Trajectory Optimization for Vehicles Using Control Vector Parameterization and Nonlinear Programming

Dr.ing. thesis

Inge Spangelo

Department of Engineering Cybernetics
The Norwegian Institute of Technology

1994

Report 94-111-W
Department of Engineering Cybernetics
The Norwegian Institute of Technology
N-7034 Trondheim, Norway

Preface

The present thesis is submitted in partial fulfillment of the requirements for the *Doktor ingeniør* degree at the Norwegian Institute of Technology (NTH). The work was carried out in the period from September 1989 to November 1994. Professor Olav Egeland, Department of Engineering Cybernetics, the Norwegian Institute of Technology, has been my supervisor.

I am very grateful to my supervisor Professor Olav Egeland for his good advices and guidance during the research period.

This work is a part of the MOBATEL research program at the Norwegian Institute of Technology. I am very grateful to Professor Jens Glad Balchen for stimulating discussions on the MOBATEL concept and on my own doctoral work. I will also acknowledge his good leadership of the MOBATEL research program which has successfully resulted in a number of *Doktor ingeniør* dissertations.

The first 24 months I worked as a research assistant at the Department of Engineering Cybernetics. In this period I participated in research activities on robot manipulator kinematics together with a fellow student J. Richard Sagli and Professor Olav Egeland. I appreciate this period of stimulating co-operation very much. The publication (Spangelo, Sagli & Egeland 1993) is a result of this work.

I am very grateful to Professor Thor I. Fossen for stimulating discussions and valuable advices concerning modelling of marine vehicles, and I am most grateful to him for proof reading the manuscript.

The last four years I have shared my office with my student-colleague and friend Ola-Erik Fjellstad. I am most grateful to him for his encouragement during all these years and for useful (and not so useful) discussions. I also appreciate his developed sense of humor which has been most helpful in periods of frustration and stress.

I am also very grateful to my friend and former student-colleague Gunleiv Skofteland for proof reading the manuscript.

I am most thankful to my colleagues and the staff at the Department of Engineering Cybernetics and SINTEF Automatic Control for all assistance and support.

I gratefully acknowledge

- NFR for my three year scholarship.

- Statoil, Saga, Simrad, Bentech, Seatex and Silicon Graphics for financial support and professional advice via the MOBATEL research program.
- The MOBATEL board for financial support to attend conferences in Nice, France, Sydney, Australia and Palo Alto, CA.
- *Norges Tekniske Høgskoles Fond* for financial support to attend a conference in Sacramento, CA.

Inge Spangelo

Trondheim, Norway

Summary

This thesis contains a study of optimal trajectories for vehicles. Highly constrained nonlinear optimal control problems have been solved numerically using control vector parameterization and nonlinear programming.

Control vector parameterization with shooting has been described in detail to provide the reader with the theoretical background for the methods which have been implemented, and which are not available in standard text books. Theoretical contributions on accuracy analysis and gradient computations have also been presented.

Optimal trajectories have been computed for underwater vehicles controlled in all six degrees of freedom by DC-motor driven thrusters. A class of nonlinear optimal control problems including energy-minimization, possibly combined with time minimization and obstacle avoidance, has been developed. A program system has been specially designed and written in the C language to solve this class of optimal control problems. Control vector parameterization with single shooting was used. This special implementation has made it possible to perform a detailed analysis, and to investigate numerical details of this class of optimization methods which would have been difficult using a general purpose CVP program system. The results show that this method for solving general optimal control problems is well suited for use in guidance and control of marine vehicles.

The use of numerical methods to solve general optimal control problems for marine vehicles is a new area of research. For other vehicles like aerospace vehicles, however, the use of advanced numerical methods in trajectory optimization has been an area of research for several decades. Results from rocket trajectory optimization has been studied in this work to bring knowledge from this area into the new area of trajectory optimization for marine vehicles. A case study of optimal ascent trajectories for the Ariane 5 rocket has been performed using the general purpose CVP program system PROMIS. The contributions here have been on the formulation of the problem and the study of the resulting trajectories.

Contents

Nomenclature

Abbreviations

AUV	Autonomous Underwater Vehicle
CVP	Control Vector Parameterization
DOF	Degree Of Freedom
MOBATEL	MOdel BAsed TELeoperation
NLP	NonLinear Programming (problem)
OCP	Optimal Control Problem
QP	Quadratic Programming
SVP	State Vector Parameterization
SQP	Sequential Quadratic Programming

Notation

Let $\mathbf{x} = \left(\; x_1, ..., x_n \; \right)^{\mathrm{T}}$ be a n-dimensional vector, and let l be a real valued function of x. Then, the gradient of l with respect to $\mathbf{x}$ is defined by

$$\frac{\partial l(\mathbf{x})}{\partial \mathbf{x}} = \left(\; \frac{\partial l(\mathbf{x})}{\partial x_1} \quad . \quad . \quad . \quad \frac{\partial l(\mathbf{x})}{\partial x_n} \; \right)$$

Let $\mathbf{f}(\mathbf{x}) = \left(\; f_1(\mathbf{x}), ..., f_n(\mathbf{x}) \; \right)^{\mathrm{T}}$ be a m-dimensional vector valued function of $\mathbf{x}$, then the Jacobian of $\mathbf{f}$ is defined by

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_1} & . & . & . & \frac{\partial \mathbf{f}_1(\mathbf{x})}{\partial x_n} \\ . & & . & & . \\ . & & . & & . \\ . & & . & & . \\ \frac{\partial \mathbf{f}_n(\mathbf{x})}{\partial x_1} & . & . & . & \frac{\partial \mathbf{f}_n(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

The Measure of Performance

The term **objective functional** is used as the measure of performance in an OCP. The objective functional is to be minimized with respect to variables in an infinite dimensional function space.

The term **objective function** is used as the measure of performance in an NLP. The objective function is to be minimized with respect to variables in a finite dimensional euclidean space. When the control vector is parameterized the objective functional J_0 becomes an objective function J.

J_0	Objective functional of an OCP
J	Objective function, resulting from J_0 when the control vector is parameterized
Jf	Intermediate value of J printed from the optimizer
J^*	Final (converged) value of the objective function
$F(\mathbf{p})$	The objective function of an NLP

J and $F(\mathbf{p})$ are therefore used alternately to denote the objective function.

Frequently Used Mathematical Symbols

It should be noted that some local definitions may contradict the definitions below.

Latin symbols

$\mathbf{A}_{GL}$	Matrix defining the general linear constraints of the QP
$\mathbf{A}_a$	Matrix defining active constraints in the QP
$\mathcal{A}$	Kuhn-Tucker matrix
$\mathbf{B}$	Control input matrix
$\tilde{\mathbf{B}}$	Thruster configuration matrix
$\mathbf{c}(\cdot)$	Vector of (nonlinear) constraints in the NLP
$\mathbf{C}(\cdot)$	Jacobian matrix of $\mathbf{c}(\cdot)$ in the NLP, or matrix defining Coriolis and centripetal terms
$\mathbf{c}_E(\cdot)$	Vector of (nonlinear) equality constraints in the NLP
$\mathbf{c}_I(\cdot)$	Vector of (nonlinear) inequality constraints in the NLP
$\mathbf{D}$	Matrix of dissipative terms
$\mathbf{d}$	Search direction in the NLP and vector of variables in the QP
$\mathbf{d}_z$	Defining null-space part of $\mathbf{d}_z$
$\mathbf{d}_y$	Defining column-space part of $\mathbf{d}_z$
$\delta\mathbf{d}$	Search direction in the QP

$\mathbf{f}(\cdot)$	Vector of right hand side functions in the state space equations
$F(\cdot)$	The objective function of the NLP
$\mathcal{F}(\mathbf{p})$	Function in NLP, objective or constraint, resulting from OCP
$\bar{\mathcal{F}}(\mathbf{p})$	Discretized $\mathcal{F}(\mathbf{p})$
f_{IM}	Numerical integration method of order p_{oI}
$\mathbf{g}$	Transpose of objective function gradient in the NLP, vector of restoring forces and moments
$\mathbf{g}_I(\cdot) \in \mathbf{R}^r$	Path inequality constraints in the OCP
$\mathbf{H}$	Quasi Newton update of the Hessian matrix
$\mathcal{H}(\cdot)$	Hamiltonian function
h_{seq}	Sequence of integration step sizes $h_1, ..., h_{n_s}$
$\mathbf{J}(\mathbf{x}_1)$	Jacobi matrix defining the velocity transformation
$L(\cdot)$	Lagrangian term in the objective functional
$\mathcal{L}(\cdot)$	Lagrangian term in $\mathcal{F}(\mathbf{p})$
$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$	Lagrangian function of the NLP
$M(\cdot)$	Merit function
$\mathbf{M}$	Inertia matrix
$\mathbf{p}$	Vector of optimizable parameters
$\mathbf{Q}$	Transformation matrix, TQ-factorization
$\mathbf{R}$	Choleskey-factor matrix
$\mathbf{R}_{x,\phi}$	Rotation matrix
$\mathbf{T}$	T-matrix in the TQ-factorization
t	Time variable
t_f	Final time j
$t_j(t_{i,j})$	Control time node number j (for control number i)
$\mathbf{u}(t)$	Vector of control variables
$\mathbf{x} \in \mathbf{R}^n$	Vector of optimizable parameters of the NLP
$\mathbf{x}(t)$	Vector of state variables, the state vector
$\mathbf{x}_1(t) = \boldsymbol{\xi}$	Vector of global position and orientation parameters
$\mathbf{x}_2(t) = \boldsymbol{\nu}$	Vector of velocities
$\mathbf{x}_0$	Initial value of the state vector
$\mathbf{x}_f$	Final value of the state vector
$\mathbf{Y}$	Column-space matrix for the active constraints
$\mathbf{Z}$	Null-space matrix for the active constraints
V	Velocity
$\mathbf{W}(\mathbf{x})$	Hessian matrix of the objective function
w	Weighting of t_f in the OCP

Greek symbols

α^k	Step size in iteration number k of the NLP
α_i	Weighting factors in the OCP
$\boldsymbol{\lambda}$	Vector of Lagrange multipliers of the NLP
ρ	Penalty parameter in the NLP, density of the water
$\boldsymbol{\mu}$	Vector of Lagrange multipliers of the QP
σ	Step size in the QP
$\phi(\alpha, \rho)$	Merit function as a function of α
ε_i	Local error coefficient for $\mathcal{F}(\mathbf{p})$
ε_i^M	Local error coefficient for $\mathcal{F}_M(\mathbf{p})$
δ_j	Forward difference perturbation of p_j
ϵ_{FD}	Error in the forward difference approximation of the partial derivatives of $\mathcal{F}(\mathbf{p})$
$\bar{\epsilon}_{FD}$	Error in the forward difference approximation of the partial derivatives of $\bar{\mathcal{F}}(\mathbf{p})$
$\boldsymbol{\tau}$	Vector of forces and moments
$\phi(\cdot)$	Mayer term in the objective functional
$\varphi(\cdot)$	Mayer term in $\mathcal{F}(\mathbf{p})$

Miscellaneous

$\bar{(\cdot)}$	Usually, updated vector, matrix or parameter
$(\cdot)_{FR}$	Corresponds to free variables in the QP
$(\cdot)_{FX}$	Corresponds to fixed variables in the QP
$(\cdot)^*$	Optimal/converged value of
$(\cdot)_e$	Extended vector

Chapter 1

Introduction

Optimal control is important in a large number of applications, and successful implementations have been reported in the literature. In particular the well-known linear quadratic optimal control problem has found wide acceptance, mainly due to the existence of an analytic solution based on the Riccati equations.

It is clear, however, that the full potential of optimal control has so far been difficult to utilize. This is mainly due to the lack of reliable an easily implementable numerical algorithms which are necessary for general optimal control problems (OCP) with nonlinear dynamics, path constraints and nonquadratic objective functionals. In addition the available computer hardware has been limited.

The concept of vehicle contains a large class of mechanical systems. Included in the concept are moving objects like airplanes, rockets, surface vehicles, ships and submarines. For economical and environmental reasons, energy saving is a very important aspect when vehicles are considered. Many types of vehicles can only use a limited amount of energy to carry out their missions, because new supply of energy during the mission is impossible, or at least to expensive to be realistic. Typically, the size of a mission will then be limited by the amount of energy that the vehicle can carry. The maximum size of a mission that a vehicle can accomplish will be referred to as the operation radius of the vehicle. If a certain number of tasks or a certain amount of work is to be carried out by a vehicle, the number of missions that are necessary will depend on the operation radius of the vehicle. An example is the use of small underwater vehicles for offshore activities. In certain applications like operation in a cluttered environment it may be advantageous that the vehicle is untethered, and the vehicle has to be energy autonomous. It is of great economical interest that the number of tasks that the vehicle can carry out in one mission is as large as possible, while at the same time the time consumption of the mission is reasonable.

The operation radius of a vehicle can be increased significantly by using optimal control to reduce the energy consumption. Generally the resulting OCP formulated for energy minimization of a vehicle includes nonlinear equations of motion and a nonquadratic objective functional. Path constraints may also be included for

instancte to handle obstacle avoidance. Generally a numerical method has to be applied for such problems.

The solution proposed in this thesis is to use control vector parameterization (CVP) and nonlinear nonlinear programming (NLP) to compute optimal trajectories for vehicles. The use of CVP and NLP for solving trajectory optimization problems is very efficient and flexible. This class of methods has become particularly popular the last fifteen years due to the development of efficient nonlinear programming problem solvers. The CVP method preferred in this thesis is CVP with shooting, which is believed to be the most powerful for dynamical systems described by nonstiff differential equations. Payload maximization for launch vehicles and other trajectory optimization problems for aerospace applications have been among the driving applications for developing more efficient and flexible numerical methods for solving general nonlinear OCP's for the last 30 to 40 years. Some references are given in Chapters 2 and 5.

The main motivation for the present work was to use numerical methods to compute optimal trajectories for marine vehicles. This is a new application of optimal trajectory generation, which is of considerable importance in offshore and maritime activities. Rather than considering all kinds of vehicles in a general optimal control concept, optimal trajectories are computed for selected case studies, see Sections 1.4 **Outline of the Thesis**.

Other interesting applications of optimal control to mechanical systems are robot manipulators and mechanical structures. The use of CVP and NLP for solving trajectory optimization problems for robot manipulators are considered in (Lunde 1991), and for mechanical structures in e.g. (Tseng & Arora 1989).

1.1 Optimal Control

For more complex OCP's including nonlinear dynamics, nonquadratic objective functionals, and path constraints, methods for finding analytical solutions are not known. In these cases the optimal trajectory has to be computed numerically. Only an approximation of the off-line trajectory is found, and if the optimal feedback solution is to be found the second variation of the optimal control problem must be considered, see (Bryson & Ho 1975). The trajectory generation and the feedback design of a control system can therefore be considered as two separate problems. It is of course possible to generate the predescribed trajectories optimally, and to use a feedback controller based on other design criteria, for instance robust stability. In the state space predictive control schemes for process control (Balchen, Ljungquist & Strand 1992) the whole control system is considered, including both trajectory generation and feedback control. The optimal trajectories are computed numerically for a certain time interval into the future (Strand 1991).

Since optimal trajectory generation and feedback control are in fact two separate problems the computation of optimal trajectories has mainly been treated separately

from other parts of the control system. This is particularly the case in aerospace applications. It should, however, be emphasized that because of model uncertainties and unknown external disturbances, optimal trajectories must be used together with feedback control in order to get a robustly stable guidance and control scheme.

Trajectory optimization has also been treated separately from feedback control because it is computationally expensive, and consequently it is usually impossible to include on-line optimal trajectory generation in a control scheme. Therefore optimal trajectories are mainly computed off-line, and used as references for the whole mission to be executed, without the valuable possibility of online recomputations.

This thesis is mainly concerned with trajectory optimization alone. Efficiency in terms of computation speed is, however, emphasized with the object in mind of using the optimal trajectories in a guidance scheme. Computation speed versus the optimization and integration accuracies is treated comprehensively. This is the main object of concern when computationally expensive numerical optimal control solutions are to be used in real time guidance and control systems. The use of trajectory optimization in guidance and control schemes is briefly discussed in Chapter 6 on **Conclusions and Recommendations**.

1.2 The MOBATEL Program

The doctoral work reported in this thesis is a part of the MOBATEL (MOdel BAsed TELeoperation) program. The MOBATEL program is mainly concerned with model based teleoperation of an unmanned untethered underwater vehicle. An untethered underwater vehicle is energy-autonomous. The larger part of the case studies in this thesis is therefore concerned with computing optimal trajectories for autonomous underwater vehicles. An energy-autonomous vehicle will typically be battery powered, and the operation radius will be limited by the amount of energy that the vehicle can carry on board. The use of energy-optimal trajectories can therefore extend the period of operation significantly.

In MOBATEL the vehicle is intended to be remotely operated using images of the vehicle and its environment. The images are based on camera and sensor information from the vehicle. The information between the vehicle and the operator is transferred over an acoustic communication link. Because of the time delay, and possibly low bandwidth in the communication link, real time information cannot be obtained from the cameras and the sensors alone. To be able to operate the vehicle in real time, the present states of the vehicles and its environments have to be predicted. Therefore, the images are generated from mathematical models of the vehicle and its environment, and the camera and sensor information is used to update the mathematical models.

In the MOBATEL program the operators define the tasks to be performed by the vehicle using the model generated images and a control panel as man-machine interface. The role of the optimal trajectory generator in this concept will typically

be to transform the operator commanded signals into control input signals for the vehicle in an optimal way. To what degree the operator defines the task and to what degree it should be left to the optimizer is a matter of discussion.

The MOBATEL program is further described by Balchen & Fossen (1992).

1.3 The contributions of the Thesis

The contributions of the thesis are:

- The main contributions of the thesis, which are presented in Chapter 4, are on the computations of optimal trajectories for underwater vehicles controlled by DC-motor driven thrusters. Section 1.4, "Outline of the Thesis", is referred for more details concerning the contributions in Chapter 4.
  - A general class of OCP's are proposed in Section 4.2, including minimization of energy and time consumption and collision avoidance. In this connection an expression for the energy consumption for the DC-motor driven thrusters is derived.
  - Energy optimal trajectories for a 6 DOF vehicle, including collision avoidance, are computed efficiently.
  - A program system in the language C has been implemented using CVP with single shooting. This makes it possible to obtain the necessary numerical results to perform a detailed quantitative analysis of the computed optimal trajectories. For this analysis computed optimal trajectories for underwater vehicles with varying degrees of freedom are also presented.
- A systematic presentation of existing CVP with shooting methods is given in Chapter 2.2, which is not available in standard text books. The presentation is intended to provide a basis for the implementation and for applications to new problems. In addition, modern CVP methods are related to the early contributions the sixties.
- In Chapter 2.3 a new accuracy analysis for objective and constraint functions depending on differential equations, which are used in CVP with shooting, is presented. The analysis is used to conclude that the convergence accuracy in the resulting NLP is not limited by the integration accuracy of the differential equations.
- The equivalence of two gradient calculation approaches are shown, partly by the use of the new accuracy analysis. The two gradient calculation schemes are the numerical approach using forward differences and a semi-analytical approach using state sensitivity equations. Some details concerning the implementation of the state sensitivity approach is proposed, partly based on

the new accuracy analysis. In addition, the analysis explains why these two methods are more accurate than the semi-analytical approach using costate equations. (Also in Chapter 2.3.)

- A comprehensive description of the SQP method is presented in Chapter 2.4. Algorithmic details, based on several publications and technical reports and which are difficult to collect from the literature, are presented.
- A four-phase OCP with singularity avoidance for the ascent of the three stage rocket Ariane 5 is proposed in Chapter 5.3.
- New results on the combination of heat-flux and splash-down constraints in rocket ascent, based on computed optimal trajectories, are presented in Chapter 5.4

1.4 Outline of the Thesis

The remaining chapters of the thesis are organized as follows:

Chapter 2 gives a detailed description of the numerical solution of optimal control problems using control vector parameterization with shooting and nonlinear programming. A nonlinear and highly constrained class of optimal control problems are presented, and the numerical solution of this class of problems is outlined. Function and gradient computation in CVP with shooting is treated in detail, with emphasis on the accuracy analysis. In the last section nonlinear programming is discussed, and a detailed description of the sequential quadratic programming methods are given.

Chapter 3 first presents kinematic and dynamic equations of motion of a six degrees of freedom marine vehicle, e.g. an underwater vehicle. Then, models of DC-motor driven thrusters are discussed, including propeller characteristics.

Chapter 4 describes a detailed treatment of trajectory optimization for underwater vehicles.

In Section 4.2 a general class of optimal control problems for underwater vehicles controlled by DC-motor driven thrusters is formulated, including the minimization of energy and time consumption. Obstacle avoidance is included by including path constraints.

In Section 4.3 gradient computations using forward differences and the adjoint equations are compared on a 6 DOF vehicle model for a very simple task. The forward difference approach is preferred, and is used for the numerical solutions reported in the remaining sections of the chapter.

An implementation of a program system in the C language for solving OCP's using CVP with single shooting, specially designed for solving this kind of

trajectory optimization problems, is reported. The program system uses the sequential quadratic programming (SQP) subroutine E04VCF from the NAG Fortran library (NAG 1991). Choices of parameters and other implementation details are based on results from solving a simple 2 DOF benchmark problem. This is reported in detail in Section 4.4.

In Section 4.5 numerical solutions of trajectory optimization for a 6 DOF vehicle are presented. Several tasks were solved for different numbers of control parameters. The results are analyzed with respect to accuracy and CPU-time consumption. It is shown that collision avoidance, both with fixed and free final time for the task, can be solved efficiently by using trajectory optimization.

In Section 4.6 numerical solutions of trajectory optimization for reduced versions of the 6 DOF underwater vehicle model are presented. The collision avoidance problem was solved for a 5 DOF model and a 4 DOF model of the vehicle. It is shown that in order to solve the optimal control problem even more efficiently, it can sometimes be appropriate to fix the roll and pitch angles and the angular velocities around the axis x-axis and the y-axis.

Chapter 5 presents optimal trajectories for rocket ascent. Payload maximization is formulated as a four-phase optimal control problem is presented, including heat-flux and splash-down constraints. Numerical solutions of the optimal control problem are presented.

Chapter 6 contains the conclusions of the thesis and recommendations for future work.

Appendix A presents numerical results from using the state sensitivity approach for analytical gradient computations in CVP with single shooting. The method is tested on a small underwater vehicle in the horizontal plane, which is represented by a non-holonomic model.

Chapter 2

Trajectory Optimization

2.1 Introduction

In this chapter a class of general nonlinear optimal control problems are stated, and there is a detailed treatment of a class of numerical methods for solving such problems using nonlinear programming (NLP).

The main purpose of this chapter is to give a comprehensive background for the numerical implementations in later chapters, and for the description of numerical implementations. Further, this chapter is used for easy reference for details which are not found in textbooks. In addition, theoretical contributions are reported in Section 2.3.

More general problems than those stated are of course possible, but this should not be considered to limit the applicability of the methods discussed. Rather, the optimal control problem statement is restricted for the sake of clarity in the presentation.

The numerical methods presented do not need the necessary conditions for optimality. Therefore, a section on optimal control theory is not included. An extensive literature in this field exists.

In the next section a comprehensive historical review of methods for solving optimal control problems using NLP are given. This is done because it is felt that the literature lacks a historical review where modern CVP methods are related to the early contributions of the sixties. Other methods are mentioned briefly. Modern methods for solving optimal control problems can be divided in two main subgroups, control vector parameterization (CVP) with shooting, and control vector parameterization (CVP) with state vector parameterization (SVP). Both will be referred to as CVP methods. After the historical review the process of solving an optimal control problem using a general CVP method is divided in three; The parameterization scheme leading to a nonlinear program, the computation of the objective function, the constraint functions and their gradients, and finally the solution of the

nonlinear program. In the remainder of the next section parameterization schemes in the CVP with shooting class of methods are discussed. CVP with SVP is not discussed any further.

In Section 2.3 the computation of functions (objective and constraint) and gradients are treated. A generalized function is presented and methods for solving the differential equations numerically are discussed. An accuracy analysis for functions depending on differential equations, which is formalized in the context of optimization, is presented here for the first time. Three different methods for computation of gradients are presented; A forward difference scheme for numerical computation, a state sensitivity approach for semi-analytical direct computation, and a semi-analytical approach using the costate equations. The first two methods are treated in the context of the new accuracy analysis, and the equivalence of the two methods is shown. The semi-analytical methods will be referred to as analytical methods.

In Section 2.4 methods for solving NLP problems are discussed. The sequential quadratic programming (SQP) methods are treated in detail. Finally, methods for solving large and sparse problems are discussed. Section 2.4 is equally relevant for shooting and SVP methods. This section is based on several textbooks and a large collection of international publications and technical reports. The comprehensive description of the SQP method is mainly based on publications and technical reports with authors from the Stanford Optimization Library (SOL), and is written to provide the readers with algorithmic details which are not available in any text books, and difficult to collect from the literature. These algorithmic details and the practical aspects of optimization which are discussed are important for the numerical implementation of the CVP method.

2.1.1 Optimal Control Problem Statement

Consider a nonlinear dynamical system described by the following state space equations

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t), \mathbf{p}_d) \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{2.1}$$

where $\mathbf{x}(t) \in \mathbf{R}^n$ is the state vector, $\mathbf{u}(t) \in \mathbf{R}^m$ is the control vector, $\mathbf{p}_d \in \mathbf{R}^{n_d}$ is a vector of design parameters and $\mathbf{x}_0 \in \mathbf{R}^n$ is the value of the initial state vector. The controls are assumed to be piecewise continuous functions of time while $\mathbf{f}$ is continuously differentiable with respect to all its arguments. Without loss of generality the initial time is set to 0.

Consider the following class of Optimal Control Problems (OCP):

$$\min_{\mathbf{u}, \mathbf{p}_{d,o}} J_0 = \phi(\mathbf{x}(t_f), \mathbf{p}_{d,o}) + \int_0^{t_f} L(t, \mathbf{x}(t), \mathbf{u}(t), \mathbf{p}_d) dt \tag{2.2}$$

for the system (2.1). The scalar functions ϕ and L are continuously differentiable with respect to all its arguments. The vector $\mathbf{p}_{d,o}$ contains the design parameters. In addition it contains the final time if it is free, and any free initial states. J_0 is termed the objective functional, whereas ϕ will be referred to as the Mayer term and L will be referred to as the Lagrangian term in this thesis.

The final time t_f can be either fixed or free.

The elements of the initial state vector $\mathbf{x}_0$ can also be fixed or free. If any of the initial states are free, boundary constraints may be defined according to

$$\boldsymbol{\psi}_E^0(\mathbf{x}(0)) = \mathbf{0} \tag{2.3}$$

$$\boldsymbol{\psi}_I^0(\mathbf{x}(0)) \geq \mathbf{0} \tag{2.4}$$

where the vector $\boldsymbol{\psi}_E^0$ is p_{0_E}-dimensional and the vector $\boldsymbol{\psi}_I^0$ is p_{0_I}-dimensional. Note that $p_{0_E} < n - \tilde{n}$ where $\tilde{n}$ is the number of fixed initial states.

Final time boundary constraints can be defined in a similar way, according to

$$\boldsymbol{\psi}_E^f(t_f, \mathbf{x}(t_f)) = \mathbf{0} \tag{2.5}$$

$$\boldsymbol{\psi}_I^f(t_f, \mathbf{x}(t_f)) \geq \mathbf{0} \tag{2.6}$$

where the vector $\boldsymbol{\psi}_E^f$ is p_{f_E}-dimensional and the vector $\boldsymbol{\psi}_I^f$ is p_{f_I}-dimensional. Note that $p_{f_E} \leq n$.

Constraints on the design parameters can be written

$$\boldsymbol{\omega}_I(\mathbf{p}_d) \geq \mathbf{0} \tag{2.7}$$

where $\boldsymbol{\omega}_I$ is p_p-dimensional.

Path inequality constraints can be written in the form

$$\mathbf{g}_I(t, \mathbf{x}(t), \mathbf{u}(t), \mathbf{p}_d) \geq \mathbf{0} \tag{2.8}$$

where $\mathbf{g}_I$ is r dimensional.

The vector valued functions $\boldsymbol{\psi}_E^0$, $\boldsymbol{\psi}_I^0$, $\boldsymbol{\psi}_E^f$, $\boldsymbol{\psi}_I^f$ and $\mathbf{g}_I$ are assumed to be continuously differentiable with respect to all their arguments.

In addition to the nonlinear constraints simple box constraints on the controls and the design parameters may be present:

$$u_{i,min} \leq u_i(t) \leq u_{i,max}, \quad (i = 1, .., m) \tag{2.9}$$

$$p_{d,i,min} \leq p_{d,i} \leq p_{d,i,max}, \quad (i = 1, .., n_d) \tag{2.10}$$

These are special cases of Eqs. (2.7) and (2.8) respectively.

Necessary conditions for optimality for OCP's are given in e.g. (Bryson & Ho 1975) and will not be presented here.

Multi-Phase Systems

A multi-phase system is characterized by the subdivision of the total time interval $[0,\ t_f]$ into l so-called phases

$$0 < t_f^1 < ... < t_f^{l-1} < t_f^l = t_f$$

where t_f^j is the final time of phase number j, or the phase separation time between phase number j and $j+1$.

The total multi-phase optimal control problem consists of l nonlinear optimal control problems like the one defined by Eqs. (2.1– 2.10). What makes it one multi-phase problem rather than l separate problems is $l-1$ additional sets of phase connect conditions, defined according to

$$\begin{pmatrix} \mathbf{x}^{j+1}(t_f^{j+}) \\ \mathbf{u}^{j+1}(t_f^{j+}) \\ \mathbf{p}_d^{j+1} \end{pmatrix} = \mathbf{h}^j(\mathbf{x}^j(t_f^{j-}), \mathbf{u}^j(t_f^{j-}), \mathbf{p}_d^j) \quad (j = 1, ..., l-1) \tag{2.11}$$

The dimensions of the state vector, the control vector, the design parameter vector and the vector of constraint functions may vary from one phase to the next.

An example of a multi-phase system is a multistage rocket trajectory optimization problem (Jänsch & Schnepper 1991).

2.2 Numerical Solution by Control Vector Parameterization and Nonlinear Programming

2.2.1 Introduction

Numerical methods for the solution of optimal control problems (OCP) were first developed in the 1950s. Several methods have been developed which use the first order necessary conditions for optimality arising from using the calculus of variation or the minimum principle of Pontryagin on the optimal control problem, see e.g. (Athans & Falb 1966, Bryson & Ho 1975). These methods are often called function space methods. A very efficient method from this class of methods is the (indirect) multiple shooting method which solves the resulting two point boundary value

problems, see e.g. (Bulirsch 1971, Deuflhard 1980, Stoer & Bulirsch 1983)). Other methods in this class are based on using gradient information in function space, see e.g. (Lasdon, Mitter & Warren 1967, Miele 1975, Bryson & Ho 1975).

In the present work numerical solutions based on Control Vector Parameterization (CVP) is discussed. The idea is to approximate the infinite dimensional OCP by a finite dimensional optimization problem, or nonlinear programming (NLP) problem. This class of methods have become particularly popular the last fifteen years due to the development of efficient NLP solvers like the sequential quadratic programming (SQP) methods (see Section 2.4). Reviews of numerical methods for solving OCPs including both function space methods and CVP methods are given in (Strand 1991, Jänsch & Schnepper 1991).

The connection between optimal control and finite dimensional optimization has been well known for long. Hurwicz (1958) extended Kuhn-Tuckers theorems to infinite dimensional problems. Several authors have extended his idea to optimal control, and others again have investigated the connection between Kuhn-Tuckers theorem and the maximum principle of Pontryagin. A discussion of this topic is found in (Tabak & Kuo 1971). The fact that an integral can be represented as a limit of summation is used to show the above mentioned relationship. The OCP is discretized giving a finite dimensional optimization problem. As the number of discretization points goes to infinity the finite dimensional optimization problem becomes an optimal control problem, showing the equivalence.

In early parameterization methods the discretization scheme above with a finite number of discretization points were used to approximate the OCP. Both the states and the controls were treated as independent variables, and satisfying the state space equations at each discretization point was included as equality constraints (Canon, Cullum & Polak 1970, Tabak & Kuo 1971, Polak 1971). Canon et al. (1970) referred to this approach as "direct transcription". Since the beginning of the seventies this approach has been made considerably more efficient by using piecewise polynomial approximations to represent the controls and the states. This class of methods is often referred to as CVP with State Vector Parameterization (SVP). Usually the system of algebraic equations forcing the state space equations to be satisfied is obtained through collocation, and this class of methods is then referred to as direct collocation methods. Examples on direct collocation methods can be found in (Neuman & Sen 1973, Tsang, Himmelblau & Edgar 1975, Biegler 1984, Cuthrell & Biegler 1987, Hargraves & Paris 1987, Jänsch & Schnepper 1991, Betts & Huffman 1992, Betts & Huffman 1993).

Another possibility is to parameterize only the controls and computing the states by integrating the state space equations. This results in a smaller number of parameters and constraints. Rosenbrock & Storey (1966) used this approach, and represented the control by

$$u(t) = \sum_{i=0}^{n} a_i \phi_i(t) \tag{2.12}$$

where the a_i's are the free parameters and the $\phi_i(t)$'s are some basic polynomial functions. Optimal control problems including, in addition to the objective functional, only the state space equations with initial states given were considered. This resulted in unconstrained optimization problems, which were solved without using any gradient informations. Difficulties with the polynomial representation in Eq. (2.12) were reported by Hicks & Ray (1971).

Since about 1970 this class of methods have been improved by replacing the control approximation in Eq. (2.12) by piecewise polynomials. In addition to the modification of the parameterization scheme, constraints can be handled efficiently, see e.g. (Brusch 1974), and as will be further explored in the next section, partial derivatives of the objective function and the constraints with respect to the parameters can be computed. These methods are often referred to as CVP with shooting because of their similarity to the shooting methods for solving two point boundary value problems. Examples on publications on CVP with shooting are found in (Speyer, Kelley, Levine & Denham 1971, Brusch & Peltier 1973, Brusch 1974, Sirisena 1973, Sargent & Sullivan 1978, Kraft 1980, Kraft 1989, Goh & Teo 1988, Horn 1990, Jänsch & Schnepper 1991, Bock & Plitt 1984).

CVP with shooting can be either direct or indirect. In the latter case the costate equations are used to compute partial derivatives of the objective function and the constraint functions (see Section 2.3.3). In (Bock & Plitt 1984) a multiple shooting approach is applied (see Section 2.2.3), which can be regarded as a combination of CVP with shooting and CVP with SVP.

The class of NLP's approximating the class of OCP's defined in the previous section can be written

$$\min_{\mathbf{p}} F(\mathbf{p}) \tag{2.13}$$

with constraints

$$\mathbf{c}_e(\mathbf{p}) = 0 \tag{2.14}$$

$$\mathbf{c}_i(\mathbf{p}) \geq 0 \tag{2.15}$$

$$p_{i,min} \leq p_i \leq p_{i,max} \tag{2.16}$$

where

$$\mathbf{p} = \begin{pmatrix} p_1 & p_2, & \dots & p_N \end{pmatrix}^{\mathrm{T}} \tag{2.17}$$

is a finite set of parameters uniquely defining the controls $u_i(t)$, possibly the states $x_j(t)$, and possibly other free parameters in the OCP. Eq. (2.14) defines a vector of equality constraints of dimension M_E and Eq. (2.15) defines a vector of inequality constraints of dimension M_I. Eq. (2.16) defines box constraints on the parameters.

Now, the description of the CVP-method is divided in three parts:

- The parameterization procedure defines how the NLP (Eqs. 2.13 – 2.16) is constructed from the OCP. The content of the parameter vector depends on the parameterization procedure as well as the OCP at hand. If infinite dimensional path constraints (Eq. 2.8) are present the parameterization procedure has to specify how these should be represented in the NLP. Even the box constraints depend on the parameterization procedure. Parameterization procedures will be explored further in the present section.

- The calculation of the objective function and all the constraints, as well as their partial derivatives with respect to the parameters. In order to solve the NLP efficiently these calculations have to be performed at each iteration of the NLP optimization procedure. The objective function and the constraint functions will for simplicity in the sequel commonly be called functions. Function and gradient calculations in CVP-methods will be treated in the next section.

- The numerical solution of the NLP (2.13 – 2.16). Numerical methods for solving finite dimensional optimization problems, or NLP's, will be treated in Section 2.4. The treatment will concentrate on the so-called SQP methods, which the last fifteen years have been considered as the most efficient methods for solving smooth moderate sized dense NLP's (Gill, Murray & Wright 1981, Fletcher 1987).

In the remainder of this chapter only CVP with shooting is treated. Since only nonstiff problems have been solved during this doctoral work, CVP with SVP has not been implemented. Section 2.4 is, however, equally relevant for shooting and SVP methods.

2.2.2 CVP with Single Shooting

In this section CVP with shooting is presented and discussed. In these methods the controls u_i are approximated by piecewise polynomials, and the states are found by integrating the state space equations forwards in time. Free initial states, if present, are included in the parameter set, otherwise the states are totally determined by the forward integration of the state space equations, and are excluded from the parameter set. The motivation for the term "single" will implicitly be explained after CVP with multiple shooting has been defined. In the following discussion CVP with single shooting will mainly be referred to as just CVP with shooting, since most of the aspects discussed are common for both shooting approaches.

A control time grid

$$0 = t_{i,0} < t_{i,1} < \ldots < t_{i,q_i} = t_f$$

is defined for each control u_i, giving q_i distinct time intervals. These time intervals are called control switching time intervals and the time nodes in the time grids are

called control switching times, or just control time nodes. The control switching time interval j is defined to be the interval between the control time node $j-1$ and j, and the control time nodes t_{i,q_i} equal the final time t_f. In each of these time intervals the control $u_i(t)$ is approximated by a polynomial in t. The control is then uniquely defined by the control time nodes and the coefficients of the polynomials.

There are many ways to define the particular parameterization to be used. There are freedoms in the choices of both the degree and the continuity of the polynomials and in the definition of the control switching time intervals.

There are also alternative formulations of the constraints in the NLP (Eqs. 2.14–2.16) in shooting methods. The algebraic constraints in the OCP (Eqs. 2.3–2.7) are represented as they are, independently of the parameterization method. The path constraints (Eq. 2.8), however, are infinite dimensional constraints which have to be represented by finite dimensional constraints in the NLP. The box constraints, Eq. (2.9) in the OCP and Eq. (2.16) in the NLP, also have to be looked more closely into.

The Control Switching Time Intervals

The simplest choice for the control switching time intervals is to choose them equal for all controls and with nonoptimizable and equidistantly distributed time nodes.

Before the discussion is continued it is necessary to explain what is meant by nonoptimizable time nodes. If the final time t_f is fixed it trivially means that the time nodes are constant. If, however, the final time is free the values of the nonoptimizable time nodes relative to the total time interval are constant, that is

$$\frac{t_{j,i}}{t_f} = \text{constant}$$

If the time is scaled with respect to the final time in the state space equations the normalized time nodes, which are constant, are used in the integrations. This time normalization is also very convenient concerning gradient calculations (Section 2.3). Therefore, it will hereafter be assumed that the time is scaled with respect to the final time resulting in the scaled time variable

$$\tau = \frac{t}{t_f}$$

If some apriori knowledge about the structure of the optimal control inputs exists it may be desirable to use nonoptimizable, but nonequidistantly distributed control time nodes. This can be done without increasing the complexity of the parameterization procedure. It may also be desirable to apply different control switching time intervals for each of the control inputs, which makes the parameterization more complex.

It is also possible to include the control time nodes in the parameter set, or just a subset of them. Also, the time nodes can be specified to be equal for all controls or chosen independently.

Now, divide the parameter vector in four parts according to

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}_u & \mathbf{p}_t & \mathbf{p}_d & \mathbf{p}_o \end{pmatrix}^{\mathrm{T}}$$

where $\mathbf{p}_u$ and $\mathbf{p}_t$ consist of the parameters defining the controls. $\mathbf{p}_u$ defines the coefficients in the polynomials and $\mathbf{p}_t$ consists of the free control time nodes. $\mathbf{p}_d$ is the design parameter vector from the OCP. Finally, $\mathbf{p}_o$ consists of other optimizable parameters, which may be the final time t_f and any free initial states.

The scaled nonlinear dynamical system (Eq. 2.1) with parameterized control $\mathbf{u}(\tau, \mathbf{p}_u, \mathbf{p}_t)$ is now written:

$$\begin{aligned} \dot{\mathbf{x}}(\tau) &= \tilde{\mathbf{f}}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau, \mathbf{p}_u, \mathbf{p}_t), \mathbf{p}_d) \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{2.18}$$

where

$$\tilde{\mathbf{f}}(\tau, \cdot) = t_f \cdot \mathbf{f}(t_f \cdot \tau, \cdot)$$

The objective function, resulting from parameterizing the controls in the objective functional (Eq. 2.2) is written

$$\min_{\mathbf{p}} J = \phi(\mathbf{x}(t_f), \tilde{\mathbf{p}}_d) + \int_0^1 \tilde{L}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau, \mathbf{p}_u, \mathbf{p}_t), \mathbf{p}_d) d\tau \tag{2.19}$$

where

$$\tilde{L}(\tau, \cdot) = t_f \cdot L(t_f \cdot \tau, \cdot)$$

Piecewise Polynomial Controls

The simplest parameterization is just to specify the controls to be piecewise constant. This parameterization is illustrated in Figure 2.1 for control number i.

In this case the controls must be allowed to be discontinuous at the control time nodes. The part $\mathbf{p}_{u_i}$ of the control parameter vector corresponding to the control u_i now simply becomes

$$\mathbf{p}_{u_i} = \begin{pmatrix} u_{i,0} & u_{i,1} & \dots & u_{i,q_i-1} \end{pmatrix}^{\mathrm{T}} \tag{2.20}$$

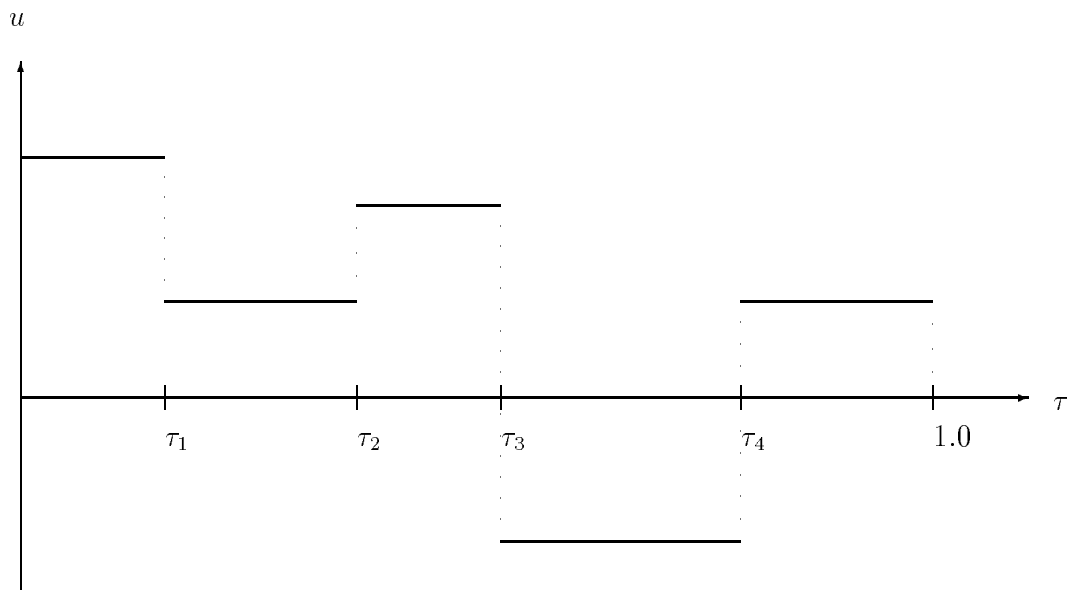

Figure 2.1: *Piecewise Constant Control Parameterization*

where $u_{i,j-1}$ is the control number i in the control switching time interval number j.

Alternatively, the controls can be approximated by piecewise linear functions. In this case the controls can either be specified to be continuous in $(0,\ t_f)$, or be allowed to be discontinuous at all the control time nodes. Both these piecewise linear parameterizations are illustrated in Figure 2.2 for control number i.

The part $\mathbf{p}_{u_i}$ of the control parameter vector corresponding to the control u_i, becomes in the case where the controls have to be continous at the control time nodes

$$\mathbf{p}_{u_i} = \begin{pmatrix} u_{i,0} & u_{i,1} & \dots & u_{i,q_i-1} & u_{i,q_i} \end{pmatrix}^{\mathrm{T}} \tag{2.21}$$

This parameterization gives

$$u_i(\tau) = u_{i,j} + \frac{u_{i,j+1} - u_{i,j}}{\tau_{i,j+1} - \tau_{i,j}}(\tau - \tau_{i,j}) \quad (i = 1:m, \quad j = 0:q_i - 1) \tag{2.22}$$

Without these continuity conditions $\mathbf{p}_{u_i}$ becomes

$$\mathbf{p}_{u_i} = \begin{pmatrix} u_{i,0} & u_{i,1}^- & u_{i,1}^+ & \dots & u_{i,q_i-1}^- & u_{i,q_i-1}^+ & u_{i,q_i} \end{pmatrix}^{\mathrm{T}} \tag{2.23}$$

where $u_{i,j}^-$ and $u_{i,j}^+$ are the left and the right limit of $u_i(\tau)$ at control time node j respectively. This parameterization gives

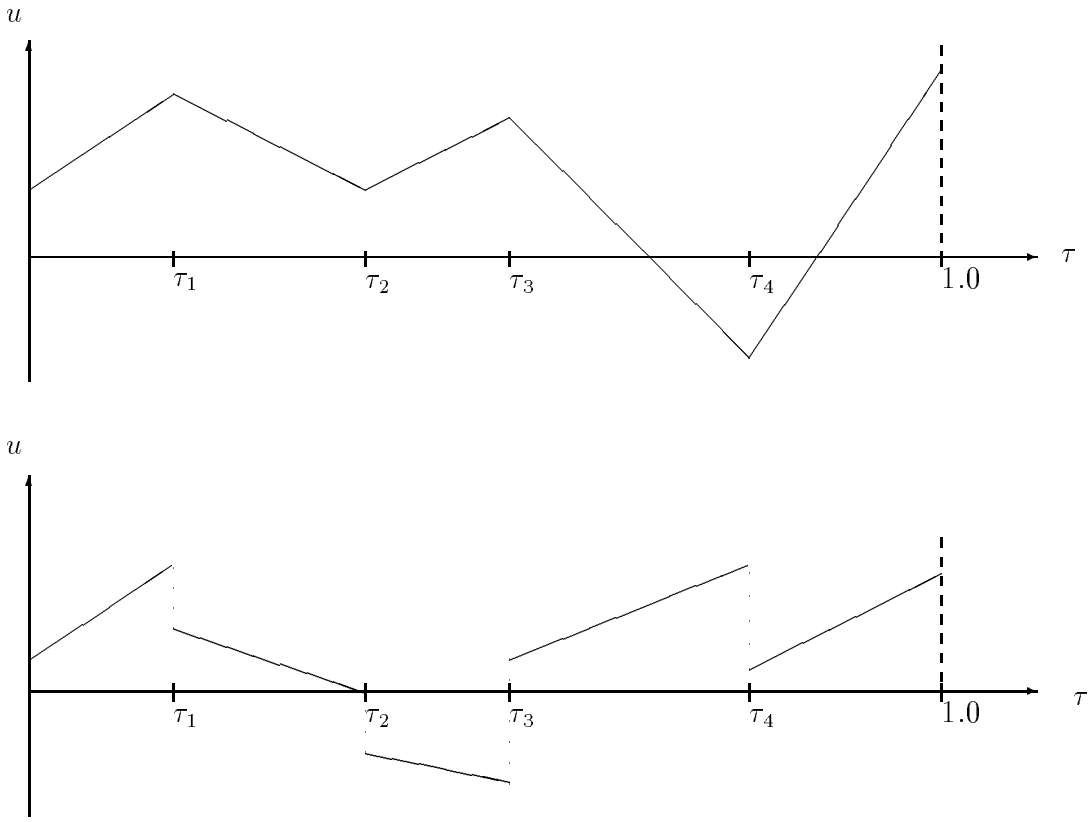

Figure 2.2: *Piecewise Linear Control Parameterization, Continuous (a) or Discontinuous at the Time nodes (b)*

$$u_i(\tau) = u^+_{i,j} + \frac{u^-_{i,j+1} - u^+_{i,j}}{\tau_{i,j+1} - \tau_{i,j}}(\tau - \tau_{i,j}) \quad (i = 1:m, \quad j = 0:q_i - 1) \tag{2.24}$$

In principle any polynomial interpolation method can be used to approximate the controls, see e.g.(Kreyszig 1978), for instance cubic splines, see e.g. (Kraft 1985).

Simple Parameter Constraints

Simple parameter constraints will primarily be box constraints, but can in general be any constraints that are independent of the states.

For piecewise constant and piecewise linear parameterization box constraints on the controls (Eq. 2.9) in the OCP lead to the same simple box constraints on the control parameters (Eq. 2.16) in the NLP. This is not necessarily true for control parameterizations with higher degrees in the approximating polynomials. The OCP box constraints may then lead to NLP constraints that are linear combinations of the parameters, and even time dependent such that one of the path constraint handling techniques presented below has to be applied.

