and a function $\alpha : \mathbb{R}^n \to \mathbb{R}$, such that

$$f(\mathbf{x}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}) + ||\mathbf{x} - \bar{\mathbf{x}}||\alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}}) \quad \text{for all } \mathbf{x} \in S, \tag{A.8}$$

where $\lim_{x \to \bar{\mathbf{x}}} \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}}) = 0$. The function f is said to be differentiable on the open set $\hat{S} \subseteq S$ if f is differentiable at each point in $\hat{S}$.

(ii) $\mathbf{f}$ is said to be *twice–differentiable* at $\bar{\mathbf{x}} \in \operatorname{int} S$ if there exists a vector $\nabla \mathbf{f}(\bar{\mathbf{x}})^{\mathrm{T}} \in \mathbb{R}^n$, an $n \times n$ symmetric matrix $H(\bar{\mathbf{x}})$, the Hessian matrix, and a function $\alpha : \mathbb{R}^n \to \mathbb{R}$, such that

$$f(\mathbf{x}) = f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}) + {}^1\!/_2(\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}} H(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}) + \\ + ||x - \bar{\mathbf{x}}||^2 \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}}) \quad \text{for all } \mathbf{x} \in S, \tag{A.9}$$

where $\lim_{x \to \bar{\mathbf{x}}} \alpha(\bar{\mathbf{x}}, \mathbf{x} - \bar{\mathbf{x}}) = 0$. The function f is said to be twice–differentiable on the open set $\hat{S} \subseteq S$ if f is twice–differentiable at each point in $\hat{S}$.

Definition A.0.5 (Pseudoconvex Function)**.** *Let S be a nonempty open set in $\mathbb{R}^n$ and let $\mathbf{f} : S \to \mathbb{R}$ be differentiable on S. The function $\mathbf{f}$ is* pseudoconvex *if*

$$\forall \mathbf{x}, \mathbf{y} \in S : \nabla \mathbf{f}(\mathbf{x})(\mathbf{y} - \mathbf{x}) \geq 0 \Rightarrow \mathbf{f}(\mathbf{y}) \geq \mathbf{f}(\mathbf{x}) \tag{A.10}$$

or, equivalently, if

$$\mathbf{f}(\mathbf{y}) < \mathbf{f}(\mathbf{x}) \Rightarrow \mathbf{f}(\mathbf{x})(\mathbf{y} - \mathbf{x}) \geq 0\,. \tag{A.11}$$

The function $\mathbf{f}$ is pseudoconcave *if $-\mathbf{f}$ is pseudoconvex.*

The function $\mathbf{f}$ is strictly pseudoconvex *if*

$$\textit{for each distint } \mathbf{x}, \mathbf{y} \in S : \nabla \mathbf{f}(\mathbf{x})(\mathbf{y} - \mathbf{x}) \geq 0 \Rightarrow \mathbf{f}(\mathbf{y}) > \mathbf{f}(\mathbf{x}) \tag{A.12}$$

or, equivalently, if

$$\textit{for each distint } \mathbf{x}, \mathbf{y} \in S : \mathbf{f}(\mathbf{y}) \leq \mathbf{f}(\mathbf{x}) \Rightarrow \mathbf{f}(\mathbf{x})(\mathbf{y} - \mathbf{x}) \geq 0\,. \tag{A.13}$$

Definition A.0.6 (Absolutely Continuous Function)**.** *Let S be a set in $\mathbb{R}^n$ with a nonempty interior and let $f : S \to \mathbb{R}$. The function f is absolutely continuous on S if*

$$\forall\, \varepsilon > 0 \quad \exists\, \delta > 0 \;:\; \sum_j (b_j - a_j) < \delta \quad \Rightarrow \quad \sum_j |f(b_j) - f(a_j)| < \varepsilon \tag{A.14}$$

for any finite or countably infinite collection of nonoverlapping subintervals $\{[a_j, b_j]\}_j \in S$.

A useful concept is the *Radon Measure* definition. Before defining it let us review other concepts.

Let $S \subset \mathbb{R}^n$ be a nonempty set. We say that a collection ξ of subsets of S is a σ–algebra on S if

$$\emptyset \in \xi\,, \quad S \setminus A \in \xi \quad \text{whenever} \quad A \in \xi\,,$$
$$\bigcup_{k \in \mathbb{N}} A_k \in \xi \quad \text{whenever} \quad A_k \in \xi \quad \text{for every} \quad k \in \mathbb{N}\,.$$

We denote by $\mathcal{B}(S)$ the intersection of all σ–algebras on S containing the open subsets of S. It turns out that $\mathcal{B}(S)$ is the smallest σ–algebra on S containing the

open subsets of S, and it is called the *σ–algebra of Borel* subsets of S and its elements are called *Borel sets*.

