SLOVAK TECHNICAL UNIVERSITY IN BRATISLAVA
Faculty of Chemical and Food Technology
Department of Information Engineering and Process Control
Radlinského 9, 812 37 Bratislava

Bc. Michal Čižniar

DYNAMIC OPTIMISATION OF PROCESSES

Diploma Work

Supervisors
Doc. Dr. Ing. Miroslav Fikar, STU Bratislava
Prof. M.A. Latifi, ENSIC Nancy

2005

SLOVENSKÁ TECHNICKÁ UNIVERZITA
Fakulta chemickej a potravinárskej technológie
Radlinského 9, 812 37 Bratislava

Katedra: **informatizácie a riadenia procesov**
Číslo: 4/KIRP/2005

Vec: **Zadanie diplomovej práce**

Meno a priezvisko študenta: **Bc. Michal Čižniar**
Meno a priezvisko vedúceho diplomovej práce**: Doc. Dr. Ing. Miroslav Fikar**
Meno a priezvisko konzultanta diplomovej práce**:**
Názov diplomovej práce:

Dynamická optimalizácia procesov

Návrh dynamickej optimalizácie procesu chemickej technológie metódou ortogonálnej kolokácie na konečných prvkoch. Vytvorenie balíka v MATLABe a jazyku C. Práca bude uskutočnená na ENSIC Nancy v rámci programu SOCRATES.

Termín odovzdania diplomovej práce: **21. mája 2005**
Diplomová práca sa odovzdáva v 3 exemplároch vedúcemu katedry.

Bratislava **1. februára 2005**

Doc. Dr. Ing. Miroslav Fikar
vedúci katedry

Prof. Ing. Dušan Bakoš, DrSc.
dekan

This work would have never been written without the help, continuous support, and encouragement of several people.
First of all, I want to express my sincere gratitude to my supervisor and head of the Department of Information and Process Control at Faculty of Chemical and Food Technology of the Slovak Technical University in Bratislava, Doc. Dr. Ing. Miroslav Fikar, for his patient guidance throughout my studies and the given opportunity to move my borders.
I gratefully thanks to Professor M. Abderazzak Latifi from ENSIC-INPL, for beeing my suppervisor in Nancy, and giving me all the guidance and support I needed.
Also I would like to express my thank to Djalal Salhi, MSc. also from ENSIC-INPL, for his excellent and tireless support and many reviews that inspired me to improve myself.
In addition, I thanks to Professor Jean-Piere Corriou from ENSIC-INPL, for giving me the opportunity to attend his lectures.
Finally, I am grateful to my parents for supporting my studies in many ways for such a long time.

Bratislava, Nancy, 2005
Michal Čižniar

Abstrakt

Táto práca sa zaoberá dynamickou optimalizáciou procesov. Pozostáva v hľadaní optimálnych profilov riadiacich a stavových veličín, ktoré optimalizujú danú účelovú funkciu vzhľadom k daným obmedzeniam. Bola vyvinutá metóda ortogonálnej kolokácie na konečných prvkoch, ktorá bola implementovaná do MATLABu. Pôvodné problémy optimalizácie sú tak konvertované do NLP problémov, na riešenie ktorých možno použiť ľubovolný program na riešenie úlohy nelineárneho programovania. Gradienty účelovej funkcie ako aj funkcie obmedzení sú počítané analyticky.

Abstract

This work deals with dynamic optimisation of processes. It consists in searching for optimal profiles of decision variables which optimise a given performance index under specified constraints. The method of orthogonal collocations of finite elements has been developed and implemented within MATLAB environment. The original optimisation problems are then converted into NLP problems which are solved using appropriate NLP solvers, i.e., SQP methods. The gradients of the performance index as well as of the constraints needed in the NLP solver are analytically computed using formal calculus. Several application are succesfully tested.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimisation problems are ubiquitous in the mathematical modeling of real world systems and cover a very broad range of applications. These applications arise in all branches of Engineering, Computer Science, Economics, Finance, Operations Research and Management Science, Chemistry, Materials Science, Astronomy, Physics, Structural and Molecular Biology, and Medicine.

Since the second world war, there has been an explosive growth in theory and techniques in all facets of optimisation. These include: combinatorial optimisation, complementary and variational inequalities, constraint logic programming, convex optimisation, nonsmooth optimisation, deterministic global optimisation, stochastic global optimisation, goal programming, multi-objective optimisation, semi-infinite programming, bilevel and multilevel linear and nonlinear optimisation, nonlinear unconstrained and constrained optimisation, cone programming, dynamic programming, generalized geometric programming, generalized convexity, game theory, interval analysis, financial optimisation, parallel optimisation, dynamic optimisation, optimal control, mixer-integer nonlinear optimisation, integer programming, linear programming, semidefinite programming, network optimisation, robust optimisation, stochastic programming, scheduling, planning, logistics, telecommunications, modeling systems and optimisation test problems and software.

However, the solution of optimisation problems with differential and algebraic equation (DAE) models still remains a difficult problem. At present, optimisation problems with nonlinear algebraic equations can be solved in straightforward way as nonlinear programs. On the other hand, unconstrained problems with differential equation models can be handled through calculus of variations. However, models that combine both of these features are currently optimised by imposing some level of approximation to the problem.

Current methods for handling these problems either apply an approximation to the control variable profile or to both the state and control profiles. A straightforward approach adopted by [18] is to parametrise the control profile (e.g., piecewise constant) over variable-length finite elements and to solve the differential equations with this parameterisation. A nonlinear programming (NLP) algorithm is then applied to the control parameters in an outer calculation loop. This feasible path approach requires the repeated and expensive

solution of the DAE systems. Also, state variable inequality constraints cannot be handled in a straightforward way. Finally, the quality of the solution is strongly dependent on the parameterisation of the control profile. Early studies with the second approach, parameterisation of both the state and control profiles, were reported by [15, 19]. Here state and control profiles and the differential equations were parameterised by use of some method of weighted residuals (e.g., orthogonal collocation). This leads to a large NLP formulation with algebraic equality constraints. However, since NLP algorithms were less developed at that time, this approach either was inefficient when compared to feasible path methods or was restricted to specialised one(e.g. linear problems).

With advances in NLP methods through the development of Successive Quadratic Programming (SQP), these NLP's could be solved more efficiently and could handle nonlinear state and control profile constraints in a straightforward manner. Large problems have been solved by [17] with orthogonal collocation on finite elements and piecewise constant approximations to the control profile. In order to obtain accurate finite element solutions, however, additional constraints were imposed by [4, 5] in the NLP formulation in order to enforce accurate state profiles. They classified the role of finite elements in terms of knot locations and breakpoints that allowed discontinuities for control profile. This led to a formulation that enforced the accurate solution of the differential equations and allowed a general description of the control profile.

The purpose of this work is to develop and discuss a NLP formulation for optimal control problems using orthogonal collocation on finite elements method that leads to the accurate solution of the general differential-algebraic optimal control problem (DAOP).

Chapter 2 deals with dynamic optimisation in general. The chapter starts with several dynamic optimisation problem definitions. In the second part, several approaches to the solution of this problems are described. Finally, this chapter ends with the NLP problem formulation.

Chapter 3 contains description of the function *dynopt*, the main function of the collection of functions which extend the capability of MATLAB Optimisation Toolbox, specifically of the constrained nonlinear minimisation routin *fmincon*. The chapter starts with tables listing general descriptions of all the input and output arguments and the parameters in the optimisation options structure, continues with the function description, and ends with some tutorial.

Chapter 4 presents and discuss the examples from literature sloved by *dynopt*.

Chapter 5 presents conclusions for this work and also it presents the goals for a future work.

Chapter 2

Dynamic Optimisation

This chapter deals with dynamic optimisation in general. The chapter starts with several dynamic optimisation problem definitions. In the second part, several approaches to the solution of this problems are described. Finally, this chapter ends with the NLP problem formulation.

2.1 Optimisation Problem Statement

The objective of dynamic optimisation is to determine, in open loop control, a set of decision variable time profiles (pressure, temperature, flow rate, current, heat duty, ...) for dynamic system that optimise a given performance index (or cost functional or optimisation criterion)(cost, time, energy, selectivity, ...) subject to specified constraints (safety, environmental and operating constraints). Optimal control refers to the determination of the best time-varying profiles in closed loop control.

2.1.1 Cost Functional

The performance index (cost functional or optimisation criterion) can in general be written in one of three forms as follows:

Bolza form

$$\mathcal{J}(\boldsymbol{u}(t)) = \mathcal{G}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\mathrm{d}t \tag{2.1}$$

Lagrange form

$$\mathcal{J}(\boldsymbol{u}(t)) = \int_{t_0}^{t_f} \mathcal{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\mathrm{d}t \tag{2.2}$$

Mayer form

$$\mathcal{J}(\boldsymbol{u}(t)) = \mathcal{G}(\boldsymbol{x}(t_f), t_f) \tag{2.3}$$

where

$\mathcal{J}$ represents optimisation criterion.

$\mathcal{G}$ represents the component of objective function evaluated at final conditions.

$\int_{t_0}^{t_f} \mathcal{F}\mathbf{d}t$ represents the component of the objective function evaluated over a period of time.

$\boldsymbol{x}(t)$ represents state profile vector.

$\boldsymbol{u}(t)$ represents control profile vector.

2.1.2 Process Model Equations

The behaviour of many of processes can in general be described by a set of ordinary differential equations (ODE) as follows:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \quad \text{over} \quad t_0 \leq t \leq t_f \tag{2.4}$$

This ODE system forms equality constraint in optimal control problem.

2.1.3 Constraints

Constraints to be accounted for typically include equality and inequality infinite dimensional, interior-point, and terminal-point constraints [8]. Moreover, they may be written in the following canonical form similar to the cost form (2.1):

$$\mathcal{J}_i(\boldsymbol{u}(t)) = \mathcal{G}_i(\boldsymbol{x}(t_i), t_i) + \int_{t_0}^{t_i} \mathcal{F}_i(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\mathrm{d}t \tag{2.5}$$

where $t_i \leq t_f, i = 1, \ldots, nc$, and nc is the number of constraints.

2.2 Optimal Control Problem Solutions

There are several approaches that can solve optimal control problems. These can be devided into analytical methods that have been used originally and numerical methods preferred nowadays.

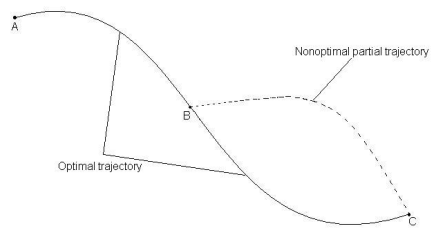

Figure 2.1: Graphical interpretation of Bellman's principle of optimality

2.2.1 Analytical Methods

One can choose from a whole series of methods to solve optimal control problems. The most important are:

- Dynamic Programming (Bellman's principle of optimality),
- Pontryagin's principle of minimum (maximum),
- Variational Calculus

Dynamic Programming

Dynamic Programming (DP) is very general method for treating variety of optimisation problems [1, 21] often used in analysis and design of automatic control systems. This method is based on the principle of optimality first formulated by Bellman.