Any constraints on $\mathbf{p}_d$ or $\mathbf{p}_o$ in the OCP will naturally lead to the same parameter constraints in the NLP.

Linear constraints on the control time nodes are present as a result of the parameterization rather than the OCP itself. For each of the control switching time intervals i there has to be a minimum distance ϵ between two time nodes

$$\tau_{i,j} - \tau_{i,j-1} \geq \epsilon \quad j = 1:q_i \tag{2.25}$$

In practice, it is more convenient to define box constraints on the control time nodes which are guaranteed to satisfy Eq. (2.25), for instance

$$\frac{j}{q_i} - \gamma \leq \tau_{i,j} \leq \frac{j}{q_i} + \gamma \quad j = 1:q_i - 1 \tag{2.26}$$

where

$$\gamma \leq \frac{0.5}{t_f} - 0.5\epsilon$$

For some parameters no box constraints are given in the OCP or implied by the parameterization. This is typical for t_f. Box constraints may nevertheless be introduced in the NLP for these parameters too, for the convenience of the optimizer.

Path Constraint Representation

The infinite dimensional path constraints in the OCP (Eq. 2.8) have to be represented by finite dimensional constraints in the NLP. For instance, define for each path constraint $\mathbf{g}_{E,i}$ (Goh & Teo 1988)

$$\tilde{\mathbf{c}}_{E,i}(\mathbf{p}) = \int_0^1 [\min\{0, \mathbf{g}_{I,i}(t_f \cdot \tau, \mathbf{x}(\tau), \mathbf{u}(\tau, \mathbf{p}), \mathbf{p}_d)\}]^2 d\tau = \mathbf{0} \tag{2.27}$$

Here, $\tilde{\mathbf{c}}_{E,i}(\mathbf{p})$ is an element of $\tilde{\mathbf{c}}_E(\mathbf{p})$ which is an r-dimensional subvector of the equality constraint vector $\mathbf{c}_E(\mathbf{p})$ in the NLP (Eq. 2.14). It is easily seen that satisfying Eq. (2.27) is equivalent to satisfying Eq. (2.8). These constraints do not satisfy the usual constraint qualifications, which usually will influence the convergence negatively by creating some oscillations around the optimum (Goh & Teo 1988). This approach will be referred to as the integral approach.

Alternatively a path constraint time grid consisting of k time nodes can be defined

$$0 \leq \tau_1^g < \tau_2^g < ... < \tau_k^g \leq 1$$

where the path constraints have to be satisfied. Now, define

$$\tilde{\mathbf{c}}_{I,j}(\mathbf{p}) = \mathbf{g}_I(t_f \tau_j^g, \mathbf{x}(\tau_j^g), \mathbf{u}(\tau_j^g, \mathbf{p}), \mathbf{p}_d) \geq \mathbf{0} \tag{2.28}$$

where the $\tilde{\mathbf{c}}_{I,j}(\mathbf{p})$'s are k r-dimensional subvectors constituting $\tilde{\mathbf{c}}_I(\mathbf{p})$ which is a subvector of the $\mathbf{c}_I(\mathbf{p})$ in the NLP. This approach only approximates the path constraints (Eq. 2.8) because there is no guarantee that the path constraints will be satisfied between the time nodes. However, continuity properties of the state space equations (Eq. 2.18) will often impose some bounds on how much the path constraints can be violated between the time nodes. It is of course possible to use different time nodes for different path constraints, if that should be necessary. The approach, which will be referred to as the time grid approximation approach, has been successfully implemented in the optimization packages TOMP (Kraft 1989) and PROMIS (Jänsch & Schnepper 1991) and by Strand (1991).

2.2.3 CVP with Multiple Shooting

The multiple shooting method is a very efficient method for solving two point boundary value problems, see e.g. (Deuflhard 1980). Bock & Plitt (1984) proposed to use this technique in CVP, resulting in the direct multiple shooting method.

The total time interval (0, 1) is divided into s shooting intervals

$$0 = \tau_0^S < \tau_1^S < ... < \tau_{s-1}^S < \tau_s^S = 1$$

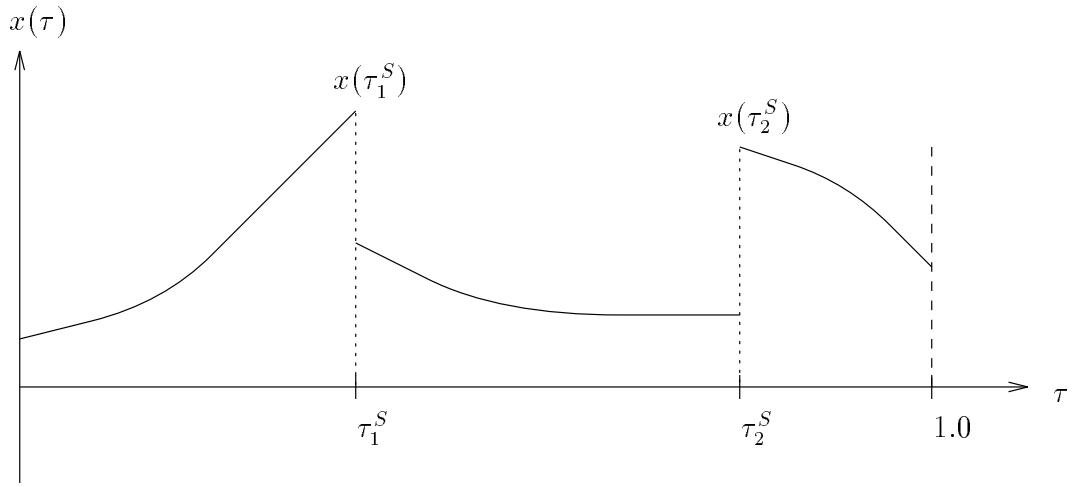

Figure 2.3: *Multiple Shooting*

where τ_j^S is a shooting node, which is fixed. In each shooting interval the state space equations are integrated separately from τ_j^S to τ_{j-1}^S. Multiple shooting is illustrated in Figure 2.3.

For each internal shooting node j a new set of initial states $\mathbf{x}_0^j$ have to be defined as a set of $(s-1)n$ new parameters, referred to as shooting parameters. These parameters have to satisfy a set of shooting constraints

$$\mathbf{x}(\tau_j^S) = \mathbf{x}_0^j \quad \text{for } j = 1, ..., s-1 \tag{2.29}$$

which forces the converged states to be continuous at the shooting nodes.

All the control switching time intervals defined above usually have to have control time nodes that coincide with the shooting nodes, such that in multiple shooting one can say that for each control i each shooting interval j is divided into $q_{i,j}$ control switching time intervals. It is of course possible to have only one switching time interval in all shooting intervals for all controls.

Since the state space equations are integrated independently in each phase, each parameter will affect the resulting trajectory in one phase only, provided that some extra modifications are made. Firstly, for each phase j a set of n_p design parameter vectors $\mathbf{p}_{d,j}$, replacing the global parameter vector $\mathbf{p}_d$, have to be defined. With the additional shooting constraints

$$\mathbf{p}_{d,j} = \mathbf{p}_{d,j+1} \quad j = 1, ..., s-1 \tag{2.30}$$

the new design parameter definition will be equivalent to the original definition. If the controls are constrained to be continuous at the control time nodes the control parameters acting on both sides of a shooting node have to be duplicated. The continuity at the shooting nodes are maintained with the additional shooting con-

straints

$$\mathbf{u}_j(\tau_j^S) = \mathbf{u}_{j+1}(\tau_j^S) \quad j = 1, ..., s-1 \tag{2.31}$$

Finally, if the integral approach (Eq. 2.27) is applied, each constraint $\tilde{\mathbf{c}}_{E,k}(\mathbf{p})$ has to be divided into s constraints, one for each shooting interval.

The parameter vector acting in shooting interval j will be denoted $\mathbf{p}_j$. The following sparsity pattern for the NLP is now achieved (Jänsch & Schnepper 1991):

- The Jacobian of the equality constraints $\mathbf{c}_E$ is block bidiagonal

$$\frac{\partial \mathbf{c}_E}{\partial \mathbf{p}} = \begin{pmatrix} \mathbf{C}_E^{1,1} & \mathbf{C}_E^{2,1} & 0 & \dots & 0 \\ 0 & \mathbf{C}_E^{2,2} & \mathbf{C}_E^{3,2} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{C}_E^{s-1,s-1} & \mathbf{C}_E^{s,s-1} \\ 0 & \dots & 0 & 0 & \mathbf{C}_E^{s,s} \end{pmatrix}$$

  with $\mathbf{c}_{E,i}$ denoting the equality constraints gathered from shooting mesh i and

$$\mathbf{C}_E^{j,i} = \frac{\partial \mathbf{c}_{E,i}}{\partial \mathbf{p}_j}$$

  The off diagonal blocks are linear, and nonzero only for the shooting constraints.

- The Jacobian of the inequality constraints $\mathbf{c}_I$ is block diagonal

$$\frac{\partial \mathbf{c}_I}{\partial \mathbf{p}} = \begin{pmatrix} \mathbf{C}_I^1 & & & \\ & \mathbf{C}_I^2 & \mathbf{0} & \\ & \mathbf{0} & \ddots & \\ & & & \mathbf{C}_I^s \end{pmatrix}$$

  with $\mathbf{c}_{I,j}$ denoting the inequality constraints gathered from shooting mesh j and

$$\mathbf{C}_I^j = \frac{\partial \mathbf{c}_{I,j}}{\partial \mathbf{p}_j}$$

- The Hessian matrix of the cost function is block diagonal. Since all off diagonal elements of the Jacobians are constant or zero the Hessian matrix of the Lagrangian function of the NLP is also block diagonal (the Lagrangian function is defined in Section 2.4)

$$\frac{\partial^2 L}{\partial \mathbf{p}^2} = \begin{pmatrix} \mathbf{H}_1 & & & \\ & \mathbf{H}_2 & \mathbf{0} & \\ & \mathbf{0} & \ddots & \\ & & & \mathbf{H}_s \end{pmatrix}$$

where

$$\mathbf{H}_j = \frac{\partial^2 L}{\partial \mathbf{p}_j^2}$$

Using multiple instead of single shooting the additional parameters and equality constraints makes the resulting NLP larger. However, there are several advantages with this approach. Firstly, the integration of the state space equations are more robust due to reduced sensitivity with respect to parameter perturbations. Secondly, the gradient computations are cheaper when the forward difference approach or the state sensitivity approach are applied (see the next section). Thirdly, the sparsity pattern imposed on the resulting NLP can be utilized.

Multi-Phase System

The generalization to multi-phase systems is for the convenience of notation treated here under CVP with multiple shooting, but can naturally be solved by single shooting as well.

The modifications of the parameterization scheme necessary to handle multi-phase systems are in principle rather simple. The phase connect conditions (Eq. 2.11) have the same structure as the shooting constraints. If the free phase separation time node parameters are replicated for each shooting interval, like the other global parameters, the sparsity pattern on the Jacobian is maintained.

For multi-phase systems it is convenient to scale the time t independently for each phase. The scaled time in phase j is then defined as $(j = 1, ..., l)$

$$\tau^j = \frac{t - t_f^{j-1}}{t_f^j - t_f^{j-1}}$$

and the scaled state space equations become

$$\begin{aligned} \dot{\mathbf{x}}^j(\tau^j) &= \tilde{\mathbf{f}}^j(\tau^j, \mathbf{x}^j(\tau^j), \mathbf{u}^j(\tau^j, \mathbf{p}_u^j, \mathbf{p}_t^j), \mathbf{p}_d^j) \\ \mathbf{x}^j(0) &= \mathbf{x}_0^j \end{aligned} \tag{2.32}$$

where

$$\tilde{\mathbf{f}}^j(\tau^j, \cdot) = (t_f^j - t_f^{j-1})\mathbf{f}^j((t_f^j - t_f^{j-1}) \cdot \tau^j, \cdot)$$

2.3 Function and Gradient Computations in CVP with Shooting

2.3.1 Function Computations

In this section $\mathcal{F}(\mathbf{p})$ denotes a function which may be the objective function (Eq. 2.13) or a nonlinear equality or inequality constraint function (Eqs. 2.14 and 2.15). For simplicity, only single-phase systems are considered. The generalization to multi-phase problems is straightforward. Normalized time is consequently used as an independent variable in this section, but with t as normalized time variable instead of τ.

$\mathcal{F}(\mathbf{p})$ can generally be written (Goh & Teo 1988)

$$\mathcal{F}(\mathbf{p}) = \varphi(t_c, \mathbf{x}(t_c), \mathbf{x}_0, \mathbf{p}_d) + \int_0^{t_c} \mathcal{L}(t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}), \mathbf{p}_d)dt \tag{2.33}$$

where t_c is a characteristic time for the function. $\varphi(\cdot)$ and $\mathcal{L}(\cdot)$ are generalized Mayer and Lagrangian terms respectively.

The quantities in Eq. (2.33) depend on what $\mathcal{F}(\mathbf{p})$ represents, and are defined for each of the possible cases according to:

- If $\mathcal{F}(\mathbf{p})$ represents the objective function $F(\mathbf{p})$, then

  $$t_c = 1, \quad \varphi(\cdot) = \phi(\cdot), \quad \mathcal{L}(\cdot) = \tilde{L}(\cdot)$$

  where $\phi(\cdot)$ and $\tilde{L}(\cdot)$ are defined in Eq. (2.19).

- If $\mathcal{F}(\mathbf{p})$ represents path constraint number i, where the path constraints are represented by the integral approach (see Eq. 2.27), then

  $$t_c = 1, \quad \varphi(\cdot) = 0, \quad \mathcal{L}(\cdot) = [\min\{0, \mathbf{g}_{I,i}(t_f \cdot t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}))\}]^2$$

  for path constraint i.

- If $\mathcal{F}(\mathbf{p})$ represents path constraint number i at path constraint time node number j, where the path constraints are approximated by the time grid approximation approach (see Eq. 2.27), then

  $$t_c = t_j, \quad \mathcal{L}(\cdot) = 0, \quad \varphi(\cdot) = \mathbf{g}_{I,i,j}(t_f \cdot t_j, \mathbf{x}(t_j), \mathbf{u}(t_j, \mathbf{p}))$$

  for path constraint number i at time node number j.

- If $\mathcal{F}(\mathbf{p})$ represents final time boundary constraint number i (see Eqs. 2.5–2.6), then

$$t_c = 1, \quad \mathcal{L}(\cdot) = 0, \quad \varphi(\cdot) = \boldsymbol{\psi}_i^f(t_f, \mathbf{x}(1))$$

  where $\boldsymbol{\psi}^f = \begin{pmatrix} (\boldsymbol{\psi}_E^f)^{\mathrm{T}} & (\boldsymbol{\psi}_I^f)^{\mathrm{T}} \end{pmatrix}^{\mathrm{T}}$.

Simple design parameter constraints and initial boundary constraints are easy to compute, since they do not depend on the state space equations through the states. They are not considered in the context of Eq. (2.33).

The major part of the effort needed to compute these functions are the integration of the state space equations, and for some of them the integral in Eq. (2.33).

The computation procedure can be described as follows.

- The time nodes from all the time grids are collected and sorted in increasing order resulting in the total time grid

$$0 = t_0^T < t_1^T < \ldots < t_{q_T}^T = 1$$

  where

$$q_T = q + k$$

  where k is the number of time nodes in the path constraint approximation time grid and q is the number of control time nodes.

- The state space equations and possible integral parts of the functions are integrated numerically forwards in time, independently for each phase. The integration procedure is organized so that the time nodes in the total time grid coincide with end points of integration time steps, also for time nodes that are not characteristic times for any functions. This is done because the lack of smoothness of the control functions may otherwise corrupt the accuracy properties of the numerical integration method.

  In principle, any numerical method for solving initial value problems can be used for integrating the state space equations. The possible integral parts of the functions are better integrated together with the states by the same integration method, with the integral of the Lagrangian functionals acting as extra states. The state space equations together with the possible Lagrangian functionals will later be referred to as the extended state space equations, with extended state vector

$$\mathbf{x}_e = \begin{pmatrix} \mathbf{x} \\ I_{\mathcal{L}_1} \\ \vdots \\ I_{\mathcal{L}_{n_l}} \end{pmatrix} \quad \text{where} \quad I_{\mathcal{L}_i} = \int_0^t \mathcal{L}_i(\cdot)dt$$

where n_l is the number of functions with Lagrangian parts. $n_l = 1$ if the path constraints are represented by the time grid approximation approach.

- The functions are computed at their respective characteristic times as the integration goes forth.

Accuracy

The error in computing $\mathcal{F}(\mathbf{p})$ is assumed to be dominated by the error in the numerical integration procedure because the error in the pure analytical algebraic calculations are normally very small. These latter errors are small enough to be ignored in an optimization problem with moderate accuracy specifications. The error is therefore assumed to be given as a function of the error in computing the extended states $\mathbf{x}_e$.

Optimization accuracy is discussed in (Gill et al. 1981). An optimization problem is usually formulated with mathematical functions which approximate the real world problem that is to be optimized. It is emphasized in (Gill et al. 1981) that if the mathematical functions are crude approximations of the real world problem, this does not mean that the numerical solution of the optimization problem has to be crude. That is, the mathematically formulated optimization problem can be solved with accuracy specifications which are independent of the accuracy to which the mathematically formulated optimization problem approximates the real world problem. The accuracy in the numerical solution of the optimization problem is, however, limited by the accuracy to which the mathematical functions are represented numerically.

In this work it is proposed to use these ideas to analyse the accuracy of numerical solution of OCP's using CVP with shooting. The analysis, which is believed to be a new contribution, is presented in the following.

Instead of considering the NLP in terms of functions $\mathcal{F}(\mathbf{p})$ of the type in Eq. (2.33), the NLP will be considered in terms of functions $\mathcal{F}_M(\mathbf{p})$ approximating the original $\mathcal{F}(\mathbf{p})$. It is then required that $\mathcal{F}_M(\mathbf{p})$ has the necessary smoothness properties of the optimization problem, i.e. the smoothness properties that $\mathcal{F}(\mathbf{p})$ is supposed to have.

Let f_{IM} denote an arbitrary integration method, i.e. a method for numerical solution of differential equations. Together with a sequence of n_s integration step sizes $h_1, \ldots, h_{n_s}$, f_{IM} defines a discretization scheme for the numerical solution of the differential equations. The discretization scheme approximates the vector of state functions $\mathbf{x}_e$, which are infinite dimensional, by a vector $\bar{\mathbf{x}}_e$ of n_s-dimensional tables. Let the sequence of step sizes be denoted by

$$h_{seq} = h_1, \ldots, h_{n_s}$$

The function $\mathcal{F}(\mathbf{p})$, at least partly depending indirectly on the parameters through

$\mathbf{x}_e(t_c)$, is now approximated by a function $\bar{\mathcal{F}}(\mathbf{p})$, depending indirectly on the same parameters through $\bar{\mathbf{x}}_e(t_c)$. This can be written

$$\mathcal{F}(\mathbf{p}) = \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM}) + \Delta(\mathbf{p}, \epsilon_I(\mathbf{p}, h_{seq}; f_{IM})) \tag{2.34}$$

where Δ is a function of the error ϵ_I in the integrator. Assume that f_{IM} is an integration method of order p_{oI}. Then, from basic numerical analysis (Hairer, Nörsett & Wanner 1987a), assuming $\mathcal{F}$ is sufficiently smooth, Eq. (2.34) can be written

$$\mathcal{F}(\mathbf{p}) = \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM}) + \sum_{i=1}^{n_s} \varepsilon_i(\mathbf{p}) h_i^{p_{oI}+1} \tag{2.35}$$

where $\varepsilon_i(\mathbf{p})$ is a coefficient defining the local error. It is easily seen that if the sequence of step sizes h_{seq} is kept fixed for the entire optimization, $\bar{\mathcal{F}}(\mathbf{p})$ is a function of $\mathbf{p}$ only and, accordingly, it is a function of the sum of a finite series of algebraic expressions. $\bar{\mathcal{F}}(\mathbf{p})$ is therefore the function $\mathcal{F}_M(\mathbf{p})$ considered in the NLP. The numerical representation of $\mathcal{F}_M(\mathbf{p})$, say $\mathcal{F}_N(\mathbf{p})$, can therefore be expected to approximate $\mathcal{F}_M(\mathbf{p})$ with a very high accuracy compared to the integration accuracy.

If an adaptive step size integrator is used, however, the sequence of step sizes h_{seq} will generally not be fixed during the optimization, and therefore $\bar{\mathcal{F}}$ will be a function of both the parameters $\mathbf{p}$ and the sequence h_{seq}. Considered as a function of $\mathbf{p}$ only, $\bar{\mathcal{F}}$ will therefore generally vary during the optimization, and can therefore not be considered as the function $\mathcal{F}_M(\mathbf{p})$.

To this end define $\bar{\mathcal{F}}^*(\mathbf{p}, h^*_{seq}; f_{IM})$ as the function of $\mathbf{p}$ approximating $\mathcal{F}(\mathbf{p})$ at the theoretical solution $\mathbf{p}^*$. This means that if the true theoretical solution $\mathbf{p}^*$ were available, and if the integration method f_{IM} were used on the extended state space equations, h^*_{seq} would have been the resulting sequence of step sizes. Therefore, define

$$\mathcal{F}_M(\mathbf{p}) = \bar{\mathcal{F}}^*(\mathbf{p}, h^*_{seq}; f_{IM})$$

Since the functions to be computed during the optimization are defined differently from $\mathcal{F}_M(\mathbf{p})$, because of different step size sequences h_{seq}, the numerical representation of $\mathcal{F}_M(\mathbf{p})$, $\mathcal{F}_N(\mathbf{p})$, cannot be expected to approximate $\mathcal{F}_M(\mathbf{p})$ with an accuracy as high as in the case of fixed step sizes. The accuracy to which $\bar{\mathcal{F}}$ approximates $\mathcal{F}_M$ is now more interesting than the accuracy to which $\bar{\mathcal{F}}$ approximates $\mathcal{F}$. This approximation can be written in the same form as the approximation in Eq. (2.35),, i.e.

$$\mathcal{F}_M(\mathbf{p}) = \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM}) + \sum_{i=1}^{n_s} \varepsilon_i^M(\mathbf{p}) h_i^{p_{oI}+1} \tag{2.36}$$

where $\varepsilon_i^M(\mathbf{p})$ is a coefficient defining the local error.

This suggests that a fixed step size integrator, like the standard 4th order Runge-Kutta method, should be preferred to an adaptive step size integrator because high optimization accuracy specifications can be applied even with moderate integration accuracy. A variable step size method has, however, the well known advantage that the accuracy can be automatically controlled, and will therefore be more flexible and more efficient than the fixed step size integrator. Therefore, assuming that the accuracy of the integrator needs to be considerably higher than the convergence accuracy specifications in the NLP, a variable step size integrator can be used with comfort. In addition, it is reasonable to believe that $\bar{\mathcal{F}}$ approximates $\bar{\mathcal{F}}^*$ more accurately than it approximates $\mathcal{F}$, i.e. that $\mathcal{F}_N$ approximates $\mathcal{F}_M$ with higher accuracy than the integration accuracy. This because it is reasonable to believe that h_{seq} will approach h^*_{seq} close to $\mathbf{p}^*$, and as a result the terms ε_i^M in Eq. (2.36) will be less than the terms ε_i in Eq. (2.35). For instance if the first j step sizes in h_{seq} and h^*_{seq} are equal, the corresponding first j ε_i^M's are zero. However, if an optimization accuracy that is higher than the desirable integration accuracy is needed, the step size sequence in the integrator has to be frozen close the solution, i.e. the step size sequence has to be fixed for the last part of the iteration sequence.

This accuracy analysis will be used further in the following subsections, where gradient computation is discussed.

Finally, it should be emphasized tha the choice of numerical integration method for solving the extended state space equations is not trivial and depends on the specific problem at hand. For instance a stiff problem depends mainly on the stability properties of the method, while for a nonstiff problem the accuracy of the integration method is the main concern, see e.g. (Hairer et al. 1987a, Hairer, Nörsett & Wanner 1987b). In this work only nonstiff problems have been considered. Optimizing a stiff problem the use of a direct collocation method will often be preferred to the shooting methods.

2.3.2 Numerical Gradient Calculations by Forward Differences

The partial derivatives of the function $\mathcal{F}(\mathbf{p})$ with respect to the parameters $\mathbf{p}$ can be calculated numerically by forward differences

$$\frac{\partial \mathcal{F}(\mathbf{p})}{\partial p_j} = \frac{\mathcal{F}(p_1, ..., p_j + \delta_j, ..., p_N) - \mathcal{F}(\mathbf{p})}{\delta_j} + \epsilon_{FD} \tag{2.37}$$

where δ_j is a small perturbation of p_j called the finite difference interval and ϵ_{FD} is the error in the forward difference scheme. Forward differences was used in CVP with single shooting by Speyer et al. (1971).

The error ϵ_{FD} in the forward difference scheme is given by, see e.g. (Gill et al. 1981)

$$\epsilon_{FD} = \alpha_1 \delta_j + \alpha_2 \frac{1}{\delta_j} \tag{2.38}$$

The first term on the right hand side of Eq. (2.38), the truncation error, is due to the truncation of the Taylor series using the forward difference scheme. The second term, the condition error, is due to the error in computing $\mathcal{F}$. This means that α_1 equals the second derivative in order of magnitude, while α_2 is proportional to the absolute error of the computed $\mathcal{F}$.

It is a common design rule that if a function is of the same order of magnitude as its second derivative, in general, the number of correct digits in the forward difference approximation of the derivative of a function is half the number of correct digits in the function itself, provided (Gill et al. 1981)

$$\delta \simeq \sqrt{\epsilon_R}$$

Here, δ is the optimal perturbation and ϵ_R is the relative error in the function calculation.

When the partial derivatives of $\mathcal{F}$ are computed by forward differences the number of correct digits that can be expected in the partial derivatives is less than half the number of correct digits that can be expected in $\mathcal{F}$. This suggests that if the initial value problem is not solved very accurately, and if the forward difference scheme is used as a black box differentiator, the estimates of the partial derivatives will be rather crude.

Fortunately, this deficiency can be avoided by using the accuracy analysis in the previous subsection, and instead of considering forward differences of $\mathcal{F}$ forward differences of $\bar{\mathcal{F}}$ should be considered, i.e.

$$\begin{aligned} &\frac{\partial \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM})}{\partial p_j} = \\ &\frac{\bar{\mathcal{F}}(p_1, ..., p_j + \delta_j, ..., p_N, h_{seq}; f_{IM}) - \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM})}{\delta_j} + \bar{\epsilon}_{FD} \\ &= \frac{\partial \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM})}{\partial p_j}|_{FD} + \bar{\epsilon}_{FD} \end{aligned} \tag{2.39}$$

where $\bar{\epsilon}_{FD}$ is the forward difference error for $\bar{\mathcal{F}}$. If the same step size sequence is used for the nominal and all the perturbed solutions of $\bar{\mathcal{F}}$, the condition error in the forward difference computation will be in the same order of magnitude as the error in computing $\bar{\mathcal{F}}$. Since this error is very small compared to the accuracy specifications, the forward difference interval can be assumed to be so large that the total error will be dominated by the truncation error

$$\bar{\epsilon}_{FD} = O(\delta_j) \tag{2.40}$$

This means that the partial derivatives of $\bar{\mathcal{F}}$ can be computed very accurately. Given this accuracy and the accuracy to which $\bar{\mathcal{F}}$ approximates $\mathcal{F}_M$ an expression for the accuracy to which the partial derivatives of $\bar{\mathcal{F}}$ approximate the partial derivatives of $\mathcal{F}_M$ can be found, i.e. from Eqs. (2.39 and 2.36)

$$\frac{\partial \mathcal{F}_M(\mathbf{p})}{\partial p_j} = \frac{\partial \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM})}{\partial p_j}|_{FD} + \bar{\epsilon}_{FD} + \sum_{i=1}^{n_s} \frac{\partial \varepsilon_i^M(\mathbf{p})}{\partial p_j} h_i^{p_{oI}+1} \tag{2.41}$$

This expression shows that if the forward difference perturbations are small enough and the functions $\bar{\mathcal{F}}$ and $\mathcal{F}_M$ are smooth the accuracy to which the partial derivatives of $\mathcal{F}_M$ are computed is in the same order of magnitude as the accuracy to which $\mathcal{F}_M$ is computed. Horn (1990) also shows that partial derivatives of the functions in CVP with single shooting can be computed to approximately the same accuracy as the functions themselves by forward differences integrating all the perturbation trajectories together with the nominal, using the same step sizes for all trajectories. Similar observations have been made by Bock (1983) in connection with parameter estimation problems. However, in (Horn 1990, Bock 1983) only the accuracies to which the original function $\mathcal{F}$ and its gradients can be computed are considered. In this work the mathematical representation of $\mathcal{F}$, $\mathcal{F}_M$, which is the important one in optimization, is of main concern. In both cases smoothness of $\mathcal{F}$ with respect to the parameters is a condition for the accuracy of the partial derivatives to be in the same order of magnitude as the function.

The fact that the time steps have to be the same in the perturbed and the unperturbed solutions obviously makes it difficult to compute derivatives with respect to free time nodes. As mentioned normalization of the time with respect to t_f is very convenient in the case of free t_f, because then perturbing the final time does not impose the perturbation of a time node, but rather a scaling factor. If the control time nodes are free, but equal for all controls, a scaling approach can obviously also simplify the computation of partial derivatives with respect to control time nodes. If the control time grids for the different controls are different this is, however, not possible. Then, the perturbation of the time nodes are constrained by the step sizes in the integrator.

Computing gradients in CVP-methods are as mentioned generally computationally expensive, and the forward difference scheme (Eq. 2.39) is no exception. However, the computational costs can be reduced by using the fact that the parameters in $\mathbf{p}_u$ and $\mathbf{p}_t$ only affect parts of the trajectory. For instance, for a control parameter that starts to affect a control at control time node number j, the perturbed trajectory for that parameter needs only to be integrated from t_j to the end of the respective shooting interval. This also means that the perturbed trajectories for parameters in $\mathbf{p}_u$ and $_{\mathbf{t}}$ are generally much cheaper to compute in multiple than in single shooting, and even if the number of parameters is higher in multiple than in single shooting the total gradient computation may be cheaper.

2.3.3 Analytical Gradient Calculations Using State Sensitivity Equations

Partial derivatives of functions defined by state space equations with respect to parameters can be computed using the sensitivity equations of the state space equations. A gradient calculation scheme using the state sensitivity equations in CVP with shooting will now be presented. The partial derivative of $\mathcal{F}$ as it is defined in Eq. (2.33) with respect to the parameter p_j can be written

$$\begin{aligned}\frac{\partial \mathcal{F}(\mathbf{p})}{\partial p_j} = \quad & \frac{\partial \varphi}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial p_j}(t_c) + \frac{\partial \varphi}{\partial p_j} + \int_0^{t_c}(\frac{\partial \mathcal{L}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial p_j}(t) + \\ & \frac{\partial \mathcal{L}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial p_j}(t) + \frac{\partial \mathcal{L}}{\partial p_j})dt \end{aligned} \tag{2.42}$$

The problem of finding the partial derivatives in Eq. (2.42) except $\frac{\partial \mathbf{x}}{\partial p_j}$ is straightforward.

Now, differentiate both sides of the dynamic equations (2.18) with respect to p_j:

$$\frac{\partial \dot{\mathbf{x}}}{\partial p_j} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial p_j} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial p_j} \tag{2.43}$$

Because $\dot{\mathbf{x}}$ is a function of the independent variables t and p_j, $j = 1, .., N$ (the p_j's are independent of t)

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt} = \frac{\partial \mathbf{x}}{\partial t} \tag{2.44}$$

and

$$\frac{d(\frac{\partial \mathbf{x}}{\partial p_j})}{dt} = \frac{\partial(\frac{\partial \mathbf{x}}{\partial p_j})}{\partial t} = \frac{\partial(\frac{\partial \mathbf{x}}{\partial t})}{\partial p_j} = \frac{\partial(\frac{d\mathbf{x}}{dt})}{\partial p_j} = \frac{\partial \dot{\mathbf{x}}}{\partial p_j} \tag{2.45}$$

Define

$$\mathbf{y}_j = \frac{\partial \mathbf{x}}{\partial p_j} \tag{2.46}$$

The partial derivative of $\mathbf{x}$ with respect to p_j is now given by the following set of linear time-varying differential equations (linear in $\mathbf{y}_j$ and $\frac{\partial \mathbf{u}}{\partial p_j}$):

$$\dot{\mathbf{y}}_j = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{y}_j + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial p_j} \tag{2.47}$$

$$\mathbf{y}_j(0) = \frac{\partial \mathbf{x}_0}{\partial p_j}$$

The partial derivatives are now found by solving Eq. (2.47) for each parameter ($j = 1, .., N$) and using $\mathbf{y}_j = \frac{\partial \mathbf{x}}{\partial p_j}$ in finding the partial derivatives in Eq. (2.42). It is easily seen that $\mathbf{y}_j$ has to be integrated for the same part of the trajectory as the perturbed solution of the states in the forward difference approach. It is also seen that the Lagrangian functional part of Eq. (2.42) can be integrated together with Eq. (2.47), just as for the extended state space equations.

Also now, of course, the function $\bar{\mathcal{F}}$ is computed rather than $\mathcal{F}$. Now, instead of differentiating Eq. (2.33) and Eq. (2.18) leading first to Eq. (2.43) and then Eq. (2.47), the difference equations resulting from the numerical integration of the extended state space equations are differentiated. It is easily seen that this leads to the same difference equations as if the same numerical integration method f_{IM} and the same step size sequence h_{seq} as used for solving the state space equations had been used to solve Eqs. (2.47) and (2.42). From Eq. (2.36) it is seen that the partial derivative of $\mathcal{F}_M$ with respect to p_j is given by

$$\frac{\partial \mathcal{F}_M(\mathbf{p})}{\partial p_j} = \frac{\partial \bar{\mathcal{F}}(\mathbf{p}, h_{seq}; f_{IM})}{\partial p_j} + \sum_{i=1}^{n_s} \frac{\partial \varepsilon_i^M(\mathbf{p})}{\partial p_j} h_i^{p_{oI}+1} \tag{2.48}$$

which means that the forward difference computations of the previous subsection is an approximation to this method, and that the approximation is very close.

The need to derive analytical expressions makes this method less flexible than the forward difference approach. However, computational costs can be reduced because the coefficient matrices in Eq. (2.47) for a given time t are equal for all the parameters.

The use of state sensitivity equations was proposed in dynamic response optimization of mechanical and structural systems by (Hsieh & Arora 1984), and introduced in an OCP formulation in (Tseng & Arora 1989). An accuracy analysis similar to the one above is, however, not provided.

2.3.4 Analytical Gradient Calculations Using the Costate Equations

Instead of computing the partial derivatives of the states with respect to the gradients as in Eq. (2.47), the state space equations can be treated as constraints. That is, the partial derivatives of $\mathcal{F}$ with respect to the parameters on the constraint manifold defined by the state space equations (2.18) are considered.

This gradient calculation approach was first proposed and described generally by Brusch & Peltier (1973). It has later been described in different ways by several authors. In (Sirisena 1973, Sirisena & Tan 1974) the technique is described for

a special implementation, while more general descriptions are given in e.g. (Kraft 1980, Teo, Goh & Wong 1991). The principles and the notations in the derivation of the method is very similar to those for the derivation of the necessary conditions for optimality for the original OCP, see e.g. (Athans & Falb 1966, Bryson & Ho 1975). The resulting CVP-method is usually referred to as indirect, because the costate equations are needed.

The goal now is to find the derivatives of $\mathcal{F}$ with respect to all the parameters while the state space equations (2.18) (dynamical constraints) are satisfied. This can be done by multiplying the dynamical constraints with Lagrange multipliers and defining for $\mathcal{F}$ a Hamiltonian $\mathcal{H}$ (similar to what is done when the necessary conditions for optimality are to be found):

$$\mathcal{H}(t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}), \mathbf{p}_d, \boldsymbol{\lambda}(t)) = \mathcal{L}(t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}), \mathbf{p}_d) + \boldsymbol{\lambda}(t)^{\mathrm{T}} \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}), \mathbf{p}_d) \tag{2.49}$$

The final computational scheme for finding the partial derivatives of $\mathcal{F}$ with respect to the parameters $\mathbf{p}$, which are summarized here, are derived in different ways by Brusch & Peltier (1973), Teo et al. (1991) and Kraft (1980). In the latter the optimization theory described in (Luenberger 1969) is used.

First, the Lagrange multiplier functions are found by integrating the set of costate differential equations

$$\dot{\boldsymbol{\lambda}}(t) = -\left(\frac{\partial \mathcal{H}(t, \mathbf{x}(t), \mathbf{u}(t, \mathbf{p}), \mathbf{p}_d, \boldsymbol{\lambda}(t))}{\partial \mathbf{x}}\right)^{\mathrm{T}} \tag{2.50}$$

$$\boldsymbol{\lambda}(t_c) = \left(\frac{\partial \boldsymbol{\varphi}(t_c, \mathbf{x}(t_c))}{\partial \mathbf{x}}\right)^{\mathrm{T}}$$

backwards in time from $t = t_c$ to $t = 0$.

Next, the partial derivatives, using the Lagrange multiplier functions from Eq. (2.50), are computed

$$\frac{\partial \mathcal{F}(\mathbf{p})}{\partial p_j} = \frac{\partial \boldsymbol{\varphi}}{\partial p_j} + \boldsymbol{\lambda}^{\mathrm{T}}(0) \frac{\partial \mathbf{x}_0}{\partial p_j} + \int_0^{t_c} \frac{\partial \mathcal{H}}{\partial p_j} dt \tag{2.51}$$

For the objective function and each of the nonlinear constraints, one set of costate equations are integrated backwards using the nominal $\mathbf{x}(t)$ found by forward integration of the state space equations (2.18).

It can be seen that the computational effort needed for solving Eq. (2.51) is very much like the computational effort needed for solving Eq. (2.42). It can also be seen that the computational effort needed for integrating the set of costate equations is very much like the computational effort needed for integrating the set of state sensitivity equations (Eq. 2.47). One difference is that while the costate equations

are integrated $M+1$ times where M is the number of constraints, the state sensitivity equations are integrated N times where N is the number of parameters.

A second difference is that the costate equations have to be integrated backwards in time. Therefore, they can not be integrated in parallel with the state space equations, and dense outputs for the states from the state space equations are needed. The accuracy analysis for the gradient computations also becomes more difficult.

It is difficult to judge which is the better approach of the forward difference/state sensitivity approach and the analytical costate approach because they are quite different. Computational results in (Kraft 1981) suggest that the two approaches are equally fast, but details on the implementation are not provided in the paper. It is no doubt, however, that with the forward difference approach the CVP with shooting will be more flexible, and easier to implement. The parallelity actually also makes the state sensitivity approach more easy to implement than the costate approach. The discussion above on the forward difference/state sensitivity approaches suggests that the attainable accuracy in the NLP solver can be expected to be higher for these approaches than for the costate approach, when the accuracy in the integrators are the same. If the optimal number of shooting nodes are used it is not obvious that the number of parameters will be higher than the number of constraints. If so, the forward difference/state sensitivity approaches are definitely faster. In the costate approach it is, however, possible to reduce the number of constraints by transforming the original NLP to a new equivalent NLP with much fewer constraints, by putting several constraints together to one (Goh & Teo 1988). For instance a set of final time boundary constraints on the states can be transformed to one single sum of error quadratures of the constraints. This will usually make it possible to reduce the number of constraints to considerably below the number of parameters. This will make the costate gradient computations faster than forward difference/state sensitivity gradient computations. However, there is no guarantee that this new NLP can be solved as efficiently and robustly as the original, so the total computation do not need to be faster. This is shown to be the case for the OCP in Chapter 4.3, where it is shown that the scrambling of several constraints together to one makes the resulting nonlinear equations more difficult to solve.

When the costate approach is applied, using the integral approach to represent possible path constraints makes the gradient computation of these constraints much cheaper. However, by the integral approach a path constraint is obviously also scrambled. The infinite dimensional path constraint is transformed into one single finite dimensional constraint. Difficulties in the NLP solver can therefore be expected.

The required analytical expressions can be derived without excessive effort using tools like MATHEMATICA and MAPLE. Therefore the argument against the analytical approaches that the derivations of the partial derivatives of the right hand side of the state space equations could be very difficult is not as important anymore as it has been earlier.

2.4 Nonlinear Programming

2.4.1 Introduction

Nonlinear programming (NLP) is discussed in this section. First, the NLP is restated with some theoretical definitions and results which are necessary for the discussion of numerical methods, particularly the first order necessary conditions for optimality. A brief historical review of numerical methods, with focus on the sequential quadratic programming (SQP) method, is then given. In the rest of the section this method is treated in detail. In (Kraft 1980) it was concluded that even though the SQP-methods generally were the best methods for solving moderate size NLPs, slightly better results were achieved by an augmented Lagrangian method in CVP with shooting for solving trajectory optimization problems. However, since that paper was published the SQP methods have been significantly improved. In fact Kraft (1988) implemented an SQP method for use in a direct single shooting method (Kraft 1989). In addition, the availability of commercial SQP implementations are today dominating the market for NLP solvers. In this thesis, for instance, the SQP routine E04VCF from the NAG Fortran library (NAG 1991), which is essentially the same as the routine NPSOL (Gill, Murray, Saunders & Wright 1986b) and the SQP Fortran routine SLSQP (Kraft 1989), have been applied.

In consistency with the NLP literature the vector of variables denoted by $\mathbf{x}$ will be used instead of the parameter vector $\mathbf{p}$. A general NLP can be written in the following form (Gill, Murray, Saunders & Wright 1985)

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{R}^n} F(\mathbf{x}) \\ &\mathbf{x}_{min} \le \mathbf{x} \le \mathbf{x}_{max} \\ &\boldsymbol{\ell}_{lc} \le \mathbf{A}_l\mathbf{x} \le \mathbf{u}_{lc} \\ &\boldsymbol{\ell}_{nc} \le \mathbf{c}(\mathbf{x}) \le \mathbf{u}_{nc} \end{aligned} \tag{2.52}$$

where $F(\mathbf{x})$ is a scalar objective function to be minimized, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ define the lower and upper box constraints for the optimizable parameters $\mathbf{x}$, $\mathbf{A}_l$ is an $m_l \times n$ matrix defining a set of linear combinations of the parameters to be constrained between a lower bound $\boldsymbol{\ell}_{lc}$ and an upper bound $\mathbf{u}_{lc}$, and $\mathbf{c}(\mathbf{x})$ defines m_n nonlinear constraints to be bounded between a lower bound $\boldsymbol{\ell}_{nc}$ and an upper bound $\mathbf{u}_{nc}$. Equality constraints are naturally included by letting the respective upper and lower bounds be equal to each other.

The form of the nonlinear program has, in addition to being general, the property that the three different types of constraints, box constraints on the parameters, linear and nonlinear constraints, are formulated separately. This is very convenient in the numerical solution of the problem where it is important to utilize the differences between these types of constraints. In a theoretical treatment and analysis of nonlinear programming it may be more convenient to formulate the general problem as

$$\min_{\mathbf{x}\in\mathbf{R}^n} F(\mathbf{x})$$
$$\mathbf{c}_E(\mathbf{x}) = \mathbf{0}$$
$$\mathbf{c}_I(\mathbf{x}) \geq \mathbf{0} \tag{2.53}$$

where $\mathbf{c}_E(\mathbf{x})$ is an m_E-dimensional vector of general, possibly nonlinear, equality constraints and $\mathbf{c}_I(\mathbf{x})$ is an m_I-dimensional vector of general, possibly nonlinear, inequality constraints.

The Lagrangian function L for the problem in Eq. (2.53) is defined as

$$L(\mathbf{x},\boldsymbol{\lambda}) = F(\mathbf{x}) - \begin{pmatrix} \boldsymbol{\lambda}_E^{\mathrm{T}} & \boldsymbol{\lambda}_I^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \mathbf{c}_E(\mathbf{x}) \\ \mathbf{c}_I(\mathbf{x}) \end{pmatrix} = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{c}(\mathbf{x}) \tag{2.54}$$

where $\boldsymbol{\lambda}$ is an $m = m_E + m_I$-dimensional vector of Lagrange multipliers.

Now, define the Jacobian $\mathbf{C}$ of the constraints as

$$\mathbf{C}_E(\mathbf{x}) = \frac{\partial \mathbf{c}_E}{\partial \mathbf{x}} = \nabla_{\mathbf{x}}\mathbf{c}_E$$
$$\mathbf{C}_I(\mathbf{x}) = \frac{\partial \mathbf{c}_I}{\partial \mathbf{x}} = \nabla_{\mathbf{x}}\mathbf{c}_I$$
$$\mathbf{C}(\mathbf{x}) = \begin{pmatrix} \mathbf{C}_E(\mathbf{x}) \\ \mathbf{C}_I(\mathbf{x}) \end{pmatrix} \tag{2.55}$$

and the gradient $\mathbf{g}^{\mathrm{T}}$ of the objective function

$$\mathbf{g}^{\mathrm{T}}(\mathbf{x}) = \nabla_{\mathbf{x}}F(\mathbf{x}) = \frac{\partial F(\mathbf{x})}{\partial \mathbf{x}} \tag{2.56}$$

Assume further the definition of the **active set** for a local solution $\mathbf{x}^*$ to be the set of all equality constraints and the inequality constraints that are satisfied as equalities at $\mathbf{x}^*$.

The vector $\mathbf{x}$ defines a **regular point** if it satisfies a certain regularity assumption on the constraints which here will be assumed to be the linear independency of the constraints in the active set, that is the Jacobian $\mathbf{C}_A$ of the constraints in the active set has full rank (see e.g. (Fletcher 1987) for details).

The gradient and the Hessian of the Lagrangian are respectively defined as

$$\nabla_{\mathbf{x}}L(\mathbf{x},\boldsymbol{\lambda}) = \frac{\partial L}{\partial \mathbf{x}} \tag{2.57}$$

$$\mathbf{W}(\mathbf{x}) = \frac{\partial^2 L}{\partial \mathbf{x}^2} \tag{2.58}$$

The first order necessary conditions for optimality, also called the Kuhn-Tucker conditions due to Kuhn & Tucker (1951) can be formulated as (Fletcher 1987)

Theorem 2.1 *If $\mathbf{x}^*$ is a local minimizer to the NLP (2.53) and certain regularity assumptions on the constraints hold at $\mathbf{x}^*$, then there exist Lagrange multipliers $\boldsymbol{\lambda}^*$ such that $\mathbf{x}^*$, $\boldsymbol{\lambda}^*$ satisfy the following system:*

$$\begin{aligned}
\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \nabla_{\mathbf{x}} F(\mathbf{x}^*) - \boldsymbol{\lambda}_E^{*\mathrm{T}} \mathbf{C}_E(\mathbf{x}^*) - \boldsymbol{\lambda}_I^{*\mathrm{T}} \mathbf{C}_I(\mathbf{x}^*) = \mathbf{0} \\
\mathbf{c}_E(\mathbf{x}^*) = \mathbf{0} \\
\mathbf{c}_I(\mathbf{x}^*) \geq \mathbf{0} \\
\boldsymbol{\lambda}_I^* \geq \mathbf{0} \\
\lambda_i^* c_i(\mathbf{x}^*) = 0 \quad i = 1, .., m
\end{aligned} \tag{2.59}$$

where λ_i^* is the i'th element of $\boldsymbol{\lambda}^*$ and c_i is the i'th element of $\mathbf{c} = \begin{pmatrix} \mathbf{c}_E^{\mathrm{T}} & \mathbf{c}_I^{\mathrm{T}} \end{pmatrix}^{\mathrm{T}}$

Necessary second order conditions for optimality are that in addition to Eq. (2.59), for all feasible directions $\mathbf{v}$, i.e. directions which are consistent with the constraints, it holds that

$$\mathbf{v}^{\mathrm{T}} \mathbf{W}^* \mathbf{v} \geq \mathbf{0} \tag{2.60}$$

where $\mathbf{W}^*$ is the Hessian of the Lagrangian function at the Kuhn-Tucker pair $\mathbf{x}^*, \boldsymbol{\lambda}^*$. A sufficient condition for optimality is that Eq. (2.60) holds with inequality only.

The first order conditions assume continuous differentiability of all the functions, while the second order conditions assume smooth second derivatives. Smoothness of the second derivatives are almost always assumed when local convergence properties of numerical algorithms are proved.

This short description of the theory of nonlinear programming is included because it is needed for the description and the discussion of the numerical algorithms. Eq. (2.59) are particularly important. A more comprehensive treatment of this theory can be found in modern textbooks (Fletcher 1987, Luenberger 1984).

Until the end of the 1960's the numerical methods for solving NLP's were dominated by the so-called Penalty Function (PF) and Barrier Function (BF) methods (Powell 1978b). These methods are discussed in e.g. the textbooks (Gill et al. 1981, Luenberger 1984, Fletcher 1987). These methods were popular tools in CVP-methods until the early seventies.