Let $(S, \mathcal{B}(S))$ be a Borel measure space.

Definition A.0.7 (Borel measure)**.** *A function $\mu : \mathcal{B}(S) \to \mathbb{R}$ is a* Borel measure *on S if $\mu(\emptyset) = 0$ and μ is contably additive in the sense that*

$$A = \bigcup_{k\in\mathbb{N}} A_k\,, \quad A_k \cap A_j = \emptyset\,, \quad k \neq j \quad \Rightarrow \quad \mu(A) = \sum_{k\in\mathbb{N}} \mu(A_k)\,. \tag{A.15}$$

The set of such measures will be denoted by $\mathcal{M}(S)$. We also say that a Borel measure in positive if it takes its values in $[0, \infty)$. The set of positive Borel measures is denoted by $\mathcal{M}_+(S)$.

Definition A.0.8 (Radon measure)**.** *A positive Borel measure on S that is finite on each compact subset of S is said to be a* Radon measure *on S.*

Bibliography

[ABQ+99] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright. Nonlinear predictive control and moving horizon estimation — an introductory overview. In Paul M. Frank, editor, *Advances in Control*, pages 391–449. Springer London, January 1999.

[ABZ13] Albert Altarovici, Olivier Bokanowski, and Hasnaa Zidani. A general hamilton-jacobi framework for non-linear state-constrained control problems. *ESAIM: Control, Optimisation and Calculus of Variations*, 19(2):337–357, April 2013.

[AHF11] D. Ariens, B. Houska, and H.J. Ferreau. Acado for matlab user's manual. http://www.acadotoolkit.org, 2011.

[AKP10] Aram Arutyunov, Dmitry Karamzin, and Fernando Pereira. On a generalization of the impulsive control concept: Controlling system jumps. *Discrete and Continuous Dynamical Systems*, 29(2):403–415, October 2010.

[BBCH00] John T. Betts, Neil Biehn, Stephen L. Campbell, and William P. Huffman. Compensating for order variation in mesh refinement for direct transcription methods. *Journal of Computational and Applied Mathematics*, 125(1–2):147–158, December 2000.

[BDZ13a] Olivier Bokanowski, Anna Désilles, and Hasnaa Zidani. *ROC–HJ: Reachability analysis and Optimal Control problems - Hamilton-Jacobi equations*, May 2013.

[BDZ13b] Olivier Bokanowski, Anna Désilles, and Hasnaa Zidani. *User's guide for the ROC-HJ solver: Finite Differences and Semi-Lagrangian methods*, February 2013.

[Bec11] Victor M. Becerra. *PSOPT Optimal Control Solver: User Manual*, 2011.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

[Bet01] John T. Betts. *Practical methods for optimal control using nonlinear programming*. SIAM, 2001.

[BGG+14] Frédéric Bonnans, Daphné Giorgi, Vincent Grélard, Stéphan Maindrault, and Pierre Martinon. *BOCOP User Guide*, 2014.

[BH75] Arthur Earl Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Taylor & Francis, 1975.

[BH97] John T. Betts and William P. Huffman. Sparse optimal control software socs. Technical report, Mathematics and Engineering Analysis, Boeing Information and Support Services, The Boeing Company, 1997.

[BH98] John T. Betts and William P. Huffman. Mesh refinement in direct transcription methods for optimal control. *Optimal Control Applications and Methods*, 19(1):1–21, 1998.

[Bie10] Lorenz T. Biegler. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. Society for Industrial and Applied Mathematics, September 2010.

[Bis13] M.H.A. Biswas. *Necessary Conditions for Optimal Control Problems with State Constraints: Theory and Applications*. PhD thesis, University of Porto, November 2013.

[BPd14] Biswas, M.H.A., Paiva, Luís Tiago, and de Pinho, MdR. A SEIR model for control of infectious diseases with constraints. *Mathematical Biosciences and Engineering*, 11:761–784, August 2014.

[BSS06] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, May 2006.

[CB04] E. F. Camacho and Carlos Bordons. *Model Predictive Control*. Springer, July 2004.

[CdB80] Samuel Daniel Conte and Carl de Boor. *Elementary Numerical Analysis: An Algorithmic Approach*. Mcgraw-Hill College, third edition, March 1980.

[Cla90] Frank H. Clarke. *Optimization and Nonsmooth Analysis*. SIAM, 1990.

[Cla98] Frank H. Clarke. *Nonsmooth Analysis and Control Theory*. Springer Science & Business Media, 1998.

[Cla13] Francis Clarke. *Functional Analysis, Calculus of Variations and Optimal Control*. 2013.