Bellman's principle of optimality can simply be formulated as follows: *"If there is an optimal way from A to C, then each partial way from B to C is also optimal".*

Consider following optimal control problem in Bolza form:

$$\mathcal{J}(\boldsymbol{u}(t)) = \mathcal{G}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\mathrm{d}t \tag{2.6a}$$

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.6b}$$

It is supposed, that this optimal control problem (2.6) has a solution. Consider the following function, also called Bellman's function, defined as:

$$\mathcal{V}(\boldsymbol{x}(t), t) = \min_{\boldsymbol{u}(t)} \left[\mathcal{G}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), \tau)\mathrm{d}\tau\right] \tag{2.7}$$

Differentiating (2.7) leads to Bellman's partial differential equation (2.8)

$$-\frac{\partial \mathcal{V}}{\partial t} = \min_{\boldsymbol{u}(t)} \left[\mathcal{F}(\boldsymbol{x}, \boldsymbol{u}, t) + \left(\frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}\right)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t)\right] \tag{2.8}$$

which must satisfy the boundary condition

$$\mathcal{V}(\boldsymbol{x}_f, t_f) = \mathcal{G}(\boldsymbol{x}_f, t_f) \tag{2.9}$$

Bellman's partial differential equation (2.8) together with boundary condition (2.9) represent necessary conditions for obtaining minimum of the optimisation problem (2.6).

Substitution of optimal control variable $\boldsymbol{u}^*$ into Bellman's partial differential equation (2.8) leads to formulation also known as Hamilton-Jacobi-Bellman's partial differential equation:

$$-\frac{\partial \mathcal{V}}{\partial t} = \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}^*, t) + \left(\frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}\right)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}^*, t) \tag{2.10}$$

By treating optimal control problems it is convenient to define Hamiltonian function as follows:

$$\mathcal{H}\left(\boldsymbol{x}, \boldsymbol{u}, \frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}, t\right) = \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}, t) + \left(\frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}\right)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.11}$$

after the substitution of (2.11) into (2.8), Bellman's partial differential equation takes form:

$$-\frac{\partial \mathcal{V}}{\partial t} = \min_{\boldsymbol{u}(t)} \mathcal{H}\left(\boldsymbol{x}, \boldsymbol{u}, \frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}, t\right) \tag{2.12}$$

Pontryagin's Principle of Minimum

Another very efficient approach to solution of optimal control problems is presented via Pontryagin's principle of minimum (PMP) [1, 14, 21]. There is a very close relationship between DP and PMP, which consider two totaly different approaches to the solution of the optimal control problems.

Consider the before mentioned control problem (2.6), and mark $\frac{\partial \mathcal{V}}{\partial \boldsymbol{x}}$ in (2.11) as adjoint variable $\boldsymbol{\lambda}(t)$. The appropriate Hamiltonian function takes following form:

$$\mathcal{H}\left(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t\right) = \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}, t) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.13}$$

Bellman's partial differential equation (2.12) takes after the substitution following form:

$$-\frac{\partial \mathcal{V}}{\partial t} = \min_{\boldsymbol{u}(t)} \mathcal{H}\left(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}, t\right) \tag{2.14}$$

Differentiating left and right side of $\frac{\partial \mathcal{V}}{\partial \boldsymbol{x}} = \boldsymbol{\lambda}(t)$ with respect to $\boldsymbol{x}$ separately gives

$$-\frac{\partial^2 \mathcal{V}}{\partial \boldsymbol{x} \partial t} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} + \frac{\partial^2 \mathcal{V}}{\partial \boldsymbol{x}^2} \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}} \tag{2.15a}$$

$$\dot{\boldsymbol{\lambda}} = \frac{\partial^2 \mathcal{V}}{\partial \boldsymbol{x}^2} \dot{\boldsymbol{x}} + \frac{\partial^2 \mathcal{V}}{\partial t \partial \boldsymbol{x}} \tag{2.15b}$$

From this follow canonical differential equations of the principle fo minimum (2.17)

$$\dot{\boldsymbol{x}} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}} \tag{2.16a}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} \tag{2.16b}$$

Necessary conditions for the optimisation problem (2.6) using Pontryagin's principle of minimum can be then formulated as follows:

- optimality condition for the control variable:

$$\mathbf{0} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{u}^T}, \quad \forall t \in [t_0, t_f] \tag{2.17a}$$

- definition of adjoint variables:

$$\dot{\boldsymbol{\lambda}}^T = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}^T}, \quad \forall t \in [t_0, t_f] \tag{2.17b}$$

- terminal conditions for adjoint variables:

$$\boldsymbol{\lambda}^T(t_f) = \frac{\partial \mathcal{G}}{\partial \boldsymbol{x}^T}\Big|_{t_f} \tag{2.17c}$$

Variational Calculus

Dynamic Programming and Pontryagin's principle of minimum are more general and effective than classic variational calculus. The elementary terms of variational calculus are obtained from Bellman's partial differential equation [21].

The classical variational calculus problem for the constraint (2.4) follows from the Euler-Lagrange differential equation

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} - \frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{x}}}\Big) = \mathbf{0} \tag{2.18}$$

where $\mathcal{L}$ is the Lagrange function defined as

$$\mathcal{L}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{\lambda}, t) = \mathcal{F}(\boldsymbol{x}, \boldsymbol{u}, t) + \boldsymbol{\lambda}^T[\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) - \dot{\boldsymbol{x}}] \tag{2.19}$$

Necessary conditions for the optimisation problem (2.6) using the Euler-Lagrange differential equation can then be formulated as follows:

- optimality condition for the control variable:

$$\mathbf{0} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}^T}, \quad \forall t \in [t_0, t_f] \tag{2.20a}$$

- definition of adjoint variables:

$$\dot{\boldsymbol{\lambda}}^T = -\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^T}, \quad \forall t \in [t_0, t_f] \tag{2.20b}$$

- terminal conditions for adjoint variables:

$$\boldsymbol{\lambda}^T(t_f) = \frac{\partial \mathcal{G}}{\partial \boldsymbol{x}^T}\Big|_{t_f} \tag{2.20c}$$

2.2.2 Numerical Methods

Optimality conditions mentioned in section 2.2.1 are in general case not able to provide optimum since the resulting two point boundary value problem (TPBVP) or Bellman's partial differential equation is difficult to solve numerically. Computational methods are therefore needed. We will concentrate ourselves to numerical solution of the problem by Pontryagin's approach. The numerical methods used for the solution of dynamic optimisation problems can then be grouped into two categories:

- indirect methods
- direct methods

Indirect Methods

The objective of these methods is to solve the TPBVP, thus indirectly solving the dynamic optimisation problem (2.1), (2.4). The most well known methods in this category are:

Boundary Condition Iteration (BCI) attempts to find the missing boundary conditions $\boldsymbol{\lambda}(t_0)$ by minimising the error in the boundary conditions so that equations (2.17b),(2.4) can be integrated together forward in time. It is worth noticing that this method may lead to instabilities due to the fact that the costate equations (2.4) are integrated forward in time in the opposite direction of their natural boundary conditions.

Control Vector Iteration (CVI) here the state equations (2.4) are integrated forward using a guess for the control variable profiles, and then the adjoint equations (2.17b) are integrated backward. Equations (2.17a) are then used locally to update the guessed profiles of control variables at a discrete number of points, and globally as a termination criterion. The main advantage of this method is that the adjoint equations (2.17b) are integrated backward, (e.g., in the same direction of their natural boundary conditions). However, this approach has slow convergence for many problems.

Direct Methods

In this category, we discuss two strategies:

Sequential Method often called control vector parameterisation (CVP), consists in an approximation of the control trajectory by a function of only few parameters and leaving the state equations in the form of the original ODE/DAE system [8]. Mostly, piece-wise constant control profiles are used. Hence, infinite dimensional optimisation problem in continuous control variables is transformed into a finite dimensional nonlinear programming (NLP) problem which can be solved by any gradient-based method (e.g., an SQP method). The cost functional evaluation is carried out by solving an initial value problem (IVP) of the original DAE system and gradient of

the performance index as well as of the constraints with respect to the parameters $\boldsymbol{u}$ may be evaluated by solving either the adjoint equations (2.17b) or the sensitivity equations. Moreover this method is feasible type method, e.g., the solution is improved at each iteration.

Simultaneous Method often called total discretisation method, uses the discretisation of both the control and state variables using polynomials of which the coefficients become the decision variables in a much larger NLP problem [4]. Implementation of this method is subject of this work. Unlike CVP method, the simultaneous method does not require the solution of IVPs at every iteration of the NLP. The method is however of infeasible-type, e.g., the solution is available only once the iterative process has converged.

2.3 NLP Formulation Problem

As mentioned before, the optimal control problem will be solved by complete parametrisation of both the control and the state profile vector [10, 11]. That means, that the initial linear control and state profiles are approximated by linear combination of some basis functions. It is expected here, that the basis functions are known so only the coefficients of linear combination of these fundamentals have to be optimised. In addition, each control sequence segment is defined on time interval, which length itself can be the optimised variable. It is supposed that the optimised dynamic model can be described by an ODE system.

Consider the following general control problem for $t \in [t_0, t_f]$:

$$\min_{\boldsymbol{u}(t)} \{\mathcal{G}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\mathrm{d}t\} \tag{2.21}$$

such that

$$\begin{gathered}
\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\
\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\
\boldsymbol{h}(\boldsymbol{u}(t), \boldsymbol{x}(t)) = \boldsymbol{0} \\
\boldsymbol{g}(\boldsymbol{u}(t), \boldsymbol{x}(t)) \leq \boldsymbol{0} \\
\boldsymbol{x}(t)^L \leq \boldsymbol{x}(t) \leq \boldsymbol{x}(t)^U \\
\boldsymbol{u}(t)^L \leq \boldsymbol{u}(t) \leq \boldsymbol{u}(t)^U
\end{gathered}$$

where

$\boldsymbol{h}$ – equality design constraint vector,

$\boldsymbol{g}$ – inequality design constraint vector,

$\boldsymbol{x}(t)^L, \boldsymbol{x}(t)^U$ – state profile bounds,

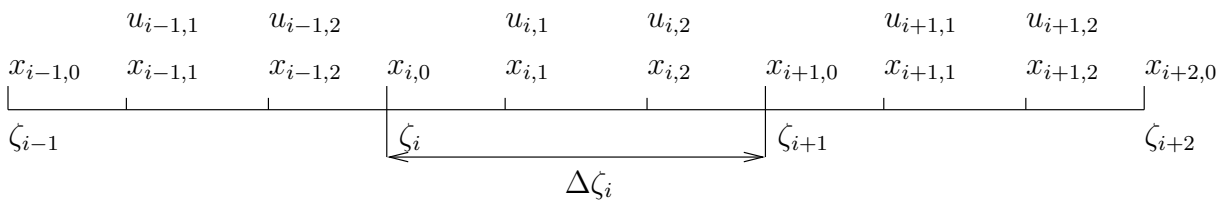

Figure 2.2: Finite-element collocation discretisation for state profiles, control profiles and element lengths

$\boldsymbol{u}(t)^L, \boldsymbol{u}(t)^U$ – control profile bounds.