The numerical difficulties of these methods lead to the development of alternative methods like the Augmented Lagrangian (AL) methods, e.g. (Rockafeller 1970,

Rockafeller 1973) and the Reduced Gradient (RG) Methods, e.g. (Sargent 1974). The RG-methods, or more generally the Projected Gradient (PG) methods, try to keep the constraints satisfied at all iterations, and are therefore said to be working in the space of feasible solutions (Luenberger 1984). It is cumbersome to satisfy the nonlinear constraints at every iteration, and therefore methods have been proposed which solve iteratively the NLP with linearized constraints in each iteration (Rosen & Kreuser 1972, Robinson 1972, Buckley 1975). The NLPs with linear constraints can then be solved by a PG-like method

The PG-like and the AL-methods consist of so-called major and minor iterations, which is also the case for the PF and BF-methods. The start of a new major iteration depends on the successful convergence of a minor iteration. In the PG-like methods the minor iterations either solve nonlinear equations or NLPs with linear constraints. The latter is in fact a combination of unconstrained optimization and linear algebra (see Chapter 5 of (Gill et al. 1981)). In the AL-methods the Lagrangian function is augmented by a penalty term resulting in an augmented Lagrangian function. In the minor iterations an unconstrained minimization of the augmented Lagrangian function is performed, while in the major iterations the Lagrange multipliers and the penalty parameter are adjusted. The AL-methods are often considered as a modification of the PF-methods since the Lagrange multiplier term was included to improve the PF-methods. However, the methods can also be viewed in the context of dual theory. Dual theory says that the problem of finding an optimal solution is a maximization problem with respect to the Lagrange multipliers. Therefore, in the major iterations of an AL-method the adjustment of the Lagrange multipliers can be performed in order to maximize the augmented Lagrangian function with respect to the Lagrange multipliers. The penalty term is added to the dual problem in order to make the method robust. A particularly good description of AL-methods both considered as dual and PF-methods is given in (Luenberger 1984). In (Hestenes 1979) a historical review of AL-methods are given.

In the Sequential Quadratic Programming (SQP) methods, also called projected Lagrangian or recursive programming methods, a Quadratic Programming (QP) problem is solved in each iteration. This method, therefore has the very significant advantage that it does not depend on the convergence of a minor problem, since the QP subproblem, if it is solvable at all, is solved exactly in a finite number of iterations. For equality constraints only, just one iteration is needed. The first SQP method (the SOLVER method) was proposed by Wilson (1963) and further interpreted by Beale (1967). For a long time this method was not very much in use, because it dependeds on the computation of the Hessian matrix of the Lagrangian function. Han (1976) proposed to use Quasi-Newton approximations to the Hessian in SQP-methods, and proved that superlinear convergence can be achieved, for instance with a Davidon-Fletcher-Powell (DFP) update, provided that the Hessian is positive definite. In (Han 1977) the convergence of these methods are made global by introducing step size control using the Absolute value penalty function as a so-called merit function

$$M(\mathbf{x},\rho) = F(\mathbf{x}) + \rho \sum_{i=1}^{m_{n_E}} |c_{E,i}| + \rho \sum_{i=1}^{m_{n_I}} \max(0, c_{I,i}) \tag{2.61}$$

which has the property that for ρ sufficiently large $\mathbf{x}^*$ is an unconstrained minimum of $M(\mathbf{x},\rho)$.

In the survey paper (Powell 1978a) the SQP method of Wilson and Han is compared to AL-methods, which Powell until 1976 considered the best method for solving NLP. The fact that the SQP methods become Newton iterations in the limit, combined with Han's new results convinced him that the SQP-method would become the superior one. The Quasi-Newton approximation to the Hessian of the Lagrangian was proposed to be updated by a BFGS-scheme, see e.g. (Luenberger 1984), which is modified such that positive definiteness is preserved even if the Hessian itself is indefinite. Good numerical results were achieved. In (Powell 1978b) superlinear convergence was proved, provided that the approximation to the Hessian of the Lagrangian projected into the active set converges to the true Hessian of the Lagrangian projected into the active set. Further discussions on global convergence using the absolute value penalty function for step-size selection are given in (Powell 1978c). In (Biggs 1975, Biggs 1978) a Quasi-Newton based SQP method is also proposed, but here the SQP is applied directly to a penalty function.

The local convergence proofs of Han and Powell are based on the assumption that as the algorithm converges to a solution the step size converges to one. However, the exact penalty function, used as a merit function, suffers from the so-called Maratos effect (Maratos 1978), which means that the step size is not guaranteed to converge to unity, and from cycling (Chamberlain 1979). To overcome these problems Schittkowski (1981) proposed to replace the Absolute value penalty function as merit function with the augmented Lagrangian function

$$\begin{aligned} M(\mathbf{x},\boldsymbol{\lambda},\rho) \;=\; & F(\mathbf{x}) - \boldsymbol{\lambda}_E^{\mathrm{T}}\mathbf{c}_E(\mathbf{x}) + \frac{1}{2}\rho\mathbf{c}_E(\mathbf{x})^{\mathrm{T}}\mathbf{c}_E(\mathbf{x}) - \\ & \sum_{i=1}^{m_{n_I}} \begin{cases} \lambda_i \mathbf{c}_{I,i}(\mathbf{x}) - \frac{1}{2}\rho\mathbf{c}_{I,i}(\mathbf{x})^2 & \text{if } \mathbf{c}(\mathbf{x})_{I,i} \le \frac{\lambda_i}{\rho} \\ \frac{\lambda_i}{2\rho} & \text{otherwise} \end{cases} \end{aligned} \tag{2.62}$$

where ρ is a non negative penalty parameter, and $\boldsymbol{\lambda}$ consists of the current estimates of the Lagrange multipliers, treated as variables in the merit function like $\mathbf{x}$. There exists a finite $\hat{\rho}$ such that for all $\rho \ge \hat{\rho}$ $\mathbf{x}^*$ is an unconstrained minimum of $M(\mathbf{x},\boldsymbol{\lambda}^*,\rho)$. In (Schittkowski 1983) Eq. (2.62) is modified to use individual penalty parameters for each of the constraints. Global convergence results are given in both papers. A Fortran subroutine NLPQL based on the algorithm in (Schittkowski 1983) was presented in (Schittkowski 1985).

For the inequality constraints that are active at the solution the augmented Lagrangian function in Eq. (2.62) is not continuously differentiable, which may cause difficulties for certain line-search techniques based on polynomial interpolation. To

overcome this problem the augmented Lagrangian merit function due to Schittkowski has been modified by introducing slack variables in (Gill et al. 1985, Gill et al. 1986b, Gill, Murray, Saunders & Wright 1986a). This merit function is treated below under the presentation of sequential quadratic programming (SQP) methods.

The most usual SQP approach is a so-called IQP strategy, where inequality constrained quadratic programs are solved in each iteration. This is also the approach to be presented here. Alternatively, an EQP strategy may be used, where instead of inequality constrained QP's, equality constrained QP subproblems are solved. This means that an active set strategy (defined later) for finding the active set is performed outside instead of inside the quadratic program. An important feature of the EQP strategy is that only the projected Hessian of the Lagrangian is needed in the quadratic program. Works on updating a quasi-Newton approximation of the projected Hessian for nonlinearly constrained optimization problems can be found in e.g. (Colemann & Conn 1982, Gabay 1982, Nocedale & Overton 1983). EQP and IQP strategies are compared in (Gill et al. 1985), and it is concluded that in some cases an EQP strategy may be more efficient than the more usual IQP strategy.

2.4.2 Sequential Quadratic Programming (SQP)

Only IQP methods are treated. The SQP methods to be discussed are based on the implementations in the NAG Fortran library (NAG 1991) and the NPSOL Fortran routine (Gill et al. 1986b), which treats the NLP in the form given in Eq. (2.52). It will, however sometimes be convenient to refer to the form in Eq. (2.53). As will become clear, inside the subproblem (QP) all the constraints in Eq. (2.52) are of interest, while outside the subproblem the general linear and box constraints are always satisfied.

Other main references for this presentation of SQP are (Gill, Murray, Saunders & Wright 1984, Gill et al. 1985, Gill et al. 1986a).

By expanding the NLP (Eq. 2.53) in a Taylor series at the point $\mathbf{x}$, and neglecting all terms of order higher than two for the Objective function F and all terms of order higher than unity for the constraints c, the NLP can be approximated by the following quadratic program (QP)

$$\begin{aligned} \min_{\mathbf{d}\in\mathbf{R}^n} F(\mathbf{x}+\mathbf{d}) &\approx \min_{\mathbf{d}\in\mathbf{R}^n} F(\mathbf{x}) + \mathbf{g}^{\mathrm{T}}(\mathbf{x})\mathbf{d} + \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{W}(\mathbf{x})\mathbf{d} \\ \mathbf{c}_E(\mathbf{x}+\mathbf{d}) &\approx \mathbf{c}_E(\mathbf{x}) + \mathbf{C}_E(\mathbf{x})\mathbf{d} = \mathbf{0} \\ \mathbf{c}_I(\mathbf{x}+\mathbf{d}) &\approx \mathbf{c}_I(\mathbf{x}) + \mathbf{C}_I(\mathbf{x})\mathbf{d} \geq \mathbf{0} \end{aligned} \tag{2.63}$$

where $\mathbf{d}$ are perturbations of the variables $\mathbf{x}$. A subproblem in the form of Eq. (2.63) are solved at the point $\mathbf{x}^k$ in every iteration k. Ignoring the term which is independent of $\mathbf{d}$ in the approximate objective function, not explicitly writing the

dependency on $\mathbf{x}^k$, and replacing the Hessian $\mathbf{W}^k$ of the Lagrangian function by a Quasi-Newton approximation $\mathbf{H}^k$, the subproblem takes the form

$$\begin{aligned} \min_{\mathbf{d}^k \in \mathbf{R}^n} & \frac{1}{2}(\mathbf{d}^k)^{\mathrm{T}}\mathbf{H}^k\mathbf{d}^k + (\mathbf{g}^k)^{\mathrm{T}}\mathbf{d}^k \\ & \mathbf{C}_E^k\mathbf{d}^k + \mathbf{c}_E^k = \mathbf{0} \\ & \mathbf{C}_I^k\mathbf{d}^k + \mathbf{c}_I^k \geq \mathbf{0} \end{aligned} \tag{2.64}$$

The necessary conditions for optimality for this QP are

$$\begin{aligned} & \mathbf{H}^k\mathbf{d}^k + \mathbf{g}^k - (\mathbf{C}^k)^{\mathrm{T}}\boldsymbol{\mu}^k = \mathbf{0} \\ & \mathbf{C}_E^k\mathbf{d}^k + \mathbf{c}_E^k = \mathbf{0} \\ & \mathbf{C}_I^k\mathbf{d}^k + \mathbf{c}_I^k \geq \mathbf{0} \\ & \boldsymbol{\mu}_I^k \geq \mathbf{0} \\ & \mu_i^k(\mathbf{C}_{I,i}^k\mathbf{d}^k + \mathbf{c}_{I,i}^k) = 0 \quad (i = 1, .., m_I) \end{aligned} \tag{2.65}$$

where $\boldsymbol{\mu}$ is the m-dimensional vector of Lagrange multipliers for the approximate QP problem and $\mathbf{C}_{I,i}^k$ is the i'th row of the Jacobian of the inequality constraints. $\boldsymbol{\mu}_I$ is the part of $\boldsymbol{\mu}$ corresponding to the inequality constraints.

It is easily seen that Eq. (2.65) is a linearization of the first order necessary conditions for the original problem (Eq. 2.59). Therefore, for equality constraints only, solving Eq. (2.65) is a Newton iteration for an equality constrained nonlinear program. The SQP method is therefore often called the Newton-Lagrange method (Fletcher 1987).

In the presentation of SQP given here the constraints of the NLP are classified in three different groups, which are treated separately. The box constraints and the linear inequality and equality constraints will be assumed to form a convex set. This means that when a search direction $\mathbf{d}$ is computed in the QP subproblem these two classes of constraints will always be satisfied after a feasible set has been found with respect to these constraints, provided that the nonnegative step length α is less than unity. Outside the QP subproblem, therefore, only the nonlinear constraints are relevant.

Quadratic Programming

The search direction $\mathbf{d}^k$ in SQP is computed by solving a QP subproblem. This subproblem is solved iteratively by solving a finite sequence of equality constrained QP problems, which is solved by solving a set of linear equations.

As mentioned above, at the solution of the NLP the set of constraints satisfied by equality is called the active set. The solution of each QP subproblem is also

associated with an active set. In SQP it is convenient to use the active set from the solution of the QP subproblem for iteration $k-1$ as an initial guess for the active set of the solution of the QP subproblem for iteration k. This because the active sets for the QP subproblems converges to the active set for the NLP, see e.g. (Gill et al. 1981, Fletcher 1987).

For convenience the constraints of the QP-subproblem will be written in the same form as Eq. (2.52), i.e.

$$\begin{aligned} \mathbf{d}_{min} \leq \mathbf{d} \leq \mathbf{d}_{max} \\ \boldsymbol{\ell}_{GL} \leq \mathbf{A}_{GL}\mathbf{d} \leq \mathbf{u}_{GL} \end{aligned} \tag{2.66}$$

where the matrix $\mathbf{A}_{GL}$ consists of the linear constraint matrix $\mathbf{A}_l$ and the Jacobian matrix $\mathbf{C}(\mathbf{x}^k)$ of the nonlinear constraints $\mathbf{c}(\mathbf{x}^k)$. The definition of the bounds in Eq. (2.66) follows easily from the definition of the bounds of the NLP-constraints.

Associated with solving the QP subproblems are a sequence of so-called **working sets**. At each iteration in a QP subproblem the working set is updated, that is constraints are added or deleted from the working set. The working set at the solution of the QP subproblem is the active set of the QP. Since the active set of one QP subproblem is used as the initial working set of the next, and the active sets of the QP subproblems converge to the active set of the NLP, only one iteration in the QP subproblem will usually be necessary near the solution of the NLP. The process of solving QP by a sequence of equality constrained QP's with respect to a sequence of working sets are called an **active set strategy**.

The active set strategy presented here is an active set feasible point QP method, where $\mathbf{d}$ is feasible (the constraints are satisfied), but $\boldsymbol{\lambda}$ is not dual feasible until the solution of the QP. Alternatively, a dual feasible active set strategy can be applied, where $\boldsymbol{\lambda}$ is always dual feasible, but $\mathbf{d}$ is not feasible until the solution of the QP, see e.g. (Goldfarb & Idnani 1983, Powell 1983).

Solving a QP subproblem at a typical iteration will be considered in the proceeding discussion, and the explicit reference to iteration number k will be omitted. The elements of $\mathbf{d}$ will be referred to as variables.

The QP subproblem can be solved in two phases, the feasibility phase and the optimality phase. In the feasibility phase a point $\mathbf{d}$ which is feasible with respect to all the constraints is found, whereas in the optimality phase an optimal solution to the quadratic program is found. The active set strategy is quite similar in the two phases, except for the difference in the definition of the objective functions.

QP problems are efficiently solved by treating box constraints and general linear constraints separately (Gill et al. 1984). To this end define $\mathbf{d}_{FX}$ as the n_{FX}-dimensional vector of variables that in the current working set are fixed to one of their respective bounds, and $\mathbf{d}_{FR}$ as the n_{FR}-dimensional vector of variables free to be optimized. Without loss of generality, write

$$\mathbf{d} = \begin{pmatrix} \mathbf{d}_{FR} \\ \mathbf{d}_{FX} \end{pmatrix}$$

Now, write the constraints in the working set as

$$\mathbf{A}_a\mathbf{d} = \begin{pmatrix} \mathbf{0} & \mathbf{I}_{FX} \\ \mathbf{A}_{FR} & \mathbf{A}_{FX} \end{pmatrix} \begin{pmatrix} \mathbf{d}_{FR} \\ \mathbf{d}_{FX} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix} \tag{2.67}$$

where $\mathbf{A}_a$ is the $m_a \times n$ Jacobian matrix of the m_a constraints in the working set. The active box constraints constitute the first n_{FX} rows of $\mathbf{A}_a$, and the active general linear constraints constitute the last $m_L = m_a - n_{FX}$ rows of $\mathbf{A}_a$. The n_{FR} first columns of $\mathbf{A}_a$ correspond to the free variables whereas the last n_{FX} columns correspond to the fixed. Accordingly $\mathbf{I}_{FX}$ is the $n_{FX} \times n_{FX}$ identity matrix, $\mathbf{A}_{FR}$ and $\mathbf{A}_{FX}$ are the $m_L \times n_{FR}$ and the $m_L \times n_{FX}$ Jacobian matrices of the general linear constraints with respect to the free and the fixed variables respectively. The m_L dimensional vector $\mathbf{b}$ is generally nonzero in the elements corresponding to nonlinear constraints.

Now, define a matrix $\mathbf{Z}$ whose columns spans the nullspace of $\mathbf{A}_a$, i.e. form a basis for the set of vectors orthogonal to the matrix $\mathbf{A}_a$ It is easily seen from Eq. (2.67) that

$$\mathbf{Z} = \begin{pmatrix} \mathbf{Z}_{FR} \\ \mathbf{0} \end{pmatrix} \tag{2.68}$$

where the columns of the $n_{FR} \times n_z$-dimensional matrix $\mathbf{Z}_{FR}$ spans the nullspace of $\mathbf{A}_{FR}$, and $n_z = n_{FR} - m_L = n - m_a$. Further, define the matrix $\mathbf{Q}$ as

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Z} & \mathbf{Y} \end{pmatrix} = \begin{pmatrix} \mathbf{Z}_{FR} & \mathbf{Y}_{FR} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{FX} \end{pmatrix} \tag{2.69}$$

where the columns of the matrix $\mathbf{Y}$ form a basis for the range space of the matrix $\mathbf{A}_a$. This gives

$$\mathbf{A}_a\mathbf{Q} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I}_{FX} \\ \mathbf{0} & \mathbf{T} & \mathbf{A}_{FX} \end{pmatrix} \tag{2.70}$$

where $\mathbf{T} = \mathbf{A}_{FR}\mathbf{Y}_{FR}$ is a full-rank $m_L \times m_L$-matrix. It is possible to use the so-called **TQ-factorization**, such that $\mathbf{T}$ is a reverse triangular matrix, i.e. $\mathbf{T}_{ij} = 0$ for $i + j \leq m_L$ (Gill et al. 1984).

Eq. (2.67) can now be written

$$\mathbf{A}_a\mathbf{Q}\begin{pmatrix}\mathbf{d}_z\\ \mathbf{d}_y\\ \mathbf{d}_{FX}\end{pmatrix}=\begin{pmatrix}\mathbf{0}\\ \mathbf{b}\end{pmatrix} \tag{2.71}$$

where, from Eq. (2.69)

$$\mathbf{d}=\mathbf{Z}\mathbf{d}_z+\mathbf{Y}\begin{pmatrix}\mathbf{d}_y\\ \mathbf{d}_{FX}\end{pmatrix} \tag{2.72}$$

The vector $\mathbf{d}_z$ is of dimension n_z and the vector $\mathbf{d}_y$ is of dimension m_a.

Since the QP subproblem is solved iteratively, define $\delta\mathbf{d}$ as the update direction of the vector $\mathbf{d}$ at a typical minor iteration, such that

$$\bar{\mathbf{d}}=\mathbf{d}+\sigma\delta\mathbf{d} \tag{2.73}$$

where σ is a positive step size to be explained later. Now, for the updated variables $\bar{\mathbf{d}}$ to satisfy the equality constraints of the working set the update direction $\delta\mathbf{d}$ has to be a linear combination of the basis vectors of the nullspace, i.e.

$$\delta\mathbf{d}=\begin{pmatrix}\delta\mathbf{d}_{FR}\\ \mathbf{0}\end{pmatrix}=\begin{pmatrix}\mathbf{Z}_{FR}\delta\mathbf{d}_z\\ \mathbf{0}\end{pmatrix} \tag{2.74}$$

where the vector $\delta\mathbf{d}_z$ is computed such that $\delta\mathbf{d}_{FR}$ is a descent direction for the objective function in the nullspace of the constraints in the current working set. The computation of $\delta\mathbf{d}_z$ depends on whether the respective minor iteration belongs to the feasibility or the optimality phase of the QP-solver.

Both the feasibility phase and the optimality phase generate updates $\delta\mathbf{d}$ in the null space of the working set. However, since the part of the initial working set corresponding to the nonlinear constraints generally are not satisfied, first an initial feasible point has to be found. Therefore, an initial direction

$$\mathbf{d}_0=\begin{pmatrix}\mathbf{d}_{FR,0}\\ \mathbf{0}\end{pmatrix}$$

which satisfies Eq. (2.67) has to be computed. From Eqs. (2.69) and (2.71)

$$\mathbf{T}\mathbf{d}_y=\mathbf{b} \tag{2.75}$$

From the definition in Eq. (2.72) an initial vector $\mathbf{d}_{FR,0}$ which is feasible with respect to the initial working set is now given by

$$\mathbf{d}_{FR,0} = \mathbf{Y}_{FR}\mathbf{T}^{-1}\mathbf{b} \tag{2.76}$$

The feasibility phase can be implemented as a linear program (LP). Consider the sum of infeasibilities

$$v(\mathbf{d}) = \sum_{\mathbf{a}_i^{\mathrm{T}}\mathbf{d}-\mathbf{u}_{lc,i}>0} (\mathbf{a}_i^{\mathrm{T}}\mathbf{d} - \mathbf{u}_{lc,i}) - \sum_{\mathbf{a}_i^{\mathrm{T}}\mathbf{d}-\boldsymbol{\ell}_{lc,i}<0} (\mathbf{a}_i^{\mathrm{T}}\mathbf{d} - \boldsymbol{\ell}_{lc,i}) \tag{2.77}$$

where $\mathbf{a}_i^{\mathrm{T}}$ is row number i of $\mathbf{A}_{GL}$. $v(\mathbf{d})$ is a linear function which is zero at any feasible point, and positive at any infeasible point. A feasible point is generated by minimizing $v(\mathbf{d})$, subject to continuing to satisfy constraints that are already satisfied. In each iteration the free variables in $\mathbf{d}_{FR}$ are changed along a descent direction $\delta\mathbf{d}_{FR}$. The step length σ along $\delta\mathbf{d}_{FR}$ is taken as $\min(\sigma_1, \sigma_2)$, where σ_1 is the maximum step that can be taken without violating one of the satisfied constraints that are not in the current working set, and σ_2 is the step to the furthest currently violated constraint along $\delta\mathbf{d}_{FR}$. In both cases a new constraint is added to the working set, and the working set has to be updated. Several violated constraints may become satisfied. In (Gill et al. 1985) the following search direction is suggested

$$\delta\mathbf{d}_{FR} = -\mathbf{Z}_{FR}\mathbf{Z}_{FR}^{\mathrm{T}}(\nabla_{FR}v(\mathbf{d}))^{\mathrm{T}} \tag{2.78}$$

Note that since an initial vector $\mathbf{d}_0$ which is feasible with respect to the initial working set is computed by Eq. (2.76), the box constraints and the general linear constraints from the NLP will generally not be satisfied in the feasibility phase.

Further discussion on linear programming can be found in e.g. (Luenberger 1984, Fletcher 1987). See also E04UCF, E04NCF and E04NAF from the NAG manual (NAG 1991).

After the feasibility phase, an optimal solution $\mathbf{d}$ of the QP problem is to be found iteratively starting from a feasible point. In each iteration an update direction $\delta\mathbf{d}$ which is feasible with respect to the constraints in the current working set is to be computed such that $\mathbf{d} + \delta\mathbf{d}$ is an optimal point for the current equality constrained QP. The step length σ along $\delta\mathbf{d}_{FR}$ is taken as $\min(\sigma_1, 1)$, where σ_1 is the maximum step that can be taken without violating one of the constraints not in the current working set. This means that if $\sigma_1 < 1$, a new constraint will become active before the solution of the equality constrained QP is reached. The respective constraint is added to the working set, and a new iteration starts with the updated working set. If $\sigma = 1$ an optimum of the current equality constrained QP is found. Then the Lagrange multipliers for the inequality constraints in the working set are tested for optimality. If all the Lagrange multipliers corresponding to inequality constraints at their lower bounds are nonnegative, and all the Lagrange multipliers corresponding to inequality constraints at their upper bounds are nonpositive, the optimal solution of the QP subproblem has been found. If, however, at least one of the Lagrange multipliers corresponding to inequality constraints at their lower bounds is negative,

or at least one of the Lagrange multipliers corresponding to inequality constraints at their upper bounds is positive, the objective function can be reduced by deleting this (or these) inequality constraint(s) from the working set. In an active set strategy all the inequality constraints with non-optimal Lagrange multipliers with absolute value greater than some specified value may be deleted from the working set. Alternatively, however, only the inequality constraint with the non-optimal Lagrange multiplier with the largest absolute value may be deleted from the working set.

The first order necessary conditions for optimality for an equality constrained QP results in the following set of linear equations for the update direction $\delta\mathbf{d}$

$$\mathcal{A}\delta\mathbf{d}_e = -\mathbf{b}_e \tag{2.79}$$

where, the matrix $\mathcal{A}$ is referred to as the Kuhn-Tucker matrix

$$\mathcal{A} = \begin{pmatrix} \mathbf{H} & \mathbf{A}_a^{\mathrm{T}} \\ \mathbf{A}_a & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{H}_{FR} & \mathbf{H}_{OD}^{\mathrm{T}} & \mathbf{0} & \mathbf{A}_{FR}^{\mathrm{T}} \\ \mathbf{H}_{OD} & \mathbf{H}_{FX} & \mathbf{I}_{FX} & \mathbf{A}_{FX}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{I}_{FX} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{FR} & \mathbf{A}_{FX} & \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{2.80}$$

The vector $\delta\mathbf{d}_e$ is

$$\delta\mathbf{d}_e = \begin{pmatrix} \delta\mathbf{d}_{FR} \\ \delta\mathbf{d}_{FX} \\ -\boldsymbol{\mu}_B \\ -\boldsymbol{\mu}_l \end{pmatrix} \tag{2.81}$$

Because the residuals of the constraints in the working set are zero the vector $\mathbf{b}_e$ becomes

$$\mathbf{b}_e = \begin{pmatrix} \mathbf{g}_{FR} + \mathbf{H}_{FR}\mathbf{d}_{FR} + \mathbf{H}_{OD}^{\mathrm{T}}\mathbf{d}_{FX} \\ \mathbf{g}_{FX} + \mathbf{H}_{OD}\mathbf{d}_{FR} + \mathbf{H}_{FX}\mathbf{d}_{FX} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{2.82}$$

Here, the Lagrange multipliers have been partitioned into two, those who correspond to the box constraints ($\boldsymbol{\mu}_B$), and those who correspond to the general linear constraints ($\boldsymbol{\mu}_l$). The Hessian matrix $\mathbf{H}$ has been partitioned into four blocks.

Now, define

$$\mathbf{S} = \begin{pmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{m_a} \end{pmatrix} \tag{2.83}$$

where $\mathbf{I}_{m_a}$ is the $m_a \times m_a$-dimensional identity matrix. The matrix $\mathbf{Q}$ was defined in Eq. (2.69).

Now, assume the transformation

$$\delta\mathbf{d}_e = \mathbf{S}\delta\bar{\mathbf{d}}_e = \mathbf{S}\begin{pmatrix} \delta\mathbf{d}_z \\ \delta\mathbf{d}_y \\ \delta\mathbf{d}_{FX} \\ -\boldsymbol{\mu}_B \\ -\boldsymbol{\mu}_l \end{pmatrix} = \begin{pmatrix} \mathbf{Z}_{FR}\delta\mathbf{d}_z + \mathbf{Y}_{FR}\delta\mathbf{d}_y \\ \delta\mathbf{d}_{FX} \\ -\boldsymbol{\mu}_B \\ -\boldsymbol{\mu}_l \end{pmatrix}$$

$$\mathbf{S}^{\mathrm{T}}\mathcal{A}\mathbf{S}\delta\bar{\mathbf{d}}_e = \begin{pmatrix} \mathbf{Q}^{\mathrm{T}}\mathbf{H}\mathbf{Q} & \mathbf{Q}^{\mathrm{T}}\mathbf{A}_a^{\mathrm{T}} \\ \mathbf{A}_a\mathbf{Q} & \mathbf{0} \end{pmatrix}\delta\bar{\mathbf{d}}_e = -\mathbf{S}^{\mathrm{T}}\mathbf{b}_e \tag{2.84}$$

Since all the constraints in the working set are satisfied as equality constraints in the optimality phase

$$\delta\mathbf{d}_{FX} = \delta\mathbf{d}_y = \mathbf{0} \tag{2.85}$$

Therefore, Eq. (2.84) becomes

$$\begin{pmatrix} \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Z}_{FR} & \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Y}_{FR} & \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{H}OD^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \\ \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Z}_{FR} & \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Y}_{FR} & \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{H}_{OD}^{\mathrm{T}} & \mathbf{0} & \mathbf{T}^{\mathrm{T}} \\ \mathbf{H}_{OD}\mathbf{Z}_{FR} & \mathbf{H}_{OD}\mathbf{Y}_{FR} & \mathbf{H}_{FX} & \mathbf{I}_{FX} & \mathbf{A}_{FX}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{FX} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T} & \mathbf{A}_{FX} & \mathbf{0} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \delta\mathbf{d}_z \\ \mathbf{0} \\ \mathbf{0} \\ -\boldsymbol{\mu}_B \\ -\boldsymbol{\mu}_l \end{pmatrix} = \begin{pmatrix} -\mathbf{Z}_{FR}^{\mathrm{T}}(\mathbf{g}_{FR} + \mathbf{H}_{FR}\mathbf{d}_{FR} + \mathbf{H}_{OD}^{\mathrm{T}}\mathbf{d}_{FX}) \\ -\mathbf{Y}_{FR}^{\mathrm{T}}(\mathbf{g}_{FR} + \mathbf{H}_{FR}\mathbf{d}_{FR} + \mathbf{H}_{OD}^{\mathrm{T}}\mathbf{d}_{FX}) \\ -\mathbf{g}_{FX} - \mathbf{H}_{OD}\mathbf{d}_{FR} - \mathbf{H}_{FX}\mathbf{d}_{FX} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{2.86}$$

Before continuing, it should be observed that

$$\mathbf{q} = \mathbf{g} + \mathbf{H}\mathbf{d} \quad \text{or}$$

$$\begin{pmatrix} \mathbf{q}_{FR} \\ \mathbf{q}_{FX} \end{pmatrix} = \begin{pmatrix} \mathbf{g}_{FR} + \mathbf{H}_{FR}\mathbf{d}_{FR} + \mathbf{H}_{OD}^{\mathrm{T}}\mathbf{d}_{FX} \\ \mathbf{g}_{FX} + \mathbf{H}_{OD}\mathbf{d}_{FR} + \mathbf{H}_{FX}\mathbf{d}_{FX} \end{pmatrix} \tag{2.87}$$

is the gradient of the current equality constrained QP.

The matrices which are explicitly represented in the QP are, in addition to $\mathbf{A}_{FX}$, usually the transformation matrices $\mathbf{Z}_{FR}$, $\mathbf{Y}_{FR}$, $\mathbf{T}$ and $\mathbf{H}_Q = \mathbf{Q}^{\mathrm{T}}\mathbf{H}\mathbf{Q}$. This means that, since $\mathbf{H}$ is not explicitly represented in the QP, the components from $\mathbf{H}\mathbf{d}$ on the right hand side of Eq. (2.86) are not explicitly available. If, however, the vectors $\mathbf{d}_z$ and $\mathbf{d}_y$ such that

$$\mathbf{d} = \begin{pmatrix} \mathbf{Z}_{FR}\mathbf{d}_z + \mathbf{Y}_{FR}\mathbf{d}_y \\ \mathbf{d}_{FX} \end{pmatrix} \tag{2.88}$$

are available, the QP gradient $\mathbf{q}$ is easily obtained. The vectors $\mathbf{d}_z$ and $\mathbf{d}_y$ are particularly easily obtained if the matrix $\mathbf{Q}$ is orthogonal, achieved for instance by Housholder reflections (Golub & van Loan 1989), because then

$$\mathbf{d}_z = \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{d}_{FR} \text{ and } \mathbf{d}_y = \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{d}_{FR} \tag{2.89}$$

Alternatively, observe that the gradient $\mathbf{q}_j$ at iteration j are given by

$$\mathbf{q}_j = \mathbf{g} + \mathbf{H}\mathbf{d}_j = \mathbf{g} + \mathbf{H}(\sum_{i=0}^{j-1} \sigma_i \delta \mathbf{d}_i) = \mathbf{g} + \mathbf{H}(\sum_{i=0}^{j-1} \sigma_i \mathbf{Z}_i \delta \mathbf{d}_{z,i}) \tag{2.90}$$

Therefore, the vector on the right hand side of Eq. (2.86), i.e. the gradient of the current EQP can be updated at each iteration as

$$\mathbf{Q}^{\mathrm{T}}\bar{\mathbf{q}} = \mathbf{Q}^{\mathrm{T}}\mathbf{q} + \sigma \mathbf{Q}^{\mathrm{T}}\mathbf{H}\mathbf{Z} \cdot \delta \mathbf{d}_z \tag{2.91}$$

or

$$\begin{aligned} \mathbf{Z}_{FR}^{\mathrm{T}}\bar{\mathbf{q}}_{FR} &= \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{q}_{FR} + \sigma \mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Z}_{FR} \cdot \delta \mathbf{d}_z \\ \mathbf{Y}_{FR}^{\mathrm{T}}\bar{\mathbf{q}}_{FR} &= \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{q}_{FR} + \sigma \mathbf{Y}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Z}_{FR} \cdot \delta \mathbf{d}_z \\ \bar{\mathbf{q}}_{FX} &= \mathbf{q}_{FX} + \sigma \mathbf{H}_{OD}\mathbf{Z}_{FR} \cdot \delta \mathbf{d}_z \end{aligned} \tag{2.92}$$

In addition to the updates in Eq. (2.92), the vector $\mathbf{Qq}$ has to be updated after each iteration as a result of changes in the working set. It is also necessary to perform these updates in the feasibility phase, even if neither $\mathbf{H}_Q$ nor $\mathbf{q}$ is needed in this phase.

The following algorithm describes one iteration of the feasibility phase of the QP subproblem, starting with $\mathbf{d}$, $\mathbf{Z}_{FR}$, $\mathbf{Y}_{FR}$, $\mathbf{T}$ and $\mathbf{H}_Q$

1. Compute $\delta \mathbf{d}_{FR}$ by solving the set of linear equations

$$\mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{H}_{FR}\mathbf{Z}_{FR}\delta \mathbf{d}_{FR} = -\mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{q}_{FR} \tag{2.93}$$

2. Compute the update $\mathbf{Q}\bar{\mathbf{q}}$ of $\mathbf{Qq}$ either by computing it directly using Eq. (2.88), or update it from Eq. (2.91). One possibility is to use the first approach in the first iterate of the optimality phase, and using the latter approach later.

3. Compute the update of $\mathbf{d}_{FR}$ from Eq. (2.73).

4. If $\sigma = 1$:

   Compute the Lagrange multipliers by solving the set of linear equations

$$\begin{aligned} \mathbf{T}^{\mathrm{T}}\boldsymbol{\mu}_L &= \mathbf{Y}_{FR}^{\mathrm{T}}\bar{\mathbf{q}}_{FR} \\ \boldsymbol{\mu}_B &= \bar{\mathbf{q}}_{FX} - \mathbf{A}_{FX}^{\mathrm{T}}\boldsymbol{\mu}_L \end{aligned} \tag{2.94}$$

   Eq. (2.94) is achieved by using Eq. (2.92) with $\sigma = 1$ and Eq. (2.86).

   If the Lagrange multipliers are optimal: terminate, convergence is achieved. If not, remove one (or more) of the inequality constraints from the working set, and go to step 5.

   If $\sigma < 1$:

   Add the inequality constraint that has become active to the working set, and go to step 5.

5. Update the matrices $\mathbf{Z}_{FR}$, $\mathbf{Y}_{FR}$, $\mathbf{T}$ and $\mathbf{H}_Q$. If a box constraint is added or removed, the columns of $\mathbf{A}_a$ are altered, i.e. a column is moved from $\mathbf{A}_{FR}$ to A_{FX} or the reverse. If, however, a general linear inequality constraint is added or removed, the rows of A_a are altered. Permutation of the matrices are necessary when box constraints are added or removed because of the redefinition of $\mathbf{d}$. The details of the updates of these matrices as a result of changes in the working set are given in (Gill et al. 1984). Because of the changes in the matrix $\mathbf{Q}$, $\mathbf{Qq}$ also has to be updated.

The matrix $\mathbf{H}_Q$ is represented by its upper-triangular Cholesky factor $\mathbf{R}$ (Golub & van Loan 1989), defined such that

$$\mathbf{B}_Q = \mathbf{R}^{\mathrm{T}}\mathbf{R} \tag{2.95}$$

This makes no principal difference to the algorithm, but is very convenient in solving the set of linear equations given in Eq. (2.93).

A very important problem has been excluded from the algorithm above, namely the fact that even if the NLP is well defined and well conditioned, the QP subproblems far from the solution may become both undefined and ill-conditioned.

If the NLP is well defined, a feasible point with respect to the general linear constraints and the box constraints exists (see below for the SQP algorithm). Any inconsistency of a QP subproblem is a result of the linearized nonlinear constraints.

An inconsistent QP subproblem is observed in the feasibility phase if optimal Lagrange multipliers are achieved in the linear program even if the objective function $v(\mathbf{d})$ is positive. The algorithm may then continue until the minimum sum of infeasibilities are found. At this point $|\mu_j| \leq 1$ for $j = 1, .., m_a$, see E04NCF and E04UCF

of (NAG 1991), because constraints corresponding to Lagrange multipliers that are larger than unity are removed from the active set and allowed to be violated.

Several approaches for resolving inconsistent QP subproblems in SQP have been proposed, where most of them generate a search direction designed to minimize a weighted combination of the QP objective function and a function of the residual vector of the constraints. Biggs (1975), who optimizes directly on a penalty function, uses an approach like this. Powell (1978c), Schittkowski (1983) and Tone (1983) introduce extra slack variables in the constraints in order to allow some constraint violations. Penalty terms are introduced in the objective function to minimize these violations. In the algorithm presented in (Schittkowski 1983) these extra terms in the SQP to resolve inconsistency are introduced only when necessary. Resolving inconsistency in QP subproblems is also treated by Fletcher (1987).

In (Gill et al. 1985) a short presentation of approaches for resolving inconsistent QP subproblems is given, and it is stated that an approach in the class above is used in NPSOL. It is, however, not specified any further details in (Gill et al. 1985, Gill et al. 1986b, NAG 1991).

A natural solution would be to add a penalty term to the objective function in the optimality phase, if the objective function $v(\mathbf{d})$ is not zero at the termination of the feasibility phase. Only the set of nonlinear constraints not satisfied at the termination of the feasibility phase would be allowed to be violated, and if these constraints are excluded from the working set, no slack variables will be necessary.

In (Gill et al. 1985) a completely different approach for resolving inconsistency is also proposed, namely to solve the linearized set of constraint equations as a least squares problem.

Another problem in SQP is that even if the NLP is well conditioned at the solution, ill-conditioned QP subproblems far from the solution may occur. In NPSOL and the NAG-routines, see E04NCF of (NAG 1991), the conditioning of the working set is controlled by not allowing constraints that would result in a large value of the estimated condition number of $\mathbf{T}$ in the TQ-factorization to be included in the working set. A cheap condition estimator is used, namely the ratio of the largest to the smallest diagonals of $\mathbf{T}$. The constraints are allowed to be violated by a small number, and of several candidates to be included in the working set, only the one that makes the largest angle with the search direction is included in the working set. Even if the active set of one QP subproblem is well conditioned, the initial working set of the next may become ill-conditioned. It may therefore also be necessary to exclude constraints from the initial working set. More on conditioning of the working set can be found in (Gill et al. 1985, Gill et al. 1981). See also (Fletcher 1987).

Step Size Selection – Merit Function

After the search direction $\mathbf{d}^k$ is computed by the quadratic program a step size α^k has to be computed in order to find an improved solution

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k \tag{2.96}$$

The problem of defining what is an improved solution is not trivial for a constrained problem. Before the final solution is reached, the nonlinear constraints are generally not satisfied, and some kind of compromise between reduction of the objective function and reduction in constraint violations is needed. For this purpose the concept of merit function is defined, which is similar to a penalty function. In (Gill et al. 1986a) an augmented Lagrangian function with slack variables $M(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}, \rho)$ is presented as a merit function

$$\begin{aligned} M(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}, \rho) &= F(\mathbf{x}) - \boldsymbol{\lambda}_E^{\mathrm{T}} \mathbf{c}_E(\mathbf{x}) + \frac{1}{2}\rho \mathbf{c}_E(\mathbf{x})^{\mathrm{T}} \mathbf{c}_E(\mathbf{x}) - \boldsymbol{\lambda}_I^{\mathrm{T}}(\mathbf{c}_I(\mathbf{x}) - \mathbf{s}) \\ &\quad + \frac{1}{2}\rho(\mathbf{c}_I(\mathbf{x}) - \mathbf{s})^{\mathrm{T}}(\mathbf{c}_I(\mathbf{x}) - \mathbf{s}) \end{aligned} \tag{2.97}$$

Here, ρ is a penalty parameter which is constant during the line search, while the vector triple

$$\begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \mathbf{s} \end{pmatrix}$$

is considered as the vector of variables to be updated, that is in each iteration a sufficient decrease in the merit function along the search direction

$$\begin{pmatrix} \mathbf{d} \\ \boldsymbol{\xi} \\ \mathbf{t} \end{pmatrix}$$

is to be found, and at each iteration $k+1$ the variables are updated according to

$$\begin{pmatrix} \mathbf{x}^{k+1} \\ \boldsymbol{\lambda}^{k+1} \\ \mathbf{s}^{k+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}^k \\ \boldsymbol{\lambda}^k \\ \mathbf{s}^k \end{pmatrix} + \alpha^k \begin{pmatrix} \mathbf{d}^k \\ \boldsymbol{\xi}^k \\ \mathbf{t}^k \end{pmatrix} \tag{2.98}$$

The search direction is directly or indirectly achieved from the QP subproblem. The direction of the multipliers $\boldsymbol{\xi}^k$ is given by

$$\boldsymbol{\xi}^k = \boldsymbol{\mu}^k - \boldsymbol{\lambda}^k$$

where $\boldsymbol{\mu}^k$ is achieved from the QP subproblem (Eq. 2.64).

The vector of slack variables $\mathbf{s}$ is introduced to get smooth first derivatives for the augmented Lagrangian function, also for inequality constraints. At the beginning of iteration k the element i of the vector $\mathbf{s}$ is taken as

$$s_i = \begin{cases} \max(0, c_i) & \text{if } \rho = 0 \\ \max(0, c_i - \frac{\lambda_i}{\rho}) & \text{otherwise.} \end{cases} \tag{2.99}$$

For all the other magnitudes given, the vector $\mathbf{s}$ given by Eq. (2.99) yields a minimum for the merit function with respect to the slack variables alone, subject to the restriction $\mathbf{s} \geq \mathbf{0}$. The update $\mathbf{t}$ is given by

$$\mathbf{t} = \mathbf{C}_I\mathbf{d} + \mathbf{c}_I - \mathbf{s} \tag{2.100}$$

such that $\mathbf{s}+\mathbf{t}$ is the residual of the inequality constraints from the QP subproblem, and it follows that

$$\mathbf{s} + \mathbf{t} \geq \mathbf{0} \text{ and } \boldsymbol{\mu}^{\mathrm{T}}(\mathbf{s} + \mathbf{t}) = 0$$

The penalty parameter ρ is constant during the line search, and possibly updated at the beginning of each iteration. In (Gill et al. 1986a) theory on global convergence of the SQP algorithm is presented. The procedure for updating the penalty parameter is based directly on this theory and on the object of keeping it as low as possible. According to Eq. (2.98) the merit function can be considered as a function of the step length α, where ρ is a parameter. Therefore, define

$$\phi(\alpha, \rho) = M(\mathbf{x} + \alpha\mathbf{d}, \boldsymbol{\lambda} + \alpha\boldsymbol{\xi}, \mathbf{s} + \alpha\mathbf{t}, \rho) \tag{2.101}$$

The theory is based on the existence of a $\hat{\rho}$ such that the derivative of ϕ with respect to α at $\alpha = 0$ is strictly negative for all $\rho \geq \hat{\rho}$. Furthermore, a step bounded away from zero exists such that a sufficient decrease condition exists, independent on iteration number and the size of ρ. Initially the penalty parameter is taken as zero.

Now, define

$$\phi'(\alpha, \rho) = \frac{d\phi(\alpha, \rho)}{d\alpha}$$

Then, at the beginning of each iteration the condition

$$\phi'(0, \rho) \leq -\frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{H}\mathbf{d} \tag{2.102}$$

is tested, and the new penalty parameter $\bar{\rho}$ is defined by

$$\bar{\rho} = \begin{cases} \rho & \text{if Eq. (2.102);} \\ \max(\hat{\rho}, 2\rho) & \text{otherwise.} \end{cases} \tag{2.103}$$

For details on the existence of a $\hat{\rho}$ and global convergence theorems and proofs, see (Gill et al. 1986a). In the presentation above, one single penalty parameter has been used for all the constraints. It may of course also be possible to have one penalty parameter for each of the constraints. This is the case for NPSOL (Gill et al. 1986b) and the NAG routines E04UCF and E04VCF (NAG 1991). Even if the norm of the penalty parameters in general is increasing, some of the penalty parameters may decrease from one iteration to the next.

A step length α can now be found by some step size selection procedure using the function $\phi(\alpha, \rho)$. A step length selection procedure can be said to consist of two parts, firstly a set of tests which must be satisfied for the step size to be accepted, and secondly a procedure for selecting points along the search direction (a sequence of α's) to be tested.

In SQP the step length $\alpha = 1$ is first tested, and if the respective criteria at $\alpha = 1$ is not satisfied positive α's less than unity are tested.

Two usual tests to be satisfied are, see e.g. (Gill et al. 1981)

1. A sufficient decrease in ϕ is expected

$$\phi(\alpha, \rho) - \phi(0, \rho) \leq \gamma \alpha \phi'(0, \rho) \tag{2.104}$$

   where $0 < \gamma \leq \frac{1}{2}$, for instance $\gamma = 0.01$

2.

$$|\phi'(\alpha, \rho)| \leq -\eta \phi'(0, \rho) \tag{2.105}$$

   where $0 < \eta < 1$ specifies the accuracy of the line search. $\eta = 0.1$ is considered an accurate line search whereas $\eta = 0.9$ is considered an inaccurate line search. The latter is often the default, or recommended value in an SQP. If $\eta = 0$ the line search would have had to be exact.

Because $\alpha = 1$ is an upper bound this value will be accepted if the first test is satisfied and

$$\phi'(1, \rho) \leq \eta \phi'(0, \rho) \tag{2.106}$$

even if the second test is not satisfied.

If $\alpha = 1$ is not accepted, several procedures exist for selecting points to be tested. In NPSOL and the SQP routines in NAG a safeguarded cubic interpolation scheme is chosen. This procedure is very robust, but it is computationally expensive if

many points are rejected because gradients have to be computed for all the test points. If for efficiency a step size selection procedure that does not need gradients are to be used, the accuracy test (Eq. 2.105 or Eq. 2.106) has to be replaced by an alternative accuracy test which does not require $\phi'(\alpha, \rho)$ to be computed. See for instance (Schittkowski 1983) for an alternative step size selection procedure where $\phi'(\alpha, \rho)$ when $\alpha \neq 0$ is not needed.

If the step size selection procedure fails it is necessary to include an additional test to stop the procedure and preventing it from testing an infinite sequence of decreasing step sizes. For instance

$$0 < \delta \leq \|\alpha^k \mathbf{d}^k\| \tag{2.107}$$

where δ defines the minimum distance allowed between $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$, depending on the accuracy to which F can be computed.

Suitable references on step size selection in general are (Gill & Murray 1974, Gill et al. 1981, Luenberger 1984, Fletcher 1987).