[Dre65] Stuart E. Dreyfus. *Dynamic programming and the calculus of variations*. Academic Press, 1965.

[EF00] William R. Esposito and Christodoulos A. Floudas. Deterministic global optimization in nonlinear optimal control problems. *Journal of Global Optimization*, 17(1-4):97–126, September 2000.

[FA03] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. In *Control, 21st Benelux Meeting on Systems and Control, Veidhoven*, pages 1–23, 2003.

[FGK90] Robert Fourer, David M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, May 1990.

[FGK02] Robert Fourer, David M. Gay, and Brian W. Kernighan. *The AMPL Book*. Second edition, November 2002.

[FH10] Wilhelm Forst and Dieter Hoffmann. *Optimization: theory and practice*. Springer, July 2010.

[FKvW10] Paola Falugi, Eric Kerrigan, and Eugene van Wyk. *Imperial College London Optimal Control Software: User Guide*. Imperial College London London England, June 2010.

[Flo99] Christodoulos A. Floudas. *Deterministic Global Optimization: Theory, Methods and Applications*. Springer, December 1999.

[FM03] Fernando A.C.C. Fontes and L. Magni. Min-max model predictive control of nonlinear systems using discontinuous feedbacks. *IEEE Transactions on Automatic Control*, 48(10):1750–1755, October 2003.

[Fon99] Fernando A.C.C. Fontes. *Optimisation–Based Control of Constrained Nonlinear Systems*. PhD thesis, Centre for Process Systems Engineering and Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, London, August 1999.

[Fon01] Fernando A.C.C. Fontes. A general framework to design stabilizing nonlinear model predictive controllers. *Systems and Control Letters*, 42(2):127–143, 2001.

[Fon03] Fernando A.C.C. Fontes. Discontinuous feedbacks, discontinuous optimal controls, and continuous-time model predictive control. *International Journal of Robust and Nonlinear Control*, 13(3-4):191–209, March 2003.

[FP12a] Fernando A.C.C. Fontes and Fernando Lobo Pereira. Model predictive control of impulsive dynamical systems. In *Nonlinear Model Predictive Control*, volume 4, pages 305–310, August 2012.

[FP12b] S.L. Fraga and F.L. Pereira. Hamilton-jacobi-bellman equation and feedback synthesis for impulsive control. *IEEE Transactions on Automatic Control*, 57(1):244 –249, January 2012.

[Fra09] Sérgio Loureiro Fraga. *Impulsive feedback control: a constructive approach*. PhD thesis, Universidade do Porto, 2009.

[Ger12] Matthias Gerdts. *Optimal Control of Odes and Daes*. De Gruyter, January 2012.

[GMS08] Philip E. Gill, Walter Murray, and Michael A. Saunders. *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming*, 2008.

[GP11] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control*. 2011.

[GSD06] Graham C. Goodwin, María M. Seron, and José A. de Doná. *Constrained Control and Estimation: An Optimisation Approach*. Springer Science & Business Media, March 2006.

[HFD11a] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[HFD11b] B. Houska, H.J. Ferreau, and M. Diehl. An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range. *Automatica*, 47(10):2279–2285, 2011.

[HFVQ13] B. Houska, H.J. Ferreau, M. Vukov, and R. Quirynen. ACADO Toolkit User's Manual. http://www.acadotoolkit.org, 2013.

[HP95a] R. Horst, Panos M. Pardalos, and Nguyen Van Thoai . *Introduction to Global Optimization*. 3rd edition, 1995.

[HP95b] R. Horst and Panos M. Pardalos. *Handbook of Global Optimization*, volume 1. Springer, 1995.

[HT99] R. Horst and N. V. Thoai. DC programming: Overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, October 1999.

[JTB04] S. L. Campbell J. T. Betts. Initialization of direct transcription optimal control software. pages 3802 – 3807 vol.4, 2004.

[KL13] Christian Kirches and Sven Leyffer. TACO: a toolkit for AMPL control optimization. *Mathematical Programming Computation*, 5(3):227–265, September 2013.

[KM95] I. Kolmanovsky and N.H. McClamroch. Developments in nonholonomic control problems. *IEEE Control Systems*, 15(6):20–36, 1995.

[KPP14] Igor Kornienko, Luís Tiago Paiva, and Maria do Rosário de Pinho. Introducing state constraints in optimal control for health problems. *Procedia Technology*, 17:415–422, 2014.

[Kro93] V. F. Krotov. Global methods in optimal control theory. In Alexander B. Kurzhanski, editor, *Advances in Nonlinear Dynamics and Control: A Report from Russia*, number 17 in Progress in Systems and Control Theory, pages 74–121. Birkhäuser Boston, January 1993.