In order to derive the NLP problem the differential equations are converted into algebraic equations using collocation on finite elements. Residual equations are then formed and solved as a set of algebraic equations. These residuals are evaluated at the shifted roots of Legendre polynomials. The procedure is then following: Consider the initial-value problem over a finite element i with time $t \in [\zeta_i, \zeta_{i+1}]$:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \qquad t \in [t_0, t_f] \tag{2.22}$$

The solution is approximated by Lagrange polynomials over element i, $\zeta_i \leq t \leq \zeta_{i+1}$ as follows:

$$\boldsymbol{x}_{K+1}(t) = \sum_{j=0}^{K} \boldsymbol{x}_{ij}\phi_j(t); \qquad \phi_j(t) = \prod_{k=0,j}^{K} \frac{(t - t_{ik})}{(t_{ij} - t_{ik})} \tag{2.23}$$

$$\text{in element } i \qquad i = 1, \ldots, \text{NE}$$

$$\boldsymbol{u}_K(t) = \sum_{j=1}^{K} \boldsymbol{u}_{ij}\theta_j(t); \qquad \theta_j(t) = \prod_{k=1,j}^{K} \frac{(t - t_{ik})}{(t_{ij} - t_{ik})} \tag{2.24}$$

$$\text{in element } i \qquad i = 1, \ldots, \text{NE}$$

Here $k = 0, j$ means k starting form 0 and $k \neq j$, NE represents the number of elements. Also $\boldsymbol{x}_{K+1}(t)$ is a $(K+1)$th degree piecewise polynomial and $\boldsymbol{u}_K(t)$ is Kth degree piecewise polynomial. The difference in orders is due to the existence of the initial conditions for $\boldsymbol{x}(t)$, for each element i. Also, the Lagrange polynomial has the desirable property that (for $\boldsymbol{x}_{K+1}(t)$, for example):

$$\boldsymbol{x}_{K+1}(t_{ij}) = \boldsymbol{x}_{ij} \tag{2.25}$$

which is due to the Lagrange condition $\phi_k(t_j) = \delta_{kj}$, where δ_{kj} is the Kronecker delta. This polynomial form allows the direct bounding of the states and controls, e.g., path constraints can be imposed on the problem formulation.

By using K point orthogonal collocation on finite elements as shown in Figure 2.2, and by defining the basis functions, so that they are normalised over the each element

$\Delta\zeta_i(\tau \in [0,1])$, one can write the residual equation as follows:

$$\boldsymbol{r}(t_{ik}) = \sum_{j=0}^{K} \boldsymbol{x}_{ij} \frac{\dot{\phi}_j(\tau_k)}{\Delta\zeta_i} - \boldsymbol{f}(t_{ik}, \boldsymbol{x}_{ik}, \boldsymbol{u}_{ik}) \tag{2.26}$$
$$i = 1, \ldots, \mathrm{NE}$$
$$j = 0, \ldots, K$$
$$k = 1, \ldots, K$$

where $\dot{\phi}_j(\tau_k) = \mathrm{d}\phi_j/\mathrm{d}\tau$, and together with $\phi_j(\tau)$, $\theta_j(\tau)$ terms (basis functions), they are calculated beforehand, since they depend only on the Legendre root locations. Note that $t_{ik} = \zeta_i + \Delta\zeta_i\tau_k$. This form is convenient to work with when the element lengths are included as decision variables. The element lengths are also used to find possible points of discontinuity for the control profiles and to insure that the integration accuracy is within a numerical tolerance. Additionally, the continuity of the states is enforced at element endpoints (interior knots $\zeta_i, i = 2, ..., \mathrm{NE}$), but it is allowed that the control profiles to have discontinuities at these endpoints. Here

$$\boldsymbol{x}^i_{K+1}(\zeta_i) = \boldsymbol{x}^{i-1}_{K+1}(\zeta_i) \tag{2.27}$$
$$i = 2, \ldots, \mathrm{NE}$$

or

$$\boldsymbol{x}_{i0} = \sum_{j=0}^{K} \boldsymbol{x}_{i-1,j}\phi_j(\tau = 1) \tag{2.28}$$
$$i = 2, \ldots, \mathrm{NE}$$
$$j = 0, \ldots, K$$

These equations extrapolate the polynomial $\boldsymbol{x}^{i-1}_{K+1}(t)$ to the endpoints of its element and provide an accurate initial conditions for the next element and polynomial $\boldsymbol{x}^i_{K+1}(t)$.

At this point a few additional comments concerning construction of the control profile polynomials must be made. Note that these polynomials use only K coefficients per element and are of lower order than the state polynomials. As a result these profiles are constrained or bounded only at collocation points. The constraints of the control profile are carried out by bounding the values of each control polynomial at both ends of the element. This can be done by writing the equations:

$$\boldsymbol{u}^L_i \leq \boldsymbol{u}^i_K(\zeta_i) \leq \boldsymbol{u}^U_i \quad i = 1, \ldots, \mathrm{NE} \tag{2.29}$$
$$\boldsymbol{u}^L_i \leq \boldsymbol{u}^i_K(\zeta_{i+1}) \leq \boldsymbol{u}^U_i \quad i = 1, \ldots, \mathrm{NE} \tag{2.30}$$

Note that since the polynomial coefficients of the control exist only at collocation points, enforcement of these bounds can be done by extrapolating the polynomial to the endpoints

of the element. This is easily done by using:

$$\boldsymbol{u}_K^i(\zeta_i) = \sum_{j=1}^{K} \boldsymbol{u}_{ij}\theta_j(\tau = 0) \qquad i = 1, \ldots, \mathrm{NE} \tag{2.31}$$

and

$$\boldsymbol{u}_K^i(\zeta_{i+1}) = \sum_{j=1}^{K} \boldsymbol{u}_{ij}\theta_j(\tau = 1) \qquad i = 1, \ldots, \mathrm{NE} \tag{2.32}$$

Adding these constraints affects the shape of the final control profile and the net effect of these constraints is to keep the endpoint values of the control profile from varying widely outside their ranges $[\boldsymbol{u}_i^L, \boldsymbol{u}_i^U]$.

The NLP formulation consists of the ODE model (2.21) discretised on finite elements, continuity equation for state variables, and any other equality and inequality constraints that may be required. It is given by

$$\min_{\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, \Delta\zeta_i} \left[\mathcal{G}(\boldsymbol{x}_f, t_f) + \sum_{i=1}^{NE} \sum_{j=1}^{K} \boldsymbol{w}_{ij} \mathcal{F}(\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, \Delta\zeta_i) \right] \tag{2.33}$$

such that

$$\boldsymbol{x}_{10} - \boldsymbol{x}_0 = \boldsymbol{0} \tag{2.34}$$

$$\boldsymbol{r}(t_{ik}) = \boldsymbol{0} \quad i = 1, \ldots, \mathrm{NE} \quad k = 1, \ldots, K \tag{2.35}$$

$$\boldsymbol{x}_{i0} - \boldsymbol{x}_{K+1}^{i-1}(\zeta_i) = \boldsymbol{0} \quad i = 2, \ldots, \mathrm{NE} \tag{2.36}$$

$$\boldsymbol{x}_f - \boldsymbol{x}_{K+1}^{NE}(\zeta_{NE+1}) = \boldsymbol{0} \tag{2.37}$$

$$\boldsymbol{u}_i^L \leq \boldsymbol{u}_K^i(\zeta_i) \leq \boldsymbol{u}_i^U \quad i = 1, \ldots, \mathrm{NE} \tag{2.38}$$

$$\boldsymbol{u}_i^L \leq \boldsymbol{u}_K^i(\zeta_{i+1}) \leq \boldsymbol{u}_i^U \quad i = 1, \ldots, \mathrm{NE} \tag{2.39}$$

$$\boldsymbol{u}_{ij}^L \leq \boldsymbol{u}_K(\tau_j) \leq \boldsymbol{u}_{ij}^U \quad i = 1, \ldots, \mathrm{NE} \quad j = 1, \ldots, K \tag{2.40}$$

$$\boldsymbol{x}_{ij}^L \leq \boldsymbol{x}_{K+1}(\tau_j) \leq \boldsymbol{x}_{ij}^U \quad i = 1, \ldots, \mathrm{NE} \quad j = 0, \ldots, K \tag{2.41}$$

$$\Delta\zeta_i^L \leq \Delta\zeta_i \leq \Delta\zeta_i^U \quad i = 1, \ldots, \mathrm{NE} \tag{2.42}$$

$$\sum_{i=1}^{NE} \Delta\zeta_i = \zeta_{\mathrm{total}} \tag{2.43}$$

$$\boldsymbol{h}(\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, t_{ij}) = \boldsymbol{0} \tag{2.44}$$

$$\boldsymbol{g}(\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, t_{ij}) \leq \boldsymbol{0} \tag{2.45}$$

$$\boldsymbol{h}_f(\boldsymbol{x}_f) = \boldsymbol{0} \tag{2.46}$$

$$\boldsymbol{g}_f(\boldsymbol{x}_f) \leq \boldsymbol{0} \tag{2.47}$$

where

i – refers to the element,

j, k – refers to the collocation point,

$\boldsymbol{w}_{ij}$ – positive quadrature weights,

$\Delta\zeta_i$ – finite-element lengths $i = 1, \ldots, \mathrm{NE}$,

$\boldsymbol{x}_f = \boldsymbol{x}(t_f)$ – the value of the state at the final time $t = t_f$,

$\boldsymbol{h}_f$ – the nonlinear equality contstraint evaluated at the final time t_f,

$\boldsymbol{g}_f$ – the nonlinear inequality constraint evaluated at the final time t_f,

$\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}$ – the collocation coefficients for the state and control profiles

Problem (2.29) can be now solved by any large-scale nonlinear programming solver.