Quasi-Newton Update

Powell (1978b) proposed a modified BFGS-update $\mathbf{H}$, originally proposed for unconstrained problems, to approximate the Hessian of the Lagrangian $\mathbf{W}$. The convergence results of Powell (1978b) were improved by Boggs, Tolle & Wang (1982), and based on these results the following modified BFGS-update is proposed by Gill et al. (1985);

Starting with a positive definite matrix $\mathbf{H}^0$, for instance the identity matrix, compute $\mathbf{H}^{k+1}$, termed $\bar{\mathbf{H}}$, from $\mathbf{H}^k$, termed $\mathbf{H}$ according to

$$\bar{\mathbf{H}} = \mathbf{H} - \frac{1}{\mathbf{r}^{\mathrm{T}}\mathbf{H}\mathbf{r}}\mathbf{H}\mathbf{r}\mathbf{r}^{\mathrm{T}}\mathbf{H} + \frac{1}{\mathbf{y}^{\mathrm{T}}\mathbf{r}}\mathbf{y}\mathbf{y}^{\mathrm{T}} \tag{2.108}$$

where $\mathbf{r} = \bar{\mathbf{x}} - \mathbf{x}$. For an unmodified BFGS the vector $\mathbf{y}$ would have been taken as $\mathbf{y}_L$, the change in the gradient of the Lagrangian function

$$\mathbf{y}_L = \bar{\mathbf{g}} - \bar{\mathbf{C}}^{\mathrm{T}}\boldsymbol{\mu} - \mathbf{g} + \mathbf{C}^{\mathrm{T}}\boldsymbol{\mu} \tag{2.109}$$

The update in Eq. (2.108) assures that when $\mathbf{H}$ is positive definite then $\bar{\mathbf{H}}$ is also positive definite if and only if $\mathbf{y}^{\mathrm{T}}\mathbf{r} > 0$. However, since $\mathbf{x}^*$ is not neccessarily an unconstrained minimum of the Lagrangian function, $\mathbf{y}_L^{\mathrm{T}}\mathbf{r} > 0$ may rarely or even never be true. Therefore if $\mathbf{y}_L^{\mathrm{T}}\mathbf{r}$ is not sufficiently positive, an attempt is made to perform the update with a vector $\mathbf{y}$ of the form

$$\mathbf{y} = \mathbf{y}_L + \omega(\bar{\mathbf{C}}^{\mathrm{T}}\bar{\mathbf{c}} - \mathbf{C}^{\mathrm{T}}\mathbf{c}) \tag{2.110}$$

where ω is a positive scalar. In (Gill et al. 1985) the scalar

$$\tilde{\omega} = -\frac{-\mathbf{y}_L^{\mathrm{T}}\mathbf{r}}{\mathbf{r}^{\mathrm{T}}(\bar{\mathbf{C}}^{\mathrm{T}}\bar{\mathbf{c}} - \mathbf{C}^{\mathrm{T}}\mathbf{c})}$$

is computed if $\mathbf{y}_L^{\mathrm{T}}\mathbf{r} < 0$. If $\tilde{\omega}$ is negative the update is skipped, otherwise the update is performed with $\mathbf{y}$ defined in Eq. (2.110) and $\omega > \tilde{\omega}$.

A similar approach is taken in the Fortran routines E04VCF and E04UCF in the NAG Fortran library. However, rather than using one single ω, one non negative ω_i for each of the terms in the vector $\bar{\mathbf{C}}^{\mathrm{T}}\bar{\mathbf{c}} - \mathbf{C}^{\mathrm{T}}\mathbf{c}$ is defined. If $\mathbf{y}^{\mathrm{T}}\mathbf{r} > 0$ can not be achieved this way, the update is performed with a scaled $\mathbf{y}_L$. It is not specified in (NAG 1991) how the ω_i's are to be found, or what is meant by a scaled $\mathbf{y}_L$.

It should be observed that in this update, only active nonlinear constraints are necessary.

In practice it is more robust and efficient to work with matrices in factorized forms. The matrix $\mathbf{H}$ is for instance represented by its Choleskey factor. The rank-2 BFGS-update of $\mathbf{H}$ is equivalent to a rank-1 update of its Cholesky factor (Dennis & Schnabel 1981). This rank-1 update may also be used in SQP (Gill et al. 1985).

In E04VCF and E04UCF of (NAG 1991) the Cholesky factor of the matrix $\mathbf{H}_Q = \mathbf{Q}^{\mathrm{T}}\mathbf{H}\mathbf{Q}$ (see Eq. 2.95) is updated instead of $\mathbf{H}$. The only difference from the presentation above is that $\mathbf{r}_Q = \mathbf{Q}^{\mathrm{T}}\mathbf{d}$ and $\mathbf{y}_Q = \mathbf{Q}^{\mathrm{T}}\mathbf{y}$ are used instead of $\mathbf{r}$ and $\mathbf{y}$ respectively.

Termination Criteria

The development of convergence criteria is thoroughly treated in (Gill et al. 1981). Assume a point $\bar{\mathbf{x}}$ close to the optimal point $\mathbf{x}^*$. Using Taylor series expansions of the objective function F and the projected gradient around $\mathbf{x}^*$ it is shown that if

$$|F(\mathbf{x}^*) - F(\bar{\mathbf{x}})| \approx \epsilon_A \tag{2.111}$$

then

$$\begin{aligned} \|\mathbf{x}^* - \bar{\mathbf{x}}\|_2^2 &\approx \frac{2\epsilon_A}{\mathbf{v}_z\mathbf{Z}^{\mathrm{T}}\mathbf{W}(\mathbf{x}^*)\mathbf{Z}\mathbf{v}_z} \\ \|\mathbf{g}_z(\bar{\mathbf{x}})\|_2^2 &\approx 2\epsilon_A\frac{\|\mathbf{Z}^{\mathrm{T}}\mathbf{W}(\mathbf{x}^*)\mathbf{Z}\mathbf{v}_z\|_2^2}{\mathbf{v}_z^{\mathrm{T}}\mathbf{Z}^{\mathrm{T}}\mathbf{W}(\mathbf{x}^*)\mathbf{Z}\mathbf{v}_z} \end{aligned} \tag{2.112}$$

where $\mathbf{v}_z$ is an arbitrary vector in the null-space of the active constraints of unit length, and ϵ_A is the accuracy specification for the objective function. This means that for a well scaled problem it is reasonable to expect about half the accuracy in

the projected gradients and the variables as in the objective functions. For linearly constrained optimization problems convergence criteria may be based on both Eq. (2.111) and (2.112). For nonlinearly constrained optimization problems, however, Eq. (2.111) can not be used, since methods for solving such problems are generally not descent with respect to the objective function. In addition to Eq. (2.112), termination criteria for the nonlinear constraints are needed.

Convergence criteria for both unconstrained, linearly constrained and nonlinearly constrained optimization problems are presented in (Gill et al. 1981). The criteria are based on examining properties of the solution after iteration number k, probably compared to iteration number $k-1$. In all cases more than one convergence criteria have to be satisfied at the same time, to increase robustness (reduce the probability of accepting a wrong solution).

In E04VCF and E04UCF of (NAG 1991) the following three convergence criteria have to be satisfied before the solution after iteration number k is accepted, namely

$$\alpha^{k-1}\|\mathbf{d}^{k-1}\|_2 \leq \sqrt{\epsilon_A}(1+\|\mathbf{x}^k\|_2) \tag{2.113}$$

$$\|\mathbf{Z}_{FR}^{\mathrm{T}}\mathbf{g}_{FR}^k\|_2 \leq \sqrt{\epsilon_A}(1+\max(1+|F(\mathbf{x}^k)|,\|\mathbf{g}_{FR}^k\|_2)) \tag{2.114}$$

$$|res_j| \leq ftol_j \quad \text{for all } j \tag{2.115}$$

where ϵ_A is the optimization accuracy, $|res_j|$ is the violation and $ftol_j$ is the error tolerance of the jth active nonlinear constraint. In addition to these three convergence criteria the Lagrange multipliers for the inequality constraints have to be optimal, but this is for an IQP based SQP satisfied in the QP subproblem.

An SQP Algorithm

Starting with an initial $\mathbf{x}^I$ define the following algorithm to solve the general NLP

1. Define an initial positive definite matrix $\mathbf{H}_0$ to approximate the Hessian matrix by a Quasi-Newton scheme, e.g. the identity matrix.

   Set (all) penalty parameter(s) to zero.

   Find from $\mathbf{x}^I$ an initial set of variables $\mathbf{x}_0$ which is feasible with respect to the box constraints and the general linear equality and inequality constraints of the NLP. This can for instance be done by the linear programming approach presented for the feasibility phase of the QP solution above. If such an $\mathbf{x}_0$ does not exist, that is the box constraints and the linear constraints are not consistent, terminate the optimization with an error message.

   Compute the objective function and its gradients, and all the nonlinear constraints and their gradients at $\mathbf{x}^0$.

   In the iterative procedure below, $\boldsymbol{\lambda}^0 = \boldsymbol{\mu}^0$.

   Then, for $k = 0, 1, ...$

2. If all the convergence criteria are satisfied, terminate successfully. Otherwise, proceed.

3. Solve the QP subproblem with $\mathbf{x}^k$, $\boldsymbol{\lambda}^k$ and $\mathbf{H}^k$ as input. Outputs from QP are $\mathbf{d}^k$, $\boldsymbol{\xi}^k = \boldsymbol{\mu}^k - \boldsymbol{\lambda}^k$, $\mathbf{s}^k$ and $\mathbf{t}^k$.

4. Compute the derivative of the merit function ($\phi'(0, \rho)$), and test if Eq. (2.102) is satisfied. The penalty parameter(s) are possibly updated according to Eq. (2.103), such that the new ρ satisfies Eq. (2.102).

5. Set $\tilde{\alpha} = 1$, and compute $\tilde{\mathbf{x}}^{k+1}$, $\tilde{\boldsymbol{\lambda}}^{k+1}$ and $\tilde{\mathbf{s}}^{k+1}$ according to Eq. (2.98).

   Compute the objective function and its gradient, and all the nonlinear constraints and their gradients at $\tilde{\mathbf{x}}^{k+1}$.

   Compute $\phi(1, \rho)$, and $\phi'(1, \rho)$. If the sufficient decrease test (Eq. 2.104) and the accuracy test (Eq. 2.106) are satisfied, set $\alpha = 1$ and accept $\tilde{\mathbf{x}}^{k+1}$, $\tilde{\boldsymbol{\lambda}}^{k+1}$ and $\tilde{\mathbf{s}}^{k+1}$ as $\mathbf{x}^{k+1}$, $\boldsymbol{\lambda}^{k+1}$ and $\mathbf{s}^{k+1}$. Then proceed to the next step.

   If $\tilde{\alpha} = 1$ is not accepted, perform a line search to select a positive step size $\alpha < 1$ that satisfies the sufficient decrease test (Eq. 2.104) and the accuracy test (Eq. 2.105).

   At each intermediate $\tilde{\alpha}$ in the line search the objective function and its gradient, and all the nonlinear constraints and their gradients have to be computed.

   If an intermediate $\tilde{\alpha}$ should fail to satisfy (Eq. 2.107) the algorithm terminates with a failure.

6. Perform a quasi-Newton update of the approximate Hessian to compute $\mathbf{H}^{k+1}$, or rather a factorization of $\mathbf{H}^{k+1}$.

7. Set $k = k + 1$, and go to step 2.

2.4.3 Practical Aspects of Optimization

Practical aspects of optimization are thoroughly treated in Chapter 8 of (Gill et al. 1981), and also in (Gill et al. 1985). Comments on accuracy and on scaling, which are emphasized in Chapter 4 of this thesis, are presented here.

Firstly, it should be observed that the local convergence and accuracy properties presented above usually assumes that the objective function and all the constraint functions are at least twice continuously differentiable. If these conditions are not satisfied at isolated points these properties will probably be degraded. The NAG routines E04VCF and E04UCF will, however, usually solve the optimization problems if there are only isolated discontinuities away from the solution (NAG 1991). In Chapter 4 of this thesis problems with objective functions which are discontinuous in the second derivatives at isolated points are solved.

It immediately seems natural that the expected accuracy in the converged solution cannot be expected to be higher than the accuracy to which gradients and functions

are computed. In Section 2.3 accuracy in CVP with shooting for solving optimal control problems is discussed.

In Chapter 4 of this thesis neither the nonsmoothness of the second derivatives of the objective functions nor the limited accuracy in function calculations cause any problems. Low accuracy specifications in the NLP may be a problem itself. As explained in (Gill et al. 1981) the accuracy specifications of the convergence criteria do not imply the same accuracy in the solution, and for low accuracy specifications the probability of mismatch is higher than if the accuracy specification is higher.

Scaling

Theoretically, the SQP method is scaling independent, except for the initial value of the Hessian approximation $\mathbf{H}^0$. However, the initial value of $\mathbf{H}^0$ relative to its updates is very important. Obviously, the magnitudes of the vectors $\mathbf{r}$ and $\mathbf{y}$, which are scaling dependent, influence the values of the $\mathbf{H}^k$'s. Therefore, scaling will to some extent decide which subproblems that are solved, and even theoretically, the iterate sequence depends on scaling. In practice, the solution of the iteration sequence is of course not scaling independent.

The conditioning of the NLP is affected by scaling. Usually, $\mathbf{H}^0$ is chosen as the identity matrix. If for instance the magnitudes of $\mathbf{r}$ is very low compared to the magnitude of $\mathbf{y}$, the updated $\mathbf{H}$ will soon become ill-conditioned (see Eq. 2.108). In addition to the relation between parameter, function and gradient values relative to each other, the relative importance of the different parameters on the objective function also affect the conditioning of $\mathbf{H}$. In a well-scaled problem the variables should have equal importance in the objective function. If the constraints are poorly scaled, the Jacobian matrix of the constraints will become ill-conditioned. This will, for instance, have a very negative influence on the active set strategy in the QP subproblem.

The convergence criteria (Eqs. 2.113–2.115) depend on the definition of "small" and "large". For different scalings the optimizer will terminate at different points.

Based on these facts, it is a common design rule that the variables should be of similar magnitude and of order unity. The constraints should then be individually scaled to give the best probable conditioning of the Jacobian matrices. The objective function should be of order unity.

Linear transformations of the variables and the functions are the most common form of scaling, see (Gill et al. 1981). Linear transformations include constant and proportional terms. The constant terms are included to cancel constant terms in the functions, or probably to shift the variables from varying between a high constant value to varying between zero or unity. Hereafter, only proportional diagonal terms in the scaling will be considered. Define $\mathbf{x}_s$, F_s, $\mathbf{c}_s$ as the scaled variables $\mathbf{x}$, scaled objective function F and scaled constraints $\mathbf{c}$ respectively. Then,

$$\mathbf{x} = \mathbf{D}_1 \mathbf{x}_s \tag{2.116}$$

$$\mathbf{c}_s = \mathbf{D}_2 \mathbf{c} \tag{2.117}$$

$$F_s = D_3 F \tag{2.118}$$

where $\mathbf{D}_1$ and $\mathbf{D}_2$ are diagonal matrices and D_3 is a scalar. The gradient $\nabla_{\mathbf{x}_s} F_s$ and the Hessian matrix $\nabla^2_{\mathbf{x}_s} F_s$ of the scaled objective function are related to the original gradient $\nabla_{\mathbf{x}} F$ and Hessian matrix $\nabla^2_{\mathbf{x}} F$ by

$$\nabla_{\mathbf{x}_s} F_s = D_3 \nabla_{\mathbf{x}} F \mathbf{D}_1$$

$$\nabla^2_{\mathbf{x}_s} F_s = D_3^2 \mathbf{D}_1 \nabla^2_{\mathbf{x}} F \mathbf{D}_1 \tag{2.119}$$

Similarly, for the scaled constraint Jacobian $\mathbf{C}_s$ and original Jacobian $\mathbf{C}$

$$\mathbf{C}_s = \mathbf{D}_2 \mathbf{C} \mathbf{D}_1 \tag{2.120}$$

It is shown in Chapter 4.4 that this proportional scaling has a significant impact on the performance of the optimizer when trajectory optimization problems are solved using CVP with shooting.

2.4.4 Optimization of Sparse Problems

In this thesis only dense NLP-solvers have been applied. This is also the most natural choice when single shooting is applied. In Chapter 4 where CVP with single shooting is applied to short time tasks for underwater vehicles it is emphasized that for larger tasks a multiple shooting strategy may be more adequate. Long time optimal control tasks parameterized using CVP with multiple shooting, like CVP with SVP or direct collocation, typically lead to large and sparse nonlinear programs (see Section 2.2.3). The techniques outlined in Section 2.4.2 are very efficient for solving small or moderate sized dense NLPs. For large and sparse problems, however, this is generally not true.

For small problems the computational costs are dominated by function and gradient computations. Therefore, the goal of the NLP solver should be to minimize the number of iterations needed to find a local minimum. For large problems, however, the linear algebra needed to compute a descent direction will tend to dominate the computational costs. As a result an NLP solver for large problems will try to reduce the linear algebra needed, at the cost of using more iterations. Typically, an active set strategy for QP-solvers used for sparse problems tries to avoid destroying the sparsity pattern of the problem, in order to be able to use sparse linear algebraic methods.

Methods for solving large-scale NLPs will not be treated in detail here since only dense small-scale problems have been solved. However, to put the work reported in this thesis into a larger context, and to emphasize the importance of sparse NLP solvers in solving OCP by CVP with a large number of parameters efficiently, a short presentation of the principles and some important references will be given.

In Chapter 5.6 of (Gill et al. 1981) large-scale optimization with respect to linear constraints are considered. The simplex method for solving large scale linear programming (LP) problems are modified to general nonlinear objective functions. In addition to basic and nonbasic variables a set of superbasic variables have to be defined when the objective function is nonlinear. The nonbasic variables are fixed at one of their bounds, the basic variables are computed to satisfy the general linear constraints, and the superbasic variables are computed to yield a descent direction for the NLP. A quasi-Newton algorithm in the space of feasible directions are used. Instead of updating an approximation of the full Hessian, an approximation of the projected Hessian is updated. The projected Hesian is generally dense, also when the Hessian is sparse. Therefore, only the Jacobian matrix part of the Kuhn-Tucker matrix (defined by the linearized Kuhn-Tucker equations) are sparse. Another major difference between this method and the method outlined in the previous subsection is that the null-space matrix of the matrix defining the active constraints is defined in terms of a partitioning of the variables, and not on compuationally expensive (orthogonal) transformations. The Fortran subroutine MINOS (Murtagh & Saunders 1993) is based on this method. MINOS include the possibility of nonlinear constraints, which is based on solving sub-problems with linear constraints, and require function computations for both minor and major iterations.

In Chapter 6.7.2 of (Gill et al. 1981) an SQP strategy for nonlinearly constrained large-scale optimization problems is presented. The nonlinear constraints will in a QP-subproblem generally not be satisfied, which has the consequence that it is not sufficient to retain an approximation of the projected Hessian only. This and other issues in large-scale SQP methods are addressed in (Gill, Murray & Saunders 1994). In the Fortran subroutine SNOPT (Gill, Murray & Saunders 1993) an IQP type strategy for large-scale SQP is used, where updates of an approximation to a transformed Hessian is used to overcome the problem that all of the Hessian is needed in the active set strategy. The transformed Hessian includes the projected Hessian as a principal submatrix.

The methods based on using quasi-Newton approximations of the reduced Hessian suffer from the fact that this matrix is dense. Therefore, sparse linear algebra can not be used for the part of the Kuhn-Tucker matrix containing the reduced Hessian. Consequently the computational cost of the linear algebra needed to solve the QP subproblem grows with N_z^3, where N_z is the number of degrees of freedom, i.e. the number of parameters minus the number of active constraints.

A method for solving sparse QP problems using Schur-complements is described in (Gill, Murray, Saunders & Wright 1990). This method which is intended for use in an SQP strategy for NLP's where the second derivatives of the Lagrangian are

available and the Hessian is sparse, utilizes the sparsity of both the Hessian and the constraints. Betts & Huffman (1993) use this method in an SQP method for CVP with SVP, where the Hessian of the Lagrangian is computed by sparse forward differences. The sparsity pattern of the direct transcription method is preserved, and consequently the computational cost of the linear algebra needed to solve the QP subproblem grows linearly with the number of degrees of freedom.

Bock & Plitt (1984) proposed an SQP method which utilizes the structure in CVP with multiple shooting (see Section 2.2.3). The QP subproblem is reduced by using a condensing algorithm which eliminates the shooting constraints and the shooting parameters. This strategy does not utilize the sparsity-pattern of the QP subproblem totally, but reduces the dimension of the subproblem to the dimension which would result if a single shooting method with the same control parameterization had been used. A high-rank BFGS-update of the approximate Hessian is proposed, which preserves the sparsity pattern of the Hessian. It is, however, not explained how this is utilized in the QP subproblem. Whereas the condensing algorithm is constructed specially for the multiple shooting method, the high-rank quasi-Newton method is general and could also be used for other sparse NLP problems like certain direct collocation methods. The high-rank BFGS-update of the approximate Hessian has been implemented in the SQP subroutine SQPHR in the direct multiple shooting program PROMIS (Jänsch & Schnepper 1991). SQPHR is a modification of the routine SLSQP by Kraft (1988).

2.5 Conclusions

The numerical solution of a general class of nonlinear and highly constrained OCP's using CVP with shooting has been treated in detail. A historical review of CVP methods has been given. Piecewise constant and piecewise linear parameterizations of the controls have been presented. It is also shown how the OCP constraints can be represented/approximated in the NLP.

An accuracy analysis for the objective and the constraint functions calculations in CVP with shooting, which is believed to be a new contribution, has been presented. The analysis was used to conclude that the accuracy of the converged solution of the NLP is not necessarily limited by the accuracy of the numerical solution of the differential equations. Further, three different schemes for computing the gradients of the objective and the constraint functions are presented; A numerical approach using forward differences, an analytical approach using the state sensitivity equations, and an analytical approach using the costate equations. The new accuracy analysis was used to investigate the accuracy of the forward difference approach and the state sensitivity approach. It was also used to show the equivalence of the two approaches.

It has not been concluded which is the better of the methods. A high number of constraints speaks in favor of the forward difference and the state sensitivity approaches,

whereas a high number of parameters speaks in favor of the costate approach. If multiple shooting is used and the time grid approximation approach is used to represent path constraints the number of parameters are not necessarily higher than the number of constraints. In that case forward differences or the state sensitivity equations, which are more accurate and robust, should be preferred. These two methods are also more flexible than the costate approach. Forward differences are definitely the most flexible method since it does not need the derivatives of the state space equations.

Finally, methods for NLP problems have been discussed, and the SQP method has been treated in detail. The presentation is based on several different publications and technical reports, and is given to provide the reader with algorithmic details which are not found in any text books, and which is difficult to collect from the literature.

Chapter 3

Mathematical Models of Marine Vehicles

3.1 Introduction

In this chapter the mathematical models for the vehicles which are used for optimization in Chapter 4 are presented in a general form. The modelling is divided into two parts, i.e. the modelling of vehicles and the modelling of the thrusters, which are used as control devices. The vehicle modelling, which is presented in the next section, is mainly based on (Fossen 1994). In Section 3.3, thruster models are discussed. First propeller characteristics are discussed in general terms, then a dynamical model of a DC-motor driven thruster is presented.

3.2 Vehicle Models

3.2.1 Kinematic Equations of Motion

The dynamic equations of a marine vehicle is described in a vehicle-fixed coordinate system. The transformation from the vehicle-fixed to the earth-fixed reference frame is described by the relation

$$\dot{\boldsymbol{\xi}} = \mathbf{J}(\boldsymbol{\xi})\boldsymbol{\nu} \tag{3.1}$$

where $\boldsymbol{\xi}$ contains the global position coordinates and the parameters describing orientation, and

$$\boldsymbol{\nu} = \begin{pmatrix} u & v & w & p & q & r \end{pmatrix}^{\mathrm{T}} \tag{3.2}$$

is the velocity of the vehicle in vehicle-fixed coordinates. Here u, v, and w are the linear velocities in the vehicle x, y and z directions, and p, q and r are the angular velocities about the vehicle x, y and z axes.

It is convenient to define

$$\boldsymbol{\xi} = \begin{pmatrix} \boldsymbol{\xi}_1^{\mathrm{T}} & \boldsymbol{\xi}_2^{\mathrm{T}} \end{pmatrix}^{\mathrm{T}}$$

$$\boldsymbol{\xi}_1 = \begin{pmatrix} x & y & z \end{pmatrix}^{\mathrm{T}}$$

Here x, y and z are the global position coordinates and $\boldsymbol{\xi}_2$ is a three or four dimensional vector describing the orientation of the vehicle.

A common three-parameter description of the orientation is the roll, pitch and yaw angles, ϕ, θ and ψ (Abkowitz 1969). The Jacobian $\mathbf{J}(\boldsymbol{\xi})$ is given by

$$\mathbf{J} = \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{pmatrix} \tag{3.3}$$

$\mathbf{R}$ is the 3×3 orthogonal rotation matrix (Spong and Vidyasagar 1989)

$$\mathbf{R} = \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi}$$

$$= \begin{pmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi s\theta c\phi \\ s\psi c\theta & c\psi c\phi + s\psi s\theta s\phi & -c\psi s\phi + s\theta s\theta c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{pmatrix} \tag{3.4}$$

where $c(\cdot) = \cos(\cdot)$ and $s(\cdot) = \sin(\cdot)$. $\mathbf{E}$ is a 3×3 transformation matrix defined as

$$\mathbf{E} = \begin{pmatrix} 1 & s\phi s\theta/c\theta & c\phi s\theta/c\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{pmatrix} \tag{3.5}$$

which in general is non-orthogonal. It is then assumed that

$$|\theta| < \frac{\pi}{2}$$

so that the Jacobian $\mathbf{J}(\boldsymbol{\xi})$ is bounded.

The kinematics of an underwater vehicle can also be represented by quaternions. This is treated in detail in (Fjellstad 1994).

3.2.2 6 DOF Dynamic Equations of Motion

Underwater vehicles are in the following assumed to be rigid bodies. A rigid body in three dimensional space has six degrees of freedom. In addition to the rigid body dynamics the vehicle is influenced by radiation-induced hydrodynamic forces (Faltinsen 1990), environmental disturbance forces and propulsion forces. The radiation-induced forces are due to the rigid body moving in the fluid (water). Environmental disturbance forces can be generated by wind, waves and ocean currents.

The environmental disturbance forces will often be unknown apriori and impossible to include in a model meant for off-line computations of optimal trajectories. However, an approximate value of a steady state ocean current can be included.

In Chapter 2 in (Fossen 1994) a general six degree of freedom model of a marine vehicle under influence of radiation-induced hydrodynamic forces is developed. It is assumed that the rigid body forces and the hydrodynamic forces can be superposed, and the complete model is written in the following compact vectorial form

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\xi}_2) = \boldsymbol{\tau} \tag{3.6}$$

Here $\mathbf{M}$ is the positive definite 6×6 inertia matrix containing vehicle inertia and hydrodynamic added inertia. $\mathbf{C}$ is a 6×6 matrix so that $\mathbf{C}\boldsymbol{\nu}$ is the vector of coriolis and centripetal forces due to vehicle and hydrodynamic added inertia. $\mathbf{D}$ is a 6×6 matrix containing coefficients describing dissipative hydrodynamic terms. $\mathbf{g}(\boldsymbol{\xi}_2)$ is the 6 dimensional vector of restoring forces and moments caused by gravity and buoyancy. The vector $\boldsymbol{\tau}$ contains the propulsion and environmental forces. A detailed description of the matrix and vector elements in Eq. (3.6) is given in (Fossen 1994). For a comprehensive treatment of marine hydrodynamics see also (Faltinsen 1990) and (Newman 1977).

3.2.3 A Small Remotely Operated Underwater Vehicle

For a small underwater vehicle the influence of a slowly varying nonrotating ocean current can be approximated by defining a 6-dimensional vector of relative velocity, see (Fossen 1994)

$$\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c \tag{3.7}$$

where

$$\boldsymbol{\nu}_c = \begin{pmatrix} u_c & v_c & w_c & 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}}$$

is the velocity of the fluid. The equations of motion are then written

$$\mathbf{M}\dot{\boldsymbol{\nu}}_r + \mathbf{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{g}(\boldsymbol{\xi}_2) = \tilde{\mathbf{B}}\boldsymbol{\tau}(\boldsymbol{\nu}_r, \mathbf{u}) \tag{3.8}$$

where $\mathbf{M}$, $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{g}$ are the same as in Eq. (3.6). $\mathbf{u}$ is a m dimensional vector of control inputs and $\boldsymbol{\tau}$ is a 6 dimensional vector of control forces including all the hydrodynamic terms. $\tilde{\mathbf{B}}$ is a constant $6\times m$ input matrix determined by the locations of the control devices, which may be propulsion devices like thrusters and possibly also rudders or control surfaces. In this work a small remotely operated vehicle operating in a cluttered environment at low velocities is considered. Therefore the vehicle is assumed to be controlled by thrusters in all six degrees of freedom.

The system can be described in state space form by defining a 12-dimensional state vector

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \tag{3.9}$$

where $\mathbf{x}_1 = \boldsymbol{\xi}$ and $\mathbf{x}_2 = \boldsymbol{\nu}$.

This gives (assuming $\boldsymbol{\nu}_c$ to be constant)

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \boldsymbol{\nu}_c) + \mathbf{B}\boldsymbol{\tau}(\mathbf{x}_2, \boldsymbol{\nu}_c, \mathbf{u}) \tag{3.10}$$

where

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \mathbf{J}(\mathbf{x}_1)\mathbf{x}_2 \\ \mathbf{M}^{-1}[-\mathbf{C}(\mathbf{x}_2, \boldsymbol{\nu}_c)(\mathbf{x}_2 - \boldsymbol{\nu}_c) - \mathbf{D}(\mathbf{x}_2, \boldsymbol{\nu}_c)(\mathbf{x}_2 - \boldsymbol{\nu}_c) - \mathbf{g}(\mathbf{x}_1)] \end{pmatrix} \tag{3.11}$$

and

$$\mathbf{B} = \begin{pmatrix} \mathbf{0} \\ \mathbf{M}^{-1}\tilde{\mathbf{B}} \end{pmatrix} \tag{3.12}$$

The vehicle operates as mentioned above at low speeds. It is also assumed that the vehicle has three planes of symmetry. The off-diagonal terms of the mass matrix can then be neglected. Hence the following simple expressions for $\mathbf{M}$ and $\mathbf{C}$ can therefore be assumed (Fossen 1994)

$$\mathbf{M} = \mathrm{diag}\{m - X_{\dot{u}}, m - Y_{\dot{v}}, m - Z_{\dot{w}}, I_x - K_{\dot{p}}, I_y - M_{\dot{q}}, I_z - N_{\dot{r}}\} \tag{3.13}$$

$$\mathbf{C} = \begin{pmatrix} 0 & -mr & mq & 0 & -Z_{\dot{w}}w & Y_{\dot{v}}v \\ mr & 0 & -mp & Z_{\dot{w}}w & 0 & -X_{\dot{u}}u \\ -mq & mp & 0 & -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \\ 0 & -Z_{\dot{w}}w & Y_{\dot{v}}v & 0 & (I_z - N_{\dot{r}})r & -(I_y - M_{\dot{q}})q \\ Z_{\dot{w}}w & 0 & -X_{\dot{u}}u & -(I_z - N_{\dot{r}})r & 0 & (I_x - K_{\dot{p}})p \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 & (I_y - M_{\dot{q}})q & -(I_x - K_{\dot{p}})p & 0 \end{pmatrix} \tag{3.14}$$

Here m is the mass of the vehicle I_x, I_y and I_z are the moments of inertia around the body x, y and z axes respectively. $X_{\dot{u}}$, $Y_{\dot{v}}$, $Z_{\dot{w}}$, $K_{\dot{p}}$, $M_{\dot{q}}$ and $N_{\dot{r}}$ are the added inertias for motion in the u, v, w, p, q and r directions, respectively.

A rough approximation to the damping forces of the vehicle is to assume that they are non-coupled, the vehicle has three planes of symmetry, and that terms higher than second order can be neglected (Fossen 1994). This approximation will be made here, and accordingly the drag matrix $\mathbf{D}$ will have a diagonal structure:

$$\begin{aligned} \mathbf{D}(\boldsymbol{\nu}) = -\mathrm{diag}\{&X_u + X_{u|u|}|u|, Y_v + Y_{v|v|}|v|, Z_w + Z_{w|w|}|w|, \\ &K_p + K_{p|p|}|p|, M_q + M_{q|q|}|q|, N_r + N_{r|r|}|r|\} \end{aligned} \tag{3.15}$$

The vehicle is assumed to be neutrally buoyant ($B = W$), which means that the restoring forces are zero. The center of gravity is located in the origin, which gives the following expression for the vector $\mathbf{g}$:

$$\mathbf{g} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ y_B Bc\theta c\phi - z_B Bc\theta s\phi \\ -x_B Bc\theta c\phi - z_B Bs\theta \\ x_B Bc\theta s\phi + y_B Bs\theta \end{pmatrix}^{\mathrm{T}} \tag{3.16}$$

where x_B, y_B and z_B is the x-, y- and z-coordinates of the center of buoyancy, respectively. B is the buoyancy, and W is the weight of the vehicle.

In Chapter 4 a thruster configuration resulting in the following control matrix is assumed

$$\tilde{\mathbf{B}} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & \ell_3 & \ell_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\ell_5 & \ell_6 \\ -\ell_1 & \ell_2 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.17}$$

where ℓ_i $(i = 1, 2 \ldots 6)$ are the thruster offsets providing the moment arms.

3.3 Thrusters as Control Devices

3.3.1 Introduction

Examples of control devices for marine vehicles are water jets, thrusters, main propellers of ships, variable buoyancy systems for submarines, rudders and other control surfaces.

In this section thrusters and propellers for underwater vehicles are considered. The input models for these two types of control devices can be considered as consisting of two parts. The first part is the propeller characteristics which are algebraic relations between the propeller's rotational velocity n and the propeller's thrust T and load torque Q. The second part is the dynamical model of the motor (rotating machinery) providing the rotational velocity n.

3.3.2 Propeller Characteristics

Using propellers as propulsion devices it is difficult to find adequate models describing the characteristics of the propellers that are not overly complicated. Especially when the propellers are intended to be used over a wide range of operation, like in dynamic positioning of ships and for small remotely operated underwater vehicles.

In normal operation the rotational velocity of the propeller n and the advance velocity of the vehicle V_A are both positive. The load torque Q from the propeller, and the thrust force T are then usually written (see e.g. Todd & Taylor (1967))

$$Q = \rho D^5 K_Q(J_a) n^2 \tag{3.18}$$

$$T = \rho D^4 K_T(J_a) n^2 \tag{3.19}$$

where ρ is the mass density of water, D is the diameter of the propeller, K_Q and K_T are the torque and the thrust coefficients of the propeller, and J_a is the advance ratio:

$$J_a = \frac{V_A}{nD} \tag{3.20}$$

Experimental characteristics for these coefficients as a function of the advance ratio typically show an approximately linear decrease in K_T and K_Q as J_a increases (see Fossen (1994), Todd & Taylor (1967)).

Blanke (1982) explained these linear relations theoretically and developed the following equations for a propeller in an ideal fluid (n and V_A both positive):

$$Q = Q_{nn}n^2 + Q_{nv}nV_A$$

$$T = T_{nn}n^2 + T_{nv}nV_A \tag{3.21}$$

where T_{nn}, Q_{nn}, T_{nv} and Q_{nv} were constant coefficients (the two latter negative).

van Lammeren, van Manen & Oosterveld (1969) have studied the performance of the Wageningen B-screw series over the entire region of possible propeller operations. They define the so-called four quadrant measurements for the propellers, and determine propeller characteristics as a function of the hydrodynamic pitch angle

$$\beta = \arctan(\frac{V_A}{0.7\pi nD}) \tag{3.22}$$

The four quadrants are:

- First quadrant: speed ahead ($V_A > 0$), revolutions ahead ($n > 0$)

$$0° \leq \beta \leq 90°$$

- Second quadrant: speed ahead ($V_A > 0$), revolutions astern ($n < 0$)

$$90° \leq \beta \leq 180°$$

- Third quadrant: speed astern ($V_A < 0$), revolutions astern ($n < 0$).

$$180° \leq \beta \leq 270°$$

- Fourth quadrant: speed astern ($V_A < 0$), revolutions ahead ($n > 0$).

$$270° \leq \beta \leq 360°$$

The torque and load torque coefficients in (van Lammeren et al. 1969) are defined differently from the ones in Eqs. (3.18) and (3.19) because the hydrodynamic pitch angle is defined differently from the advance velocity

In the first and the third quadrants, where V_A and n have equal signs, the experimental results in (van Lammeren et al. 1969) correspond to the bilinear relationship between propeller velocity and propeller torque and load given in Eq. (3.21). In the second and fourth quadrants, however, the characteristics are more complicated.

Experimental results for the thrusters of the Norwegian Experimental Remotely Operated Vehicle (NEROV) given in (Fossen 1991) and (Sagatun 1992) are similar

to the results in (van Lammeren et al. 1969). In the form which the results are given by van Lammeren et al. (1969) the functions $T(n, V_A)$ and $Q(n, V_A)$ will be very complicated. Therefore a definition similar to the one in Eqs. (3.18) and (3.19) would be preferable. Fossen (1991) and Sagatun (1992) have made the following definition of torque and thrust coefficients:

$$Q = \rho D^5 K_Q(J_a) n|n| \tag{3.23}$$

$$T = \rho D^4 K_T(J_a) n|n| \tag{3.24}$$

In general one can not expect the thrust and torque coefficients to be equal in the first and the third quadrant (i.e. symmetrical), since positive thrust usually is larger than negative thrust. In this work, however, it will, without loss of generality, be assumed that the thrusters are symmetrical. This is approximately true for the NEROV-thrusters. If, in addition, it is assumed that the bilinear relationships are valid in the second and third quadrants too, Eq. (3.21) can be modified to

$$Q = Q_{|n|n} n|n| + Q_{|n|v}|n|V_A \tag{3.25}$$

$$T = T_{|n|n} n|n| + T_{|n|v}|n|V_A \tag{3.26}$$

The results in (Fossen 1991) and (Sagatun 1992) indicate that the error made when using (3.26) in the second and fourth quadrant is negligible if J_a has a small negative value (V_A small compared to n). It is reasonable to assume that large negative values for J_a are rare for an ROV operating at small velocities.

The advance velocity of the propeller is related to the velocity component of the vehicle acting along the axis of the propeller through the wake fraction number w according to

$$V_A = (1 - w)V \tag{3.27}$$

where typical values for w are in the interval $0.1 - 0.4$ (Fossen 1994). Assuming w to be a constant, and assuming the thrust and the torque not to be influenced by the velocity component acting normal to V are reasonable approximations (Todd & Taylor 1967). In fact, ducting the propellers makes the latter approximation quite good.

The relations between V and the vehicle velocity components u, v and w are for each thruster given by the thruster configuration matrix B (see Eq. 3.17)

$$V_1 = \quad V_2 \quad = u$$

$$\begin{aligned} V_4 &= -V_3 &&= v \\ V_5 &= V_6 &&= w \end{aligned} \tag{3.28}$$

where V_i corresponds to thruster number i ($i = 1, ..., 6$).

3.3.3 Dynamical Model for a DC-motor Driven Thruster

The dynamical model of a DC-motor is written

$$L_a \frac{d}{dt} i_a = -R_a i_a - 2\pi K_M n + u_a \tag{3.29}$$

$$2\pi J_m \frac{d}{dt} n = K_M i_a - Q \tag{3.30}$$

where L_a is the armature inductance, R_a is the armature resistance, u_a is the armature voltage, K_M is the motor torque constant, J_m is the moment of inertia of motor and thruster, n is the velocity of the motor in revolutions per second and Q is the load torque of the motor. The load torque for a motor driving a propeller is given i Eq. (3.23).

DC-motors are usually controlled by velocity feedback. The dynamics of the DC-motors are much faster than the dynamics of the vehicle and it is therefore assumed that

$$n_d \approx n \tag{3.31}$$

where n_d is the commanded reference velocity of the motor. This means that the n_i's can be considered as the physical input for the model in Eq. (3.10). The choice of the control input representation $u_i(n_i)$ is discussed in Chapter 4.

Chapter 4

Energy-Optimization for Autonomous Underwater Vehicles with DC-motor Driven Thrusters

4.1 Introduction

In this chapter numerical solutions of optimal control problems for underwater vehicles with DC-motor driven thrusters are presented. The optimal control problems were solved using control vector parameterization (CVP) with single shooting. The controls were parameterized by piecewise constant and piecewise linear polynomials. Fixed and equidistantly distributed control time nodes were used. The resulting nonlinear programs were solved using sequential quadratic programming (SQP).

The use of energy-optimal trajectories can extend the operation radius of an autonomous underwater vehicle by reducing the energy consumption. This type of vehicle can be used for underwater inspection, maintenance, repair and telerobotics. In certain applications like operation in a cluttered environment it may be advantageous that the vehicle is untethered, and batteries have to be used for small vehicles.

In the next section the problem of trajectory generation for underwater vehicles is formulated as a class of optimal control problems. An expression for the energy consumption for a DC-motor driven thruster is developed using the dynamical models presented in Chapter 3.3, and the control input representation for the thrusters is discussed. It should be noted that the expression for the energy consumption of the thrusters can be generalized to any propeller based control device. If the energy loss in the motor which is driving the propeller is not negligible compared to the losses due to the interaction between the propeller and the water, more terms may have to be included in the expression. In addition, motor dynamics may have to be included to the dynamic equations of motion. In (Spangelo & Egeland 1993) thruster modelling and control input representations for the thrusters in the context of trajectory optimization were also discussed. The resulting class of optimal control problems,

which is derived for the type of vehicles presented in Chapter 3.2, includes in addition to the energy-minimization the possibilities of time minimization and collision avoidance. Collision avoidance, or obstacle avoidance, is solved by introducing path constraints.

In Section 4.3 the results from the paper (Spangelo & Egeland 1992a) are presented, where analytical and forward difference gradient generation is compared. Energy-optimal trajectories for a simple surge and sway motion without path constraints and with fixed final time were computed for an underwater vehicle with the same parameters as in (Fossen 1991), excluding centripetal and Coriolis forces (see Chapter 3.2.2). In this particular case there is no major difference in the computational efficiency of the two approaches, but it is concluded that for general problems the use of forward differences is preferable. The use of optimal trajectories are further compared to the use of quadratic velocity profiles in the x and y directions, and it is shown that energy consumption can be reduced significantly by introducing optimal control. The resulting nonlinear programs were solved using the SQP-package SLSQP (Kraft 1988, Kraft 1989).

In Section 4.4 the numerical implementation used to obtain the results presented in the rest of the chapter is presented. Gradients were computed using the forward difference scheme, whereas the state space equations were integrated using a variable step-size Runge-Kutta method, and the resulting nonlinear programs were solved using the subroutine E04VCF from the NAG Fortran library (NAG 1991). The implementation includes several optimization parameters, and the tuning of these parameters are discussed by extensive testing on a simple 2 DOF benchmark problem. The model was obtained by extracting the motion in the x and y directions of the vehicle from the rest of the full 6 DOF model presented in Section 4.5. The objective functional included energy-minimization, possibly with free final time. Path constraints were not included. The model is simple, but there is a close relationship to the more complete models, especially in the equality of the objective functionals. Conclusions are presented concerning choices of optimization parameters, which were used as guidelines for the rest of the chapter. Results obtained by tuning some of the parameters are discussed only briefly, while for other parameters complete details are given. Especially the choice of the right scaling parameters for this class of optimization problems is treated comprehensively.

In Section 4.5 optimal trajectories for the full 6 DOF model of the type presented in Chapter 3.2 are discussed. Numerical results with and without free final time, and with and without collision avoidance are presented. Piecewise linear and piecewise constant control parameterizations are compared, for different number of parameters. The piecewise linear parameterization is concluded to be the better of the two. Section 4.5 is based on (Spangelo & Egeland 1994).

Based on the results and the discussion in Section 4.5 optimal trajectories for reduced versions of the 6 DOF model were computed for the case where both collision avoidance and free final time were included. The piecewise linear parameterization of the parameters was used. The results are presented in Section 4.6.

Energy-optimal trajectories for underwater vehicles were first presented in (Spangelo & Egeland 1992b), where control vector parameterization (CVP) with direct shooting was compared with the conjugated gradient penalty method in function space by Lasdon et al. (1967). It was shown that the CVP-method was completely superior to the function space method for the problem at hand. The implementation of the conjugated gradient method for the energy-optimization of underwater vehicles are not presented in this thesis.

Both the NAG-routine E04VCF, which is essentially the same as NPSOL (Gill et al. 1986b), and SLSQP are well suited for solving trajectory optimization problems. The NAG-routine was chosen because it is a standard commercially available program package, and because detailed information from the optimization is easily available for the user. In addition the convergence accuracy specifications are more flexible, and the implementation details of this routine is more accessible.

I have implemented the models and the numerical methods, except for the NLP solvers, in the C language. Crude estimates of the CPU-time consumption of the numerical solutions are given, sometimes also for parts of the program. This is mainly done to investigate the possibilities of using the method in a guidance scheme, and to identify the bottlenecks of the method. A guidance scheme is not presented in this chapter, but in Chapter 6 "Conclusions and Recommendations" aspects of control and guidance are discussed.

4.2 Formulation of a Class of Optimal Control Problems

4.2.1 Energy Consumption for a DC-motor Driven Thruster

By assuming the dynamics of the armature circuit to be fast Eq. (3.29) can be approximated by

$$u_a = R_a i_a + 2\pi K_M n \tag{4.1}$$

Dissipated electrical power is

$$P = u_a i_a = R_a i_a^2 + 2\pi K_M n i_a \tag{4.2}$$

The first term on the right hand side is due to dissipation in the armature resistance, and the second term represents the conversion from electrical to mechanical power. The first term is assumed to be much smaller than the second and is therefore neglected. Inserting

$$i_a = \frac{2\pi J_m}{K_M}\dot{n} + \frac{Q}{K_M} \tag{4.3}$$

which is obtained from Eq. (3.30), into the second term of Eq. (4.2) gives

$$P = (2\pi)^2 J_m n\dot{n} + 2\pi Q n \tag{4.4}$$

The first term on the right hand side is power needed to accelerate the propeller, and the last term is power dissipation due to the interaction between propeller and water. This motivates the following approximation of the power dissipation for motor i:

$$P_i = 2\pi Q_i n_i = 2\pi\rho D_i^5 K_{Qi}(J_a) n_i^2 |n_i| \tag{4.5}$$

This expression is recognized as the mechanical power from a shaft with rotational velocity n and load torque Q.

4.2.2 Control Input Representation for the Thruster

According to the discussion in Chapter 3.3 a simple choice for the control inputs would be

$$u_i = n_i|n_i| \tag{4.6}$$

The reference inputs to the velocity controlled actuators could then simply be calculated by (see Eq. 3.31)

$$n_{di} = \text{sign}(u_i)\sqrt{|u_i|} \tag{4.7}$$

According to Eq. (3.26) this would give the following relation between the control inputs and thrusts

$$T = T_{|n|n} u_i + T_{|n|v}\sqrt{|u_i|}V_i \tag{4.8}$$

This transformation is likely to corrupt the convergence of the optimizer for several reasons. Firstly the transformation is not unique (see discussion below). Secondly it is not Lipschitz continuous for $u_i = 0$.

An alternative choice for control input representation is simply

$$u_i = T_i \tag{4.9}$$

This means that the model will be linear in the inputs which is favorable for the convergence properties. The reference inputs to the velocity controlled actuators now have to be calculated by solving the second order equations

$$n_{di}|n_{di}| - \alpha|n_{di}| - \beta u_i = 0 \tag{4.10}$$

where from Eq. (3.26)

$$\alpha = -\frac{T_{|n|v}(1-w)V_i}{T_{|n|n}} \geq 0$$

$$\beta = \frac{1}{T_{|n|n}} > 0 \tag{4.11}$$

In Figure 4.1 the thrust as a function of rotational velocity is given for positive, negative and zero advance velocity for a typical thruster. It can be seen that if the thrust and the advance velocity for the thruster have different signs and small enough absolute values, then Eq. (4.10) will have two real solutions, and the transformation between the propeller velocity and the propeller thrust will therefore not be unique. A reasonable solution to this problem is to always choose the same sign on the propeller revolution as on the computed thrust.

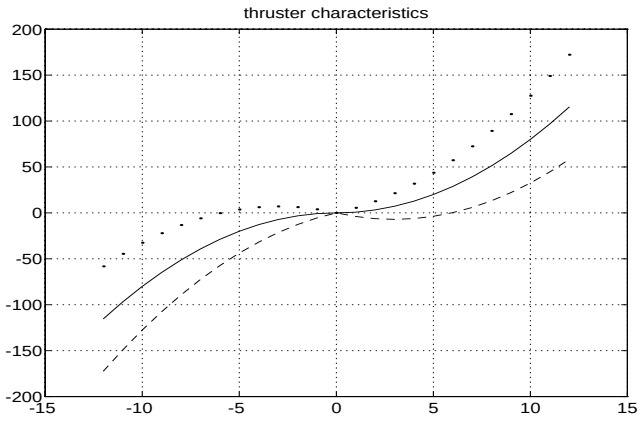

Figure 4.1: *Thrust T as a function of rotational velocity n for $V_A < 0$ (dotted), $V_A = 0$ (solid) and $V_A > 0$ (dashed).*

Choosing the input representation (4.9) it can be seen from Eqs. (3.23), (3.24), (4.9) and (4.5) that the power from thruster i as a function of the input ($u_i = T_i$) becomes

$$P_i = \frac{2\pi K_{Qi}(J_a)}{\sqrt{\rho} D_i |K_{Ti}(J_a)|^{1.5}} |u_i|^{1.5} \tag{4.12}$$

In the sequel this will be approximated as

$$P_i = \alpha_i |u_i|^{1.5} \tag{4.13}$$

where α_i is a constant. This approximation is crude compared to the approximation in the computation of the thrust. This is, however, not unreasonable because (4.13) will only be used in the objective functional, while the thrust is an input in the state space equations. In trajectory optimization it is much more important to have an accurate state space model than to have an accurate objective functional.

A problem with Eq. (4.13) is that the function is not twice continuously differentiable for $u_i = 0$. Actually the derivative of the function is not Lipschitz continuous. This problem will be addressed in more detail in Section 4.4.