[LLC13] Ziena Optimization LLC. *KNITRO Documentation*, 2013.

[MB05] Ian M. Mitchell, Alexandre M. Bayen, and TomlinClaire J. . A time-dependent hamilton–jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, July 2005.

[MHL99] Manfred Morari and Jay H. Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4–5):667–682, May 1999.

[MM90] D.Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35(7):814–824, July 1990.

[MM93] H. Michalska and D.Q. Mayne. Robust receding horizon control of constrained nonlinear systems. *IEEE Transactions on Automatic Control*, 38(11):1623–1633, November 1993.

[MRRS00] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.

[NBW11] Tim Nicolayzik, Christof Büskens, and Dennis Wassel. Nonlinear optimization in space applications with WORHP. 2011.

[NL10] Rachael Miller Neilan and Suzanne Lenhart. An introduction to optimal control with an application in disease modeling. *Modeling paradigms and analysis of disease transmission models*, 75:67–81, 2010.

[Pai13] Luís Tiago Paiva. Optimal control in constrained and hybrid nonlinear system: Solvers and interfaces. Technical report, Faculdade de Engenharia, Universidade do Porto, 2013.

[PF13] Luís Tiago Paiva and Fernando A.C.C. Fontes. Mesh refinement strategy for optimal control problems. *AIP Conference Proceedings*, 1558:590–593, October 2013.

[PF14a] Luís Tiago Paiva and Fernando A. C. C. Fontes. Time–mesh refinement in optimal control problems for nonholonomic vehicles. *Procedia Technology*, 17:178–185, 2014.

[PF14b] Luís Tiago Paiva and Fernando A.C.C. Fontes. Adaptive time-mesh refinement in optimal control problems with constraints. *Discrete and Continuous Dynamical Systems*, 2014. (Submitted for Publication).

[PF14c] Luís Tiago Paiva and Fernando A.C.C. Fontes. Mesh refinement in optimal control: Nonholonomic vehicles manoeuvre problems, January 2014.

[PHR14] Michael A. Patterson, William W. Hager, and Anil V. Rao. A ph mesh refinement method for optimal control. *Optimal Control Applications and Methods*, February 2014.

[Pin96] János D. Pintér. *Global Optimization in Action - Continuous and Lipschitz Optimization: Algorithms, Implementations.* 1996.

[Pin10] Heitor Pina. *Métodos Numéricos.* Escolar Editora, 3rd edition, 2010.

[PR02] Panos M. Pardalos and H. Edwin Romeijn. *Handbook of Global Optimization*, volume 2. Springer, 2002.

[PR13] Michael A. Patterson and Anil V. Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software*, 2013.

[PR14] Michael A. Patterson and Anil V. Rao. *GPOPS-II A General-Purpose MATLAB Software for Solving Multiple-Phase Optimal Control Problems*, May 2014.

[QB03] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, July 2003.

[Raw00] J.B. Rawlings. Tutorial overview of model predictive control. *IEEE Control Systems*, 20(3):38–52, June 2000.

[RL92] O. Rosen and R. Luus. Global optimization approach to nonlinear optimal control. *Journal of Optimization Theory and Applications*, 73(3):547–562, June 1992.

[RLL+09] Davide Martino Raimondo, Daniel Limon, Mircea Lazar, Lalo Magni, and Eduardo Ferná ndez Camacho. Min-max model predictive control of nonlinear systems: A unifying overview on stability. *European Journal of Control*, 15(1):5–21, 2009.

[RRTP76] J. Richalet, A. Rault, J. L. Testud, and J Papon. Algorithmic control of industrial processes. In *4th IFAC symposium on identification and system parameter estimation*, pages 1119–1167, 1976.

[RV99] F. Rampazzo and R. B. Vinter. A theorem on existence of neighbouring trajectories satisfying a state constraint, with applications to optimal control. *IMA Journal of Mathematical Control and Information*, 16(4):335–351, December 1999.

[Tec] Boeing Research & Technology. *SOCS User's Guide*.

[Var72] P.P. Varaiya. *Notes on Optimization*. Van Nostrand Reinhold, 1972.

[Vin00] Richard B. Vinter. *Optimal Control*. Springer, 2000.

[WB06] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, March 2006.

[Wei08] Thomas Weise. Global optimization algorithms – theory and application, 2008.

[wor12] *Tutorial for WORHP 2.0*, 2012.

[wor13] *WORHP User Manual*, 2013.

[Wä14] Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*, 2014.

[ZT11] Yiming Zhao and Panagiotis Tsiotras. Density functions for mesh refinement in numerical optimal control. *Journal of Guidance, Control, and Dynamics*, 34(1):271–277, January 2011.