To solve this problem within MATLAB, the Optimization Toolbox was used. This includes several programs for treating optimisation problems. In this case function *fmincon* was choosen. This can minimise/maximise given objective function with respect to nonlinear equality and inequality constraints. In order to use this function it was neccessary to create and program series of additional functions. These additional functions together with *fmincon* are formed within *dynopt* [20] which is simple for user to employ. This function is presented in next chapter.

Chapter 3

Description of the *dynopt* Function

This chapter contains description of the function *dynopt*, the main function of the collection of functions which extend the capability of MATLAB Optimisation Toolbox, specifically of the constrained nonlinear minimisation routin *fmincon*. The chapter starts with tables listing general descriptions of all the input and output arguments and the parameters in the optimisation options structure, continues with the function description, and ends with some tutorial.

3.1 Function Arguments

These tables describe arguments used by *dynopt*: the first describes input arguments, the second describes the output arguments, and the third describes the optimisation options parameters structure `options` which is given by MATLAB.

ncoli	– number of collocation points + 1
ncolt	– number of collocation points + 2
ni	– number of intervals
nx	– number of state variables
nu	– number of control variables

Table 3.1: Some predefined variables which are used for function description

Table 3.1 describes defines variables which are used to simplify *dynopt's* description.

Table 3.2: Input arguments

Argument	Description
ncol	The number of collocation points.
`delta_t`	The (ni–by–1) vector of initial lengths of intervals.
`tf`	The final time, if the time is not specified use empty brackets [ ].
`uinit`	The (nu–by–ni) matrix of control variable initial values.
`bdx`	The state variables bounds matrix with size (nx–by–2), for example [***lbx ubx***].
`bdu`	The control variables bounds (nu–by–2) matrix, for example [***lbu ubu***].
`bdt`	The time interval bounds (1–by–2) matrix, for example [lbdt ubdt]).
`objfun`	The function to be optimised. *Objfun* is the name of an M-file. For more information about this input argument, see section 3.2.4.
`confun`	The function that computes the nonlinear equality and inequality constraints. *Confun* is the name of an M-file. For more information about this input argument, see section 3.2.4.
`process`	The function that describes given process. *Process* is the name of an M-file. For more information about this input argument, see section 3.2.4.
`options`	An optimisation options parameter structure that defines parameters used by the optimisation functions. This parameter is defined by MATLAB for all optimisation routins of MATLAB Optimisation Toolbox. For information about the parameters which are important for *dynopt*, see Table 3.4 or the individual function reference pages.

Table 3.3: Output arguments

Argument	Description
`x`	The solution found by the optimisation function. If exitflag > 0, then `x` is a solution otherwise, `x` is the value the optimisation routine was at when it terminated prematurely.
`fval`	The value of the objective function *objfun* at the solution `x`.
`exitflag`	The exit condition. > 0 indicates that the function converged to a solution `x`, 0 indicates that the maximum number of function evaluations or iterations was reached, < 0 indicates that the function did not converge to a solution.

Continued on next page

Table 3.3: concluded from previous page

Argument	Description
`output`	An output structure that contains information about the results of the optimisation. `output.iterations` gives the information about the number of iteration, `funcCount` gives the information about the number of function evaluations, `output.algorithm` returns the used algorithm, `output.stepsize` returns the taken final stepsize (medium-scale algorithm only), `output.firstorderopt` gives the information about a measure of first-order optimality (large-scale algorithm only).
`lambda`	The Lagrange multipliers at the solution `x`. `lambda` is a structure where each field is for a different constraint type. `lamdba.lower` for the lower bounds lb, `lambda.upper` for the upper bounds ub, `lambda.ineqlin` for the linear inequalities, `lambda.eqlin` for the linear equalities, `lambda.ineqnonlin` for the nonlinear inequalities, `lambda.eqnonlin` for the nonlinear equalities.
`grad`	The value of the gradient of objfun at the solution `x`.

Table 3.4: Optimisation options parameters

Parameter Name	Description
`DerivativeCheck`	Compare user-supplied analytic derivatives (gradients) to finite differencing derivatives (medium-scale algorithm only), default value: 'off'.
`Diagnostics`	Print diagnostic information about the function to be minimised or solved, default value: 'off'.
`DiffMaxChange`	Maximum change in variables for finite difference derivatives (medium-scale algorithm only), default value: 0.1000.
`DiffMinChange`	Minimum change in variables for finite difference derivatives (medium-scale algorithm only), default value: 1.0000e-008.
`Display`	Level of display. 'off' displays no output, 'iter' displays output at each iteration, 'final' displays just the final output, default value: 'final'.
`GradConstr`	Gradients for the nonlinear constraints defined by user, default value: 'off'.

Continued on next page

Table 3.4: concluded from previous page

Parameter Name	Description
`GradObj`	Gradient for the objective function defined by user, default value: 'off'.
LargeScale	User large-scale algorithm if possible, default value: 'on'.
`MaxFunEvals`	Maximum number of function evaluations allowed, default value: '100*numberofvariables'.
`MaxIter`	Maximum number of iterations allowed, default value: 400.
`TolCon`	Termination Tolerance on the constraint violation, default value: 1.0000e-006.
`TolFun`	Termination Tolerance on the function value, default value: 1.0000e-006.
`TolX`	Termination Tolerance on x, default value: 1.0000e-006.
`TypicalX`	Typical x values (large-scale algorithm only), default value: 'ones(numberofvariables,1)'.

Function parameters described in Tables 3.3, 3.4 are implicitly given by MATLAB Optimisation Toolbox for all it's subroutins. They also present just parameters usefull for *dynopt* through funciton *fmincon*.

3.2 Function Description

3.2.1 Purpose

The actual version of *dynopt* is able to solve dynamic optimisation problems which cost functions can be expressed by the Mayer form. The problem formulation can be described by following set of DAEs:

$$\min_{\boldsymbol{u}(t)} \mathcal{G}(\boldsymbol{x}(t_f), t_f)$$

such that

$$\begin{gathered}
\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\
\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\
\boldsymbol{h_f}(\boldsymbol{x}(t_f), \boldsymbol{u}(t_f)) = \boldsymbol{0} \\
\boldsymbol{g_f}(\boldsymbol{x}(t_f), \boldsymbol{u}(t_f)) \leq \boldsymbol{0} \\
\boldsymbol{x}(t)^L \leq \boldsymbol{x}(t) \leq \boldsymbol{x}(t)^U \\
\boldsymbol{u}(t)^L \leq \boldsymbol{u}(t) \leq \boldsymbol{u}(t)^U
\end{gathered}$$

where

$\mathcal{G}(\boldsymbol{x}(t_f), t_f)$ – component of objective function evaluated at final conditions,

$\boldsymbol{h_f}$ – equality design constraint vector at final time t_f,

$\boldsymbol{g_f}$ – inequality design constraint vector at final time t_f,

$\boldsymbol{x}(t)$ – state profile vector,

$\boldsymbol{u}(t)$ – control profile vector,

$\boldsymbol{x}_0$ – initial conditions for state vector,

$\boldsymbol{x}(t)^L, \boldsymbol{x}(t)^U$ – state profile bounds,

$\boldsymbol{u}(t)^L, \boldsymbol{u}(t)^U$ – control profile bounds.

3.2.2 Syntax

```
x = dynopt(ncol,delta_t,tf,uinit,bdx,bdu,bdt,objfun,confun,process,options)
[x,fval] = dynopt(...)
[x,fval,exitflag] = dynopt(...)
[x,fval,exitflag,output] = dynopt(...)
[x,fval,exitflag,output,lambda] = dynopt(...)
[x,fval,exitflag,output,lambda,grad] = dynopt(...)
```

3.2.3 Description

```
x = dynopt(ncol,delta_t,tf,uinit,bdx,bdu,bdt,objfun,confun,...
    process,options)
```

starts with the initial lengths of intervals (`delta_t` can be scalar or vector), initial control values for each interval (`uinit` is matrix) for defined collocation point number (`ncol` is scalar) and final time value (is scalar, set `tf` = [ ] if the final time is not defined), and minimises `objfun` evaluated in the final time t_f subject to the final time nonlinear inequalities or equalities defined in `confun` by given process model defined in `process` with the optimisation parameters specified in the structure `options`, the defined set of lower upper bounds on the state variables (`bdx` is matrix), control variables (`bdu` is matrix), and time (`bdt` is also matrix) so that the solution is always in the range of this bounds. Set `bdx` = [ ] and/or `bdu` = [ ] if no bounds exist.

```
[x,fval] = dynopt(...)
```

returns the value of the objective function *objfun* at the soluton `x`.

```
[x,fval,exitflag] = dynopt(...)
```

returns a value exitflag that describes the exit condition of *dynopt*.

```
[x,fval,exitflag,output] = dynopt(...)
```

returns a structure `output` with information about the optimisation.

```
[x,fval,exitflag,output,lambda] = dynopt(...)
```

returns a structure `lambda` whose fields contain the Lagrange multipliers at the solution `x`.

```
[x,fval,exitflag,output,lambda] = dynopt(...)
```

returns the value of the gradient of the objective function at the solution `x`.

3.2.4 Arguments

The arguments passed into the function are described in Table 3.2. The arguments returned by the function are described in Table 3.3. Details relevant to *dynopt* are included below for `objfun`, `confun`, `process`.

`objfun` The function to be minimised. `objfun` is a string containing the name of an M-file function, e.g., *objfun.m*. Whereas *dynopt* optimises a given performance index at final conditions, thus *objfun* takes a scalar `t` - final time t_f, scalar/vector `x` - state variable(s) evaluated at coresponding final time t_f, and scalar/vector `u` - control variable(s) evaluated at coresponding final time t_f, and returns a scalar value `f` of the objective function evaluated at these values. The M-file function has to have the following form:

```
function f = objfun (t,x,u)

f = [...];     % compute the function value at t, x, u
```

If the gradients of the objective function can also be computed and options.GradObj is 'on', as set by options = optimset('GradObj','on') then objfun is a string containing the name of an M-file function, e.g., *objfungrad.m*. The function *objfungrad* must return, in the second, third, and fourth output argument, the gradient value `Dft`, `Dfx`, `Dfu` at `t`, `x`, `u`.

```
function [f,Dft,Dfx,Dfu] = objfungrad (t,x,u)

f = [...];      % compute the function value at t, x, u
Dft = [...];    % compute the gradient evaluate at t
Dfx = [...];    % compute the gradient evaluate at x
Dfu = [...];    % compute the gradient evaluate at u
```

The gradients `Dft`, `Dfx`, `Dfu` are the partial derivatives of `f` at the points `t`, `x`, `u`. That means, `Dft` is the partial derivative of `f` with respect to the `t`, the ith component of `Dfx` is the partial derivative of `f` with respect to the ith component of `x`, the ith component of `Dfu` is the partial derivative of `f` with respect to the ith component of `u`.