4.2.3 The Resulting Optimal Control Problem with Possible Inclusion of Time-Optimization and Obstacle Avoidance

An optimal control problem is formulated where the vehicle is specified to move from an initial state $\mathbf{x}_0$ (position, orientation and velocity) to a final state $\mathbf{x}_f$ with minimum energy consumption. The ocean current $\boldsymbol{\nu}_c$ is for simplicity and without loss of generality assumed to be zero.

In the resulting optimal control problem the time consumption can be either fixed or free. In this work it will be assumed that the initial time is $t_0 = 0$, which means that the time consumption equals the final time t_f. If the final time is fixed the objective functional will consist of the energy consumption alone. If the final time is free it has to be included in the objective functional with a weighting factor w, and the objective functional becomes a weighted combination of energy and time consumption.

Obstacle avoidance can be solved by including inequality path constraints. A mathematical constraint is defined surrounding the physical constraint. For instance cylinders can be defined as mathematical constraints around the legs of an oil platform.

To avoid collision the mathematical constraints have to include some margin to the physical obstacle. The infinite dimensional path constraints will be represented as finite dimensional constraints by the time-grid approximation (see Chapter 2.2.2). If the margins between the mathematical constraints and the physical constraints are large enough compared to the distance between the grid-points, the limitations of the vehicle velocities will guarantee collision avoidance. It is then assumed that the initial and the final times of the time-grid is chosen such that collision cannot occur outside the time-grid.

Physical limitations in the actuators lead to constraints in the input variables. The power delivered to a DC motor is bounded by a maximum value. These limitations

are included as simple box constraints on the input variables. This is a reasonable approximation according to the approximation in Eq. (4.13). Since it is assumed that the bandwidth of the actuator dynamics is much higher than the bandwidth of the vehicle dynamics, box constraints on $\dot{u}_i$ are not included.

The resulting optimal control problem can then be formulated as follows: Find the trajectory, that is the state and control time histories, which minimize the objective functional

$$\min_{\mathbf{u}} J_0 = \mathrm{w} \cdot t_f + \int_0^{t_f} \sum_{i=1}^{m} \alpha_i |u_i(t)|^{1.5} dt \tag{4.14}$$

subject to

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{x}_0 \\ \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u} \\ \mathbf{x}(t_f) &= \mathbf{x}_f \\ \mathbf{g}_I(\mathbf{x}) &\geq \mathbf{0} \\ |u_i| &\leq |u|_{max} \quad (i = 1, ..., m) \end{aligned} \tag{4.15}$$

The factors α_i is included either to reflect possible differences in the thrusters or to merge more than one thruster into one control input (see Section 4.6). Hereafter, all thrusters are assumed to be equal.

4.3 Comparing Forward Differences with Analytical Gradient Computations Using Costate Equations

This section is based on (Spangelo & Egeland 1992a), and describes the generation of optimal trajectories for the simple surge and sway motions of an autonomous underwater vehicle. The main purpose is to compare the CVP approach using direct shooting and forward difference generation of the gradients with the use of analytically generated gradients. The resulting optimal trajectories are finally compared to a heuristic trajectory with quadratic velocity profiles in the x- and y-directions.

The numerical results in this section are obtained from an earlier implementation (Spangelo & Egeland 1992a) using the NLP-solver SLSQP by Kraft (1988), see also (Kraft 1989), a fixed step size integrator and no scaling. As opposed to the results in the remainder of the chapter, which are obtained from the latest version of an implementation using the NAG Fortran routine E04VCF, a variable step size numerical integration scheme, and systematically chosen scaling.

The complete 6 DOF model described in Chapter 3.2.3 was used, but the matrix $\mathbf{C}$, defining the Coriolis and centripetal terms in the dynamic equations of motion, is for simplicity chosen to be zero.

Numerical results in terms of the CPU-time consumption are not comparable to corresponding results in other sections, since the results are older, and consequently obtained on a slower computer.

4.3.1 The Optimal Control Problem: Coupled Surge and Sway Motions

A motion increment of 5 m in 10 s in the initial surge and sway directions was studied. The initial and final states were

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{0} \\ \mathbf{x}(10) &= \mathbf{x}_f = \begin{pmatrix} 5 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \end{aligned} \tag{4.16}$$

No box constraints or path constraints were included, and the final time was fixed. This resulted in the following optimal control problem

$$\min_{\mathbf{u}} J_0 = \int_0^{10} \sum_{i=1}^{6} |u_i(t)|^{1.5} dt \tag{4.17}$$

subject to Eq. (4.16) and

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u} \tag{4.18}$$

Vehicle Model Parameters

The hydrodynamical parameters were found in (Fossen 1991). These parameters are slightly different from those used in the rest of the chapter (see definition of the parameters in Chapter 3.2).

Numerical values for the hydrodynamic derivatives (added inertias) are

$X_{\dot{u}} = -30$ kg	$X_u = -80$ kg/s	$X_{u\|u\|} = -120$ kg/m
$Y_{\dot{v}} = -1100$ kg	$Y_v = -110$ kg/s	$Y_{v\|v\|} = -200$ kg/m
$Z_{\dot{w}} = -80$ kg	$Z_w = -100$ kg/s	$Z_{w\|w\|} = -150$ kg/m
$K_{\dot{p}} = -15$ kgm^2	$K_p = -30$ kg m/s	$K_{p\|p\|} = -50$ kg m
$M_{\dot{q}} = -20$ kgm^2	$M_q = -40$ kg m/s	$M_{q\|q\|} = -40$ kg m
$N_{\dot{r}} = -1$ kgm^2	$N_r = -10$ kg m/s	$N_{r\|r\|} = -15$ kg m

Other numerical values are

$m = 185$ kg	$I_x = 25$ kg m^2	$x_B = 0.0$ m	$\ell_1 = 0.4$ m	$\ell_4 = -0.02$ m
$B = W = 1800$ N	$I_y = 29$ kg m^2	$y_B = 0.0$ m	$\ell_2 = 0.4$ m	$\ell_5 = 0.43$ m
	$I_z = 28$ kg m^2	$z_B = -0.04$ m	$\ell_3 = 0.21$ m	$\ell_6 = 0.43$ m

These parameters result in the following model for the vehicle, with the dynamic equations of motion in component form:

$$
\begin{aligned}
\dot{\mathbf{x}}_1 &= \mathbf{J}(\mathbf{x}_1)\mathbf{x}_2 \\
\dot{x}_7 &= -(\frac{16}{43} + \frac{24}{43}|x_7|)x_7 + \frac{1}{215}(u_1 + u_2) \\
\dot{x}_8 &= -(\frac{22}{59} + \frac{40}{59}|x_8|)x_8 + \frac{1}{295}(-u_3 + u_4) \\
\dot{x}_9 &= -(\frac{20}{53} + \frac{30}{53}|x_9|)x_9 + \frac{1}{265}(u_5 + u_6) \\
\dot{x}_{10} &= -(\frac{3}{4} + \frac{5}{4}|x_{10}|)x_{10} - \frac{9}{5}\sin(x_4)\cos(x_5) + \frac{21u_3 + 2u_4}{4000} \\
\dot{x}_{11} &= -(\frac{40}{49} + \frac{40}{49}|x_{11}|)x_{11} - \frac{72}{49}\sin(x_5) + \frac{43}{4900}(-u_5 + u_6) \\
\dot{x}_{12} &= -(\frac{10}{29} + \frac{15}{29}|x_{12}|)x_{12} + \frac{4}{290}(-u_1 + u_2)
\end{aligned}
\tag{4.19}
$$

The Jacobian matrix J and the state vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are defined in Chapter 3.

4.3.2 Numerical Implementation

CVP with (direct) single shooting was applied using a constant and equidistantly distributed control time grid, which was equal for all the 6 controls. The controls were parameterized using piecewise linear interpolation (see Chapter 2.2.2).

The SQP subroutine SLSQP (Kraft 1988, Kraft 1989) was used as NLP-solver. This routine uses a less rigid step size selection strategy than the NAG routine E04VCF, which means that the gradients of the objective and the constraint functions are only computed once for each iteration, i.e. function values are only needed at intermediate steps in the step size selection procedure. SLSQP does not separate between general linear and nonlinear constraints. Box constraints are specified separately by the user, but how they are treated in the routine is not clear. A parameter *acc* defining the convergence criteria is specified. It defines both the accuracy of the converged optimal solution, and the tolerance of the constraint violation where the L_1-norm is used. The exact definition of the convergence criteria is not known. It should be noted that SLSQP performs very well, but less information on how it works is provided to the user than from E04VCF.

The standard 4th order Runge-Kutta method, with fixed step-size, was implemented and used for the numerical solution of the state space equations. The 4th order Simpson's rule of integration was implemented and used for the numerical integration of the objective function. No scaling of the state space equations and the Lagrangian part of the objective function was introduced, since the final time was fixed. Forward difference approximations of the gradients were computed using the same step size for the perturbed trajectories as for the nominal unperturbed trajectory (see Chapter 2.3.2).

Analytical Gradient Calculations Using the Costate Equations

General expression for the analytical gradient calculations using the costate equations are given in Chapter 2.3.4, where the general expression for the functions to be differentiated can be written in the form given by Eq. (2.33). The Hamiltonian functions (see Eq. 2.49) for the objective and the constraint functions of the optimal control problem given by Eqs. (4.16), (4.17) and (4.18) is

$$\mathcal{H}(\mathbf{x}(t),\mathbf{u}(t,\mathbf{p}),\boldsymbol{\lambda}(t)) = \gamma \sum_{i=1}^{6} |u_i(t)|^{1.5} + \boldsymbol{\lambda}(t)^{\mathrm{T}}(\mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u}) \tag{4.20}$$

where $\gamma = 1$ for the objective function and zero for all the constraint functions (all final time boundary constraints).

The costate equations (see Eq. 2.50) now becomes

$$\begin{aligned} \dot{\boldsymbol{\lambda}} &= -\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\boldsymbol{\lambda} \\ \boldsymbol{\lambda}(10) &= (\frac{\partial \varphi(\mathbf{x}(10))}{\partial \mathbf{x}})^{\mathrm{T}} \end{aligned} \tag{4.21}$$

The gradients of the functions are given by Eq. (2.51). The partial derivatives of the objective function with respect to the parameters become

$$\frac{\partial F(\mathbf{u}(t,\mathbf{p}))}{\partial u_{i,j}} = \int_{t_{j-1}}^{t_{j+1}} 1.5\mathrm{sign}(u_i)\sqrt{|u_i|}\frac{\partial u_i}{u_{i,j}}dt \tag{4.22}$$

where u_i is control number i and $u_{i,j}$ is the parameter defining u_i at control time number j (see Chapter 2.2.2, Eq. 2.21). The costates are identically zero for the objective function, since in this case the value of $\varphi(\mathbf{x}(10))$ in Eq. (4.22) is zero .

The partial derivatives of the final time boundary constraints with respect to the parameters, on the other hand, depend on the states. Therefore, one set of costate equations has to be solved for each of the constraints.

Two different representations of the boundary constraints have been implemented

1. No transformation is performed, i.e.

$$\mathbf{c}(\mathbf{p}) = \mathbf{x}(10) - \mathbf{x}_f \tag{4.23}$$

There are twelve boundary constraint, which means that Eq. (4.21) has to be solved twelve times. The boundary constraints of set number k of costate equations are

$$\boldsymbol{\lambda}_k(10) = \begin{cases} 1 \text{ in component } k \\ 0 \text{ elsewhere} \end{cases} \tag{4.24}$$

the partial derivative of constraint number k with respect to parameter number j is now given by

$$\frac{\partial c_k}{\partial u_{i,j}} = \int_{t_{j-1}}^{t_{j+1}} (\boldsymbol{\lambda}_k^{\mathrm{T}}\mathbf{B})_i \frac{\partial u_i}{\partial u_{i,j}} dt \tag{4.25}$$

2. Alternatively the final time boundary constraints can be transformed into a quadrature, se e.g. (Goh & Teo 1988), resulting in only on constraint

$$c(\mathbf{p}) = (\mathbf{x}(10) - \mathbf{x}_f)^{\mathrm{T}}(\mathbf{x}(10) - \mathbf{x}_f) \tag{4.26}$$

The costate equations only have to be solved once, with boundary constraints

$$\boldsymbol{\lambda}(10) = 2 \cdot (\mathbf{x}(10) - \mathbf{x}_f) \tag{4.27}$$

The partial derivative of c with respect parameter number j is now given by

$$\frac{\partial c}{\partial u_{i,j}} = \int_{t_{j-1}}^{t_{j+1}} (\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{B})_i \frac{\partial u_i}{\partial u_{i,j}} dt \tag{4.28}$$

From Eq. (2.22) it is seen that the partial derivative of u_i with respect to $u_{i,j}$ is

$$\frac{\partial u_i}{\partial u_{i,j}} = \begin{cases} \frac{t-t_{j-1}}{t_j-t_{j-1}}; & t_{j-1} < t < t_j \\ 1 - \frac{t-t_j}{t_{j+1}-t_j}; & t_j < t < t_{j+1} \\ 0; & \text{otherwise} \end{cases} \tag{4.29}$$

4.3.3 Numerical Results

Computational results were obtained on a SUN 4/75 station. The computational effort in terms of CPU-seconds and the value of the converged objective function were compared for the two approaches of gradient calculations, and for different numbers of parameters. In the case of analytical gradient calculations the constraints were handled both as plain final time constraints and as an error quadrature (Eq. 4.26).

The accuracy was set to

$$acc = 0.01$$

A good estimate of the accuracy in fixed step size integrators are not easily obtained, but after some tuning effort using forward difference gradient calculations a step size $h = 0.25$ was chosen, which appeared to be a good compromise between integration accuracy and computational costs. After some tuning effort the forward-difference perturbation in the forward difference gradient calculation was chosen as

$$pert_i = \max\{\tfrac{h^5}{10}, |p_i|\tfrac{h^5}{10}\} \tag{4.30}$$

In all tests all the parameters were initially chosen to be zero.

Table 4.3.3 shows computational results using numerical gradient calculation (NG), analytical gradient calculations and plain constraints(AG), and analytical gradient calculations and error quadrature (QAG). The first column indicates which method. The computations were done with different numbers of parameter per control (# par). In the third column the converged objective functions are shown (J^*). The numbers of iterations (# iter) and function calculations (# fc) are shown in column three and four. A function calculation consists of computing the objective function and the constraint violations. In the three last columns of Table 4.3.3 computational costs in terms of CPU-seconds are shown, i.e. the total CPU-time (tot. CPU), CPU-time needed to calculate a new search direction in the SQP (sqp CPU), and CPU-time needed to calculate the gradients for each iteration (gc CPU). A gradient calculation consists of computing the gradient of the objective function and the constraint jacobian matrix. The CPU-time needed to calculate a function was about 0.05 seconds in all cases, and is not shown in the table.

In the QAG case a different OCP is solved than in the AG and the NG case. Even if the two different OCP's are equivalent in the sence that the optimal solutions are the same, this has a great impact on the numerical behaviour.

First, observe the case with only two parameters per control, which give a total of twelve parameters. Since the final time conditions imply twelve equality constraints in the NG and the AG case there are no degrees of freedom left for optimization and the problem is just that of solving a set of nonlinear equations. This set of nonlinear equations were solved in four iterations and with # fc = 4 in both cases. In the QAG case the twelve final time conditions are "scrambled" into only one nonlinear equality constraint. Remember that satisfying this constraint is equivalent to satisfying all the twelve constraints in the NG and the AG case. One may be tempted to conclude from Table 4.3.3 that in the QAG case another solution to the boundary conditions have been found, since the value of the objective function is considerably lower. However, it is easily understood that satisfying the error quadrature constraint to within an accuracy of 0.01 is easier than satisfying the L_1 norm of the twelve boundary conditions to within the same accuracy. Therefore the

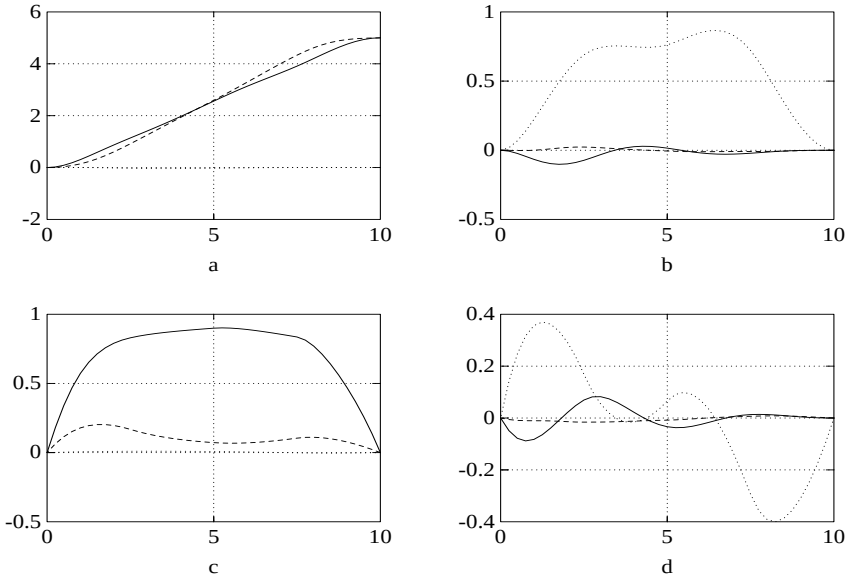

Figure 4.2: *Computed optimal trajectories*

QAG case with two parameters per control was recomputed with $acc = 0.001$. The results were

$$J^* = 25.91 \cdot 10^3$$
$$\text{iter} = 89 \text{ and } \#\text{ fc} = 177$$

The boundary conditions are considerably more difficult to satisfy than in the NG and the AG case. After 57 iterations $J^* = 16.59 \cdot 10^3$ with $acc = 0.01$, which shows how slowly the objective function grows towards the final solution. This was also observed during optimizations with more than two parameters per control. Whereas with simple boundary conditions as constraints only a couple of iterations were needed before the objective function was higher than the converged, with the error quadrature representation it increased slowly until convergence. The boundar conditions are much easier to approach when they are represented in the original form than when they are "scrambled" together into an error quadrature, which results in a higher number of iterations.

Due to numerical reasons, some differences in the results (J^*, # iter, # fc) occured also between the AG and the NG case. In the case of 5 parameters the calculations with AG were performed with the half step length ($h = 0.125$) in the integrators, which gave results much closer to NG (the same numbers of iterations and function calculations and the same objective function). That the NG method is more accurate than the AG method can also be expected from the discussion in Chapter 2.3.

The choice of an optimal number of parameters is a compromise between total CPU-time effort and optimization accuracy. From Table 4.3.3 it is seen that in the case of numerical gradient calculations the converged objective function with five parameters is only about 3% higher than with eleven parameters, whereas the

	# par	J^*	# iter	# fc	tot. CPU	sqp CPU	gc CPU
NG	11	$12.58 \cdot 10^3$	18	27	65	1.6	2.1
NG	9	$12.69 \cdot 10^3$	13	20	35	0.9	1.8
NG	6	$12.87 \cdot 10^3$	8	9	11.7	0.3	1.2
NG	5	$13.00 \cdot 10^3$	8	8	9.5	0.2	1.1
NG	3	$14.40 \cdot 10^3$	4	4	4.3	0.07	0.7
NG	2	$26.12 \cdot 10^3$	4	4	1.9	0.03	0.5
AG	11	$12.58 \cdot 10^3$	18	27	42	1.6	0.7
AG	9	$12.60 \cdot 10^3$	17	23	29	0.9	0.7
AG	6	$12.87 \cdot 10^3$	8	9	8.1	0.3	0.7
AG	5	$12.96 \cdot 10^3$	10	15	8.7	0.2	0.7
AG	3	$14.39 \cdot 10^3$	6	6	4.5	0.07	0.7
AG	2	$26.12 \cdot 10^3$	4	4	2.6	0.03	0.7
QAG	11	$12.23 \cdot 10^3$	21	39	29	1.2	0.07
QAG	9	$12.14 \cdot 10^3$	21	39	18	0.7	0.07
QAG	6	$12.18 \cdot 10^3$	25	49	10.4	0.25	0.07
QAG	5	$12.31 \cdot 10^3$	24	48	7.8	0.17	0.07
QAG	3	$13.34 \cdot 10^3$	27	55	5.8	0.04	0.07
QAG	2	$16.59 \cdot 10^3$	29	57	5.3	0.02	0.07

Table 4.1:

total CPU-time effort is only about 15%. More than 5 or 6 parameters seem to be unnecessary.

From Table 4.3.3 it is seen that with five parameters per control the NG case is moderately slower than the AG case which again is is moderately slower than the QAG case. It is also seen from the table that the reason for this is the computational costs for the gradient calculations. With the accuracy considerations taken into account (smaller step sizes for the costate equations should have been used) it is, however, no reason to conclude that the NG method is slower than the methods with analytical gradient calculation schemes, with this number of parameters. For a larger number of parameters, introducing multiple shooting would have reduced the computational costs of the numerical gradient calculation considerably, but the computational costs of the analytical gradient calculation would have been the same.

A very simple task has been solved in this section, which is advantageous for the analytical gradient calculation scheme. Obstacle avoidance in trajectory optimization for an underwater vehicle is very efficiently handled by introducing path constraints and approximating them using the time grid approach (see Section 4.5). This does not increase the computational costs for the numerical gradient computations considerably, but would have increased the computational costs of analytical gradient computations significantly. To overcome this problem, the integral approach to represent the path constraint can be introduced (Eq. 2.27) in the case of analytical gradient computations. Problems with this approach are presented in Chapter 2.2

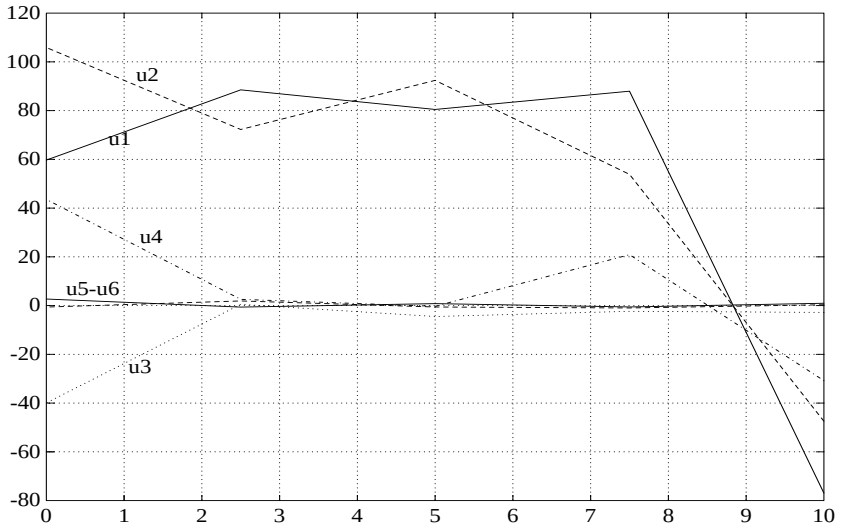

Figure 4.3: *Computed optimal thrusts*

and 2.3.

In addition the numerical approach is more flexibel because the costate equations are not needed.

Therefore, I conclude that numerical gradient calculations using forward differences are more efficient and robust than analytical gradient calculations using the costate equations when optimal trajectories for underwater vehicles are computed.

Optimal trajectories and optimal controls computed with the NG-method and with five parameters are shown in Figures 4.2-4.3.

Finally a heuristic trajectory was computed. A solution with quadratic velocity profiles in the x and y directions was used. The resulting controls are shown in Figure 4.4a. The acceleration(a), velocity(v) and the trajectory(p) are shown in Figure 4.4b. The resulting objective function was $J_h = 24.88 \cdot 10^3$, which is almost twice the computed optimal solutions.

4.4 Numerical Implementation: A 2 DOF Benchmark Problem

In this section the numerical implementation of the direct single shooting method will be presented and discussed. A small benchmark problem will be tested on the optimization program in order to investigate properties, and to draw some conclusions about which scaling to choose, and which values some optimization parameters should have. The conclusions which are drawn in this section are used in later sec-

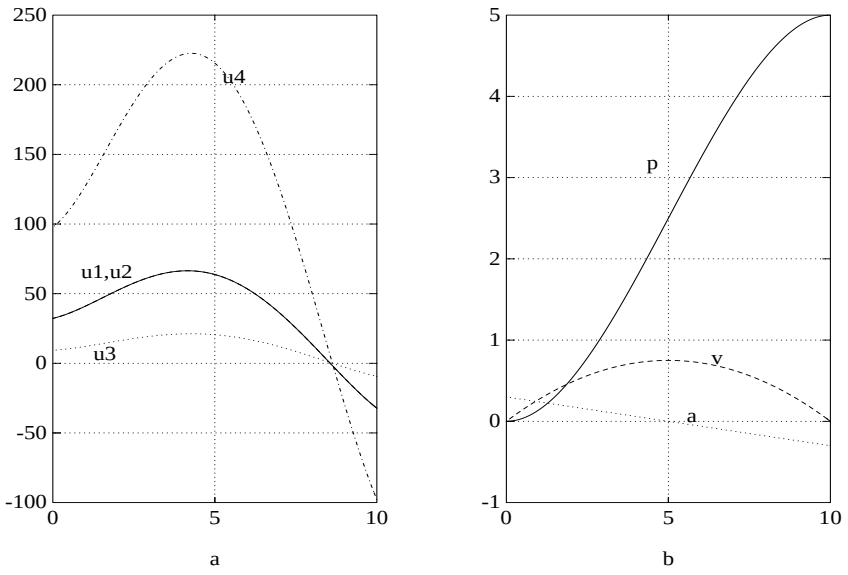

Figure 4.4: *Computed heuristic thrusts and trajectories*

tions where more realistic models are optimized. Therefore, in order to inherit important properties, the model and the optimization problems which are solved in this section have many properties similar to the physical problems in the later sections.

Scaling is an important issue in optimization in order to get good numerical properties for the solution (Gill et al. 1981). Scaling for the type of optimization problems solved in this chapter will be investigated in the context of the scaling discussion in Chapter 2.4.3. First the problem will be solved without any scaling at all. Then scaling will be introduced based on the numerical results without scaling.

Some of the implementation issues like parameterization will not be investigated in the context of the simple benchmark problem in this section. The number of parameters and the type of parameterization is fixed. The control parameters are initially set to zero. Also the introduction of path constraints will be postponed to later sections.

4.4.1 Numerical Implementation

The optimization problems considered in this chapter are solved by direct single shooting. The numerical implementation includes the possibility of choosing between piecewise linear and piecewise constant parameterization of the controls. In both cases simple box constraints on the controls lead to the same simple box constraints on the parameters (see Chapter 2.2.2). Initially I tried to include the control time nodes in the parameter set. This only gave convergence problems and no improvements in the objective functional. The implementation of the control time nodes as parameters is complex (see Chapter 2.2.2), so the difficulties may have

been a result of errors in the implementation. Therefore, the problem has also been tested using the program package PROMIS (Jänsch & Schnepper 1991), but the same problems with including the control time nodes as parameters were observed with this implementation. Therefore, the control time grid was fixed. For simplicity the time nodes were chosen to be equidistantly distributed, and equal for all controls. The number of control parameters per control is chosen initially.

In addition to the control parameters the possibility of including the final time t_f as an optimizable parameter was implemented. The extended state space equations are scaled by t_f, and the scaling factor is included in the parameter set.

A Runge-Kutta-Fehlberg 4/5th order differential equation solver was implemented for the numerical integration of the state space equations and the integral part of the objective functional, i.e. the extended state space equations equations. The derivatives are computed by the forward difference approach. The nominal unperturbed and the perturbed trajectories are integrated together using the same sequence of step sizes. The step size selection procedure is based only on the nominal trajectory. It is controlled by user defined parameters, giving bounds on the relative (*relerr*) and absolute (*abserr*) local errors. Observe that the global errors, defining the accuracy of the functions are generally larger. The step sizes are bounded above by a user defined parameter *Hmax*. This is done to avoid problems in special cases where the nominal trajectory can be computed exactly for any sequence of step sizes.

The path constraints were approximated by the time grid approach (see Chapter 2.2.2).

Initially, an initialization routine is called for defining initial values of the control parameters, and possibly the final time. The content of the routine is problem dependent. For the benchmark problem all the control parameters are set to zero.

The Fortran subroutine E04VCF from the NAG fortran library, which uses a sequential quadratic programming (SQP) method, is used as nonlinear program solver (NAG 1991). The routine E04UCF from the same library should also be consulted. These two routines are essentially equal, but most of the mutual features are better described in E04UCF. Some important optimization parameters, including some from E04VCF, are described below. For other parameters in E04VCF, see (NAG 1991).

Optimization Parameters

Important parameters for the optimization are investigated in order to choose adequate values. These are user defined parameters from E04VCF, parameters to define scaling and parameters related to the integrator and the forward difference approximation of the gradients. The parameters are

- **orthog**: E04VCF-logic-parameter which defines if the TQ-factorization should be orthogonal (*orthog*= 1) or not (*orthog*= 0). In E04UCF the TQ-factorization

is always orthogonal. This makes the optimization more robust, and this choice has been used consequently in the sequel.

- **epsaf**: E04VCF-parameter which defines a good bound on the error in computing the function values at the initial point.

- **eta:** For the step size selection. Defines how accurately α_k should approximate a univariate minimum of the merit function along $\mathbf{d}_k$. The recommended value 0.9 seemed to be a very good choice, and was used consequently.

- **ftol:** Accuracy in the final converged objective function. The magnitude of *ftol* was investigated thoroughly in this work.

- **featol(nctotl):** Array of tolerances for constraint violations, one for each constraint. *nctotl* is the number of constraints.

- **featol1:** All the tolerances corresponding to nonlinear constraints in *featol* are set to *featol1* for simplicity. A low number ($= 10^{-10}$) is chosen for the box constraints.

- **relerr:** The relative error in the variable stepsize integrator, which is related to *ftol* through the expression

$$relerr = \frac{ftol}{relftol} \tag{4.31}$$

where **relftol** is the specified parameter. This parameter is critical. On one hand it is desirable that the accuracy in function evaluations is much higher than *ftol*. On the other hand, integrating with high accuracy can be very computationally expensive since integrating takes a major part of the computation time.

- **abserr** $= 10^{-9}$: The absolute error in the variable stepsize integrator, which is included for robustness.

- **Hmax** $= \frac{0.5}{t_f}$: A maximum value on the step size in the integrator.

- **pertf:** The perturbation $\mathbf{pert}_j$ of the parameter p_j is defined as

$$pert_j = |p_j| \cdot pertf \tag{4.32}$$

where *pertf* is a small fraction of *relerr* ($\frac{1}{100}$). As long as *pertf* is considerably smaller than *relerr*, and $pert_j$ is considerably larger than the machine precision, the actual value of this quantity does not seem to be critical.

- **jscale:** The scaling factor for the objective function, i.e. the original objective function is multiplied by *jscale*.

Outputs from the Optimization Program

For each iteration the following information is given in tabular form:

- ITN(k): The current iteration number.
- ITQP: The number of minor iterations in the QP sub-problem.
- α_k: The step size taken from $\mathbf{p}_{k-1}$ to $\mathbf{p}_k$. α_0 is defined to be zero.
- Numf: The number of times the objective function has been calculated so far.
- Merit: The current value of the merit function.
- BND: The number of active boundary constraints in the working set.
- LC: The number of active linear constraints in the current working set.
- NC: The number of active nonlinear constraints in the current working set.
- NZ: The dimension of the matrix $\mathbf{Z}$, or the null space with the current working set. This number equals the number of parameters minus the total number of active constraints in the current working set, and is therefore an estimate of the number of degrees of freedom in the optimization problem.
- norm GF: The Euclidean norm of $\mathbf{g}_{FR}$, the gradient of the objective function with respect to the free parameters.
- norm GZ: The Euclidean norm of $\mathbf{Z}^{\mathrm{T}}\mathbf{g}_{FR}$, the gradient of the objective function with respect to the free parameters projected into the nul space with the current working set.
- cond H: A lower bound on the condition number of the Hessian approximation.
- cond HZ: A lower bound on the condition number of the Hessian approximation projected into the null space ($\mathbf{Z}^{\mathrm{T}}\mathbf{H}\mathbf{Z}$).
- cond T: A lower bound on the condition number of the matrix $\mathbf{T}$ in the QT-transformation of the constraints in the current working set. This matrix is an indicator of how good the mutual scaling of the constraints is.
- norm C: The Euclidean norm of the residuals of the nonlinear constraints that are violated or in the current working set.
- ρ: The Euclidean norm of the vector of penalty parameters used in the merit function.
- CONV: A three letter indication of the status of the three convergence tests, which respectively indicates whether:

    1. The sequence of iterates has converged (Eq. 2.113).

2. The projected gradient (norm Gz) is sufficiently small (Eq. 2.114).
3. Norm C is sufficiently small. (Eq. 2.115).

A T indicates that the test is satisfied, an F indicates that it is not. All the tests have to be satisfied for convergence.

In E04VCF, the status of a fourth test is also printed, indicating the status of some convergence test for the Lagrange multipliers. This test is, however, ignored in the program. It is probably a bug in the routine that this test is included in the print-outs of NAG. It is removed from the tables in this section.

- Jf: The objective function, printed at each point in the line search. The last one is the value at the next iteration. (Not a value printed by NAG.)

The following values are given after convergence:

- $\mathbf{p}$: The vector of parameter values.
- $\boldsymbol{\lambda}_{VAR}$: The vector of estimates of the Lagrange multipliers corresponding to the box constraints on the variables.
- $\mathbf{c}(\mathbf{p})$: The vector of the residuals of the nonlinear constraints.
- $\boldsymbol{\lambda}_{CON}$: The vector of estimates of the Lagrange multipliers corresponding to the nonlinear constraints.
- J^*: The final objective function.
- CPU-time: Estimated CPU-time consumption of the optimization.
- # rhs. calc.: Total number of times the right hand side of the extended state space equations were calculated.
- $\mathbf{C}(\mathbf{p})$: The Jacobian matrix of the nonlinear constraints.
- $\mathbf{g}(\mathbf{p})$: The gradient of the objective function with respect to the parameters.

4.4.2 The 2 DOF Optimization Problem

The following optimization problem is chosen:

$$\min_{\mathbf{u}} J_0 = \mathrm{w} \cdot t_f + \int_0^{t_f} (|u_1(t)|^{1.5} + |u_2(t)|^{1.5}) dt \tag{4.33}$$

subject to

$$\begin{aligned}
\dot{x}_1 &= x_3 \\
\dot{x}_2 &= x_4 \\
\dot{x}_3 &= -\frac{70}{215}x_3 - \frac{100}{215}|x_3|x_3 + \frac{1}{215}u_1 \\
\dot{x}_4 &= -\frac{100}{265}x_4 - \frac{200}{265}|x_4|x_4 + \frac{1}{265}u_2
\end{aligned} \tag{4.34}$$

where

$$\begin{aligned}
\mathbf{x}(0) &= \mathbf{0} \\
\mathbf{x}(t_f) &= \mathbf{x}_f
\end{aligned} \tag{4.35}$$

and

$$|u_i| \leq 80 \tag{4.36}$$

This is an optimal control problem in the class defined in Section 4.2.3 for a two degrees of freedom model extracted from the six degrees of freedom ROV model presented in Chapter 3.2.3. The linear motion in the x and y directions are extracted from the model. The same model parameters as in Chapter 4.5.2 are used.

The controls are approximated by a piecewise linear parameterization, and five parameters for each of the two controls are chosen. The control parameters are all chosen to be zero, partly for simplicity, but mainly because a value zero for a parameter is the only value for which the objective function does not possess the desirable smoothness properties. It is of interest to see which consequenses this have, and how these can be handled in the best way.

4.4.3 Numerical Results for Fixed Final Time

The final time was initially fixed to be $t_f = 15$ s and

$$\mathbf{x}_f = \begin{pmatrix} 5 & 5 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \tag{4.37}$$

First, some general observations about the optimization parameters were made. First it was concluded that a relative error in the integrator that was $\frac{1}{20}$ of the accuracy in the final converged objective function was a reasonable choice. Therefore,

$$relftol = \frac{ftol}{20}$$

A reasonable choice for the value of *epsaf* could now be one tenth of *ftol*. That choice was first examined. However, this choice turned out to be bad. The reason for that is that the lower bound on the legal changes in the parameters, and hence on the step size α_k is a function of *epsaf*. The reason for this is to avoid the functions to change less than the expected accuracy in the function calculations. In the optimizer it is simply assumed that if the changes get to small, improvements in the current search direction is not possible, and the optimizer terminates with a failure. However, choosing a much smaller value for *epsaf* and hence allowing the iterate to be carried out with a very small step size, the optimization always recovered in the next iterate. Since the line search procedure is not influenced by *ftol* the above choice for *epsaf* actually gave convergence when high accuracy in the optimizer was requested, while failure occurred for more crude tolerances. The following choice was finally made:

$$epsaf = 10^{-7}$$

which after extensive testing seemed to be a very good choice. The problems with step size selection occurred at the first iteration where all the parameters were zero, and for the worst scaling choice that was tested (see Table 4.4).

No Scaling

First the consequences of no scaling were investigated. Printed outputs from each iteration of an optimization with convergence specifications

$$ftol = featol1 = 10^{-2}$$

are shown in Table 4.2, and in Eq. (4.38) the converged results of the optimization are shown

$$\begin{aligned}
\mathbf{p} &= \begin{pmatrix} 53.42 & 80.00 & 46.23 & 73.35 & 43.00 & 68.12 & 42.52 & -32.33 & -49.03 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & -2. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.00143 & -0.00145 & -0.54543E-05 & -0.5777183E-05 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} -0.0014 & -0.0014 & -0.5454E-05 & -0.5777E-05 \end{pmatrix}^{\mathrm{T}} \\
J^*(\mathbf{p}) &= 11610.19 \\
\text{CPU-time} &= 0.170000 \\
\#\text{ rhs. calc.} &= 10932 \\
\mathbf{C}(\mathbf{p}) &= \begin{pmatrix} 0.01388 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 0.00825 & 0.00000 & 0.00000 \\ 0.02521 & 0.00000 & 0.00001 & 0.00000 \\ 0.00000 & 0.01472 & 0.00000 & 0.00000 \\ 0.02513 & 0.00000 & 0.00018 & 0.00000 \\ 0.00000 & 0.01462 & 0.00000 & 0.00006 \\ 0.02439 & 0.00000 & 0.00324 & 0.00000 \\ 0.00000 & 0.01464 & 0.00000 & 0.00187 \\ 0.00786 & 0.00000 & 0.00593 & 0.00000 \\ 0.00000 & 0.00516 & 0.00000 & 0.00396 \end{pmatrix}^{\mathrm{T}} \\
\mathbf{g}(\mathbf{p}) &= \begin{pmatrix} 20.087 & 38.509 & 37.082 & 27.462 & -4.5341 & 24.804 & 48.246 & 47.039 & 36.958 & -4.0361 \end{pmatrix}^{\mathrm{T}}
\end{aligned} \tag{4.38}$$

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 1.8D-2	0.0D+0 6.33D-8	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 6117.5 Jf = 1076.1						
1	1 4.6D+1	3.1D-1 5.05D+0	3 1.D+0	9.2361D+3 1.D+0	0 1.D+1	0 5.18D+0	4 5.6D+2	6 TFF
		Jf = 8725.7						
2	3 9.3D+1	1.0D+00 7.70D+0	4 1.D+0	1.2976D+04 1.D+0	0 1.D+1	0 1.06D+0	4 2.0D+3	6 TFF
		Jf = 11570.9						
3	1 1.0D+2	1.0D+0 6.22D+0	5 1.D+0	1.1691D+4 1.D+0	0 1.D+1	0 4.28D-2	4 9.0D+2	6 TFF
		Jf = 11653.6						
4	2 1.0D+2	1.0D+0 2.87D+0	6 2.D+0	1.1656D+4 1.D+0	1 1.D+1	0 8.16D+0	4 5.0D+1	5 TFT
		Jf = 11610.2						
5	1 1.0D+2	1.0D+0 1.33D+0	7 2.D+0	1.1617D+4 2.D+0	1 1.D+1	0 2.03D-3	4 1.2D+1	5 TTT

Table 4.2:

The same printed outputs are shown for a higher accuracy in the convergence specifications

$$ftol = featol1 = 10^{-5}$$

in Table 4.3 and Eq. (4.39).

$$
\begin{aligned}
\mathbf{p} &= \begin{pmatrix} 58.17 & 80.00 & 42.97 & 71.77 & 43.44 & 67.56 & 42.90 & 76.50 & -32.54 & -50.29 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & -2. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.21541E-05 & -0.52959E-07 & -0.72565E-07 & -0.34688E-08 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} 1493. & 3236. & -2716. & -5248. \end{pmatrix}^{\mathrm{T}} \\
Jf^{*} &= 11612.52 \\
\text{CPU-time} &= 0.33 \\
\#\ \text{rhs. calc.} &= 21246 \\
\mathbf{C}(\mathbf{p}) &= \begin{pmatrix} 0.01375 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 0.00828 & 0.00000 & 0.00000 \\ 0.02539 & 0.00000 & 0.00001 & 0.00000 \\ 0.00000 & 0.01481 & 0.00000 & 0.00000 \\ 0.02520 & 0.00000 & 0.00018 & 0.00000 \\ 0.00000 & 0.01462 & 0.00000 & 0.00006 \\ 0.02434 & 0.00000 & 0.00323 & 0.00000 \\ 0.00000 & 0.01452 & 0.00000 & 0.00184 \\ 0.00785 & 0.00000 & 0.00592 & 0.00000 \\ 0.00000 & 0.00513 & 0.00000 & 0.00395 \end{pmatrix}^{\mathrm{T}} \\
\mathbf{g}(\mathbf{p}) &= \begin{pmatrix} 20.483 & 37.950 & 37.001 & 27.681 & -4.4245 & 24.719 & 47.868 & 46.976 & 37.420 & -4.1446 \end{pmatrix}^{\mathrm{T}}
\end{aligned}
\tag{4.39}
$$

The optimization performs well in each iteration, both in the case of crude and tight convergence specifications. This is natural since the conditioning of the optimization problem is good. The condition number of the matrix $\mathbf{T}$ is not larger than about

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 5.6D-4	0.0D+0 2.00D-9	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 6117.44 Jf = 1075.85						
1	1 4.6D+1	3.1D-1 5.07D+0	3 1.D+0	9.2346D+3 1.D+0	0 1.D+1	0 5.18D+0	4 5.6D+2	6 FFF
		Jf = 8725.41						
2	3 9.3D+1	1.0D+0 7.67D+0	4 1.D+0	1.2974D+4 1.D+0	0 1.D+1	0 1.06D+0	4 2.0D+3	6 TFF
		Jf = 11570.87						
3	1 1.0D+2	1.0D+0 6.22D+00 1.D+0	5 1.D+0	1.1691D+4 1.D+1	0 4.28D-2	0 9.1D+2	4 TFF	6
		Jf = 11653.69						
4	2 1.0D+2	1.0D+0 2.87D+0	6 2.D+0	1.1656D+4 1.D+0	1 1.D+1	0 8.17D+0	4 5.1D+1	5 TFF
		Jf = 11610.22						
5	1 1.0D+2	1.0D+0 1.34D+0	7 2.D+0	1.1617D+4 2.D+0	1 1.D+1	0 2.03D-3	4 1.2D+1	5 TFF
		Jf = 11615.00						
6	1 1.0D+2	1.0D+0 1.01D+0	8 2.D+0	1.1615D+4 4.D+0	1 1.D+1	0 3.80D-5	4 5.9D+0	5 TFF
		Jf = 11611.85						
7	1 1.0D+2	1.0D+0 3.08D-1	9 2.D+0	1.1613D+4 5.D+0	1 1.D+1	0 3.26D-4	4 5.9D+0	5 TFF
		Jf = 11612.52						
8	1 1.0D+2	1.0D+0 2.48D-1	10 2.D+0	1.1613D+4 4.D+0	1 1.D+1	0 2.16D-6	4 5.9D+00	5 TTT

Table 4.3:

10, because the nonlinear constraints are mutually well conditioned with respect to perturbations of the parameters. This can be seen from the Jacobian matrices. The Hessian matrix is also well conditioned since the parameters are equally important in the objective function, and parameter and gradient values are of the same magnitude. The absolute value of the Lagrange multiplier for the active box constraint is large compared to the absolute values of the Lagrange multipliers for the nonlinear constraints. In another case, a large variation in the magnitude of the Lagrange multipliers as a result of bad scaling could have given problems in the active set strategy.

The parameters and the objective function are not scaled to be within the order of magnitude 1. It may seem to be unnecessary to satisfy this common rule of scaling. The convergence properties of the optimization problem is very good in the sense that convergence is reached in few iterations even for high accuracy in the convergence specifications, and that the line search is everywhere well behaved.

However, in addition to the possible advantages for the active set strategy, scaling can be advantageous for the sake of balancing the convergence criteria. From the column CONV in Figures 4.2 and 4.3 it can be seen that convergence criteria 1 is

extremely easily satisfied. This is natural since the value of the objective function is large compared to the norm of its gradient (see Eq. 2.114). In practice there is only two convergence criterias. For more advanced problems better initial estimates of the parameters may be necessary. This can have the effect that the constraints are satisfied at iteration zero, and the optimization will terminate successfully even if it is far from optimum. This because convergence criteria 3 is not efficient before after the first iteration.

Scaling only the Parameters

Next, all the control parameters were scaled by the factor $\frac{1}{80}$ such that they all are constrained to be in the interval $[-1\ 1]$. Printed outputs from an optimization with convergence tolerances

$$ftol = featol1 = 10^{-2}$$

are shown in Table 4.4 and Eq. (4.40)

$$\begin{aligned}
\mathbf{p} &= \begin{pmatrix} 0.76248 & 1.00000 & 0.51163 & 0.88937 & 0.56350 & 0.85812 & 0.51786 & 0.94895 & -0.39764 & -0.62542 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & -174. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.00170 & -0.433088E-03 & -0.140187E-03 & -0.604289E-06 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} 1491. & 3239. & -2760. & -5223. \end{pmatrix}^{\mathrm{T}} \\
J^* &= 11609.84 \\
\text{CPU-time} &= 0.64 \\
\#\ \text{rhs. calc.} &= 42432 \\
\mathbf{C}(\mathbf{p}) &= \begin{pmatrix} 1.09453 & 0.00000 & 0.00003 & 0.00000 \\ 0.00000 & 0.66345 & 0.00000 & 0.00000 \\ 2.03566 & 0.00000 & 0.00071 & 0.00000 \\ 0.00000 & 1.18583 & 0.00000 & 0.00012 \\ 2.01218 & 0.00000 & 0.01461 & 0.00000 \\ 0.00000 & 1.16721 & 0.00000 & 0.00453 \\ 1.95380 & 0.00000 & 0.26062 & 0.00000 \\ 0.00000 & 1.16232 & 0.00000 & 0.14770 \\ 0.62985 & 0.00000 & 0.47483 & 0.00000 \\ 0.00000 & 0.41102 & 0.00000 & 0.31583 \end{pmatrix}^{\mathrm{T}} \\
\mathbf{g}(\mathbf{p}) &= \begin{pmatrix} 1656.5 & 3014.3 & 2977.3 & 2182.6 & -365.29 & 1974.8 & 3823.4 & 3772.3 & 2996.7 & -319.81 \end{pmatrix}^{\mathrm{T}}
\end{aligned} \tag{4.40}$$

Now the value of the objective function and the norm of its gradient are in the same order of magnitude. However, the parameter values are very small compared to the norm of the gradient of the objective function, and consequently the derivative of the merit function. The third term on the right hand side of Eq. (2.108) will therefore be large, and consequently dominate the quasi-Newton update of the Hessian approximation. This matrix therefore soon becomes ill-conditioned. It can be seen that the convergence properties of the optimization is not good. Many more iterations are needed to achieve convergence, and the step sizes in the line searches are small. At the first iteration an acceptable univariate decrease is hard to get. It can also be seen that it is hard to find the correct active set, and at the first iterations the maximum number of parameters (6) are assumed to be active at a

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 1.3D+1	0.0D+0 4.77D-5	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 6117.5 Jf = 2.568D-3 Jf = 1.324D-5						
1	6 1.3D+1	1.7D-6 0.00D+0	4 4.D+4	1.5141D+1 1.D+0	6 1.D+2	0 7.48D+0	4 8.6D-1	0 TFT
		Jf = 9265.9						
2	4 3.3D+3	1.0D+0 0.00D+0	5 2.D+4	1.3380D+5 1.D+0	6 8.D+0	0 2.91D+0	4 9.4D+4	0 TFF
		Jf = 12842.1						
3	2 3.8D+3	1.0D+0 0.00D+0	6 2.D+3	7.0704D+4 1.D+0	6 9.D+0	0 2.34D+0	4 9.4D+4	0 TFF
		Jf = 14824.8 Jf = 11407.4						
10	4 6.2D+3	1.0D-1 1.82D+2	20 9.D+2	1.1616D+4 2.D+0	3 1.D+1	0 3.29D-1	4 3.0D+3	3 TFT
		Jf = 12651.2 Jf = 11134.8						
11	2 7.3D+3	7.0D-2 3.01D+1	22 2.D+3	1.1613D+4 8.D+0	2 1.D+1	0 3.1D-01	4 3.0D+3	4 TFT
		Jf = 11608.6 Jf = 11251.0						
12	3 6.9D+3	2.5D-1 2.03D+1	24 2.D+3	1.1612D+4 8.D+0	2 1.D+1	0 2.80D-1	4 3.0D+3	4 TFT
		Jf = 11643.4 Jf = 11262.8						
13	4 7.9D+3	3.4D-2 2.18D+1	26 2.D+3	1.1612D+4 8.D+0	1 1.D+1	0 2.14D-1	4 3.0D+3	5 TFT
		Jf = 11609.8						
14	1 8.0D+3	1.0D+0 4.43D+1	27 2.D+3	1.1613D+4 3.D+1	1 1.D+1	0 1.77D-3	4 3.5D+3	5 TTT

Table 4.4:

boundary value. It can be seen that the Lagrange multipliers are far from unity. Unnecessary problems with the active set strategy is a natural consequence of bad scaling that leads to high Lagrange multipliers. This is the main reason why the first convergence criteria never becomes active here.