`confun` The function that computes the nonlinear inequality constraints g(t,x,u)<=0 marked as output argument `c` and nonlinear equality constraints h(t,x,u)=0, marked as output argument `ceq` at the final conditions. `confun` is a string containing the name of an M-file function, e.g., *confun.m*. *confun* takes a scalar `t` - time value coresponding to the final time t_f, scalar/vector `x` - state variable value(s) corresponding to the value of `t`, and scalar/vector `u` - control variable value(s) coresponding to the value of `t`, and returns two arguments, a vector `c` of the nonlinear inequalities evaluated at `t`, `x`, `u`, and a vector `ceq` of the nonlinear equalities evaluated at `t`, `x`, `u`. For example, if confun='confun' then the M-file *confun.m* would have the form:

```
function [c,ceq] = confun(t,x,u)

c = [...];    % compute nonlinear inequalities at t, x, u
ceq = [...];  % compute nonlinear equalities at t, x, u
```

If the gradients of the constraints can also be computed and the `options.GradConstr` is 'on', as set by `options` = optimset('GradConstr','on') then `confun` is a string containing the name of an M-file function, e.g., *confungrad.m*. The function *confungrad* must return, in the second, third, ..., eighth output argument, the gradient value `Dct`, `Dcx`, `Dcu`, `Dceqt`, `Dceqx`, `Dcequ` at `t`, `x`, `u`.

```
function [c,ceq,Dct,Dcx,Dcu,Dceqt,Dceqx,Dcequ] = confungrad(t,x,u)

c = [...];       % compute nonlinear inequalities at t, x, u
ceq = [...];     % compute nonlinear equalities at t, x, u
Dct = [...];     % compute the gradient of inequalities at t
Dcx = [...];     % compute the gradient of inequalities at x
Dcu = [...];     % compute the gradient of inequalities at u
Dceqt = [...];   % compute the gradient of equalities at t
Dceqx = [...];   % compute the gradient of equalities at x
Dcequ = [...];   % compute the gradient of equalities at u
```

The gradients `Dct`, `Dcx`, `Dcu` are the partial derivatives of `c` at the points `t`, `x`, `u`. That means, `Dct` is the partial derivative of `c` with respect to `t`, the ith component of `Dcx` is the partial derivative of `c` with respect to the ith component of `x`, the ith component of `Dcu` is the partial derivative of `c` with respect to the ith component of `u`, and the gradients `Dceqt`, `Dceqx`, `Dcequ` are the partial derivatives of `c` at the points `t`, `x`, `u`.

`process` The function wich describes process model, that means the right hand sizes of differential equations. `process` is a string containing the name of an M-file function, e.g., *process.m*. *process* takes a time `t`, scalar/vector of state variable `x`, scalar `flag` and scalar/vector of control variable `u` coresponding to time `t` and returns `sys` values

according to `flag` value evaluated at time `t`. The M-file function has to be written in the following form:

```
function sys = process(t,x,flag,u)

switch flag,
    case 0
        sys = [...];    % right hand sizes of the ODE
    case 1
        sys = [...];    % jacobian with respect to x
    case 2
        sys = [...];    % jacobian with respect to u
    case 3
        sys = [...];    % jacobian with respect to t
    case 4
        sys = [...];    % initial values of state variables
    otherwise
        error(['unhandled flag = ',num2str(flag)]);
end
```

3.2.5 Algorithm

Large-scale optimisation By default *dynopt* will choose the large-scale algorithm if the user supplies the gradient in objfun (and `GradObj` is 'on' in `options`) and if only upper and lower bounds exists or only linear equality constraints exist. This algorithm is a subspace trust region method and is based on the interior-reflective Newton method described in [3]. Each iteration involves the approximate solution of a large linear system using the method of preconditioned conjugate gradients (PCG). See the trust-region and preconditioned conjugate gradient method descriptions in the Large-Scale Algorithms chapter in [2].

Medium-scale optimisation *dynopt* uses through the *fmincon* Sequential Programming (SQP) method. In this method, a Quadratic Programming (QP) subproblem is solved at each iteration. An estimate of the Hessian of the Lagrangian is updated at each iteration using the BFGS formula [3].

A line search is performed using a merit function similar to that proposed by [9]. The QP subproblem is solved using an active set strategy similar to that described in [7]. A full description of this algorithm is found in the Constrained optimisation section of the Introduction to algorithms chapter of the Optimization Toolbox manual. See also the SQP implementation section in the Introduction to Algorithms chapter for more details on the algorithm used.

3.3 Tutorial

This section discusses the *dynopt* application, definitions of input argument functions *process*, *objfun*, *confun* for numerically calculated gradients examples, and *objfungrad*, *confungrad* for analytically calculated gradients examples.

3.3.1 Constrained Example with Bounds

Consider the problem of starting and stopping a car in minimum time for a fixed distance (300 units). This problem was treated by [4, 10, 11]. The given system:

$$\dot{x}_1 = u, \quad x_1(0) = 1 \tag{3.1}$$

$$\dot{x}_2 = x_1, \quad x_2(0) = 0 \tag{3.2}$$

has to be optimised for $-2 \leq u(t) \leq 1$ with the cost function:

$$\min_{\boldsymbol{u}(t)} \; J = t_f \tag{3.3}$$

subject to the constraints:

$$x_1(t_f) = 0 \tag{3.4}$$

$$x_2(t_f) = 300 \tag{3.5}$$

$x_1(t),\ x_2(t)$ – state vectors representing speed and distance,

$u(t)$ – control vector representing acceleration.

Function *process*, *obfun*, *confun* Definitions

Problem (3.3) is described by two differential equations which together with the initial values of state variables should be defined in *process*.

Step1: Write an M-file *process.m*

```
function sys = process(t,x,flag,u)

switch flag,
    case 0
        sys = [u;x(1)];
    case 1
        sys = [0 1;0 0];
    case 2
        sys = [1 0];
    case 3
```

```
        sys = [0 0];
    case 4
        sys = [0;0];
    otherwise
        error(['unhandled flag = ',num2str(flag)]);
end
```

dynopt optimises a given performance index subject to the constraints, both evaluated at the final conditions. Thus the input arguments of *objfun* and *confun* are as follows: `t` - scalar value representing t_f, `x` - scalar/vector of state variable(s) evaluated at corresponding final time t_f, `u` - scalar/vector of control variable(s) evaluated at corresponding final time t_f. *objfun* should be defined as follows:

Step2: Write an M-file *objfun.m*

```
function f = objfun (t,x,u)

f=[t];
```

The given final time constraints should be written in M-file *confun* as follows:

Step3: Write an M-file *confun.m*

```
function [c,ceq] = confun(t,x,u)

c=[];
ceq=[x(1);
     x(2)-300];
```

Step4: Invoke *dynopt* After the problem has been defined in the functions, user has to invoke the *dynopt* function by writing an M-file *car1.m* as follows:

```
options = optimset('LargeScale','off','Display','iter');
options = optimset(options,'TolFun',1e-4);
options = optimset(options,'TolCon',1e-4);
options = optimset(options,'TolX',1e-4);
[x,fval,exitflag,output]=dynopt(2,[20;20],[],[0.8 -1],[],[-2 1],...
                 [1e-1 40],'objfun','confun','process',options);
```

For this example 2 collocation points, 2 intervals with initial lengths 20 units for each interval have been choosen. Final time t_f was given as optimised cost function variable by the problem definition so it has been set to [ ], the contol variable initial values were set to 0.8 for first time interval, and -1 for the second one. The state variable bounds have been set to [ ]. The control variable bounds have been set within given interval (-2,1). Lower

and upper bounds to the time interval lengths were set to 1e-1 and 40, the summation of given initial lengths.

279 function evaluations in 13 iterations were needed to obtain the solution:

```
x = 20.0000 10.0000 1.0000 1.0000 -2.0000 -2.0000 0 4.2265 15.7735 ...
    0 8.9316 124.4017 20.0000 15.7735 4.2265 200.0000 237.7992 ...
    295.5342]
```

The objective function at the solution $\boldsymbol{x}$ is returned in `fval`:

```
fval =
    30
```

The `exitflag` tells if the algorithm converged. An exitflag > 0 means a local minimum was found:

```
exitflag =
    1
```

More details about optimisation are given by the `output` structure. In this example, the default selection of the large-scale algorithm has been turned off, so the medium-scale algorithm is used. Also all termination tolerances have been changed. For more information about `options` and *dynopt* input and output arguments, see section 3.1.

In order to get from $\boldsymbol{x}$ plotable control and state variable profiles use additional function *makegraphs* as follows:

```
[time,state,control] = makegraphs(x,2,2,1,2)
```

The optimal control trajectory is shown in Figure 3.1. The actual state trajectories are shown in Figure 3.2 and Figure 3.3.

3.3.2 Constrained Example with Gradients and Bounds

A path constraint will be placed on the before defined problem (3.3) by setting un upper bound on the speed of 10 units. This example will be also solved by supplying analytical gradients. Ordinarily the medium-scale minimisation routines use numerical gradients calculated by finite-difference approximation. This procedure systematically perturbs each of the variables in order to calculate function and constraint partial derivatives. Alternatively, a function to compute partial derivatives analytically can be provided by the user. Typically, the problem is solved more accurately and efficiently if such a function is provided.

Function *process*, *obfungrad*, *confungrad* Definitions

Step1: Write an M-file *process.m* The M-file *process.m* remains without changes. It is the same as in section 3.3.1.