Scaling the Parameters and the Objective Function

It is obvious that the objective function also has to be scaled. It is a rule of thumb that the magnitude of the objective function should be in the order no larger than unity in the region of interest. Following this rule a good choice for scaling is

$$jscale = 10^{-4}$$

$$\mathrm{p} = \begin{pmatrix} 0.48260 & 0.83932 & 0.69280 & 0.99397 & 0.63878 & 0.99816 & 0.39801 & 0.76963 & -0.33330 & -0.54347 \end{pmatrix}^{\mathrm{T}}$$

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 1.3D-3	0.0D+0 4.77D-9	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 0.61175						
1	3 4.7D-1	1.0D+0 7.78D-2	2 1.D+0	1.3393D+0 1.D+0	2 1.D+1	0 2.26D+0	4 8.7D-2	4 TFF
		Jf = 1.13196						
2	3 8.2D-1	1.0D+0 8.99D-2	3 1.D+0	1.1803D+0 1.D+0	0 1.D+1	0 1.96D-1	4 2.2D-1	6 TFF
		Jf = 1.17822						
3	1 8.3D-1	1.0D+0 7.21D-2	4 2.D+0	1.1788D+0 2.D+0	0 1.D+1	0 2.98D-3	4 6.5D-1	6 TTT

Table 4.5:

$$
\begin{aligned}
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.00136 & -0.00258 & -0.27001E-03 & -0.57590E-03 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} 0.1547 & 0.3280 & -0.3141 & -0.5767 \end{pmatrix}^{\mathrm{T}} \\
J^* &= 1.178217 \\
\text{CPU-time} &= 0.11 \\
\#\ \text{rhs. calc.} &= 6204 \\
\mathbf{C}(\mathbf{p}) &= \begin{pmatrix} 1.14617 & 0.00000 & 0.00003 & 0.00000 \\ 0.00000 & 0.67828 & 0.00000 & 0.00000 \\ 1.95885 & 0.00000 & 0.00067 & 0.00000 \\ 0.00000 & 1.15726 & 0.00000 & 0.00012 \\ 1.94656 & 0.00000 & 0.01639 & 0.00000 \\ 0.00000 & 1.13373 & 0.00000 & 0.00531 \\ 2.01390 & 0.00000 & 0.27703 & 0.00000 \\ 0.00000 & 1.20238 & 0.00000 & 0.15807 \\ 0.64288 & 0.00000 & 0.48228 & 0.00000 \\ 0.00000 & 0.42099 & 0.00000 & 0.32162 \end{pmatrix}^{\mathrm{T}} \\
\mathbf{g}(\mathbf{p}) &= \begin{pmatrix} 0.14946 & 0.32399 & 0.31329 & 0.19823 & -0.03758 & 0.18991 & 0.39611 & 0.39611 & 0.27756 & -0.03518 \end{pmatrix}^{\mathrm{T}}
\end{aligned}
\tag{4.41}
$$

In Table 4.5 and Eq. (4.41) printed outputs from an optimization with this scaling and with convergence tolerances

$$ftol = featol1 = 10^{-2}$$

are shown.

$$
\begin{aligned}
\mathbf{p} &= \begin{pmatrix} 0.65623 & 1.00000 & 0.57000 & 0.89008 & 0.54114 & 0.86008 & 0.54585 & 0.94623 & -0.41204 & -0.62418 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & -0.0175 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.10128E-03 & -0.27524E-04 & -0.32447E-05 & -0.39947E-05 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} 0.1500 & 0.3241 & -0.2734 & -0.5232 \end{pmatrix}^{\mathrm{T}} \\
J^* &= 1.161627 \\
\text{CPU-time} &= 0.16 \\
\#\ \text{rhs. calc.} &= 10932
\end{aligned}
$$

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 4.0D-4	0.0D+0 1.44D-9	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 0.61175						
1	3 4.7D-1	1.0D+0 7.78D-2	2 1.D+0	1.3389D+0 1.D+0	2 1.D+1	0 2.26D+0	4 8.7D-2	4 FFF
		Jf = 1.13196						
2	3 8.2D-1	1.0D+0 8.99D-2	3 1.D+0	1.1803D+0 1.D+0	0 1.D+1	0 1.96D-1	4 2.2D-1	6 FFF
		Jf = 1.17822						
3	1 8.3D-1	1.0D+0 7.21D-2	4 2.D+0	1.1788D+0 2.D+0	0 1.D+1	0 2.98D-3	4 6.5D-1	6 FFF
		Jf = 1.16808						
4	2 8.1D-1	1.0D+0 4.01D-2	5 2.D+0	1.1687D+0 1.D+0	1 1.D+1	0 8.44D-2	4 6.5D-1	5 TFF
		Jf = 1.16072						
5	1 8.0D-1	1.0D+0 1.20D-2	6 2.D+0	1.1619D+0 2.D+0	1 1.D+1	0 4.71D-3	4 6.5D-1	5 TFF
		Jf = 1.16163						
6	1 8.0D-1	1.0D+0 9.09D-3	7 2.D+0	1.1616D+0 2.D+0	1 1.D+1	0 1.05D-4	4 6.5D-1	5 TTT

Table 4.6:

$$\mathbf{C(p)} = \begin{pmatrix} 1.11670 & 0.00000 & 0.00003 & 0.00000 \\ 0.00000 & 0.66336 & 0.00000 & 0.00000 \\ 2.02522 & 0.00000 & 0.00068 & 0.00000 \\ 0.00000 & 1.18539 & 0.00000 & 0.00012 \\ 2.00609 & 0.00000 & 0.01417 & 0.00000 \\ 0.00000 & 1.16682 & 0.00000 & 0.00455 \\ 1.94022 & 0.00000 & 0.25706 & 0.00000 \\ 0.00000 & 1.16291 & 0.00000 & 0.14785 \\ 0.62695 & 0.00000 & 0.47315 & 0.00000 \\ 0.00000 & 0.41116 & 0.00000 & 0.31592 \end{pmatrix}^{\mathrm{T}} \tag{4.42}$$

$$\mathbf{g(p)} = \begin{pmatrix} 0.15939 & 0.30635 & 0.297610.22246 & -0.03621 & 0.19751 & 0.38251 & 0.37743 & 0.29935 & -0.03202 \end{pmatrix}^{\mathrm{T}}$$

In Table 4.6 and Eq. (4.42) printed outputs from an optimization with the same scaling, but with convergence tolerances

$$ftol = featol1 = 10^{-3}$$

are shown. It can be seen that the optimization performance is very good for this scaling. For the most accurate of these convergence tolerances convergence criteria 1 is active, and should therefore be preferred for the sake of robustness.

The results were compared to a scaling of the objective function

$$jscale = 10^{-3}$$

$$\mathbf{p} = \begin{pmatrix} 0.75615 & 1.00000 & 0.50325 & 0.87858 & 0.54759 & 0.85855 & 0.55078 & 0.95793 & -0.41551 & -0.63079 \end{pmatrix}^{\mathrm{T}}$$

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 1.3D-2	0.0D+0 4.77D-8	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 6.1175 Jf = 0.0769 Jf = 0.0109						
1	4 7.5D-1	1.5D-2 2.11D-2	4 3.D+0	1.5491D+0 1.D+0	3 2.D+1	0 7.18D+0	4 8.8D-2	3 TFT
		Jf = 8.4493						
2	7 5.8D+0	1.0D+0 1.02D+0	5 2.D+0	1.4710D+1 2.D+0	2 1.D+1	0 2.76D+0	4 3.1D+0	4 FFF
		Jf = 10.0991						
3	4 5.6D+0	1.0D+0 1.83D+0	6 3.D+0	1.2937D+1 2.D+0	3 1.D+1	0 1.02D+0	4 3.1D+0	3 FFF
		Jf = 11.4575						
4	3 8.0D+0	1.0D+0 7.12D-1	7 3.D+0	1.1787D+1 2.D+0	1 1.D+1	0 1.71D-1	4 3.5D+0	5 TFF
		Jf = 11.5758						
5	1 7.9D+0	1.0D+0 4.90D-1	8 4.D+0	1.1664D+1 2.D+0	1 1.D+1	0 3.16D-2	4 3.5D+0	5 TFF
		Jf = 11.5952						
6	1 8.0D+0	1.0D+0 9.04D-2	9 3.D+0	1.1614D+1 2.D+0	1 1.D+1	0 6.81D-3	4 3.5D+0	5 TTT

Table 4.7:

$$
\begin{aligned}
\boldsymbol{\lambda}_{VAR} &= \begin{pmatrix} 0. & -0.1787 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}} \\
\mathbf{c}(\mathbf{p}) &= \begin{pmatrix} -0.005013 & -0.004586 & -0.33222E-03 & -0.37374E-03 \end{pmatrix}^{\mathrm{T}} \\
\boldsymbol{\lambda}_{CON} &= \begin{pmatrix} 1. & 3. & -3. & -5. \end{pmatrix}^{\mathrm{T}} \\
J^* &= 11.60 \\
\text{CPU-time} &= 0.23 \\
\#\ \text{rhs. calc.} &= 14004 \\
\mathbf{C}(\mathbf{p}) &= \begin{pmatrix} 1.09946 & 0.00000 & 0.00003 & 0.00000 \\ 0.00000 & 0.66474 & 0.00000 & 0.00000 \\ 2.04856 & 0.00000 & 0.00070 & 0.00000 \\ 0.00000 & 1.18936 & 0.00000 & 0.00012 \\ 2.01779 & 0.00000 & 0.01407 & 0.00000 \\ 0.00000 & 1.16733 & 0.00000 & 0.00449 \\ 1.93640 & 0.00000 & 0.25640 & 0.00000 \\ 0.00000 & 1.15975 & 0.00000 & 0.14727 \\ 0.62643 & 0.00000 & 0.47294 & 0.00000 \\ 0.00000 & 0.41059 & 0.00000 & 0.31566 \end{pmatrix}^{\mathrm{T}} \\
\mathbf{g}(\mathbf{p}) &= \begin{pmatrix} 1.6479 & 2.9890 & 2.9596 & 2.2356 & -0.36323 & 1.9711 & 3.7721 & 3.0086 & -0.32052 \end{pmatrix}^{\mathrm{T}}
\end{aligned}
\tag{4.43}
$$

for the same tolerances. The results are shown in Table 4.7 and Eq. (4.43) ($ftol = featol1 = 10^{-2}$) and Table 4.8 and Eq. (4.44) ($ftol = featol1 = 10^{-3}$) respectively. Some more iterations were needed with this scaling. While $\alpha_0 = 1$ for $jscale = 10^{-4}$, the line search now resulted in a small step length, and two extra function computations were needed. An advantage with $jscale = 10^{-3}$ is that convergence criteria 1 is more easily activated. This is because of the two unity parts of the criteria (see Eq. 2.114).

$$
\mathbf{p} = \begin{pmatrix} 0.73237 & 1.00000 & 0.53028 & 0.90236 & 0.56278 & 0.84354 & 0.51633 & 0.95098 & -0.39662 & -0.62616 \end{pmatrix}^{\mathrm{T}}
$$

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 4.0D-3	0.0D+0 1.44D-8	1 1.D+0	0.0000D+0 1.D+0	0 2.D+1	0 7.07D+0	4 0.0D+0	6 TFT
		Jf = 6.1175 Jf = 0.0768 Jf = 0.0108						
1	4 7.4D-1	1.5D-2 2.10D-2	4 3.D+0	1.5399D+0 1.D+0	3 2.D+1	0 7.18D+0	4 8.7D-2	3 TFT
		Jf = 8.4674						
2	7 5.8D+0	1.0D+0 1.03D+0	5 2.D+0	1.4807D+1 2.D+0	2 1.D+1	0 2.77D+0	4 3.2D+0	4 FFF
		Jf = 10.0943						
3	4 5.6D+0	1.0D+0 1.82D+0	6 3.D+0	1.2944D+1 2.D+0	3 1.D+1	0 1.02D+0	4 3.2D+0	3 FFF
		Jf = 11.4584						
4	3 8.0D+0	1.0D+0 7.12D-1	7 3.D+0	1.1788D+1 2.D+0	1 1.D+1	0 1.70D-1	4 3.6D+0	5 FFF
		Jf = 11.5760						
5	3 7.9D+0	1.0D+0 4.88D-1	8 4.D+0	1.1663D+1 2.D+0	1 1.D+1	0 3.13D-2	4 3.6D+0	5 FFF
		Jf = 11.5953						
6	1 8.0D+0	1.0D+0 9.03D-2	9 3.D+0	1.1614D+1 2.D+0	1 1.D+1	0 6.78D-3	4 3.6D+0	5 TFF
		Jf = 11.6117						
7	1 8.0D+0	1.0D+0 3.30D-2	10 3.D+0	1.1613D+1 2.D+0	1 1.D+1	0 6.32D-4	4 3.6D+0	5 TTT

Table 4.8:

$$\boldsymbol{\lambda}_{VAR} = \begin{pmatrix} 0. & -0.1635 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}^{\mathrm{T}}$$

$$\mathbf{c(p)} = \begin{pmatrix} -0.62065E-03 & -0.10707E-03 & -0.52433E-04 & -0.15842E-05 \end{pmatrix}^{\mathrm{T}}$$

$$\boldsymbol{\lambda}_{CON} = \begin{pmatrix} 1. & 3. & -3. & -5. \end{pmatrix}^{\mathrm{T}}$$

$$J^* = 11.61$$

$$\text{CPU-time} = 0.23$$

$$\#\text{ rhs. calc.} = 15570$$

$$\mathbf{C(p)} = \begin{pmatrix} 1.09994 & 0.00000 & 0.00003 & 0.00000 \\ 0.00000 & 0.66200 & 0.00000 & 0.00000 \\ 2.02934 & 0.00000 & 0.00070 & 0.00000 \\ 0.00000 & 1.18318 & 0.00000 & 0.00012 \\ 2.00816 & 0.00000 & 0.01463 & 0.00000 \\ 0.00000 & 1.17028 & 0.00000 & 0.00456 \\ 1.95468 & 0.00000 & 0.26081 & 0.00000 \\ 0.00000 & 1.16331 & 0.00000 & 0.14768 \\ 0.63001 & 0.00000 & 0.47490 & 0.00000 \\ 0.00000 & 0.41096 & 0.00000 & 0.31579 \end{pmatrix}^{\mathrm{T}} \tag{4.44}$$

$$\mathbf{g(p)} = \begin{pmatrix} 1.6401 & 3.0349 & 2.9838 & 2.1797 & -0.36504 & 1.9793 & 3.8365 & 3.75659 & 2.9946 & -0.31928 \end{pmatrix}^{\mathrm{T}}$$

General Discussion

The nonlinear constraints in these optimization problems are well scaled, and the influence of the parameters are absolutely balanced. The only scaling possibly needed is to get the parameters and the objective function to unit order of magnitude. How-

ever, for a model with larger differences in the dynamic properties of the different degrees of freedom this may not be true. It may then be advantageous to scale the constraints mutually, or to use a scaled vehicle model. The most obvious consequence of bad scaling is the possible difficulty in updating the penalty parameters in the line search. For the case of parameter scaling only, the norm of ρ (see column ρ of Table 4.4) becomes very small compared to the values of the objective function computed in the line search.

In the sequel the parameters will be scaled to be within 1 and -1. The objective function will be scaled to be in the order of magnitude 1. It should be noted that the first objective is also achieved by redefining the control inputs to be within 1 and -1, and increasing their influence on the model by a factor of 80 (multiplying the B-matrix by 80). The Lagrangian part of the objective functional in Eq. (4.33) will also be changed, and consequently the scaling should be different. This was done in (Spangelo & Egeland 1994).

The value of *featol1* has been chosen equal to *ftol* in these optimizations for simplicity, even if cruder values for the nonlinear constraints would physically be more adequate. This because this value affects the result only if it is so strict that the other convergence criteria are satisfied before it is. In later sections the choice for this value will be based on physical arguments. The value of *ftol*, however, is important. In the optimizations in this section, the computation time is low for any value of *ftol*. But for more complicated models this value should not be to low. On the other hand, if the convergence criteria is crude, the risk of termination far from optimum increases. The results from this section will later be used as guidelines for choosing a proper optimization tolerance.

4.4.4 Numerical Results for Free Final Time

Finally the optimization problem was solved with free final time t_f. The control parameters were scaled to be within -1 and 1, and the scaling for the objective function was chosen to be 10^{-4}, which was concluded to be a good scaling for the free final time case. Scaling the objective function with 10^{-3} was also tried here, but clearly did not perform as well as the ones with 10^{-4}. The tolerances used if not otherwise mentioned, were *ftol* $= 10^{-3}$ and *featol1* $=10^{-2}$.

A physically not obvious problem is to define box constraints for the final time parameter. Independently of the weighting factor, the optimizer always wants to reduce the time consumption in the first iterate when the initial guess for the control parameters are zero. This because the partial derivative of the objective function with respect to t_f is always positive while the partial derivatives of all the constraints with respect to t_f are zero when all the control parameters are zero. It was observed that the optimizer usually tried to reduce t_f in the first iterate also for other initial values of the control parameters. The initial guess for the final time was therefore chosen as the lower bound for this parameter. It is then assumed that the initial guess for the final time is low. As shown below, this is the case for the final time

of 15 s, used in the fixed final time case, and used as initial guess in the free final time case. The upper bound on the final time was chosen to be four times the lower bound, which numerically seemed to be a good choice. On the other hand, it also gives a large interval of feasible values for t_f. In contrast to the box constraints on the control parameters, the box constraints on the final time parameter is chosen more based on numerical than physical arguments.

Optimizations were performed with and without scaling of the final time parameter. Table 4.9 shows numerical results for optimizations without scaling of the final time parameters for different values of the weighting factor w. For quantities that coincide with those in the tables for the fixed final time case, converged values are shown here. In the last line *ftol* is 10^{-4}. For this w the optimization with *ftol* = 10^{-3} converged prematurely.

w	J^*	t_f	# rhs. calc.	ITN	Numf	cond H	cond HZ	cond T
10	0.3638	60.0	46958	18	26	1.D+4	2.D+0	4.D+1
100	0.8200	36.46	28295	15	17	2.D+3	1.D+3	3.D+1
500	1.9215	15.18	6496	3	4	2.D+0	2.D+0	1.D+1
500(2)	1.8266	19.25	28643	15	16	2.D+2	1.D+2	1.D+1

Table 4.9:

Table 4.10 shows the same numerical results for optimizations with the final time parameter scaled to be between 1 and 4. There was no problem to converge to the correct t_f for w = 500 in this case. It is also seen that the optimization performance is generally better with this scaling than without. From the rightmost column of Table 4.9 and 4.10 it is seen that the conditioning of the constraints is not much affected by the scaling of t_f. It is, however, moderately affected by the final value of t_f. This is quite natural since the influence of the control parameters on the position coordinate part of the constraint jacobian matrix increase more rapidly with increasing t_f than their influence on the velocity part.

The conditioning of the Hessian approximation matrix is significantly improved by the scaling of the final time parameter. The final time parameter varies between 15 and 60 without scaling, whereas the control parameters only vary between -1 and 1. This affects the conditioning of the Hessian negatively since the second term of the BFGS-update in Eq. (2.108) is dominated by the final time parameter. In the diagonal of the Hessian-factorization (which can be printed from E04VCF, but not shown here), the term corresponding to the final time parameter becomes very small compared to the other terms.

The two last lines in Table 4.9 illustrate how slowly the objective function is changed in the direction of the final time parameter. This problem is resolved by the scaling of the final time parameter. In the case of w = 10 the conditioning of the Hessian approximation matrix is not good even with the scaling of the final time parameter.

In Table 4.11 the same optimizations as in Table 4.10 are performed with the difference that the initial guess for t_f is 30 s., while the box constraints still are 15 and 60

w	J^*	t_f	# rhs. calc.	ITN	Numf	cond H	cond HZ	cond T
10	0.3641	60.0	32042	12	17	7.D+2	2.D+0	4.D+1
100	0.8192	37.62	20395	9	12	5.D+1	6.D+0	3.D+1
500	1.8289	18.75	9760	5	6	5.D+0	4.D+0	2.D+1

Table 4.10:

seconds. Now the conditioning of the Hessian is significantly improved when w = 10, whereas it is degraded when w = 500. This indicates that it is the variation of the final time parameter during the optimization and not the value of the converged t_f that is important for the conditioning of the Hessian approximation. This further confirms the importance of the initial guess for and the box constraints on t_f. It suggests that maybe tighter bounds on t_f should be chosen.

w	J^*	t_f	# rhs. calc.	ITN	Numf	cond H	cond HZ	cond T
10	0.3639	60.0	33456	8	11	3.D+1	1.D+0	4.D+1
100	0.8205	36.54	24160	6	8	1.D+1	1.D+0	3.D+1
500	1.8262	19.15	30256	8	10	2.D+1	4.D+0	2.D+1

Table 4.11:

The change of the objective function in the direction of the final time parameter is small for the scaled problem too. That is shown by the difference in converged final time in Table 4.10 and 4.11. This in spite of the fact that the partial derivatives of the constraints and the objective function with respect to the final time parameter are not small compared to the other partial derivatives. In the case of w = 500 in Table 4.10, the jacobian of the constraints and the gradient of the objective function are (the eleventh parameter is the final time parameter):

$$\mathbf{C(p)} = \begin{pmatrix} 1.55056 & 0.00000 & 0.00001 & 0.00000 \\ 0.00000 & 0.93428 & 0.00000 & 0.00000 \\ 2.87484 & 0.00000 & 0.00034 & 0.00000 \\ 0.00000 & 1.70341 & 0.00000 & 0.00006 \\ 2.82802 & 0.00000 & 0.00952 & 0.00000 \\ 0.00000 & 1.67329 & 0.00000 & 0.00318 \\ 2.70397 & 0.00000 & 0.25484 & 0.00000 \\ 0.00000 & 1.64775 & 0.00000 & 0.15094 \\ 0.91240 & 0.00000 & 0.54269 & 0.00000 \\ 0.00000 & 0.60222 & 0.00000 & 0.36468 \\ 4.15960 & 4.12140 & -0.13239 & -0.12703 \end{pmatrix}^{\mathrm{T}}$$

$$\mathbf{g(p)} = \begin{pmatrix} 0.167 & 0.319 & 0.316 & 0.260 & -0.022 & 0.207 & 0.400 & 0.398 & 0.335 & -0.0187 & 1.463 \end{pmatrix}^{\mathrm{T}} \quad (4.45)$$

In Table 4.12 printed outputs from this optimization is shown.

4.4.5 Integration Accuracy Lower Than Optimization Accuracy

It is natural that it is not possible to obtain a higher accuracy in the optimal objective function than the accuracy to which the objective and constraint functions

ITN (k)	ITQP norm GF	α_k norm GZ	Numf cond H	Merit cond HZ	BND cond T	LC Norm C	NC ρ	NZ CONV
0	1 4.0D-4	0.0D+0 1.44D-9	1 1.D+0	7.5000D-1 1.D+0 2.D+1	1 7.07D+0	0 0.0D+0	4 TFT	6
		Tf = 15.00						
		Jf = 1.3617						
1	6 1.5D+0	1.0D+00 1.36D-1	2 1.D+0	2.0643D+00 1.D+0	0 1.D+1	0 2.22D+0	4 8.7D-2	7 FFF
		Tf = 18.17						
		Jf = 1.9338						
2	1 1.8D+0	1.0D+00 1.63D-1	3 4.D+0	1.8685D+00 2.D+0	0 2.D+1	0 2.07D-1	4 8.7D-2	7 FFF
		Tf = 17.98						
		Jf = 1.8571						
3	1 1.8D+0	1.0D+0 1.26D-1	4 3.D+0	1.8579D+0 3.D+0	0 2.D+1	0 3.38D-3	4 2.4D-1	7 FTF
		Tf = 18.60						
		Jf = 1.8192						
4	1 1.7D+0	1.0D+0 4.53D-2	5 3.D+0	1.8301D+0 3.D+0	0 2.D+1	0 5.09D-2	4 2.4D-1	7 TFF
		Tf = 18.75						
		Jf = 1.8289						
5	1 1.7D+0	1.0D+0 3.90D-2	6 5.D+0	1.8289D+0 4.D+0	0 2.D+1	0 2.59D-4	4 2.4D-1	7 TTT

Table 4.12:

can be calculated. However, considering the discussion about mathematical representation of "real world" models in connection with direct shooting in Chapter 2.3, this does not necessarily mean that the global accuracy in the integrator have to be higher than the optimization accuracy.

This was tested for the small 2 DOF problem, and some interesting observations were made. First, a very high optimization accuracy was tested

$$ftol = 10^{-8}$$

It turned out that whereas convergence was not achieved for a very high accuracy in the integrator, i.e. $relerr = 10^{-10}$, it was achieved for lower accuracies, i.e. $relerr = 10^{-3}$. This can be explained by the nature of the variable step size integrator. The accuracy of the functions near the solution depend on the change of the step sizes during the last iterations. It is reasonable to believe that these step sizes are more likely to converge near the solution if $relerr = 10^{-3}$ than for $relerr = 10^{-10}$, because the cruder of these two integration accuracy tolerances still gives a stable accurate integration of the differential equations.

This phenomenon is only mentioned here, and will not be utilized in the sequel. For more complicated tasks an assumption of convergence of the integration step sizes cannot be considered to be very robust. If it were necessary, however, with an optimization accuracy that were higher than the integration accuracy, this could

have been achieved by freezing the step size sequence of the integrator close to the solution.

4.5 An AUV Controlled by Thrusters in all 6 DOF

4.5.1 Introduction

In this section computation of optimal trajectories for a complete 6 degree of freedom ROV model, described in Chapter 3.2.3 is reported. This section is based on the results in (Spangelo & Egeland 1994), but with a different handling of the scaling problem.

Three different tasks were studied, one with no path constraints and fixed final time (Section 4.5.4), one with collision avoidance and fixed final time (Section 4.5.5), and one with collision avoidance and free final time (Section 4.5.6).

Both piecewise linear and piecewise constant control parameterizations were studied, and for different numbers of control parameters.

The choice of optimization parameters in the nonlinear programming solver was based on the results and the discussion in the previous section. The convergence accuracy for the objective function was chosen to be $ftol = 0.001$, and the nonlinear constraint violation tolerance *featol1* was chosen to be 0.1.

As discussed in Chapter 4.4.4 the conditioning of the constraints become worse when the optimization time increases. Scaling of the constraints was, however, not performed. It was assumed that the optimization time was small enough for this not to be a problem. For larger optimization time intervals the introduction of multiple shooting (see Chapter 2.2.3) would have been preferable, and this would solve this conditioning problem too. The parameters where scaled to be between -1.0 and 1.0, and the scaling of the objective function was chosen to be

$$jscale = 10^{-4}$$

Numerical results are given in tabular form, including the number of controls per control (# par. per control.), converged objective function (J^*), number of major iterations (# iter), number of functions and gradient calculations (# fc), number of right hand side calculations of the state space equations (# rhs calc.) and a crude estimate of the CPU-time consumption (CPU-time). The number of function and gradient evaluations (# fc) is greater than or equal to the number of iterations.

The computations were performed on a Sparc 10 station. In Sections 4.5.5 and 4.5.6 indications of which time nodes in the path constraint time grid that are active at the solution are also given. In Section 4.5.6 converged final times (t_f) and the energy parts (J_e^*) of the objective functions are given.

4.5.2 Model Parameters

The computational study in this section was based on the model structure defined in Chapter 3.2.3. This model structure is the same as that of the Norwegian Experimental Remotely Operated Vehicle (NEROV) (Fossen 1991). The parameters in (Fossen 1991) were partly found from experiments, and partly guessed. The parameters used in this work were chosen slightly different, in order to get a vehicle which was symmetric about the x-axis. After discussions with the author of (Fossen 1991) some of the parameters were changed significantly to get relations between the different degrees of freedom that are physically more realistic than the NEROV parameters that were guessed.

The vehicle is controlled in all six degrees of freedom by six DC-motor driven thrusters.

The fluid velocity was chosen to be zero in all computations.

Numerical values for the hydrodynamic derivatives (added inertias) and damping coefficients are given in Table 4.13.

$X_{\dot{u}} = -30$ kg	$X_u = -70$ kg/s	$X_{u\|u\|} = -100$ kg/m
$Y_{\dot{v}} = -80$ kg	$Y_v = -100$ kg/s	$Y_{v\|v\|} = -200$ kg/m
$Z_{\dot{w}} = -80$ kg	$Z_w = -100$ kg/s	$Z_{w\|w\|} = -200$ kg/m
$K_{\dot{p}} = -15$ kg m^2	$K_p = -30$ kg m/s	$K_{p\|p\|} = -50$ kg m
$M_{\dot{q}} = -30$ kg m^2	$M_q = -50$ kg m/s	$M_{q\|q\|} = -100$ kg m
$N_{\dot{r}} = -30$ kg m^2	$N_r = -50$ kg m/s	$N_{r\|r\|} = -100$ kg m

Table 4.13: *Hydrodynamic Derivatives and Damping Coefficients*

Numerical values for the remaining parameters are given in Table 4.14:

$m = 185$ kg	$I_x = 25$ kg m^2	$x_B = 0.0$ m	$\ell_1 = 0.4$ m	$\ell_4 = 0.21$ m
$B = W = 1800$ N	$I_y = 50$ kg m^2	$y_B = 0.0$ m	$\ell_2 = 0.4$ m	$\ell_5 = 0.43$ m
	$I_z = 50$ kg m^2	$z_B = -0.02$ m	$\ell_3 = 0.21$ m	$\ell_6 = 0.43$ m

Table 4.14: *Weight and Balance Data*

These parameters result in the following model for the vehicle.

$$\dot{\mathbf{x}}_1 = \mathbf{J}(\mathbf{x}_1)\mathbf{x}_2$$

$$\begin{aligned}
\dot{x}_7 &= \frac{53}{43}x_8x_{12} - \frac{53}{43}x_9x_{11} - (\frac{14}{43} + \frac{20}{43}|x_7|)x_7 + \frac{1}{215}(u_1 + u_2) \\
\dot{x}_8 &= -\frac{43}{53}x_7x_{12} + x_9x_{10} - (\frac{20}{53} + \frac{40}{53}|x_8|)x_8 + \frac{1}{265}(-u_3 + u_4)
\end{aligned}$$

$$
\begin{aligned}
\dot{x}_9 &= \frac{43}{53}x_7x_{11} - x_8x_{10} - (\frac{20}{53} + \frac{40}{53}|x_9|)x_9 + \frac{1}{265}(u_5 + u_6) \\
\dot{x}_{10} &= -(\frac{3}{4} + \frac{5}{4}|x_{10}|)x_{10} - \frac{9}{10}\sin(x_4)\cos(x_5) + \frac{21}{4000}(u_3 + u_4) \\
\dot{x}_{11} &= \frac{5}{8}x_7x_9 + \frac{1}{2}x_{10}x_{12} - (\frac{5}{8} + \frac{5}{4}|x_{11}|)x_{11} - \frac{9}{20}\sin(x_5) + \frac{43}{8000}(-u_5 + u_6) \\
\dot{x}_{12} &= -\frac{5}{8}x_7x_8 - \frac{1}{2}x_{10}x_{11} - (\frac{5}{8} + \frac{5}{4}|x_{12}|)x_{12} + \frac{1}{200}(-u_1 + u_2)
\end{aligned} \tag{4.46}
$$

where $\mathbf{x}_1$, $\mathbf{x}_2$ and the Jacobian $\mathbf{J}(\mathbf{x}_1)$ are defined in Chapter 3.

The box constraints for the controls used in these computations were

$$|u_i(t)| \leq 80.0\text{N} \quad i = 1, ..., 6 \tag{4.47}$$

Based on the thruster characteristics in (Fossen 1991) these box constraints correspond to a maximum propeller velocity of 10 revolutions per second at bollard pull (zero advance velocity).

4.5.3 Initialization of the Parameters

An initial set of parameters has to be specified before optimization is started. A simple solution is to set all these parameters to zero, which was chosen for the simple benchmark problem in the previous section.

For more complicated tasks and models this is an inefficient solution, and will in some cases not lead to convergence. Therefore a method for automatic computations of initial parameters, based on a feedback linearization controller, has been developed.

Using a feedback linearization controller on the model defined in Chapter 3.2.3 the controls are computed as, see e.g. (Fossen 1994)

$$\mathbf{u} = \tilde{\mathbf{B}}^{-1}(\mathbf{M}\mathbf{a}_v + \mathbf{C}(\mathbf{x}_2)\mathbf{x}_2 + \mathbf{D}(\mathbf{x}_2)\mathbf{x}_2 + \mathbf{g}(\mathbf{x}_1)) \tag{4.48}$$

where all symbols are defined in Chapter 3.2.3 except for

$$\mathbf{a}_v = \mathbf{J}(\mathbf{x}_1)^{-1}(\mathbf{a}_x - \dot{\mathbf{J}}(\mathbf{x}_1)\mathbf{x}_2) \tag{4.49}$$

where $\dot{\mathbf{J}}(\mathbf{x}_1)$ is the time derivative of the Jacobian matrix, and $\mathbf{a}_x$ is an acceleration reference in the global coordinate system. Using a PD controller $\mathbf{a}_x$ can be chosen as

$$\mathbf{a}_x = k_p\mathbf{e} + k_d\dot{\mathbf{e}} \tag{4.50}$$

Here, k_p and k_d are constant control parameters, and

$$\mathbf{e} = \mathbf{x}_d - \mathbf{x}_1 \tag{4.51}$$

where $\mathbf{x}_d$ is a filtered reference signal for the global position coordinates. Usually, using global position references in feedback linearization for a mechanical system like the one considered, the references are filtered through a second order filter. However, in the optimization problems considered here the controls do not have to be continuous. Therefore, only a first order filter was used. In addition, if one of the computed controls was outside one of its box constraints, it was set to the respective box constraint.

The initializing controls were then computed by the following algorithm, where $\lambda > 0$ is a filter parameter:

$$\begin{aligned}
\dot{\mathbf{x}}_d + \lambda \mathbf{x}_d &= \lambda \mathbf{r}_d \\
\mathbf{u}_{FL} &= \tilde{\mathbf{B}}^{-1}(k_p \mathbf{M}\mathbf{J}^{-1}(\mathbf{x}_d - \mathbf{x}_1) + k_d \mathbf{M}\mathbf{J}^{-1}\dot{\mathbf{x}}_d - k_d \mathbf{M}\mathbf{x}_2 - \mathbf{M}\mathbf{J}^{-1}\dot{\mathbf{J}}\mathbf{x}_2 \\
&+ \mathbf{C}(\mathbf{x}_2)\mathbf{x}_2 + \mathbf{D}(\mathbf{x}_2)\mathbf{x}_2 + \mathbf{g}(\mathbf{x}_1)) \\
u_i &= \begin{cases} u_{i,\max}; & u_i > u_{i,\max} \\ u_i; & u_{i,\min} \le u_i \le u_{i,\max}, \quad (i = 1, ..., m) \\ u_{i,\min}; & u_i < u_{i,\min} \end{cases}
\end{aligned} \tag{4.52}$$

where $\mathbf{x}_d$ and the state space equations were integrated by the standard 4th order Runge-Kutta method, and $\mathbf{u}$ was calculated once per time step.

The computed initializing controls were stored once per second, and the control parameters were computed from these controls multiplied by the control parameter scaling factor. For control parameter switching times located between whole seconds, linear interpolation was used.

For free final time t_f the simulations in the initializer were continued until the infinity norm of the position deviation $\mathbf{e}$ was less then some, possibly small, user specified parameter ϵ.

For reasons discussed later the possibility of defining an initial path different from a straight line was introduced by including the possibility of defining intermediate reference points. The initializing simulator is driven towards an intermediate reference point in a specified time t_n, and then driven towards another intermediate reference point in another specified time, until it is finally driven towards the specified final position. Only the x, y and z position coordinates of the intermediate points are different from zero.

The initialization parameters to be defined by the user are

- k_p and k_d: These parameters were both chosen to be 1.0 for all the optimization runs.

- numinitgrid: A parameter defining the number of intermediate reference points. This parameter is zero if there are no path constraints.
- ref: A matrix consisting of numinitgrid number of vectors, each with four elements defining an intermediate reference point, the first element the time t_n, and the three last elements the x, y and z coordinates of the reference point.
- lamfact: The filter parameter λ (see Eq. 4.52) is defined as

$$\lambda = \frac{\text{lamfact}}{\text{norme}}$$

  where norme is the maximum norm of the vector from the initial position to the first intermediate reference position. In the case of no intermediate reference position the final position is used.

  lamfact appeared to be an important tuning parameter for the optimization problems.
- ϵ: This parameter is defined only in the case of free final time t_f.

4.5.4 No Path Constraints and Fixed Final Time.

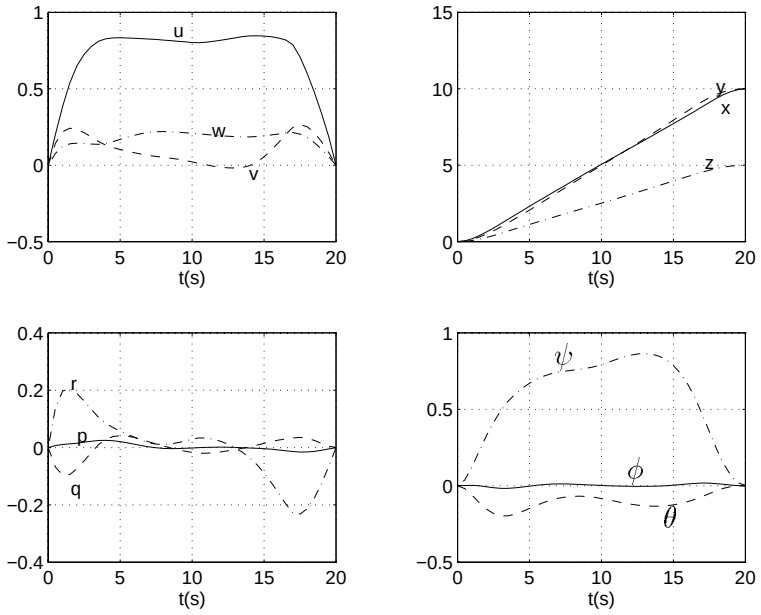

Figure 4.5: *Optimal* $\mathbf{x}(t)$*, no path constraints, fixed* t_f

Optimal trajectories were first computed for a simple task, where the vehicle was specified to move 10 m in the x and y directions and 5 m in the z direction. The

final time was fixed to 20 s and no path constraints were included. The initial and final states were

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{0} \\ \mathbf{x}(20) &= \begin{pmatrix} 10 & 10 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \end{aligned} \tag{4.53}$$

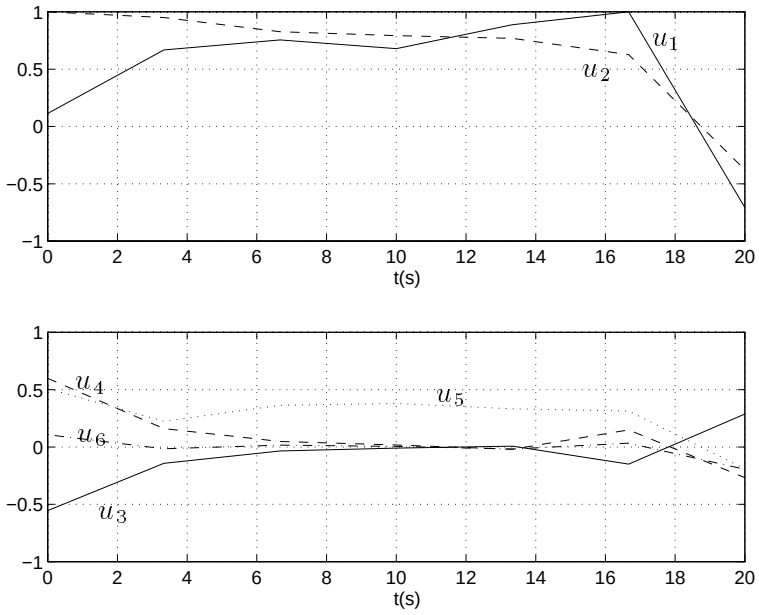

Figure 4.6: *Optimal control time histories, no path constraints, fixed* t_f

Problems occurred when the parameter lamfact was chosen to large, primarily with the active set strategy. The reason for this is that because of the resulting saturations in the feedback linearization generated controls, large errors in the Euler angles occurs which the controller cannot compensate for. For low accuracy problems the optimization terminates prematurely far from optimum. This problem is, however, easily avoided by choosing lamfact sufficiently low. A good choice seemed to be

$$\text{lamfact} = 0.5$$

Numerical results from computations using piecewise linear and piecewise constant controls are given in Table 4.15 and Table 4.16 respectively.

The optimal states and the optimal control time histories using piecewise linear controls and 7 parameters per control are shown in Figure 4.5 and 4.6 respectively.

# par. per contr.	J^*	# iter	# fc	# rhs calc.	CPU-time
4	2.284	10	17	101574	6.2
5	2.220	10	17	116966	7.0
6	2.198	11	16	126958	8.1
7	2.182	13	18	159019	10.2
8	2.177	11	19	191222	12.3

Table 4.15: *Numerical results using piecewise linear controls, no path constraints, fixed* t_f

# par. per contr.	J^*	# iter	# fc	# rhs calc.	CPU-time
4	2.696	12	21	116717	6.7
5	2.427	11	20	123621	7.2
6	2.342	12	17	124501	7.5
7	2.298	14	22	186153	11.6
8	2.273	11	16	151627	9.8

Table 4.16: *Numerical results using piecewise constant controls, no path constraints, fixed* t_f

4.5.5 Collision Avoidance and Fixed Final Time.

Secondly, collision avoidance was introduced by defining a circular path constraint in the xy-plane (cylinder in 3 dimensional space) centered at $x = y = 5$ and with radius 2 m. The corresponding constraint equation is

$$c(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2 - 2^2 \geq 0 \tag{4.54}$$

The same initial and final states, and the same final time as in Section 4.5.4 was specified. The following normalized time grid for path constraint computations was specified

$$\begin{pmatrix} 0.425 & 0.45 & 0.475 & 0.5 & 0.525 & 0.55 & 0.575 \end{pmatrix}$$

For this particular path constraint the optimization may fail if the initial parameter choice drives the vehicle through the constraint. That is because it becomes difficult for the active set strategy of the optimizer to decide which of the two different sides of the cylinder to choose. Computations with the same initialization procedure for the parameters as in the previous subsection confirmed this problem. The problem was solved by driving the vehicle towards intermediate points in the initial simulation, where the initial parameters were decided. There is of course no guarantee that a

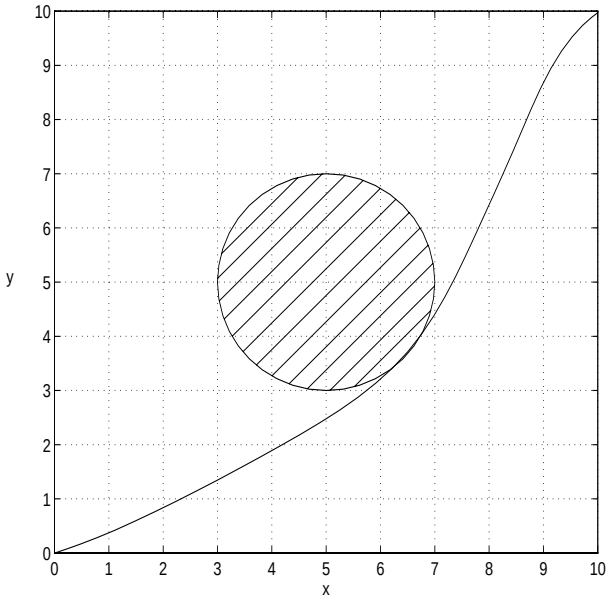

Figure 4.7: *x-y plot of optimal trajectory with the path constraint, fixed* t_f

more optimal solution will not exist on the other side of the obstacle than the chosen one. Two intermediate reference points were chosen (numinitgrid = 2)

$$\text{ref} = \begin{pmatrix} 5 & 10 \\ 10 & 10 \\ -1 & 5 \\ 5 & 5 \end{pmatrix}$$

In this case

$$\text{lamfact} = 1.0$$

appeared to be a good choice.

Numerical results from computations using piecewise linear and piecewise constant controls are given in Table 4.17 and Table 4.18 respectively.

The optimal states and the optimal control time histories using piecewise linear controls and 7 parameters per control are shown in Figure 4.8 and 4.9 respectively. An xy-plot of the optimal trajectory including the path constraint is shown in Figure 4.7.

4.5.6 Collision Avoidance and free Final Time.

Optimal trajectories for the same task as in Section 4.5.5 was computed with free final time. The parameter t_f was scaled to be within 1.0 and 4.0, where the initial

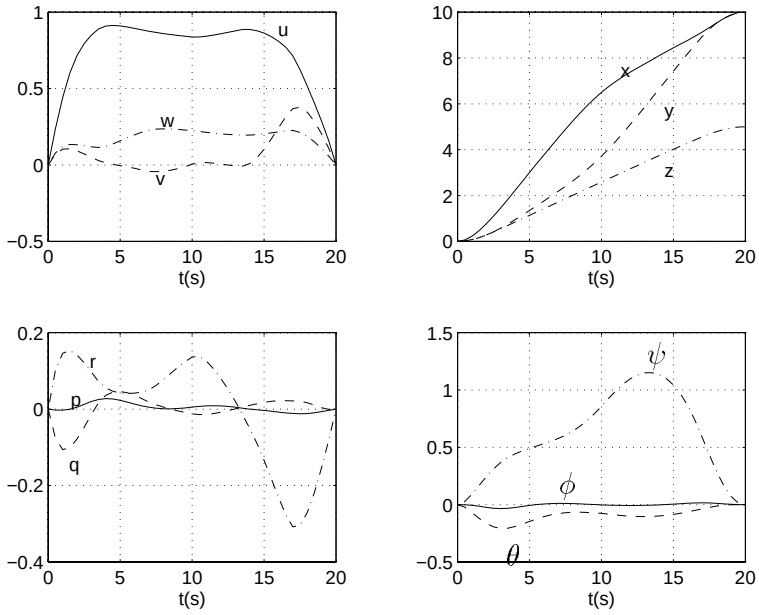

Figure 4.8: *Optimal* $\mathbf{x}(t)$*, collision avoidance, fixed* t_f

# par. per contr.	J^*	# iter	# fc	# rhs calc.	CPU-time	Active tgrid
4	2.535	19	32	194375	11.5	0.475
5	2.437	16	25	184091	11.1	0.475, 0.5
6	2.395	12	16	137891	8.6	0.475, 0.5
7	2.383	12	20	194900	12.3	0.475, 0.5
8	2.351	15	23	263807	17.8	0.475, 0.5

Table 4.17: *Numerical results using piecewise linear controls, a path constraint, fixed* t_f

guess of the parameter was the lower bound. Using the initial guess as lower bound on this parameter seemed to be the best solution numerically, and is also reasonable physically as long as the initial guess is below the optimal. The weighting factor was chosen to be

$$\mathrm{w} = 500$$

The same time grid, the same value on lamfact (1.0), and the same intermediate points ref as in Section 4.5.5 appeared to be good choices also here. In addition

$$\epsilon = 2.0$$

was chosen, which gave an initial guess $t_f = 25.0$.

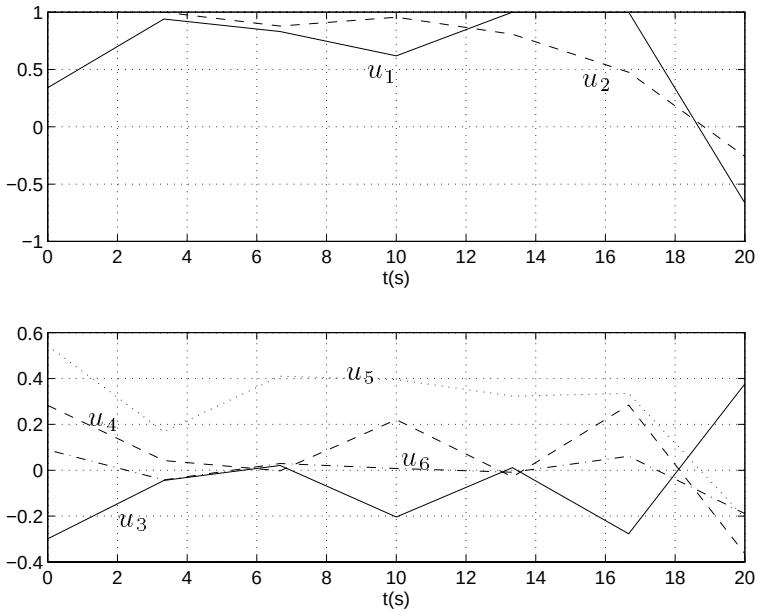

Figure 4.9: *Optimal control time histories, collision avoidance, fixed t_f*

# par. per contr.	J^*	# iter	# fc	# rhs calc.	CPU-time	Active tgrid
4	3.080	24	51	93309	5.4	0.425
5	2.678	12	17	79149	4.7	0.475, 0.5
6	2.564	16	21	93635	5.5	0.475, 0.5
7	2.513	12	14	120835	7.1	0.475, 0.5
8	2.457	17	22	921711	6.4	0.475, 0.5

Table 4.18: *Numerical results using piecewise constant controls, a path constraints, fixed t_f*

Numerical results from computations using piecewise linear and piecewise constant controls are given in Table 4.19 and Table 4.20 respectively.