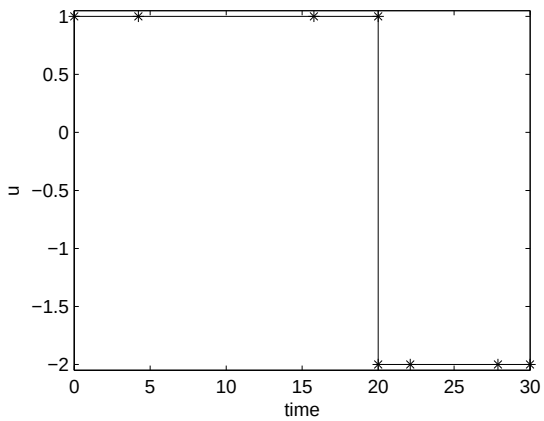

Figure 3.1: Control variable profile for example 1

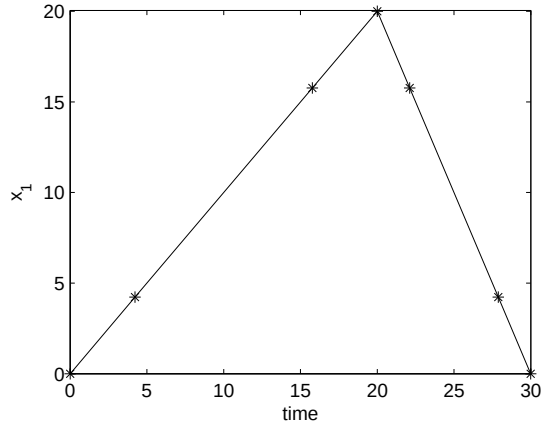

Figure 3.2: State1 variable profile for example 1

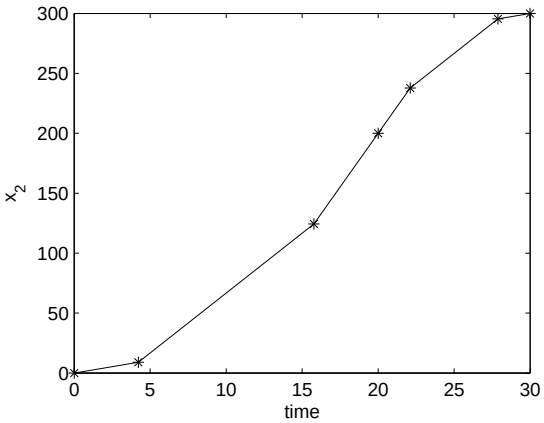

Figure 3.3: State2 variable profile for example 1

Step2: Write an M-file *objfungrad*

```
function [f,Dft,Dfx,Dfu] = objfungrad (t,x,u)

f=[t];
Dft=[1];
Dfx=[];
Dfu=[];
```

Since you are providing the gradients of the objective function in *objfungrad.m* and the gradients of the constraints in *confungrad.m*, you must tell *dynopt* that these M-files contain this additional information. Use optimset to turn the parameters `GradObj` and `GradConstr` to 'on' in our already existing options structure

```
options = optimset(options,'GradObj','on','GradConstr','on');
```

If you do not set these parameters to 'on' in the options structure, *dynopt* will not use the analytic gradients.

Step3: Write an M-file *confungrad.m*

```
function [c,ceq,Dct,Dcx,Dcu,Dceqt,Dceqx,Dcequ] = confungrad(t,x,u)

c=[];
ceq=[x(1);
     x(2)-300];
Dct=[];
Dcx=[];
Dcu=[];
Dceqt=[];
Dceqx=[1 0;0 1];
Dcequ=[];
```

Step4: Invoke *dynopt*

```
options = optimset('LargeScale','off','Display','iter');
options = optimset(options,'GradObj','on','GradConstr','on');
options = optimset (options,'TolFun',1e-4);
options = optimset (options,'TolCon',1e-4);
options = optimset (options,'TolX',1e-4);
[x,fval,exitflag,output]=dynopt(2,[10;22;10],[],[1 0 -1],[0 10;0 300],...
                 [-2 1],[1e-2 42],'objfungrad','confungrad','process',...
                 options);
```

For this example 2 collocation points, 3 intervals with initial lengths 10 units for first and last one, and 22 units for the second interval have been choosen. Final time t_f remains to be set to [ ], the contol variable initial values were set to 1 for first time interval, to 0 for the second interval, and -1 for the last one. The state variable bounds have been set to [0 10;0 300]. The control variable bounds remains the same as before. Lower and upper bounds to the time interval lengths were set to 1e-2 and 42, the summation of given initial lengths.

After 12 iteration and 52 function evaluatons, optimal value of $t_f = 37.5$ was found. The graphical interpretation of obtained optimal profiles is shown in Figure 3.4 for the control profile, in Figure 3.5 for the speed profile, and in Figure 3.6 for distance.

3.3.3 Maximisation

dynopt performes minimisation of the objective function $f(t, x, u)$. Maximisation is achieved by supplying the routine with $-f(t, x, u)$.

3.3.4 Greater than Zero Constraints

The Optimisation Toolbox assumes nonlinear inequality constraints are of the form $C_i(x) \leq 0$. Greater than zero constraints are expressed as less than zero constraints by multiplying

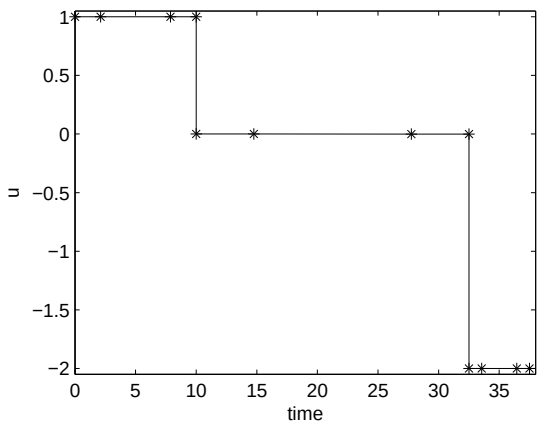

Figure 3.4: Control variable profile for example 2

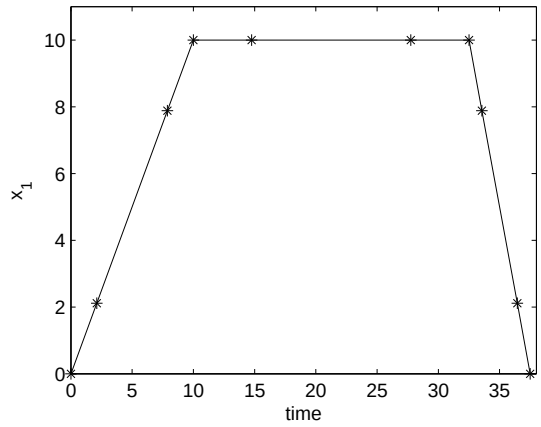

Figure 3.5: State1 variable profile for example 2

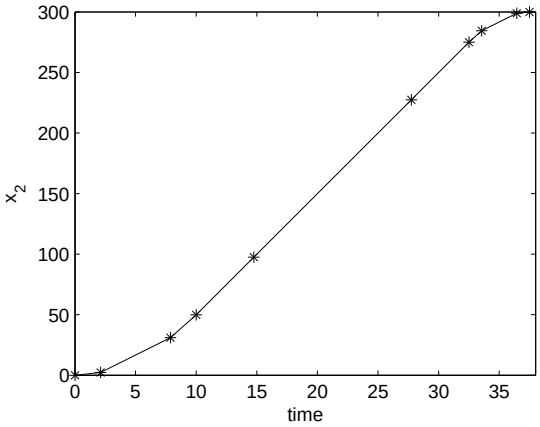

Figure 3.6: State2 variable profile for example 2

them by -1. For example, a constraint of the form $C_i(x) \geq 0$ is equivalent to the constraint $-C_i(x) \leq 0$.

Chapter 4

Case Studies

In this chapter the examples from literature solved by *dynopt* are presented and discussed. In all considered examples, 4 collocation points and 5 elements have been used to obtain accurate solution to the fourth decimal palce.

4.1 Example 1

4.1.1 Example 1a

Consider the following unconstrained problem [13, 16]

$$\min_{\boldsymbol{u}(t)} \; J = x_2(t_f) \tag{4.1}$$

such that

$$\begin{gathered} \dot{x}_1 = u, \quad x_1(0) = 1 \\ \dot{x}_2 = x_1^2 + u^2, \quad x_2(0) = 0 \\ t_f = 1 \end{gathered}$$

Problem (4.1) does not have any constraint and for this problem a minimum (0.76519) was determined by [13]. Another optimal value for the performance index was found by [16] (0.76238). The solution obtained by *dynopt* is 0.7616 by using numerical gradients and the same value of the performance index was obtained by user provided analitical gradients. As can be seen in Table 4.1, providing the gradients analytically greatly decreased the computing time. The control and state profiles are shown in Figure 4.1.

4.1.2 Example 1b

Consider the following constrained problem [13, 16]

	Numerical Gradients	Analytical Gradients
Optimal value	0.7616	0.7616
Number of iterations	43	45
Number of function evaluations	3570	353
CPU-time (s)	50.5730	18.6770

Table 4.1: Comparison between gradients calculated numerically and analytically for example 1a

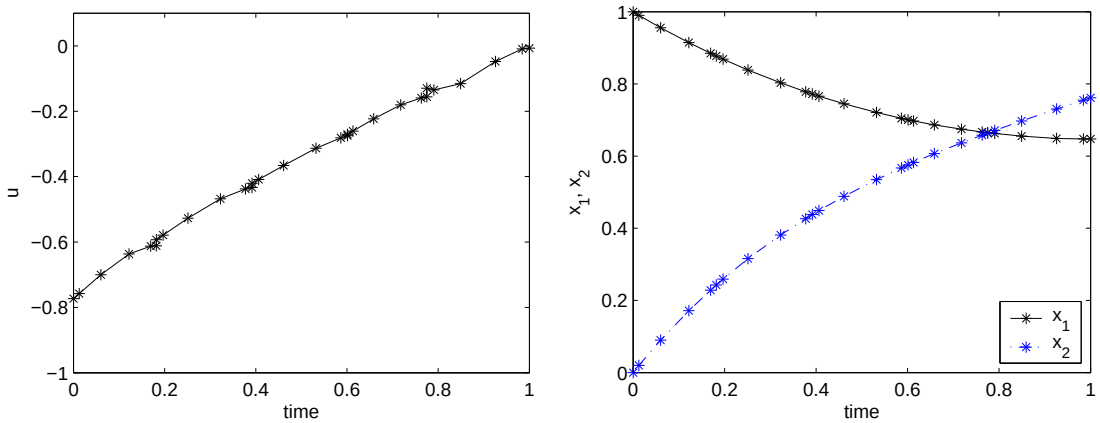

Figure 4.1: Control and state profiles obtained by *dynopt* for problem (4.1)

$$\min_{\boldsymbol{u}(t)} \; J = x_2(t_f) \tag{4.2}$$

such that

$$\begin{gathered} \dot{x}_1 = u, \quad x_1(0) = 1 \\ \dot{x}_2 = x_1^2 + u^2, \quad x_2(0) = 0 \\ x_1(1) = 0 \\ t_f = 1 \end{gathered}$$

The problem (4.2) differs from problem (4.1) with the terminal constraint. For this case the the optimal value of the performance index (0.92518) was obtained by [13] and another one (0.92547) was obtained by [16]. The solution obtained by *dynopt* in this case is shown in Table 4.2 and Figure 4.2. Also in this case the gradients computed analytically have forced the computation time.

4.2 Example 2

Consider the following nonlinear unconstrained problem [12, 16]

	Numerical Gradients	Analytical Gradients
Optimal value	0.9243	0.9243
Number of iterations	26	23
Number of function evaluations	2154	193
CPU-time (s)	32.0660	13.6790

Table 4.2: Comparison between gradients calculed numerically and analytically for example 1b

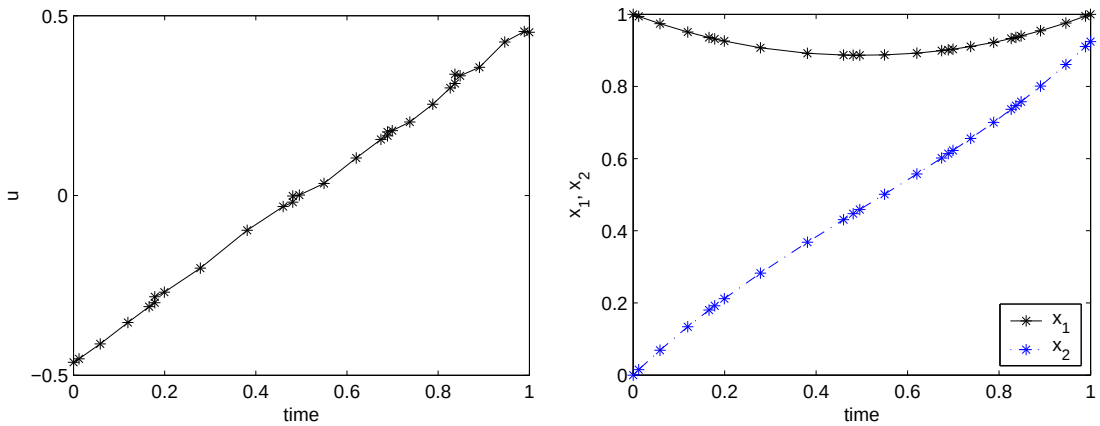

Figure 4.2: Control and state profiles obtained by *dynopt* for problem (4.2)

$$\min_{\boldsymbol{u}(t)} \; J = x_4(t_f) \tag{4.3}$$

such that

$$\begin{gathered}
\dot{x}_1 = x_2, \quad x_1(0) = 0 \\
\dot{x}_2 = -x_3 u + 16t - 8, \quad x_2(0) = -1 \\
\dot{x}_3 = u, \quad x_3(0) = -\sqrt{5} \\
\dot{x}_4 = x_1^2 + x_2^2 + 0.0005(x_2 + 16t - 8 - 0.1 x_3 u^2)^2, \quad x_4(0) = 0 \\
-4 \leq u \leq 10 \\
t_f = 1
\end{gathered}$$

Problem (4.3) is a four-state variable system treated by [12, 16]. For the state variable x_4, a value of the minimum (0.12011) was obtained by [12]. [16] computed the optimum of x_4 at final time (0.1290). With *dynopt* in this case we were able to reach the values of given performance index shown in Table 4.3 and Figure 4.3.

4.3 Example 3

Consider a tubular reactor with following parallel reaction [6, 10, 16]:

	Numerical Gradients	Analytical Gradients
Optimal value	0.1217	0.1212
Number of iterations	96	126
Number of function evaluations	12555	586
CPU-time (s)	239.5850	44.8450

Table 4.3: Comparison between gradients calculated numerically and analytically for example 2

$A \rightarrow B$
$A \rightarrow C$

$$\min_{\boldsymbol{u}(t)} \; J = -x_2(t_f) \tag{4.4}$$

such that

$$\begin{gathered} \dot{x}_1 = -[u + 0.5u^2]x_1, \quad x_1(0) = 1 \\ \dot{x}_2 = ux_1, \quad x_2(0) = 0 \\ 0 \leq u \leq 5 \\ t_f = 1 \end{gathered}$$

where

$x_1(t)$ – dimensionless concetration of A,

$x_2(t)$ – dimensionless concentration of B,

$u(t)$ – control vector

Problem (4.4) is a tubular reactor control problem where the state variable x_2 at final time has to be maximised. This problem was treated by [6, 10, 16] and the optimal value (0.57353) was reported by [6, 10], and the optimal value (0.57284) was given by [16]. Table 4.4 describes the results obtained by the orthogonal collocation on finite elements method used in *dynopt*, the optimal control and state profiles for this result are shown Figure 4.4.

	Numerical Gradients	Analytical Gradients
Optimal value	0.5727	0.5725
Number of iterations	74	68
Number of function evaluations	5951	295
CPU-time (s)	118.9810	24.4350

Table 4.4: Comparison between gradients calculated numerically and analytically for example 3

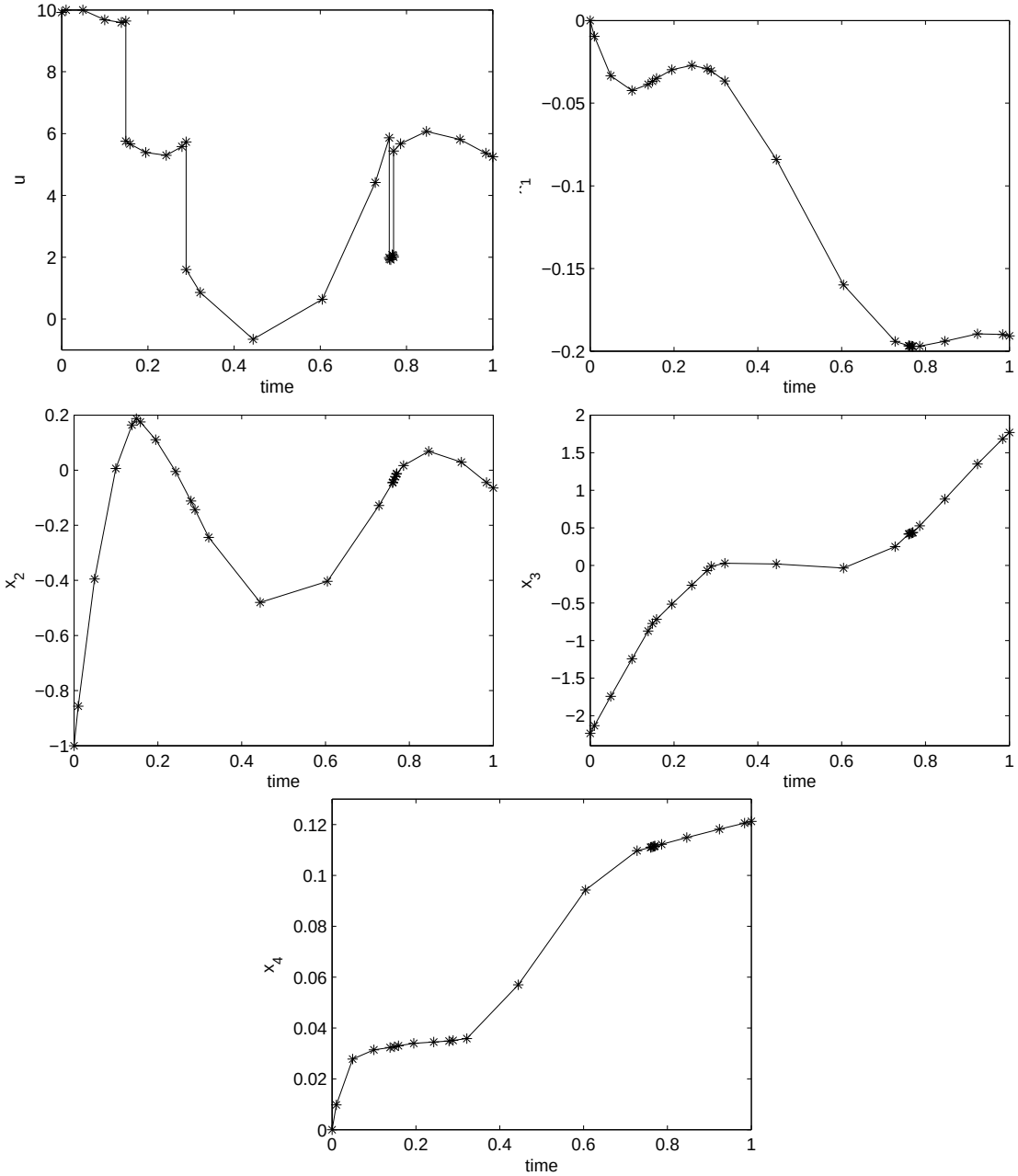

Figure 4.3: Control and state profiles obtained by *dynopt* for problem (4.3)

4.4 Example 4

Consider a batch reactor [6, 16] with the following consecutive reactions:
$A \rightarrow B \rightarrow C$

$$\min_{\boldsymbol{u}(t)} \; J = -x_2(t_f) \tag{4.5}$$

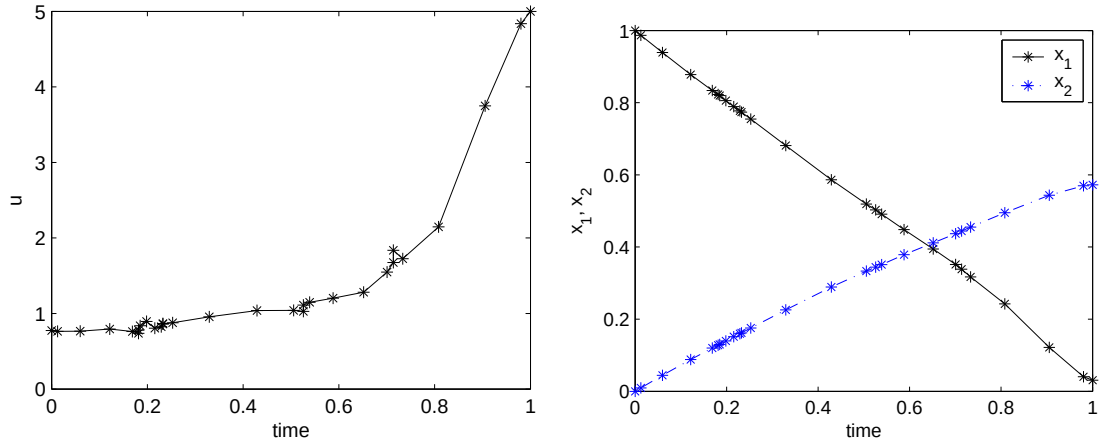

Figure 4.4: Control and state profiles obtained by *dynopt* for problem (4.4)

such that

$$\dot{x}_1 = -k_1 x_1^2 \quad x_1(0) = 1$$
$$\dot{x}_2 = k_1 x_1^2 - k_2 x_2 \quad x_2(0) = 0$$
$$k_1 = 4000 e^{(-\frac{2500}{T})}$$
$$k_2 = 620000 e^{(-\frac{5000}{T})}$$
$$298 \leq T \leq 398$$
$$t_f = 1$$

$x_1(t)$ – concentration of A,

$x_2(t)$ – concentration of B,

T – temperature.