The optimal states and the optimal control time histories using piecewise linear controls and 7 parameters per control are shown in Figure 4.11 and 4.12 respectively. An xy-plot of the optimal trajectory, including the path constraint, is shown in Figure 4.10.

4.5.7 Discussion

In Tables 4.15-4.20 numerical results are given for 4 to 8 parameters per control. The number of equality constraints are twelve. Therefore, the number of parameters have to be more than twelve. Computations with three parameters per control

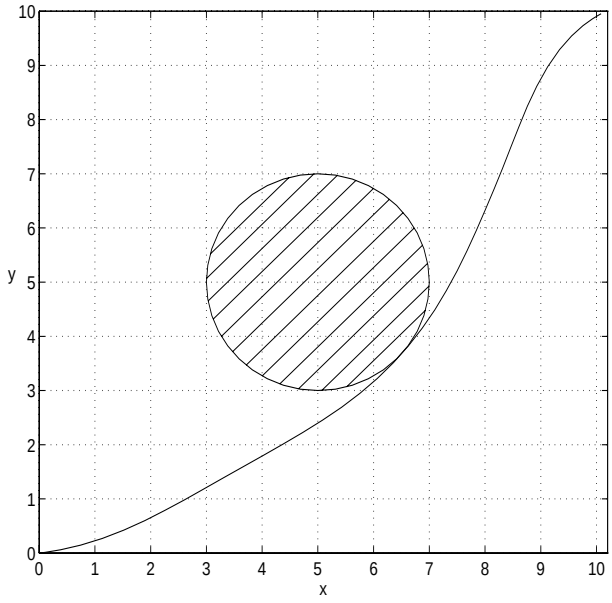

Figure 4.10: *x-y plot of optimal trajectory with the path constraint, free* t_f

	J^* (J_e^*)	# iter	# fc	# rhs calc.	t_f	CPU-time	Active tgrid
4	3.017(1.402)	38	65	543536	32.30	33.7	0.475
5	2.957(1.373)	12	21	203936	31.69	12.9	0.475, 0.5
6	2.929(1.361)	14	20	226600	31.36	15.0	0.475, 0.5
7	2.919(1.368)	13	20	250380	31.02	16.4	0.475, 0.5
8	2.912(1.395)	11	20	275813	30.34	17.7	0.475, 0.5

Table 4.19: *Numerical results using piecewise linear controls, a path constraint, free* t_f, # par. per contr. *in left column*

also gave problems, and resulted in a very large value on the objective function, if convergence was achieved at all. Also four parameters per control were difficult, and in the case of piecewise constant parameterization, free final time and inclusion of the path constraint, convergence was not achieved. Theoretically, it is more difficult to satisfy the path constraint with fixed final time, but computationally free final time is more difficult.

An interesting observation was that normally the estimated condition number of the approximated Hessian was in the order no higher than 10. This condition number did, however, increase with a decreasing number of parameters, which made the optimization problem more difficult to solve. Including path constraints and with only four parameters per control this matrix soon became very ill-conditioned. During the early iterations an "I" was printed from the optimizer, indicating that the QP sub-problem had no feasible point. The condition number of the matrix $\mathbf{T}$ in the TQ-factorization was usually between 100 and 1000. This condition number also became worse for difficult problems, but only by a decade. This condition number

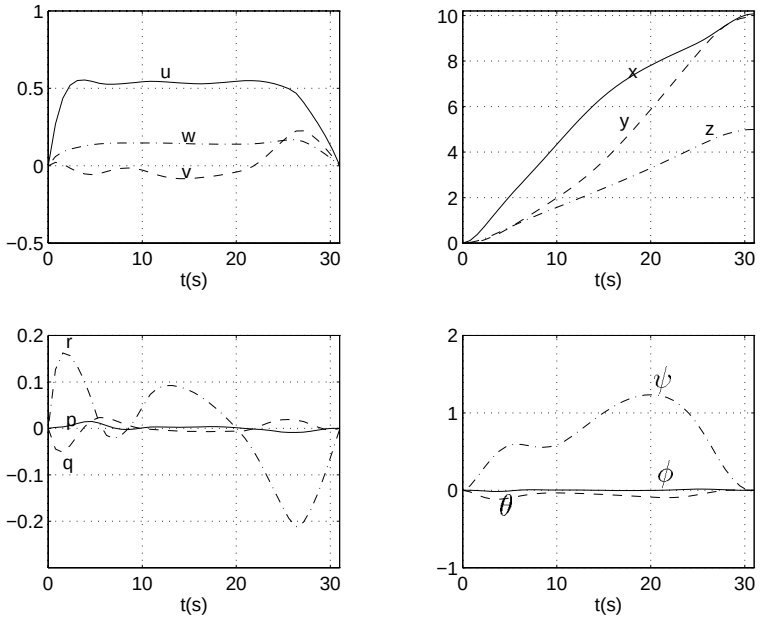

Figure 4.11: *Optimal* $\mathbf{x}(t)$*, collision avoidance, free* t_f

	J^* (J_e^*)	# iter	# fc	# rhs calc.	t_f	CPU-time	Active tgrid
5	3.086(1.413)	17	28	257653	33.47	15.8	0.45
6	3.031(1.479)	10	19	200594	31.04	12.3	0.475, 0.5
7	2.986(1.3905)	15	29	363442	31.91	22.4	0.475
8	2.972(1.372)	11	22	305951	32.01	19.0	0.475, 0.5

Table 4.20: *Numerical results using piecewise constant controls, a path constraint, free* t_f*,* # par. per contr. *in left column*

did not seem to be decided by the mutual conditioning of the constraints, but rather by the fact that some parameters had a smaller influence than others.

For the accuracy specified in the present work, it turned out that to report results from using more than eight parameters per control was not necessary.

It seems quite difficult to give any conclusions concerning the convergence speed. This is mainly due to the nonlinear nature of the problem, especially the kinematic equations of motion are complex. The choice of initial parameters was also quite important for the number of iterations.

Both gradient calculations and computations of new search directions (in E04VCF) become more expensive when the number of parameters are increased. Therefore, the number of parameters should not be chosen higher than necessary. Using seven parameters per control seemed to be a reasonable choice, and results from these

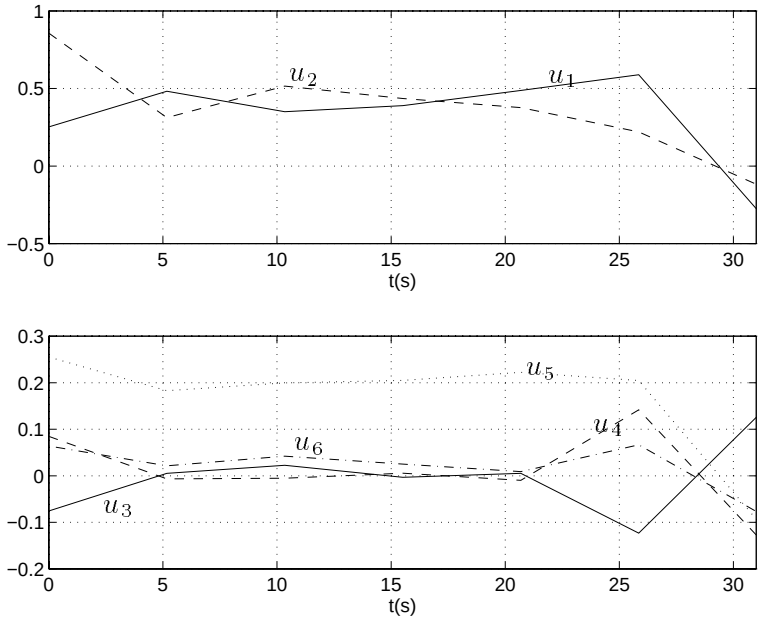

Figure 4.12: *Optimal control time histories, collision avoidance, free* t_f

computations were therefore chosen to be shown in the figures. In all the figures piecewise linear controls are chosen. The tables clearly show that piecewise linear parameterizations resulted in a lower value on the optimal objective function than piecewise constant parameterizations. In addition, piecewise constant controls are discontinuous at every switching time while piecewise linear controls are discontinuous at the initial and final times only.

Figures 4.7 and 4.10 show that the path constraint is satisfied in both cases. The time grid was chosen on the basis of physical arguments. Generally it has to be chosen large enough so that path constraint violations outside the time grid are not likely to occur, and tight enough so that violations between the time nodes have to be small compared to the margin between the obstacle and the path constraint.

The figures show that for all three tasks the motion in roll becomes small. This is to be expected, because with the given model parameters, the only coupling in the dynamic equations of motion between roll and other directions are the stable restoring moments. This because the vehicle is symmetrical around the x-axis. Nothing can be gained by roll motion. Small roll motions occur because of the couplings in the kinematic equations of motion, that is the combination of yaw angular velocity and a nonzero pitch angle gives an angular velocity in roll.

Both because of the conditioning of the constraints and the complexity of the state space equations, the optimization time, which means also the optimization tasks, are kept small. As discussed in Chapter 2, when the optimization time and the number of parameters per control becomes large efficiency can be improved significantly by introducing multiple shooting and using a sparse nonlinear programming solver.

In the next section the full 6 DOF model will be approximated by a 5 DOF model and a 4 DOF model by ignoring roll motion, and roll and pitch motion. Using considerably simpler models for the vehicles will make the optimization far more efficient and robust. The approximation that angular velocity around the x-axis and roll angle are identically zero is very good, considering the symmetry properties and the metacentric stability of the vehicle. Using different reduced models for different phases in a multiphase scheme can also be a reason for using reduced models. A complete task is then partitioned into many different small tasks.

4.6 Optimizing Reduced Models of the Thruster Controlled AUV

4.6.1 Introduction

As shown in the previous section optimizing the full six degree of freedom vehicle model can be complicated, especially the kinematic equations motion are highly nonlinear and coupled.

In this section the full 6 DOF model is approximated by reduced models. By eliminating one or two of the rotational degrees of freedom the model is considerably simplified. This is especially true for the kinematics. Consequently, the computations become more simple and robust. In addition, using reduced models can be efficient in the context of solving multiphase problems, as mentioned in the previous section.

The model reduction is an approximation, and will sometimes also lead to suboptimal solutions of the original optimization problem. But, the metacentric stability of the vehicle makes the approximation quite good. The nature of the approximations will be discussed later.

In this section only one of the tasks in the previous section has been considered, namely the one including free final time and collision avoidance. Piecewise linear approximation of the controls and 7 parameters per control was applied. The user defined optimization parameters were chosen equal to the 6 DOF case in Section 4.5.6 to get realistic comparisons.

4.6.2 5 DOF Model Setting the Roll Angle to Zero

Since the vehicle is metacentrically stable and symmetric around the x-axis the roll angle is set to zero, and thus removed from the state space equations. Now the position states are defined as before, the pitch and yaw angles are x_4 and x_5 respectively, x_6, x_7 and x_8 are the velocities in the vehicles x, y and z coordinates respectively, and x_9 and x_{10} are the angular velocities around the vehicles y and

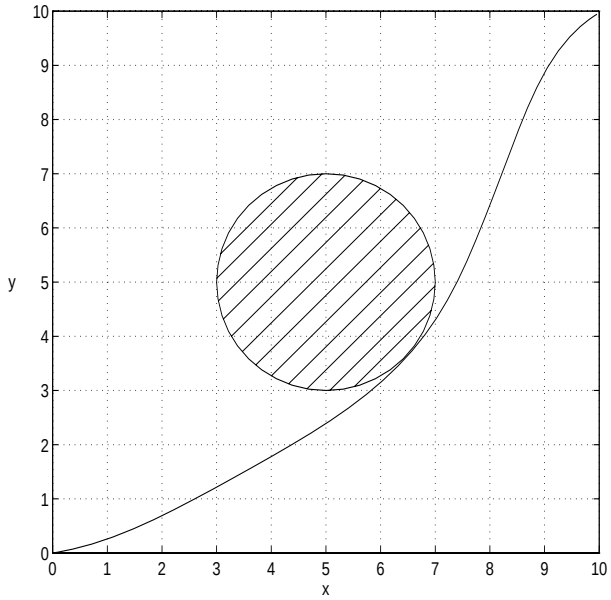

Figure 4.13: *x-y plot of optimal trajectory with the path constraint for the 5 DOF vehicle*

z coordinates respectively. By setting $u_3 = -u_4$ the control input for the angular velocity around the x-axis is zero. In the 5 DOF model the controls u_3 and u_4 are substituted by a new control u_3 which equals the old $u_4 - u_3$. In the thruster configuration matrix row 4 is removed and column 3 and 4 are substituted by a column which equals column 4 minus column 3. Because of the kinematic couplings, this is an approximation. However, the couplings give small roll rates, and the roll angle will be driven to zero because of the metacentric stability. The previously defined u_5 and u_6 now become u_4 and u_5 respectively. This results in the following set of state space equations.

$$
\begin{aligned}
\dot{x}_1 &= c(x_4)c(x_5)x_6 - s(x_5)x_7 + s(x_4)c(x_5)x_8 \\
\dot{x}_2 &= c(x_4)s(x_5)x_6 + c(x_5)x_7 + s(x_4)s(x_5)x_8 \\
\dot{x}_3 &= -s(x_4)x_6 + c(x_4)x_8 \\
\dot{x}_4 &= x_9 \\
\dot{x}_5 &= \frac{1}{c(x_4)}x_{10} \\
\dot{x}_6 &= \frac{53}{43}x_7x_{10} - \frac{53}{43}x_8x_9 - (\frac{14}{43} + \frac{20}{43}|x_6|)x_6 + \frac{1}{215}(u_1 + u_2) \\
\dot{x}_7 &= -\frac{43}{53}x_6x_{10} - (\frac{20}{53} + \frac{40}{53}|x_7|)x_7 + \frac{2}{265}u_3 \\
\dot{x}_8 &= \frac{43}{53}x_6x_9 - (\frac{20}{53} + \frac{40}{53}|x_8|)x_8 + \frac{1}{265}(u_4 + u_5) \\
\dot{x}_9 &= \frac{5}{8}x_6x_8 - (\frac{5}{8} + \frac{5}{4}|x_9|)x_9 - \frac{9}{20}\sin(x_4) + \frac{43}{8000}(-u_4 + u_5)
\end{aligned}
$$

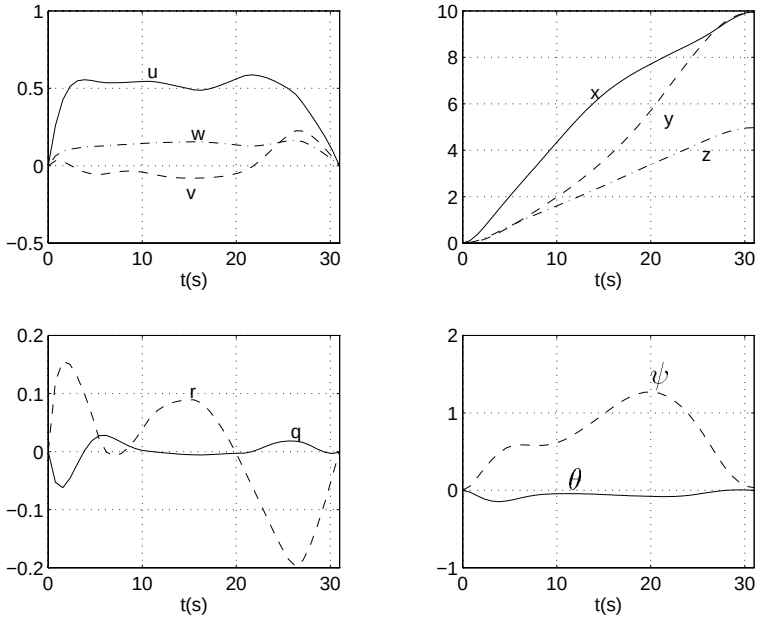

Figure 4.14: *Optimal* $\mathbf{x}(t)$ *for the 5 DOF vehicle*

$$\dot{x}_{10} = -\frac{5}{8}x_6x_7 - (\frac{5}{8} + \frac{5}{4}|x_{10}|)x_{10} + \frac{1}{200}(-u_1 + u_2) \tag{4.55}$$

where $c(\cdot) = \cos(\cdot)$ and $s(\cdot) = \sin(\cdot)$. In the objective functional (Eq. 4.14) defined in Section 4.2 $m = 5$ and $\alpha_3 = 2$.

The following results were obtained:

$$\begin{aligned} J^* &= 2.906 \\ \#\text{ iter} &= 8 \\ \#\text{ fc} &= 15 \\ t_f &= 30.98 \\ \text{CPU-time} &= 5.5s \\ \#\text{ rhs calc.} &= 143929 \end{aligned} \tag{4.56}$$

Comparing with Table 4.19, the case with seven parameters per control, the converged objective function is approximately the same as for the 6 DOF problem, actually marginally lower. This is a natural consequence of the fact that it is nothing to gain by generating a roll motion. The approximation made by assuming that there are no couplings between the remaining angular velocities and the roll rate is small, and the vehicle is metacentrically stable, so this conclusion is quite fair.

The optimal states and the optimal control time histories for the 5 DOF vehicle are

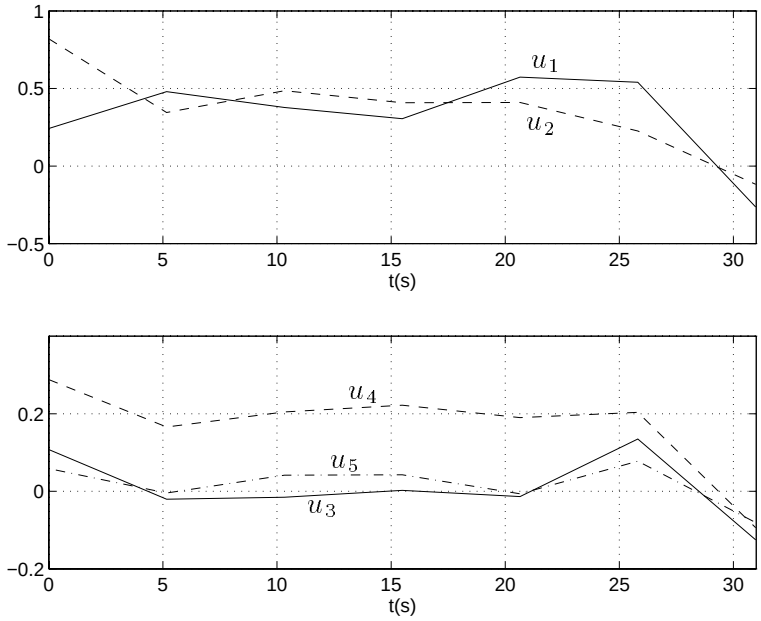

Figure 4.15: *Optimal control time histories for the 5 DOF vehicle*

shown in Figures 4.14 and 4.15 respectively. An x-y plot of the optimal trajectory including the path constraint is shown in Figure 4.13.

4.6.3 4 DOF Model Setting the Roll and Pitch Angles to Zero

The metacentric stability also makes it interesting to assume zero pitch angle in addition to zero roll angle. By setting $u_4 = u_5$ for the 5 DOF vehicle above, and defining a new u_4 which equals the old $u_4 + u_5$ the control inputs to the angular velocity around the y-axis becomes zero. By assuming $x_4 = x_9 = 0$ in the state space equations above, the kinematics will be very simple, and the assumption of zero roll and pitch angles when the angular velocities around the x- and y-axis are identically zero, are kinematically correct. However, the vehicle is not symmetrical around the y-axis. This means that firstly, the assumption that x_4 above is zero whenever $u_4 = u_5$ is only an approximation, because of the coupling term. Secondly, zero pitch angle is not optimal when the vehicle is diving. The states are redefined in the same manner as above, and the 4 DOF state space equations become

$$\begin{aligned}
\dot{x}_1 &= c(x_4)x_5 - s(x_4)x_6 \\
\dot{x}_2 &= s(x_4)x_5 + c(x_4)x_6 \\
\dot{x}_3 &= x_7 \\
\dot{x}_4 &= x_8
\end{aligned}$$

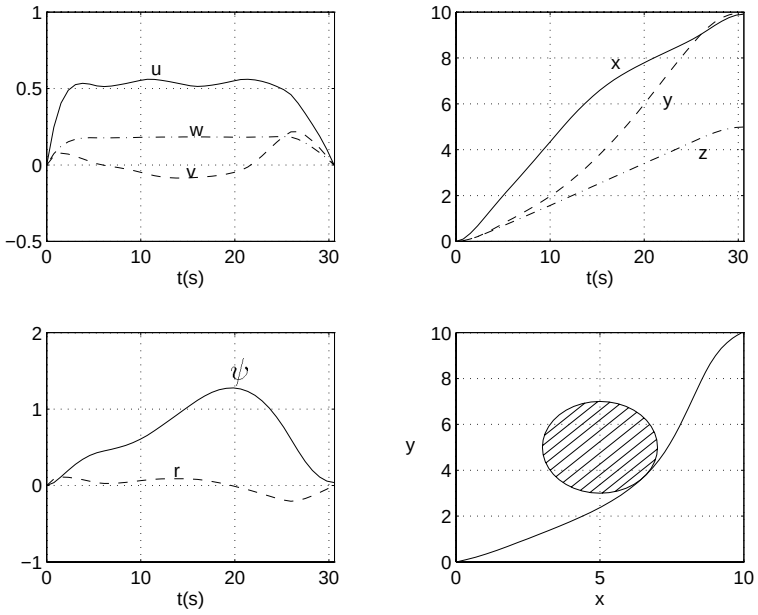

Figure 4.16: *Optimal* $\mathbf{x}(t)$ *for the 4 DOF vehicle*

$$
\begin{aligned}
\dot{x}_5 &= \frac{53}{43}x_6x_8 - (\frac{14}{43} + \frac{20}{43}|x_5|)x_5 + \frac{1}{215}(u_1 + u_2) \\
\dot{x}_6 &= -\frac{43}{53}x_5x_8 - (\frac{20}{53} + \frac{40}{53}|x_6|)x_6 + \frac{2}{265}u_3 \\
\dot{x}_7 &= -(\frac{20}{53} + \frac{40}{53}|x_7|)x_7 + \frac{2}{265}u_4 \\
\dot{x}_8 &= -\frac{5}{8}x_5x_6 - (\frac{5}{8} + \frac{5}{4}|x_8|)x_8 + \frac{1}{200}(-u_1 + u_2)
\end{aligned} \tag{4.57}
$$

In the objective functional (Eq. 4.14) defined in Section 4.2 $m = 4$ and $\alpha_3 = \alpha_4 = 2$.

The following results, which can be compared with the results for seven parameters per control in Table 4.19 and the results in Eq. (4.56), were obtained:

$$
\begin{aligned}
J^* &= 2.902 \\
\#\text{ iter} &= 9 \\
\#\text{ fc} &= 17 \\
t_f &= 30.55 \\
\text{CPU-time} &= 3.7s \\
\#\text{ rhs calc.} &= 121362
\end{aligned} \tag{4.58}
$$

The converged objective function is approximately the same as for the 6 DOF and the 5 DOF problems, actually marginally lower. It can, however, not be concluded that

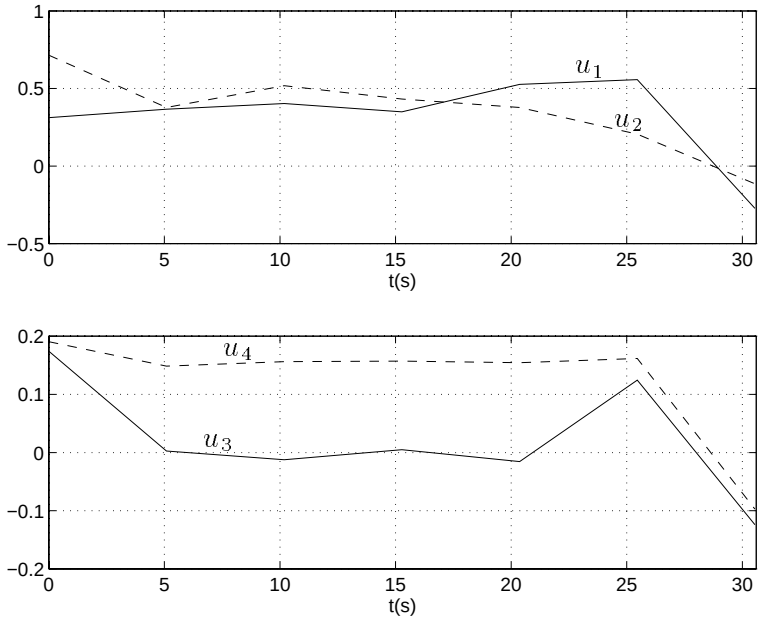

Figure 4.17: *Optimal control time histories for the 4 DOF vehicle*

there is nothing to gain by generating a pitch motion since the approximation made by removing the coupling terms in the dynamic equations of motion are rather crude. However, the fact that the vehicle is metacentrically stable makes it reasonable to believe that the loss in performance by using this simplified model is not large.

Observe that the number of right hand side calculations of the state space equations are lower in the 4 DOF case than in the 5 DOF case even though the number of function and gradient calculations are higher. This is a result of the fact that when the kinematic equations become simpler the same accuracy in the integrator can be obtained with larger step sizes.

The optimal states and the optimal control time histories for the 4 DOF vehicle are shown in Figures 4.16 and 4.17 respectively. An x-y plot of the optimal trajectory including the path constraint is included in Figure 4.16.

Finally, it should be mentioned that the conditioning of the approximate Hessian and the $\mathbf{T}$ matrices did not improve significantly when the model became simpler.

4.7 Conclusions

Energy optimal trajectories and collision avoidance for underwater vehicles can be achieved by solving nonlinear optimal control problems with path constraints.

A general class of optimal control problems (OCP) have been formulated including minimization of energy an time consumption. Obstacle avoidance was included by

including path constraints. An expression for the energy consumption of a DC-motor driven thruster has been developed, and and approximation was included in the optimal control problem.

A program system using CVP with single shooting for numerical solution of OCP's has been implemented in the C language. The program is specially designed to solve the kind of problems reported in this chapter. The program system uses the SQP subroutine E04VCF from the NAG Fortran library (NAG 1991). Optimization parameters and other implementation details were decided after extensive testing of the program system on a 2 DOF benchmark problem. The results from these tests are reported.

Energy optimal trajectories have been computed for a DC-motor driven vehicle. Several tasks were solved for different numbers of control parameters. The results have been analyzed with respect to accuracy and CPU-time consumption. It has been shown that collision avoidance, both with fixed and free final time for the task, can be solved efficiently by using trajectory optimization.

Optimal trajectories for reduced versions of the 6 DOF underwater vehicle model have also been computed. The collision avoidance problem was solved for a 5 DOF model and a 4 DOF model. It has been shown that in order to compute optimal trajectories for this type of vehicles even more efficiently, it can sometimes be appropriate to fix the roll and pitch angles and the angular velocities around the x-axis and the y-axis.

Results from a previously published paper (Spangelo & Egeland 1992a) have also been presented. It was concluded that forward difference gradient computations should be preferred to analytical gradient computations using the costate equations.

Chapter 5

Trajectory Optimization for Rocket Ascent with Heat-Flux and Splash-Down Constraints

5.1 Introduction

Trajectory optimization for aerospace vehicles has been one of the driving applications for developing more efficient and flexible numerical methods for solving highly constrained nonlinear optimal control problems for the last 30–40 years (see Chapter 2.2).

Payload maximization for launch vehicles is an important task since only a small fraction of the total lift-off mass can be utilized for conventional rockets (in the order of 1%). For this purpose the motion of the vehicle can be analyzed by looking at the motion of the center of mass only and neglecting the rotational dynamics (Miele 1962).

Rocket ascent into space maximizing the payload is a nonlinear optimal control problem including path constraints and nonlinear boundary constraints defined by the orbit. It is also a multiphase problem because the state variables are discontinuous at certain stage separation times, and the right hand sides of the differential equations may have discontinuities as well.

In this chapter optimal ascent trajectories for Ariane 5 from Kourou into a geostationary transfer orbit (GTO) with an orbital inclination of 0° are studied. The vehicle dynamic equations of motion are expressed in a flight path coordinate system. It is assumed that there is no wind in the atmosphere. The transformation from the flight path coordinate system to the body-fixed coordinate system is expressed by the angle of attack and the side slip angle. When the thrust-time histories are predetermined and the rotational dynamics of the vehicle are ignored, these two angles are the control inputs to be optimized. Rocket ascent trajectories are usually

constrained by a maximum dynamic pressure and a maximum heat-flux that the rocket can tolerate. Ariane 5 has three stages, which leads to a three phase optimal control problem. After stage separation the lower stage has to return to the earth. For Ariane 5 it is of interest to control the return of the second stage. This can be done by imposing a maximum value on the perigee altitude of the return orbit of the second stage. Publications on trajectory optimization for Ariane 5 can be found in e.g. (Landiech & Aumasson 1986), (Jänsch, Schnepper & Well 1989) and (Well 1990).

The main contribution of this chapter is to study the impact on the optimal payload of different combinations of heat-flux and splash-down constraints. Numerical results show that these two constraints are contradicting, and the results are explained by studying the computed optimal trajectories. In addition problems with the singularity of the flight path angle ($\gamma = \frac{\pi}{2}$) are resolved by introducing an extra phase at the beginning of the first stage, leading to a four phase optimal control problem. Finally, a scheme for allowing the thrust-time history of the boosters to vary is proposed. The mass-flow of the boosters is introduced as a third control. Numerical results indicate that the nominal thrust-time profile is optimal.

The optimal trajectories were calculated numerically using the optimization program package PROMIS (Jänsch & Schnepper 1991).

This chapter is based on (Spangelo & Well 1994).

5.2 Mathematical Models

5.2.1 Equations of Motion

Assuming no wind in the atmosphere and that the thrust forces act along the x-axis of the body-fixed coordinate system, the point mass equations of motion in the flight path coordinate system over a spherical, rotating earth, neglecting the centripetal force due to the rotation of the earth, are (Miele 1962)

$$\dot{\lambda} = \frac{V \mathrm{c}\gamma \mathrm{c}\chi}{r}$$

$$\dot{\tau} = \frac{V \mathrm{c}\gamma \mathrm{s}\chi}{\mathrm{c}\lambda r}$$

$$\dot{r} = V \mathrm{s}\gamma$$

$$\dot{V} = \frac{T \mathrm{c}\alpha \mathrm{c}\beta}{m} + \frac{X_W(\alpha, \beta, \phi)}{m} - g \mathrm{s}\gamma$$

$$\begin{aligned} \dot{\chi} &= \frac{V}{r} \mathrm{c}\gamma \mathrm{s}\chi \tan(\lambda) - 2\omega_E (\mathrm{c}\chi tan(\gamma) \mathrm{c}\lambda - \mathrm{s}\lambda) \\ &\quad - \frac{T \mathrm{c}\alpha \mathrm{s}\beta}{mV \mathrm{c}\gamma} - \frac{Y_W(\alpha, \beta, \phi)}{mV \mathrm{c}\gamma} \end{aligned}$$

$$\dot{\gamma} = \frac{V}{r}\mathrm{c}\gamma + 2\omega_E \mathrm{s}\chi\mathrm{c}\lambda + \frac{T\mathrm{s}\alpha}{mV} + \frac{Z_W(\alpha,\beta,\phi)}{mV} - \frac{g\mathrm{c}\gamma}{V}$$
$$\dot{m} = -\dot{b} \tag{5.1}$$

The different terms in Eq. (5.1) are

- λ and τ are geographical latitude and longitude respectively.
- V is the vehicle velocity relative to an earth fixed coordinate system, and r is its radius vector from the center of the earth,
- χ and γ are azimuth and path inclination which measure the orientation of the flight path coordinate system relative to the local horizontal system.
- T is the thrust, which is defined in Section 5.3.
- m is the mass of the vehicle and $\dot{b}$ is the mass flow of the engines, which is defined in Section 5.3.
- X_W, Y_W, Z_W are the aerodynamic forces acting on the vehicle decomposed in the flight path coordinate system.
- α, β and ϕ are the angle of attack, the side slip angle and the vehicles roll angle respectively, defining the transformation from body-fixed to flight path coordinate system.

  The transformation is defined by first a rotation $-\alpha$ about the body-fixed y-axis to the new system (x', y, z'), then a rotation β about the z' axis to the coordinate system (x_W, y'', z') is performed, and finally a rotation ϕ about the x_W axis to the flight path coordinate system (x_W, y_W, z_W). In the following it will be assumed that the roll-angle ϕ is zero, because for a vehicle like Ariane 5 it will be nothing to gain from rolling. The transformation of a vector from body-fixed to flight path coordinate system is then given by the rotation matrix (see Figure 5.1)

$$\mathbf{R}_{z,\beta} \cdot \mathbf{R}_{y,-\alpha} = \begin{pmatrix} \mathrm{c}\beta\mathrm{c}\alpha & \mathrm{s}\beta & \mathrm{c}\beta\mathrm{s}\alpha \\ -\mathrm{s}\beta\mathrm{c}\alpha & \mathrm{c}\beta & -\mathrm{s}\beta\mathrm{s}\alpha \\ -\mathrm{s}\alpha & 0 & \mathrm{c}\alpha \end{pmatrix} \tag{5.2}$$

- The rotation of the earth is

$$\omega_E = 7.2921 \cdot 10^{-5} \text{ rad/s}$$

  The gravitational acceleration g is given by

$$g(r,\lambda) = \frac{\mu_E}{r^2}(1 + 1.5 \cdot J_2(\frac{R_E}{r})^2(1 - 3\mathrm{s}^2\lambda)) \tag{5.3}$$

  where the oblateness of the earth is

$$J_2 = 0.0010826$$

and the gravitational constant of the earth is

$$\mu_E = 398602 \text{ km}^3/\text{s}^2$$

The equatorial radius of the earth is

$$R_E = 6378155 \text{ m}$$

- The following trigonometric abbreviations have been used

$$\cos(\cdot) = \text{c}$$

$$\sin(\cdot) = \text{s}$$

5.2.2 Singularity Avoidance

There are two singularities, at $V = 0$ and at $\gamma = \frac{\pi}{2}$. The velocity singularity is avoided by assuming that the velocity V is always larger than some positive value.

A rocket is usually launched vertically or almost vertically, which means that the state space equations (Eqs. 5.1) are singular at the initial time of the first stage (see Eq. 5.13). For a rocket ascended into space this singularity is only a problem in the first part of the ascent, if it is assumed that after the first part the path inclination γ is bounded away from 90°.

The singularity can be avoided by defining $\gamma(0) = \frac{\pi}{2} - \varepsilon$ where $\varepsilon > 0$. With proper controls the singularity will be avoided.

Here, we propose the following solution for the flight path inclination singularity. The azimuth χ is fixed to $\chi(0) = \chi_0$, and removed from the state space equations for the first part of the ascent. The side slip angle β is fixed to zero. This results in the following state space equations

$$\dot{\lambda} = \frac{V\text{c}\gamma\text{c}\chi_0}{r}$$

$$\dot{\tau} = \frac{V\text{c}\gamma\text{s}\chi_0}{\text{c}\lambda r}$$

$$\dot{r} = V\text{c}\gamma$$

$$\dot{V} = \frac{T\text{c}\alpha}{m} + \frac{X_W(\alpha)}{m} - g\text{s}\gamma$$

$$\dot{\gamma} = \frac{V}{r}\mathrm{c}\gamma + 2\omega_E \mathrm{s}\chi_0 \mathrm{c}\lambda + \frac{T\mathrm{s}\alpha}{mV} + \frac{Z_W(\alpha)}{mV} - \frac{g\mathrm{c}\gamma}{V}$$
$$\dot{m} = -\dot{b} \tag{5.4}$$

The vehicle is launched vertically until $t = t_v$ by fixing the angle of attack α to zero. At $t = t_v$ a pitch program starts, where α decreases linearly until $t = t_1$ and $\alpha(t_1) < 0$ is chosen such the path inclination $\gamma(t_1) < \frac{\pi}{2} - \varepsilon$, where $\varepsilon > 0$. At $t = t_1$ a switch to the original state spce equations (Eq. 5.13) is performed. In a multi-phase optimal control problem this results in an additional phase.

5.2.3 Aerodynamic Forces

The aerodynamic forces decomposed in the body-fixed coordinate system acting on the vehicle are given as

$$X_f = -qC_{D_0}S$$
$$Y_f = qC_yS$$
$$Z_f = qC_zS \tag{5.5}$$

Here, S is a reference area (22.0 m^2 for Ariane 5), q is the dynamic pressure

$$q = \frac{\rho}{2}V^2 \tag{5.6}$$

where ρ is the density of the air. The factors C_{D_0}, C_y and C_z are calculated from a vehicle specific subroutine. The latter two are given by

$$C_y = \frac{\partial C_y(M)}{\partial \beta_f}\beta_f$$
$$C_z = \frac{\partial C_N(M)}{\partial \alpha_f}\alpha_f \tag{5.7}$$

The mach number M is given by

$$M = \frac{V}{a(h)}$$

where $a(h)$ is the speed of sound and h is the altitude of the vehicle. Given altitude h as input the speed of sound $a(h)$ and the density of the air ρ is calculated from a subroutine based on the US standard atmosphere model of 1962, described in e.g. (Well 1991).

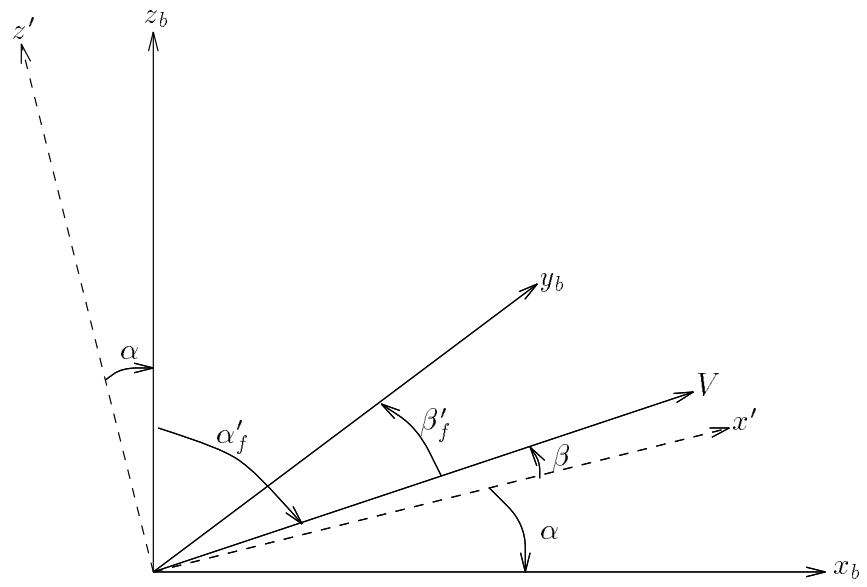

Figure 5.1: *Defining angle of attack and side slip angle*

α_f is the angle between the velocity vector (flight path x-axis) and the projection of the velocity vector in the body fixed xy-plane (direction from the flight path x-axis).

β_f is the angle between the velocity vector (flight path x-axis) and the projection of the velocity vector in the body fixed zx-plane (direction towards the flight path x-axis).

The unity vectors along the body-fixed y-axis ($\mathbf{j}_b$) and z-axis ($\mathbf{k}_b$) decomposed in the flight path coordinate system are obtained from Eq. (5.2) as

$$\mathbf{j}_b^{\mathrm{w}} = \begin{pmatrix} \mathrm{s}\beta \\ \mathrm{c}\beta \\ 0 \end{pmatrix}; \quad \mathbf{k}_b^{\mathrm{w}} = \begin{pmatrix} \mathrm{c}\beta\mathrm{s}\alpha \\ -\mathrm{s}\beta\mathrm{s}\alpha \\ \mathrm{c}\alpha \end{pmatrix} \tag{5.8}$$

The angles between the velocity vector and the body-fixed y-axis and z-axis are denoted β'_f, and α'_f respectively. This gives

$$\begin{aligned} \alpha_f &= \frac{\pi}{2} - \alpha'_f \\ \beta_f &= \frac{\pi}{2} - \beta'_f \end{aligned} \tag{5.9}$$

The angles α, β, α'_f and β'_f are given in Figure 5.1. It is further assumed that α_f and β_f are bounded between plus and minus $\frac{\pi}{2}$, which means that α'_f and β'_f are bounded between zero and π.

The cosine law gives

$$\cos(\beta'_f) = \mathbf{i}_w \cdot \mathbf{j}_b = \sin(\beta)$$

$$\cos(\alpha'_f) = \mathbf{i}_w \cdot \mathbf{k}_b = \cos(\beta) \cdot \sin(\alpha) \tag{5.10}$$

Together with Eq. (5.9) this gives

$$\sin(\beta_f) = \sin(\beta)$$
$$\sin(\alpha_f) = \cos(\beta)\sin(\alpha)$$

Together with the definitions of the directions of α_f and β_f this gives

$$\beta_f = \beta$$
$$\alpha_f = \arcsin(\cos(\beta) \cdot \sin(\alpha)) \tag{5.11}$$

5.3 Three Stage Ascent for Ariane 5

The Ariane 5 rocket is to be ascended into orbit starting from Kourou while maximizing the payload, i.e. the mass at the final time t_f

$$\min\{-m(t_f)\} \tag{5.12}$$

subject to the state space equations (Eqs. 5.1), with initial conditions

$$\lambda(0) = -52.82^\circ = -0.9219$$
$$\tau(0) = 5.25^\circ = 0.0916$$
$$r(0) = R_E + 5\text{m} = 6378160\text{m}$$
$$V(0) = 5\text{m/s}$$
$$\chi(0) = \frac{\pi}{2}$$
$$\gamma(0) = \frac{\pi}{2}$$
$$m(0) = 717348\text{kg} \tag{5.13}$$

The initial value for χ can be defined as an optimizable parameter.

To achieve this Ariane 5 has three stages, which will be defined below. The engines of the lower stages of Ariane 5 are two solid-propellant boosters and the engine H155. In this work the upper stage is the engine L5, see also (Jänsch et al. 1989) and (Flury & Prieto-Llanos 1987).

Maximizing the payload for the three stage rocket ascent is a three phase optimal control problem. The state variables can be discontinuous at the stage separation times. The aerodynamic forces are neglected in stage two and three.

The control inputs for the optimization problem using point mass equations and fixed thrust time histories are the angle of attack α and the side slip angle β.

5.3.1 A Four-Phase Optimal Control Problem

The state space equations are defined by Eq. (5.4) in the first phase, and by Eq. (5.1) in the last three phases.

Stage 1, Phase 1 and 2

The first stage is the H155 together with the two boosters. The thrust is given by

$$T = T_{H155} + 2 \cdot T_{P230} = 9.81 \cdot I_{spH155} \cdot \dot{b}_{H155} + 2 \cdot T_{P230} \tag{5.14}$$

The mass flow is given by:

$$\dot{b} = \dot{b}_{H155} + 2 \cdot \dot{b}_{P230} \tag{5.15}$$

Here,

$$I_{spH155} = 430 \text{ s}$$

is the specific impulse of the H155, and

$$\dot{b}_{H155} = 256.4 \text{ kg/s}$$
$$\dot{b}_{P230} = 1933 \text{ kg/s}$$

is the mass flow of the H155 and each of the boosters, respectively.

The thrust time history of each of the boosters, shown in Figure 5.2, is approximated by linear interpolation (Jänsch et al. 1989). We propose to define the following two phases for the first stage.

Phase 1

The initial conditions for the first phase are

$$\begin{aligned} \lambda(0) &= -52.82^\circ = -0.9219 \\ \tau(0) &= 5.25^\circ = 0.0916 \\ r(0) &= R_E + 5\text{m} = 6378160\text{m} \\ V(0) &= 5\text{m/s} \\ \gamma(0) &= \frac{\pi}{2} \\ m(0) &= 717348\text{kg} \end{aligned} \tag{5.16}$$

Possible optimizable parameters are

1. χ_0: The initial value of the azimuth. This parameter has a nominal value of $\frac{\pi}{2}$.
2. $\alpha_1 = \alpha(t_1)$: The angle of attack at the end of the pitch program.
3. t_v: The initial time for the pitch program. The nominal value is $t = 10$ s.
4. t_1: The final time of the first phase. The nominal value is $t = 20$ s.

There is one boundary condition in the first phase

$$\gamma(t_1) < 85^\circ$$

Phase 2

Connect conditions: All the 5 states from the first phase are continuous at the initial time t_1. The new state χ equals the parameter χ_0, i.e.

$$\chi(t_1) = \chi_0$$

The controls at $t = t_1$ are also continuous, i.e.

$$\alpha(t_1) = \alpha_1$$

$$\beta(t_1) = 0$$

The final time of the second phase, i.e. the first stage, is

$$t_2 = 119.177 \text{ s}$$

Stage 2, Phase 3

The second stage is defined as the flight path with the H155 after jettisoning of the boosters. The thrust is given by

$$T = T_{H155} \tag{5.17}$$

The mass flow is given by

$$\dot{b} = \dot{b}_{H155} \tag{5.18}$$

Connect conditions:

$$m(119.177\text{s}+) = 150558 \text{ kg}$$

All the other states and the controls are continuous at $t = t_2$.

The final time of the second stage, i.e. the third phase, is

$$t_3 = 600 \text{ s}$$

Stage 3, Phase 4

The third stage is the L5 after burn out of the H155 and stage separation. The thrust is given by

$$T = T_{L5} = 9.81 \cdot I_{spL5} \cdot \dot{b}_{L5} \tag{5.19}$$

The mass flow is given by

$$\dot{b} = \dot{b}_{L5} = 6.75 \text{ kg/s} \tag{5.20}$$

and the specific impulse is

$$I_{spL5} = 317 \text{ s}$$

Connect conditions:

$$m(600\text{s}+) = 13085 \text{ kg}$$

All the other states and the controls are continuous at $t = t_3$.

The final time $t = t_f = t_4$ is free. Actually, with constant mass flow, maximizing the payload is equivalent to minimizing t_f.

5.3.2 Constraints

The rocket ascent has to satisfy certain path constraints and boundary constraints, in addition to the boundary constraint on γ at $t = t_1$.

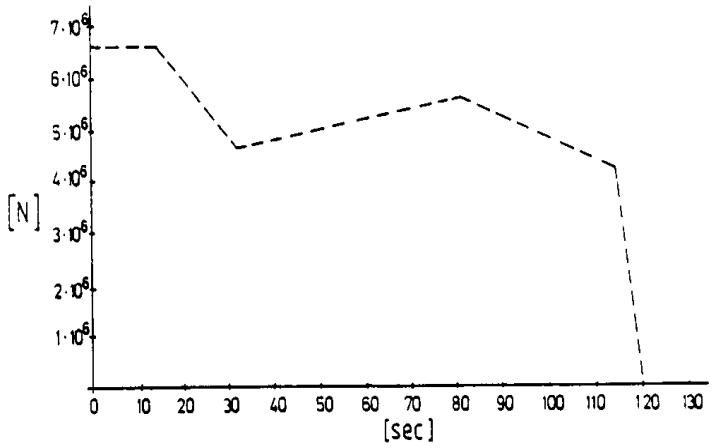

Figure 5.2: *Thrust-time history for the boosters*

Path Constraints

(see e.g. (Well 1991))

The dynamic pressure has to be bounded

$$q = \frac{\rho}{2} V^2 \leq 40kPa \tag{5.21}$$

This constraint is especially critical in the first stage.

The heat-flux

$$Q = 1.7 \cdot 10^{-4} \sqrt{\rho} V^3 \leq Q_{\text{ref}} \tag{5.22}$$

will often be limited too.