The objective in problem (4.5) is to obtain the optimal temperature profile that maximizes x_2 at the end of a specified time. The problem was solved by [10, 16] and the reported optimum (0.610775) was found by [10] and (0.61045) obtained by [16]. We were able to obtain the value of performance index 0.6102 same for both, the numerically and analytically calculed gradients. As shown in Table 4.5 the computional time needed for obtaining the optimum was shorter by analytical gradients computing method. The appropriate optimal control and state profiles are presented in Figure 4.5.

4.5 Example 5

Consider a catalytic plug flow reactor [6, 16] with the following reactions:
$A \leftrightarrow B \rightarrow C$

	Numerical Gradients	Analytical Gradients
Optimal value	0.6102	0.6102
Number of iterations	9	10
Number of function evalutations	780	32
CPU-time (s)	13.0380	3.6950

Table 4.5: Comparison between gradients calculated numerically and analytically for example 4

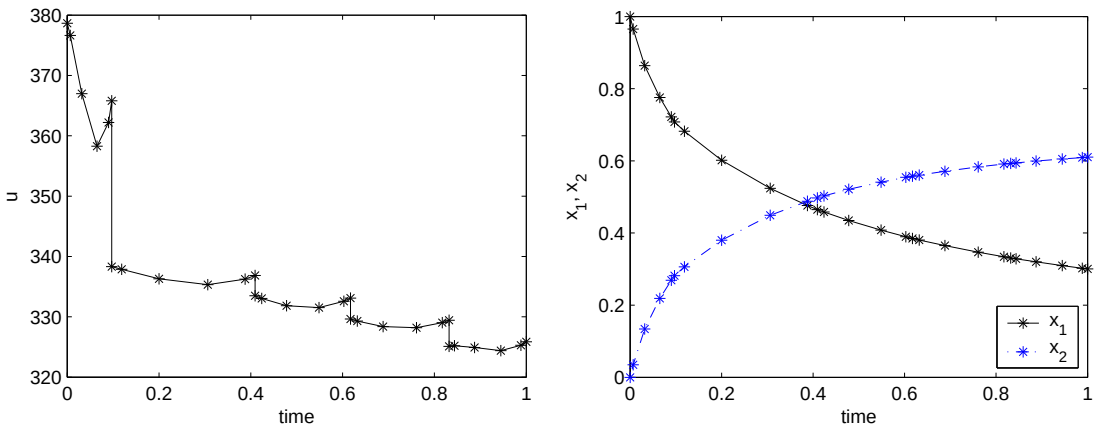

Figure 4.5: Control and state profiles obtained by *dynopt* for problem (4.5)

$$\max_{\boldsymbol{u}(t)} \; J = 1 - x_1(t_f) - x_2(t_f) \tag{4.6}$$

such that

$$\begin{gathered} \dot{x}_1 = u[10x_2 - x_1] \quad x_1(0) = 1 \\ \dot{x}_2 = -u[10x_2 - x_1] - [1-u]x_2 \quad x_2(0) = 0 \\ 0 \le u \le 1 \\ t_f = 12 \end{gathered}$$

$x_1(t)$ – mole fraction of A,

$x_2(t)$ – mole fraction of B,

$u(t)$ – fraction of type 1 catalyst.

Optimisation of problem (4.6) has also been analysed. This problem was solved by [10, 16] and the optima (0.476946, 0.47615) were found. Values obtained using *dynopt* are shown in Table 4.6. The coresponding control and state profiles are shown in Figure 4.6

Note, that all the results obtained by orthogonal collocation on finite elements method implemented within MATLAB-*dynopt* are only local in nature, since NLP solvers are only based on necessary conditions for optimality.

	Numerical Gradients	Analytical Gradients
Optimal value	0.4781	0.4781
Number of iterations	25	28
Number of function evaluations	2052	130
CPU-time (s)	34.0690	12.5780

Table 4.6: Comparison between gradients calculated numerically and analytically for example 5

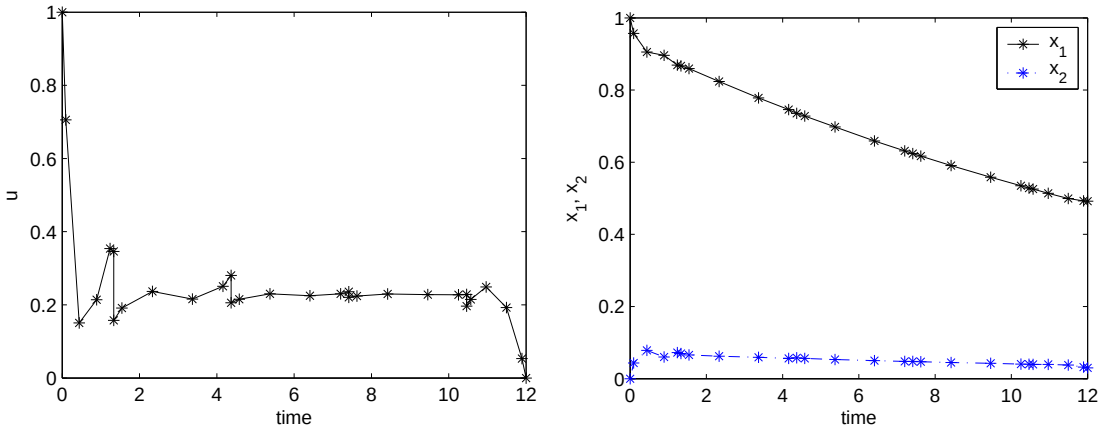

Figure 4.6: Control and state profiles obtained by *dynopt* for problem (4.6)

Chapter 5

Conclusions

The main aim of this work that was to implement a user friendly interface to dynamic optimisation based on orthogonal collocation on finite elements within the MATLAB environment. It was done by developing a MATLAB function *dynopt* for determination of optimal control trajectory by given description of the process, the cost to be minimised, subject to equality and inequality constraints. This function has been tested on 7 examples from the literature. The examples were choosen to ilustrate the ability of the dynopt package to treat the problems of varying levels of difficulty. In all the considered examples, two different methods of gradients computation were used: numerical approximation and analytical computation. The resulting performances of each method for the case studies are presented in Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6. As expected, they show that the performances of analytical computations are superior. On the other hand, as mentioned before, the optima obtained are local in nature.

The future work will be devoted to:

- enrich the knowledge already acquired until now by developing rigorous methods of dynamic and global optimisation of processes. In fact, it is well known for several years that solutions of dynamic optimisation in chemical engineering and control exhibit numerous local optima. Therefore, due to the non-convexity problems in chemical engineering, it is almost impossible to find the true global optimum and the results obtained are only local in nature. This sub-optimality can have serious consequences on economical, environmental operation, safety, and clearly show importance of the development of global optimisation methods.

- further develop the software *dynopt*, because the actual version is able to treat problems with performance index and constraints defined just at final time t_f. It means that the further development will be devoted to:

    1. create a version which will be able to treat a kind of problems that have performance index and constraints defined over a period of time $t \in [t_0, t_f]$,

2. improve its robustness and speed by incorporating public domain codes for automatic differentiation (e.g., Adifor, ...),
3. incorporate methods of global optimisation (e.g., $\alpha\beta\beta$ method, ...).

Bibliography

[1] A. E. Bryson and Y. Ho. *Applied Optimal Control*. Blaisdell Publishing Company, Massachussetts, 1969. 5, 6

[2] T. F. Coleman, M. A. Branch, and A. Grace. *Optimization Toolbox User's Guide*. MathWorks, Inc., 01 1999. 21

[3] T. F. Coleman and Y. Li. An interior, trust region approach for nonlinear minimization subject to bounds. *SIAM J. on Optimization*, 6:418–445, 1996. 21

[4] J. E. Cuthrell and L. T. Biegler. On the optimization of differential-algebraic process systems. *AIChE Journal*, 33:1257–1270, 1987. 2, 9, 22

[5] J. E. Cuthrell and L. T. Biegler. Simultaneous optimization and solution methods for batch reactor control profiles. *Computers chem. Engng.*, 13(1/2):49–62, 1989. 2

[6] S.A. Dadeo and K.B. McAuley. Dynamic optimization of constrained chemical engineering problems using dinamic programming. *Computers chem. Engng.*, 19:513–525, 1995. 30, 31, 32, 33

[7] P. E. Gill, W. Murray, and M. A. Wright. *Practical Optimization*. Academic Press, London, 1981. 21

[8] C. J. Goh and K. L. Teo. Control parametrization: A unified approach to optimal control problems with general constraints. *Automatica*, 24:3–18, 01 1988. 4, 8

[9] S. P. Han. A globally convergent method for nonlinear programming. 22:297, 1977. 21

[10] J. S. Logsdon and L. T. Biegler. Accurate solution of differential-algebraic optimization problems. *Chem. Eng. Sci.*, (28):1628–1639, 1989. 9, 22, 30, 31, 33, 34

[11] J. S. Logsdon and L. T. Biegler. Decomposition strategies for large-scale dynamic optimization problems. *Chem. Eng. Sci.*, 47(4):851–864, 1992. 9, 22

[12] R. Luus. Optimal control by dynamic programming using systematic reduction in grid size. *Int. J. Control*, 51:995–1013, 1990. 29, 30

[13] R. Luus. Application of iterative dynamic to state constrained optimal control problems. *Hung. J. Ind. Chem.*, 19:245–254, 1991. 28, 29

[14] J. Mikleš and V. Hutla. *Teória automatického riadenia (Theory of Automatic Control).* Alfa, 1986. 6

[15] C. P. Neuman and A. Sen. A suboptimal control algorithm for constrained problems using cubic splines. *Automatica*, 9:601–613, 1973. 2

[16] J. Rajesh, K. Gupta, H.S. Kusumakar, V.K. Jayaraman, and B.D. Kulkarni. Dynamic optimization of chemical processes using ant colony framework. *Computers and Chemistry*, 25:583–595, 2001. 28, 29, 30, 31, 32, 33, 34

[17] J. G. Renfro, A. M. Morshedi, and O. A. Asbjornsen. Simultaneous optimization and solution of systems described by differential/algebraic equations. *Computers chem. Engng.*, 11(5):503–517, 1987. 2

[18] R. W. H. Sargent and G. R. Sullivan. The development of an efficient optimal control package. 1977. 1

[19] T. H. Tsang, D. M. Himmelblau, and T. F. Edgar. Optimal control via collocation and non-linear programming. *Int. J. Control*, 21(5):763–768, 1975. 2

[20] M. Čižniar, M. Fikar, and M.A. Latifi. *User's Guide for MATLAB Dynamic Optimisation Code DYNOPT*, 05 2005. 13

[21] A. Víteček and M. Vítečková. *Optimální systémy řízení (Optimal Control Systems).* VŠT–Technicka Univerzita Ostrava, 2002. 5, 6, 7