Boundary Conditions

To be able to define orbital boundary constraints the velocity, the azimuth and the path inclination of the vehicle relative to an inertial system have to be computed (subscript I)

$$\begin{aligned} V_I^2 &= V^2(t_f) + r\omega_E \mathrm{c}\lambda(t_f)(r\omega_E \mathrm{c}\lambda(t_f) \\ &\quad + 2V(t_f)\mathrm{s}\chi(t_f)\mathrm{c}\gamma(t_f)) \end{aligned} \tag{5.23}$$

$$\mathrm{s}\gamma_I = \frac{V(t_f)\mathrm{s}\gamma(t_f)}{V_I}$$

$$\mathrm{c}\chi_I = \frac{V(t_f)\mathrm{c}\gamma(t_f)\mathrm{c}\chi}{V_I \mathrm{c}\gamma_I} \tag{5.24}$$

Eccentricity e is obtained from

$$e = \sqrt{1-(2-f)fc^2\gamma_I} \tag{5.25}$$

with the parameter f defined as

$$f = \frac{rV_I^2}{\mu_E} \tag{5.26}$$

The perigee $rper$ and the apogee $rapo$ of the orbit are given by

$$\begin{aligned} rper &= \frac{r\mu_E}{2\mu_E - rV_I^2}(1-e) \\ rapo &= \frac{r\mu_E}{2\mu_E - rV_I^2}(1+e) \end{aligned} \tag{5.27}$$

The shape of the orbit is defined by the perigee and the apogee altitude which give two boundary constraints

$$rper = R_E + H_{\mathrm{PER}} \tag{5.28}$$

$$rapo = R_E + H_{\mathrm{APO}} \tag{5.29}$$

A third equality constraint is to specify the inclination inc of the orbit, given by

$$\cos(inc) = \cos(\lambda(t_f))\sin(\chi_I) \tag{5.30}$$

In addition to the three equality boundary constraints at $t = t_f$, the perigee altitude at the end of the second stage can be bounded above to control the return of the second stage by controlling the return-orbit of the second stage. Therefore this inequality constraint is referred to as a splash-down constraint, and is defined by

$$H_{\mathrm{PER}}(t_3) \leq \mathcal{H}_{\mathrm{PER}} \tag{5.31}$$

where $\mathcal{H}_{\mathrm{PER}}$ is a constant (Well 1990).

For a comprehensive treatment of orbital mechanics, see e.g. (Chobotov 1990).

5.4 Computational Study

The optimal trajectories were calculated numerically using the optimization program package PROMIS (Jänsch & Schnepper 1991). PROMIS uses direct shooting, multiple or single (see Chapter 2.2). The program has an advanced user interface which makes it easy to handle multi-phase problems, path constraints and parameter initializations (Jänsch, Schnepper & Well 1994). In this work, only single shooting has been applied. In PROMIS the resulting nonlinear programs are solved using a modified version of SLSQP (Kraft 1988), namely SQPHR. This implementation, which is presented in (Jänsch & Schnepper 1991), includes a high rank version of the BFGS-update of the Hessian matrix proposed by Bock & Plitt (1984) in order to utilize the sparseness structure obtained by multiple shooting.

The optimal control problem was parametrized using single shooting and piecewise linear interpolation of the controls in all computations. Except for the single control switching time node in the first phase, the switching time nodes were chosen to be fixed.

A geostationary transfer orbit (GTO) has been studied. The perigee and apogee altitude constraints were $H_{\mathrm{PER}} = 200\mathrm{km}$ and $H_{\mathrm{APO}} = 36000\mathrm{km}$. The inclination constraint was $inc = 0°$.

The introduction of the extra phase at the start of the ascent efficiently solved the singularity avoidance problem. The optimizable parameters in the singularity avoidance phase did not noticeably influence the optimal payload, and will not be discussed any further.

5.4.1 Splash-Down and Heat-Flux Constraints

Optimal payload $m(t_f)$ is given for different combinations of heat-flux and splash-down constraints in Table 5.1. It can be seen that these two constraints are contradicting. The cost of satisfying one of them is small compared to satisfying both. In the case of neither splash-down nor heat-flux constraints both maximum heat-flux and splash-down perigee altitude were higher than any of their respecting values in Table 5.1. For instance the splash-down perigee altitude was 91 km. When a heat-flux constraint of 50 $\mathrm{kW/m^2}$ was introduced the splash-down perigee altitude increased to 106 km. On the other hand, Table 5.1 shows that only a minor decrease in optimal payload occurred. The same observations were made when only splash-down constraints were introduced, i.e. the maximum heat-flux increased significantly while only a minor decrease in the optimal payload occurred.

These results can be explained by the behavior of the rocket in a period starting about 200 seconds before the end of the second stage and lasting about 200 seconds into the third stage. In this period the flight path angle becomes negative and the rocket starts to dive into the atmosphere. The result is that the rocket gains velocity when it is heavy, which it utilizes to escape more easily from the gravitation field

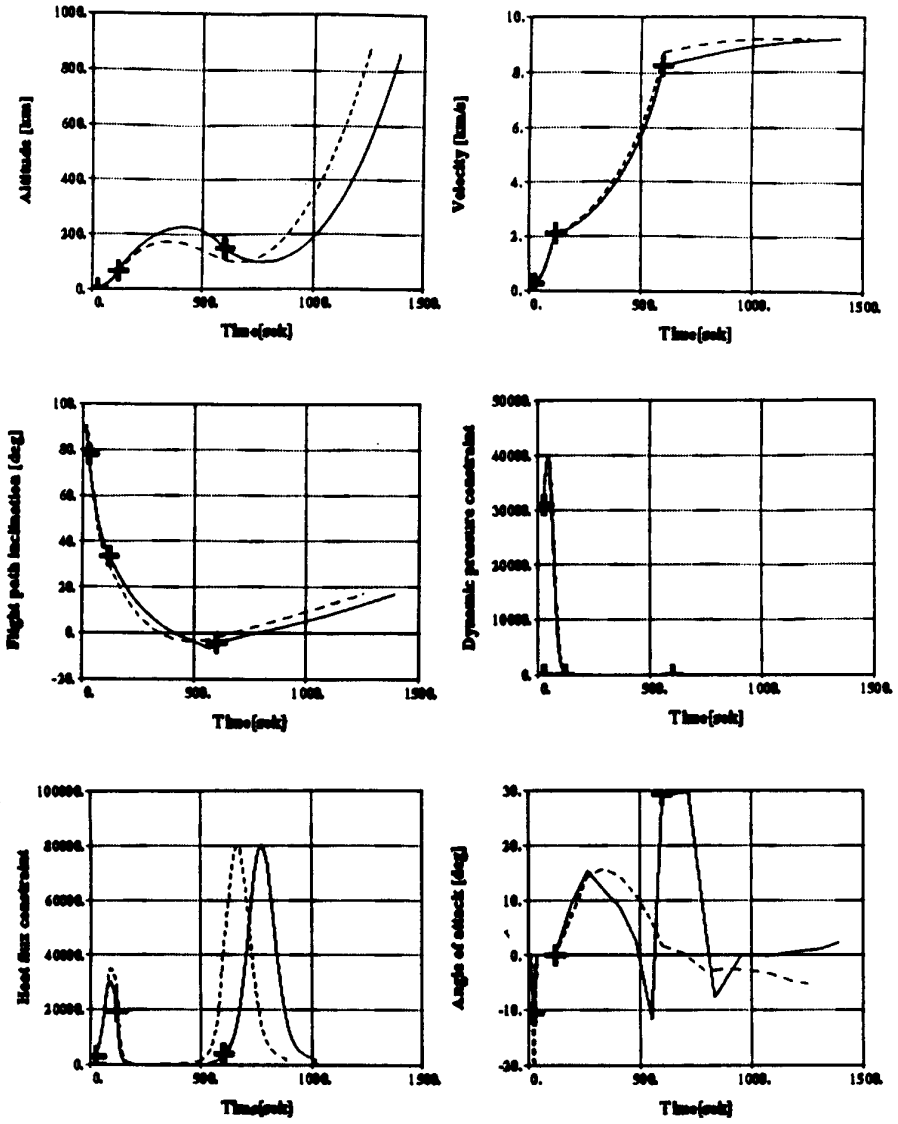

Figure 5.3: *Optimal Trajectories*

when it becomes lighter. The heat-flux constraint limits the velocity of the vehicle at its minimum altitude in the third stage, while trying to increase the minimum altitude itself. The splash-down constraint forces the shape of the trajectory to either have a low altitude or to have a very negative flight path inclination at the end of the second stage. A reduction in possible velocity can be compensated for by increasing the altitude, whereas a reduction in possible altitude can be compensated for by increasing the velocity.

Figure 5.4 shows optimal trajectories without any splash-down constraints in the case of no heat-flux constraint (dashed curves) and in the case of a maximum heat-flux od 80 kW/m^2 (solid curves). It can be seen that the speed is almost maintained when the heat-flux constraint is satisfied by increasing the minimum altitude.

When there are both heat-flux and splash-down constraints the rocket is not per-

Max. heat-flux (kW/m^2)	Max. splash-down perigee altitude (km) 60	70	80	None
$40 \cdot 10^3$	6821	7059	7421	8708
$50 \cdot 10^3$	6880	7146	7555	8714
$60 \cdot 10^3$	6927	7217	7667	8720
$70 \cdot 10^3$	6972	7283	7766	8723
$80 \cdot 10^3$	7014	7353	7871	8726
None	8638	8684	8721	8739

Table 5.1: *Optimal payload (kg) for different combinations of splash-down and heat-flux constraints*

mitted to go deep enough into the atmosphere to satisfy the splash-down constraint with a moderate negative flight path angle. Therefore, these two constraints contradict each other, resulting in a significant decrease in optimal payload when both have to be satisfied. This is illustrated in Fig. 5.3, which shows optimal trajectories of the rocket for a maximum heat-flux of 80 kW/m^2, and with (solid) and without(dashed) a maximum splash-down perigee altitude of 80 km. It can be seen in the figure that the splash-down constraint is satisfied by generating a more negative flight path inclination at the end of the third stage, which again is achieved by increasing the maximum altitude in the second stage. This has the result that the minimum altitude occurs later with a lower velocity. It can also be seen that this maneuvre is computationally expensive in that the time consumption becomes considerably higher. Note that, in the case of fixed thrust time histories and mass flows, minimization of time consumption is equivalent to maximization of payload.

Shown in both figures are altitude, velocity, flight path inclination, dynamic pressure, heat-flux, and angle of attack.

5.4.2 Varying the Thrust Time History of the Boosters

The thrust time history for each of the boosters is assumed to be given by

$$T = 9.81 \cdot I_{\mathrm{sp230}} \cdot \dot{m}_{\mathrm{TP230}} \tag{5.32}$$

where $\dot{m}_{\mathrm{TP230}} = 1933$ s is the constant mass flow. The thrust T and the specific impulse I_{sp230} are given by linear interpolation from the values in Table 5.2. The burned out mass m_{p230b} is also given in the table. The specific impulse is calculated from Eq. (5.32).

The interpolation scheme in Table 5.2 gives

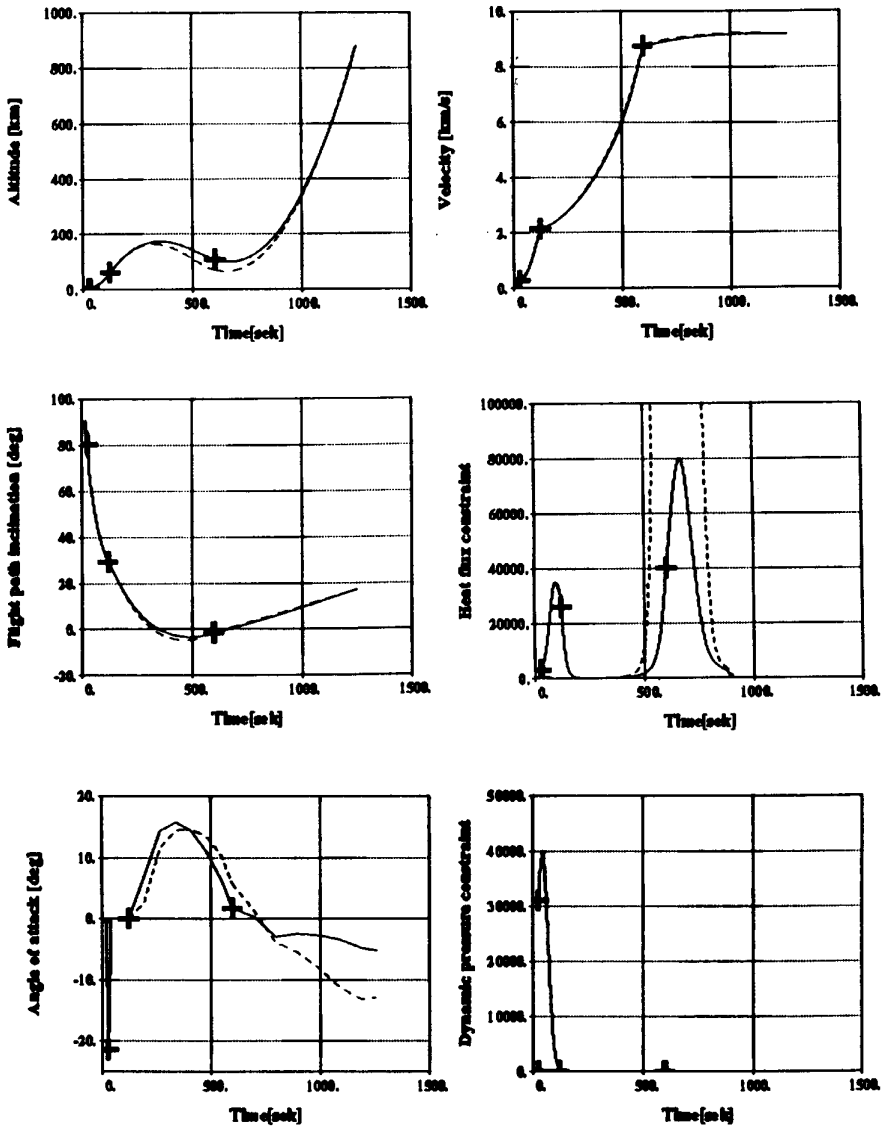

Figure 5.4: *Optimal Trajectories*

$$m_{\mathrm{p230b}}(119.177 \text{ s}) = 230369.1 \text{ kg}$$

It is now assumed that the specific impulse is a function of burned out mass and linearly interpolated according to the relations between the two quantities in Table 5.2. The mass flow is then introduced as an extra control variable. The final time of stage one (phase two) is now free to be optimized. The boundary constraint $m_{\mathrm{p230b}}(t_f) = 230369.1$ kg has to be satisfied.

Maximum heat-flux was set to 50 kW/m^2. The heat-flux constraint now also had to be included in the second phase. The third control, $\dot{m}_{\mathrm{TP230}}$ was limited to be between 1400 s and 2600 s (nominally 1933 s).

Without any splash-down constraints optimal payload was computed to 8999 kg

time (s)	Thrust (N)	m_{p230b}	I_{sp230} (kg)
0.0	$6.618 \cdot 10^6$	0.0	349.0
13.0	$6.618 \cdot 10^6$	25129.0	349.0
31.0	$4.676 \cdot 10^6$	59923.0	246.589
80.0	$5.647 \cdot 10^6$	154640.0	297.795
113.8	$4.353 \cdot 10^6$	219975.4	229.556
120.0	10^3	231960.0	0.053

Table 5.2:

which is an improvement of 3.3%. Introducing a maximum splash-down perigee altitude of 80 km the optimal payload was computed to 7607 kg, which is an improvement of only 0.7%. The nominal thrust-time history seemed to be quite optimal with both these two constraints satisfied.

5.5 Conclusions

Optimal rocket ascent trajectories have been computed using the general purpose CVP program system PROMIS. CVP with single shooting has been applied.

Payload maximization for the ascent of the Ariane 5 rocket has been formulated as a four-phase optimal control problem. Ariane 5 is a three stage rocket. The fourth phase was introduced for the purpose of singularity avoidance.

Optimal trajectories have been computed for ascending Ariane 5 into a geostationary transfer orbit. The resulting optimal payload for different combinations of heat-flux and splash-down constraints showed that these two constraints are contradicting.

Chapter 6

Conclusions and Reccomendations

6.1 Conclusions

This thesis contains a study of optimal trajectories for vehicles. Highly constrained nonlinear optimal control problems have been solved numerically using control vector parameterization and nonlinear programming.

Control vector parameterization with shooting has been described in detail to provide the reader with the theoretical background for the methods which have been implemented, and which are not available in standard text books. Theoretical contributions on accuracy analysis and gradient computations have also been presented.

Optimal trajectories have been computed for thruster controlled underwater vehicles. A class of nonlinear optimal control problems including energy-minimization, possibly combined with time-minimization and obstacle avoidance, has been developed. A program system has been specially designed and written in the C language to solve this class of optimal control problems. Control vector parameterization with single shooting was used. This special implementation has made it possible to perform a detailed analysis, and to investigate numerical details of this class of optimization methods which would have been difficult using a general purpose CVP program system.

Several optimization runs were performed for a 6 DOF underwater vehicle controlled by 6 DC-motor driven thrusters. A collision avoidance problem, where the vehicle was commanded to move 10 m in the x- and y-directions and 5 m in the z-direction while avoiding a cylindrical obstacle, was solved on a Sun Sparc 10 station using about 15 CPU-seconds. The weighted combination of energy and time consumption was optimized, and the resulting optmal time consumption for the task was about 30 seconds. The results show that this method is well suited for use in guidance and control schemes for marine vehicles.

The use of numerical methods to solve general optimal control problems for marine vehicles is a new area of research. For other vehicles like aerospace vehicles, however,

the use of advanced numerical methods in trajectory optimization has been an area of research for several decades. Results from rocket trajectory optimization has been studied in this work to bring knowledge from this area into the new area of trajectory optimization for marine vehicles. A case study of optimal ascent trajectories for the Ariane 5 rocket has been performed using the general purpose CVP program system PROMIS. The contributions here have been on the formulation of the problem and the study of the resulting trajectories.

The contributions of the thesis are stated more specifically in the introduction (Chapter 1). The work has also been related to guidance and control, and put into a larger context. In particular, it has been outlined how the work can be related to the MOBATEL program.

6.2 Recommendations for Further Works

There are many possible trajectory optimization problems for vehicles which are not solved. In this section some interesting optimal control problems for marine vehicles are mentioned. Further use of CVP with shooting is also discussed.

6.2.1 Trajectory Optimization Problems for Marine Vehicles Which are not Solved in This Thesis

As emphasized in Chapter 1 trajectory optimization for vehicles have been limited to a collection of case studies, more specifically to underwater vehicles.

The introduction of control surfaces or rudders can be handled by minor modification of the dynamic equations of motion. The energy needed to move a control surface or a rudder is generally negligible compared to the thruster energy, which means that the objective functional will be the same. The same can be said when surface vehicles are considered instead of underwater vehicles. These modifications are therefore not likely to give rise to new problems in the area of trajectory optimization for marine vehicles.

As discussed in Chapter 4, when a task consists of several different subtasks, the introduction of multi-phase optimal control problems may be of interest. In each phase different reduced models of the vehicles may be used.

The optimal control problems formulated for underwater vehicles controlled by DC-motor driven thrusters are recognized in particular by the energy part of the objective functional (Eq. 4.14). The expression for this part of the objective functional is an approximation of the energy dissipation due to the interaction between the propeller and the water. The energy dissipation in the DC-motor is neglected. In fact, for any thruster or propeller control device where the energy consumption in the motors are neglected, the objective functional defined in Eq. (4.14) would be

adequate for minimizing a weighted combination of energy and time consumption. If, however, the energy consumption in the motor or machinery cannot be neglected, an extra term has to be introduced. An interesting class of problems would be fuel optimal trajectories for ships driven by diesel engine driven propellers. Models of diesel engine speed propulsion systems can be found in (Fossen 1994). It is not trivial to tell whether the energy losses in the engine should be included in the objective functional or not. A study would have to be performed to decide this matter. If the energy losses in the engine were included, the dynamics of the propulsion system would probably also have to be included.

It should also be mentioned that there are alternatives to the thrusters and propeller systems as propulsion devices, i.e. water jets. Interesting applications would be high speed crafts like surface effect ships and foilborne catamarans (Sørensen 1993, Fossen 1994).

In addition, special tasks may also be of particular interest.

6.2.2 Further Works on the Numerical Implementation

The dynamic models of marine vehicles are typically described by nonstiff differential equations. When CVP and NLP are used, I will therefore recommend the use of shooting methods with explicit Runge-Kutta initial value problem solvers for the numerical solution of the optimal control problem.

The numerical implementations in Chapter 4 exclusively use single shooting. For long time ranging tasks which would require significantly higher numbers of parameters per control than the tasks solved in Chapter 4, the introduction of shooting nodes, i.e. using multiple shooting, would increase the efficiency of the gradient computations significantly. If the number of parameters becomes very large, multiple shooting would also be a mean to obtain the necessary sparseness to being able to solve the resulting large scale NLP efficiently. These aspects are also discussed in Chapter 2.

There are alternative parameterization strategies for the shooting approach to those implemented in this thesis (see Chapter 2). Using independent control time nodes for each of the controls, and using higher degree polynomials to represent the controls may be of interest. It is, however difficult to decide how independent control time nodes should have been defined. The optimal definition would also, of course, be task dependent. I also doubt the possibility of obtaining more than marginal improvements, if any, by using piecewise polynomials of higher degrees to represent the controls.

It should also be mentioned that there are alternative ways of including obstacle avoidance. One alternative could be to include barrier functions instead of path constraints. The path constraint approach implemented in Chapter 2 performed so well that it was decided not to try alternatives like barrier functions, which are known to have a negative effect on the properties of an optimization problem. For

reasons explained in Chapter 2, representing the path constraints by the integral approach was rejected.

6.2.3 Using Trajectory Optimization in Guidance Schemes

The numerical solution of a general optimal control problem usually results in an off-line feedforward control strategy. The resulting optimal trajectories alone are very interesting in analysis, but because of model uncertainties and unknown external disturbances this control strategy must be used in combination with feedback control in order to get a robustly stable guidance and control scheme.

Using the second variation of the optimal control problem an optimal feedback controller is obtained. A linear time-variant set of Riccati equations has to be solved (Bryson & Ho 1975). This optimal feedback control strategy is not necessarily the best in terms of stability and robustness. Another feedback control strategy would lead to a suboptimal control and guidance scheme. The feedback control design would then be completely independent of the trajectory generation. Therefore, these two parts of the guidance and control scheme are naturally treated separately.

These aspects are also discussed in the introduction of the thesis. It is mentioned here as a possibility for further works in the area of trajectory optimization for marine vehicles. I believe that a particularly important application in the future will be in guidance for use in ship autopilots. The ship guidance schemes of today are based on simple heuristic path planners, and great improvements of the task performances in terms of energy and time consumption can be expected by the use of energy optimal trajectories. The results reported in this thesis show that trajectory optimization using CVP with shooting is well suited for use in guidance and control of marine vehicles.

Even if the optimal trajectory computation and the feedback control design are independent, it is of great interest to investigate how these two parts of the total system work together, possibly also with state and parameter estimation, like in the state-space predictive control schemes for process control (Balchen et al. 1992). A guidance scheme for flight control based on optimal trajectories is considered in (Paus 1992). The perhaps most interesting subjects for future research in this area will be on practical implementations of optimal trajectory based guidance and control schemes.

Bibliography

Abkowitz, M. (1969). *Stability and Motion Control of Ocean Vehicles*, M.I.T. Press, Boston, Massachusetts.

Athans, M. & Falb, P. L. (1966). *Optimal Control. An introduction to the theory and its application*, McGraw-Hill Book Company, New York.

Balchen, G. G., Ljungquist, D. & Strand, S. (1992). State-space predictive control, *Chemical Engineering Science* **47**(4): 787–807.

Balchen, J. G. & Fossen, T. I. (1992). Model based teleoperation of an underwater vehicle over a narrow band communication link - mobatel, *International Advanced Robotics Programme (IARP), 4th Workshop on Underwater Robotics*, Genova, Italy.

Beale, E. M. L. (1967). Numerical methods, *in* J. Abadie (ed.), *Nonlinear Programming*, North-Holland, Amsterdam.

Betts, J. T. & Huffman, W. P. (1992). Application of sparse nonlinear programming to trajectory optimization, *J. Guidance, Control and Dynamics* **15**(1): 198–206.

Betts, J. T. & Huffman, W. P. (1993). Path constrained trajectory optimization using sparse sequential quadratic programming, *J. Guidance, Control and Dynamics* **16**(1): 59–68.

Biegler, L. T. (1984). Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation, *J. Comp. Chem. Eng.* **8**(3): 243–248.

Biggs, M. C. (1975). Constrained minimization using recursive quadratic programming: Some alternative sub-problem formulations, *in* L. C. W. Dixon & G. P. Szego (eds), *Towards Global Optimization*, North-Holland, Amsterdam.

Biggs, M. C. (1978). On the convergence of some constrained minimization algorithms based on recursive quadratic programming, *J. Inst. Math. Applics.* **21**: 67–81.

Bock, H. G. (1983). Recent advances in parameter identification techniques in o.d.e., *in* P. Deuflhard & E. Hairer (eds), *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, Birkhäuser, Boston, pp. 95–121.

Bock, H. G. & Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems, *Preprints of the 9th Triennal World Congress of IFAC*, Budapest, Hungary, pp. 243–247. Pergamon-Press, Oxford, pp. 1603-1608, 1985.

Boggs, P. T., Tolle, J. W. & Wang, P. (1982). On the local convergence of quasi-newton methods for constrained optimization, *SIAM J. Control and Optimization* **20**: 161–171.

Brusch, R. G. (1974). A nonlinear programming approach to space shuttle trajectory optimization, *J. Optimization Theory and Applications* **13**: 94–118.

Brusch, R. G. & Peltier, J. P. (1973). Parametric control models and gradient generation by impulsive response, *Proceedings IAF Congress*, Baku.

Bryson, A. E. & Ho, Y. C. (1975). *Applied Optimal Control. Optimization, Estimation, and Control*, Hemisphere Publishing Corporation, Washington, D.C.

Buckley, A. G. (1975). An alternative implementation of goldfarb's minimization algorithm, *Mathematical Programming* **8**: 207–231.

Bulirsch, R. (1971). Die mehrzielmethode zur numerische lösungen von nichtlineare randwertproblemen und aufgaben der optimalen steuerungen, *Technical report*, Carl-Cranz-Gesellschaft e.V., Oberpfaffenhofen, Germany.

Canon, M. D., Cullum, C. D. & Polak, E. (1970). *Theory of Optimal Control and Nonlinear Programming*, McGraw-Hill, New York.

Chamberlain, R. M. (1979). Some examples of cycling in variable metric methods for constrained minimization, *Mathematical Programming* **16**: 378–384.

Chobotov, V. A. (1990). *Orbital Mechanics*, AIAA Education Series, New York.

Colemann, T. F. & Conn, A. R. (1982). Nonlinear programming via an exact penalty function, *Mathematical Programming* **24**: 123–161.

Cuthrell, J. E. & Biegler, L. T. (1987). On the optimization of differential-algebraic process systems, *AIChE Journal* **33**(8): 1257–1270.

Dennis, J. E. & Schnabel, R. E. (1981). A new derivation of symmetric positive definite secant updates, *in* O. L. Mangasarian, R. R. Meyer & S. M. Robinson (eds), *Nonlinear Programming 4*, Academic Press, London and New York, pp. 167–199.

Deuflhard, P. (1980). Recent advances in multiple shooting techniques, *in* Gladwell & Sayers (eds), *Computational Techniques for Ordinary Differential Equations*, Academic Press, London - New York.

Faltinsen, O. M. (1990). *Sea Loads on Ships and Offshore Structures*, Cambridge University Press, Cambridge.

Fjellstad, O.-E. (1994). *Control of Unmanned Underwater Vehicles in Six Degrees of Freedom: A Quaternion Feedback Approach*, PhD thesis, Dept. of Engineering Cybernetics, The Norwegian Institute of Technology, Trondheim.

Fletcher, R. (1987). *Practical Methods of Optimization*, 2nd edition edn, John Wiley and Sons, Chichester.

Flury, W. & Prieto-Llanos, T. (1987). Ariane v performance in gto with configuration 2p230/h155/l5, *Memorandum*, ESOC.

Fossen, T. I. (1991). *Nonlinear Modeling and Control of Underwater Vehicles*, PhD thesis, University of Trondheim, NTH, Division of Eng. Cyb., Trondheim, Norway. (ITK-rapport 1991:42-W).

Fossen, T. I. (1994). *Guidance and Control of Ocean Vehicles*, John Wiley & Sons Ltd., London.

Gabay, D. (1982). Reduced quai-newton methods with feasibility improvement for non-linearly constrained optimization, *Math. Prog. Study* **16**: 18–44.

Gill, P. E. & Murray, W. (1974). Safeguarded steplength algorithms for optimization using descent methods, *Report NAC 37*, National Physical Laboratory, England.

Gill, P. E., Murray, W. & Saunders, M. A. (1993). Snopt: A sparse nonlinear optimizer, *in* W. W. Hager & D. W. Hearn (eds), *Conference on Large Scale Optimization*, pp. 113–138.

Gill, P. E., Murray, W. & Saunders, M. A. (1994). Large-scale sqp methods and their applications in trajectory optimization, *in* R. Bulirsch & D. Kraft (eds), *Computational Optimal Control*, Birkhauser Verlag, Basel Swtzerland, pp. 29–42. ISNM Vol. 115.

Gill, P. E., Murray, W., Saunders, M. A. & Wright, M. H. (1984). Procedures for optimization problems with a mixture of bounds and general linear constraints, *ACM Trans. Math. Software* **10**: 282–298.

Gill, P. E., Murray, W., Saunders, M. A. & Wright, M. H. (1985). Model building and practical aspects of nonlinear programming, *in* K. Schittkowski (ed.), *Computational Mathematical Programming*, Springer-Verlag, Berlin, pp. 209–246. (NATO ASI Series. Series F: Computer and System Science, Vol. 15).

Gill, P. E., Murray, W., Saunders, M. A. & Wright, M. H. (1986a). Some theoretical properties of an augmented lagrangian merit function, *Technical Report SOL 86-6*, Department of operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. & Wright, M. H. (1986b). User's guide for npsol: A fortran package for nonlinear programming, *Technical Report SOL 86-2*, Department of operations Research, Stanford University, California.

Gill, P. E., Murray, W., Saunders, M. A. & Wright, M. H. (1990). A schur-complement method for sparse quadratic programming, *in* M. G. Cox & S. J. Hammarling (eds), *Reliable Mathematical Computation*, Oxford University Press, pp. 113–138.

Gill, P. E., Murray, W. & Wright, M. H. (1981). *Practical Optimization*, Academic Press Limited, London, UK.

Goh, C. J. & Teo, K. L. (1988). Control parameterization: a unified approach to optimal control problems with general constraints, *Automatica* **24**(1): 3–18.

Goldfarb, D. & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programming, *Mathematical Programming* **27**: 1–33.

Golub, G. H. & van Loan, C. F. (1989). *Matrix Computations*, 2 edn, The John Hopkins University Press, Baltimore and London.

Hairer, E., Nörsett, S. P. & Wanner, G. (1987a). *Solving Ordinary Differential Equations 1. Nonstiff Problems*, Springer-Verlag, Berlin, Heidelberg.

Hairer, E., Nörsett, S. P. & Wanner, G. (1987b). *Solving Ordinary Differential Equations 2. Stiff Problems*, Springer-Verlag, Berlin, Heidelberg.

Han, S. P. (1976). Superlinearly convergent variable metric algorithms for general nonlinear programming problems, *Mathematical Programming* **11**: 263–282.

Han, S. P. (1977). A globally convergent method for nonlinear programming, *JOTA* **22**(3): 297–309.

Hargraves, C. R. & Paris, S. W. (1987). Direct trajectory optimization using nonlinear programming and collocation, *J. Guidance and Control* **10**(4): 338–342.

Hestenes, M. R. (1979). Historical overview of generalized lagrangians and augmentability, *Presented at the IIASA Task Force Meeting on Generelized Lagrangians in Systems and Economic Theory*, IIASA, Laxenberg, Austria.

Hicks, G. A. & Ray, W. H. (1971). Approximation methods for optimal control synthesis, *The Canadian Journal of Chemical Engineering* **49**: 522–528.

Horn, M. K. (1990). Solution of the optimal control problem using the software package stomp, *in* H. Siguerdidjane & P. Bernhard (eds), *Control Applications of Nonlinear Programming and Optimization 1989*, Pergamon Press, Paris, France, pp. 51–56. (IFAC Workshop Series, Number 2).

Hsieh, C. C. & Arora, J. S. (1984). Design sensitivity analysis and optimization of dynamic response, *Computer Methods in Applied Mechanics and Engineering* **43**: 195–219.

Hurwicz, L. (1958). Programming in linear spaces, *in* K. J. Arrow, L. Hurwicz & H. Uzawa (eds), *Studies in Linear and Nonlinear Programming*, University Press, Stanford, Cal., pp. 38–102.

Jänsch, C. & Schnepper, K. (1991). Optimization methods, *CCG-Course DR 4.04*, Carl-Cranz-Gesellschaft, Oberpfaffenhofen, Germany. (K. H. Well (ed): Trajectory Optimization and Guidance of Aerospace Vehicles).

Jänsch, C., Schnepper, K. & Well, K. H. (1989). Ascent and descent trajectory optimization for ariane v/hermes, *AGARD Preprints, 75th Symposium of AGARD Flight Mechanics Panel on Space Vehicle Flight Mechanics*, Luxumburg.

Jänsch, C., Schnepper, K. & Well, K. H. (1994). Multiphase trajectory optimization methods with applications to hypersonic vehicles, *in* A. Miele & A. Salvetti (eds), *Applied Mathematics in Aerospace Science and Engineering*, Plenum Publishing Corporation, New York. To appear.

Kraft, D. (1980). Comparing mathematical programming algorithm based on lagrangian functions for solving optimal control problems, *in* H. E. Rauch (ed.), *Control Applications of Nonlinear Programming*, Pergamon Press, New York, pp. 71–84.

Kraft, D. (1981). Finite-difference gradients versus error-quadrature gradients in the solution of parameterized optimal control problems, *Optimal Control Applications and Methods* **2**: 191–199.

Kraft, D. (1985). On converting optimal control problems into nonlinear programming problems, *in* K. Schittkowski (ed.), *Computational Mathematical Programming*, Springer-Verlag, Berlin, pp. 261–280. (NATO ASI Series. Series F: Computer and System Science, Vol. 15).

Kraft, D. (1988). A sofware package for sequential quadratic programming, *Forschungsbericht*, DFVLR, Oberpfaffenhofen, Germany.

Kraft, D. (1989). Tomp - fortran modules for optimal control calculations, *Lehrgang r1.06: Optimierungsverfahren - software und praktische anwendungen*, Carl-Cranz-Gesellschaft, Oberpfaffenhofen, Germany.

Kreyszig, E. (1978). *Introductory Functional Analysis with Applications*, John Wiley and Sons, New York.

Kuhn, H. W. & Tucker, A. W. (1951). Nonlinear programming, *in* J. Neyman (ed.), *Proceedings of the Second Berkley Symposium on Mathematical Statistics and Probability*, University of California Press.

Landiech, P. & Aumasson, C. (1986). Optimal trajectory of ariane v launcher with first stage fallout constraint, *Proceedings of the 37th Congress of the International Astronautical Federation*, Insbruck, Austria, IAF-86-230.

Lasdon, L. S., Mitter, S. K. & Warren, A. D. (1967). The conjugate gradient method for optimal control problem, *IEEE Trans. Aut. Control* **12**(2): 132–138.

Luenberger, D. G. (1969). *Optimization by Vector Space Methods*, John Wiley and Sons, New York.

Luenberger, D. G. (1984). *Linear and nonlinear programming*, 2 edn, Addison-Wesley, Reading, Massachusetts.

Lunde, E. (1991). *Learning Optimal Control for Robot Manipulators*, PhD thesis, University of Trondheim, NTH, Division of Eng. Cyb., Trondheim, Norway. (ITK-rapport 1991:17-W).

Maratos, N. (1978). *Exact Penalty Function Algorithms for Finite Dimensional and Control Optimization Problems*, PhD thesis, Imperial College Sci. Tech., University of London, London, UK.

Miele, A. (1962). *Flight Mechanics 1, Theory of Flight Paths*, Addison-Wesley, Reading, Mass.

Miele, A. (1975). Recent advances in gradient algorithms for optimal control problems, *J. Optimization Theory and Applications* **17**(5/6). Dedicated to Prof. A. Busemann.

Murtagh, B. A. & Saunders, M. A. (1993). Minos 5.4 user's guide, *Technical Report SOL 93-20R*, Department of operations Research, Stanford University, California.

NAG (1991). *NAG Fortran Library Manual.* Mark 15 Volume 4 E04 - Minimizing or Maximizing a Function.

Neuman, C. P. & Sen, A. (1973). A suboptimal control algorithm for constrained problems using cubic splines, *Automatica* **9**: 601–613.

Newman, J. N. (1977). *Marine hydrodynamics*, MIT Press, Massachusetts.

Nocedale, J. & Overton, M. (1983). Projected hessian updating algorithms for nonlinearly constrained optimization, *Report 95*, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York.

Paus, M. (1992). A general approach to real-time guidance of dynamic systems based on nonlinear programming, *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Hilton Head Island, SC.

Polak, E. (1971). *Computational Methods in Optimization*, Academic Press, New York.

Powell, M. J. D. (1978a). Algorithms for nonlinear constraints that use lagrange functions, *Mathematical Programming* **14**: 224–248.

Powell, M. J. D. (1978b). The convergence of variable metric methods for nonlinearly constrained optimization calculations, *in* O. L. Mangasarian, R. R. Meyer & S. M. Robinson (eds), *Nonlinear Programming 3*, Academic Press, pp. 27–63. Presented at Nonlinear Programming Symposium 3, Madison, Wisconsin, 1977.

Powell, M. J. D. (1978c). A fast algorithm for nonlinearly constrained optimization calculations, *in* G. A. Matson (ed.), *Numerical Analysis: Lecture Notes in Mathematics Vol. 630*, Springer, New York, pp. 144–157.

Powell, M. J. D. (1983). Zqpcvx a fortran subroutine for convex quadratic programming, *Report DAMTP 83/NA17*, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England.

Robinson, S. M. (1972). A quadratically convergent algorithm for general nonlinear programming problems, *Mathematical Programming* **3**: 145–156.

Rockafeller, R. T. (1970). New applications of duality in convex programming, *Presented at 7th International Symposium on Mathematical Programming*, The Hague.

Rockafeller, R. T. (1973). A dual approach to solving nonlinear programming problems by unconstrained optimization, *Mathematical Programming* **5**: 354–373.

Rosen, J. B. & Kreuser, J. (1972). A gradient projection algorithm for nonlinear constraints, *in* F. A. Lootsma (ed.), *Numerical Methods for Nonlinear Optimization*, Academic Press, London, pp. 297–300.

Rosenbrock, H. H. & Storey, C. (1966). *Computational Techniques for Chemical Engineers*, Pergamon, Oxford, UK.

Sagatun, S. I. (1992). *Modeling and Control of Underwater Vehicles*, PhD thesis, University of Trondheim, NTH, Division of Eng. Cyb., Trondheim, Norway. (ITK-rapport 1992:28-W).

Sargent, R. W. H. (1974). Reduced-gradient and projection methods for nonlinear programming, *in* P. E. Gill & W. Murray (eds), *Numerical Methods for Constrained Optimization*, Academic Press, London, pp. 149–174.

Sargent, R. W. H. & Sullivan, G. R. (1978). The development of an efficient optimal control package, *in* J. Stoer (ed.), *Optimization Techniques, Proc. of the 8th IFIP Conference on Optimization Techniques*, Springer-Verlag, Berlin.

Schittkowski, K. (1981). The nonlinear programming method of wilson, han and powell with an augmented lagrangian type line search function. part 1: Convergence analysis, *Numer. Math.* **38**: 83–114.

Schittkowski, K. (1983). On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function, *Math. Operationsforschung u. Statistik, Ser. Optimization* **14**: 197–216.

Schittkowski, K. (1985). Nlpql: A fortran subroutine solving constrained nonlinear programming problems, *Operations Research Annals* **5**: 485–500.

Sirisena, H. R. (1973). Computation of optimal controls using a piecewise polynomial parameterization, *IEEE Trans. Aut. Control,* **18**: 409–411.

Sirisena, H. R. & Tan, K. S. (1974). Computation of constrained optimal controls using parameterization techniques, *IEEE Trans. Aut. Control,* **19**: 431–433.

Sørensen, A. (1993). *Modeling and Control of SES Dynamics in the Vertical Plane*, PhD thesis, University of Trondheim, NTH, Division of Eng. Cyb., Trondheim, Norway. (ITK-rapport 1993:7-W).

Spangelo, I. & Egeland, O. (1992a). Computing energy-optimal trajectories for an autonomous underwater vehicle using direct shooting, *Mod., Id., Contr.* **13**(3): 163–174.

Spangelo, I. & Egeland, O. (1992b). Generation of energy-optimal trajectories for an autonomous underwater vehicle, *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2107–2112.

Spangelo, I. & Egeland, O. (1993). Thruster modelling and energy-optimization for an autonomous underwater vehicle, *Proceedings of the IFAC World Congress*, Sydney, Australia.

Spangelo, I. & Egeland, O. (1994). Trajectory planning and collision avoidance for underwater vehicles using optimal control, *IEEE J. Oceanic Engineering* **19**(4).

Spangelo, I., Sagli, J. R. & Egeland, O. (1993). Bounds on the largest singular value of the manipulator jacobian, *IEEE Transactions on Robotics and Automation* **9**(1): 93–96.

Spangelo, I. & Well, K. H. (1994). Rocket ascent with heat-flux and splash-down constraints, *Proceedings of the 13th IFAC Symposium on Automatic Control in Aerospace*, Palo-Alto, CA.

Speyer, J. L., Kelley, H. J., Levine, H. J. & Denham, W. F. (1971). Accelerated gradient projection technique with application to rocket trajectory optimization, *Automatica* **7**: 37–43.

Stoer, J. & Bulirsch, R. (1983). *The Multiple Shooting Method*, Springer Verlag, New York, Berlin, Heidelberg, pp. 483–498. In: Introduction to Numerical Analysis.

Strand, S. (1991). *Dynamic Optimization in State Space Predictive Control Schemes*, PhD thesis, University of Trondheim, NTH, Division of Eng. Cyb., Trondheim, Norway. (ITK-rapport 1991:11-W).

Tabak, D. & Kuo, B. C. (1971). *Optimal Control by Mathematical Programming*, Prentice-Hall, Englewood Cliffs, N.J.

Teo, K. L., Goh, C. J. & Wong, K. H. (1991). *A Unified Computational Approach to Optimal Control Problems*, Longman Scientific and Technical, London, U.K.

Todd, F. H. & Taylor, D. (1967). Resistance and propulsion, *in* J. P. Comstock (ed.), *Principles of Naval Architecture (revised)*, The Society of Naval Architects and Marine Eng., New York, chapter 7, pp. 288–462.

Tone, K. (1983). Revisons of constraint approximations in the successive qp method for nonlinear programming, *Mathematical Programming* **26**: 144–152.

Tsang, T. H., Himmelblau, D. M. & Edgar, T. F. (1975). Optimal control via collocation and nonlinear programming, *International Journal of Control* **21**(5): 763–768.

Tseng, C. H. & Arora, J. S. (1989). Optimum design of systems for dynamics and controls using sequential quadratic programming, *AIAA J.* **27**: 1793–1800.

van Lammeren, W. P. A., van Manen, J. D. & Oosterveld, M. W. C. (1969). The wagening b-sqrew series, *Trans. SNAME* pp. 269–317. (no. 330 of the NSMB, presented at annual meeting).

Well, K. H. (1990). Ariane v ascent trajectory optimization with a first-stage splash-down constraint, *in* H. Siguerdidjane & P. Bernhard (eds), *Control Applications of Nonlinear Programming and Optimization 1989*, Pergamon Press, Paris, France, pp. 27–32. (IFAC Workshop Series, Number 2).

Well, K. H. (1991). Modelling, *CCG-Course DR 4.04*, Carl-Cranz-Gesellschaft, Oberpfaffenhofen, Germany. (K. H. Well (ed): Trajectory Optimization and Guidance of Aerospace Vehicles).

Wilson, R. B. (1963). *A Simplical Method for Convex Programming*, PhD thesis, Harvard University, Cambridge, MA.

Appendix A

Numerical Results for the State Sensitivity Approach

The state sensitivity approach for analytical gradient computations in CVP with shooting has been tested on a model of a small underwater vehicle operating in the horizontal plane. The vehicle is of the type presented in Chapter 3. Good results have been achieved.

The state space equations were written

$$\begin{aligned}
\dot{x}_1 &= \cos(x_3) \cdot x_4 \\
\dot{x}_2 &= \sin(x_3) \cdot x_4 \\
\dot{x}_3 &= x_5 \\
\dot{x}_4 &= -0.37x_4 - 0.56|x_4|x_4 + u_1 + u_2 \\
\dot{x}_5 &= -0.23x_5 - 0.35|x_5|x_5 + 0.4(-u_1 + u_2)
\end{aligned} \tag{A.1}$$

where x_1 and x_2 were the global x and y position coordinates respectively, x_3 the yaw angle, x_4 the velocity in vehicle x direction (surge) and x_5 the angular velocity about the z-axis (yaw velocity). No motion along the vehicle y-axis was assumed. This results in a non-holonomic dynamical model.

The following optimal control problem was defined:

$$\begin{aligned}
\min_{\mathbf{u}} J_0 &= \int_0^{15} (u_1^2 + u_2^2) dt \\
\mathbf{x}(0) = \mathbf{0}, \quad \mathbf{x}(15) &= \begin{pmatrix} 10 & 10 & 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}} \\
|u_i(t)| &\leq 1.0 \quad i = 1, 2
\end{aligned} \tag{A.2}$$

Collision avoidance was included by a path constraint defined by a circle of radius 2 m centered at $x = y = 5$ m:

$$g_I(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 5)^2 - 4 \geq 0 \tag{A.3}$$

The time grid approximation approach was chosen to represent the path constraint in the NLP, and the time-grid was chosen to be $\begin{pmatrix} 6.5 & 7.0 & 7.5 & 8.0 & 8.5 & 9.0 \end{pmatrix}$.

The controls were approximated by piecewise linear polynomials where the switching times were constant, equal for both controls and equidistantly distributed. The box constraints on the parameters are now the same as those for the controls (Eq. A.2).

The resulting nonlinear program was solved by the program package E04VCF from the NAG fortran routine library. The desired accuracy in the optimal cost function was set to 10^{-3} and the tolerances for all the constraints were set to 10^{-2}. Integrations were performed by a Runge-Kutta-Fehlberg 4/5th order variable step size method, with a local relative accuracy of $5 \cdot 10^{-5}$.

Let parameter p_{ik} correspond to control u_k at switching time t_i, then

$$\frac{\partial u_k}{\partial p_{ik}} = \begin{cases} \frac{t - t_{i-1}}{t_i - t_{i-1}}; & t_{i-1} \leq t < t_i \\ 1 - \frac{t - t_i}{t_{i+1} - t_i}; & t_i \leq t < t_{i+1} \\ 0 & t < t_{i-1}, \;\; t > t_{i+1} \end{cases} \tag{A.4}$$

The nonautonomous linear equations (Eq. 2.47) corresponding to the model in Eq. (A.1) are

$$\begin{aligned} \dot{y}_1 &= -\sin(x_3) \cdot x_4 \cdot y_3 + \cos(x_3) \cdot y_4 \\ \dot{y}_2 &= \cos(x_3) \cdot x_4 \cdot y_3 + \sin(x_3) \cdot y_4 \\ \dot{y}_3 &= y_5 \\ \dot{y}_4 &= -(0.37 + 1.12|x_4|) \cdot y_4 + \frac{\partial u_1}{\partial p_j} + \frac{\partial u_2}{\partial p_j} \\ \dot{y}_5 &= -(0.23 + 0.7|x_5|) \cdot y_5 - 0.4(\frac{\partial u_1}{\partial p_j} - \frac{\partial u_2}{\partial p_j}) \end{aligned} \tag{A.5}$$

And the partial derivatives of the resulting objective function and the constraints with respect to the parameters are

$$\frac{\partial J}{\partial p_{ik}} = \int_{t_{i-1}}^{t_{i+1}} (\begin{pmatrix} 2u_1 & 2u_2 \end{pmatrix} \cdot \frac{\partial \mathbf{u}}{\partial p_{ik}}) dt \quad k = 1, 2 \tag{A.6}$$

$$\frac{\partial \mathbf{x}(15)}{\partial p_j} = \mathbf{y}_j(t_f) \tag{A.7}$$

and

# par. per u	J^*	# iter	# fgc	tot. CPU
5	10.86	7	12	1.1
6	10.54	7	13	1.4
7	10.43	6	11	1.4
8	10.38	8	13	2.0

Table A.1:

$$\frac{\partial g_{I,l}}{\partial p_j} = \begin{pmatrix} 2(x_1-5) & 2(x_2-5) & 0 & 0 & 0 \end{pmatrix} \cdot \mathbf{y}_j(t_l) \tag{A.8}$$

for the path constraint at grid time number l.

Starting the optimization with all the initial parameters set to zero the optimization failed after the first iteration. Therefore the model (Eq. A.1) was first simulated once with a simple feedback driving it in the direction of the desired final position. The resulting controls computed from the feedback were used to generate the initial parameter guess. The initializer chooses one side of the circular path constraint. There is of course no guarantee that a more optimal solution on the other side does not exist.

Numerical results for optimization runs with different numbers of parameters are shown in Table A. The table shows respectively the number of parameters per control (# par. per u), optimal objective function, (J^*), number of iterations (# iter), number of function and gradient calculations (# fgc) which is always greater or equal to the number of iterations, and a crude estimate of the total cpu-time consumption (tot. CPU). $t_3 = 7.5$ s was the active time node in the path constraint time grid in all cases. The computations were performed on a Sparc 10 work station.