Analyse und Restrukturierung eines Verfahrens zur direkten Lösung von Optimal-Steuerungsproblemen

(The Theory of MUSCOD in a Nutshell)

Daniel B. Leineweber

Diplomarbeit Mathematik

Betreuer: Prof. Dr. Hans-Georg Bock
Dr. Johannes Schlöder

Interdisziplinäres Zentrum
für Wissenschaftliches Rechnen (IWR)
Universität Heidelberg
1995

The Theory of MUSCOD in a Nutshell

Daniel B. Leineweber*

April 27, 1995

Abstract

MUSCOD (*MU*ltiple *S*hooting *CO*de for *D*irect Optimal Control) is the implementation of an algorithm for the *direct* solution of optimal control problems. The method is based on multiple shooting combined with a sequential quadratic programming (SQP) technique; its original version was developed in the early 1980s by Plitt under the supervision of Bock [Plitt81, Bock84]. The following report is intended to describe the basic aspects of the underlying theory in a concise but readable form. Such a description is not yet available: the paper by Bock and Plitt [Bock84] gives a good overview of the method, but it leaves out too many important details to be a complete reference, while the diploma thesis by Plitt [Plitt81], on the other hand, presents a fairly complete description, but is rather difficult to read. Throughout the present document, emphasis is given to a clear presentation of the concepts upon which MUSCOD is based. An effort has been made to properly reflect the structure of the algorithm and to give a good motivation for its fundamental parts. Of course, recent developments have been taken into account, and it should be noted that some of the material presented here has not been previously published. Proofs which are readily found elsewhere have not been included. It is hoped that this tutorial description will help new users of MUSCOD to quickly become familiar with the necessary theory, and perhaps it will also be useful for people who want to modify or extend the existing algorithm.

*Graduiertenkolleg "Modellierung und Wissenschaftliches Rechnen in Mathematik und Naturwissenschaften", Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Universität Heidelberg, Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany.

Keep it simple:
as simple as possible,
but no simpler.
— Albert Einstein

Contents

Introduction

The optimization of dynamic processes frequently requires the solution of an *optimal control problem*. That is, given a process model in terms of differential equations for the state variables, find the vector of control functions that minimizes some performance index (e.g. a functional of the state and control profiles), subject to boundary conditions and possibly additional equality or inequality constraints on the states and controls. Provided certain regularity conditions are satisfied, this infinite-dimensional problem can—at least in principle—be solved exactly using the *indirect* approach (calculus of variations, maximum principle): the controls are expressed in terms of state and adjoint variables, and then they can be found by solving a boundary value problem; for details see the standard text [Brys76].

In simple cases, as for linear-quadratic problems, this procedure is relatively straightforward, and consequently, the indirect approach has been widely applied in control engineering and related fields, where linear dynamic models have to be optimized with respect to quadratic performance criteria [Föll88]. In general, however, the numerical solution of the resulting boundary value problem tends to be very complicated. Although a number of reliable and efficient algorithms have been developed based on the multiple shooting technique (e.g. [Bock78, Bock81a]), there still remains the problem of finding appropriate initial values for the adjoint variables, since they do not have a direct physical interpretation.

The *direct* approach avoids these difficulties by solving the problem directly in terms of the control and state variables. To this end, the control functions are *parameterized*, for instance through a finite expansion using some set of suitably chosen basis functions. Most often, *piecewise* representations of the control functions are used. In the simplest case, one might try to represent each control by a piecewise constant function. The original continuous problem is thus reformulated as a finite optimization problem, and the optimal values of the control parameters can now be determined, e.g., by nonlinear programming (NLP) techniques. It is important to note that only a *suboptimal* solution to the original problem will be obtained if the exact optimal controls cannot be represented by the chosen parameterization. In practice, judgement and experience are sometimes required for finding a "good" initial parameterization that allows the exact solution to be approximated reasonably well, and can then be iteratively refined, until a satisfactory approximation is obtained. An early and quite successful implementation of a direct method has been described in [Sarg78].

In MUSCOD, the control parameterization is coupled with a *multiple shooting discretization* of the state differential equations, leading to greatly enhanced stability and efficiency compared to simple shooting. The resulting large, but structured, NLP problem is solved by a specifically tailored *sequential quadratic programming (SQP)* algorithm which uses a high rank update formula in order to preserve the block diagonal structure of the Hessian matrix and thereby significantly improve the convergence behaviour. It also makes use of an efficient method for the solution of the quadratic programming (QP) subproblems by taking advantage of the special problem structure. Observe that discretizing the state differential equations by multiple shooting and using an infeasible path optimization method allows the minimization of the objective and the solution of the dynamic model to proceed *simultaneously*, i.e., the original state differential equations are satisfied only at the final solution. At the same time, it is possible to use existing advanced ODE or DAE solvers for the integration of the differential equations. Compared to previous direct approaches (especially the "black-box" methods based on simple shooting), there is a substantial improvement of performance [Bock84, Plitt81].

Alternatively, *collocation* has been applied in order to discretize the state differential equations. This avoids integrating the differential equations repeatedly (as usually required for simple and multiple shooting). However, it considerably increases the size of the NLP problem, especially for stiff systems. Adaptive collocation schemes must be used to control the discretization error. Direct methods based on collocation have been developed by Biegler and some of his coworkers [Bieg84, Cuth87, Logs89, Logs92, Logs93], by Renfro, Morshedi, and Asbjornsen [Renf87], and by Schulz [Schu95]. Both multiple shooting and collocation have distinct advantages—neither of them can be said to be generally superior.

In Section 1, we discuss the class of optimal control problems that can be solved with MUSCOD, the specific way in which these problems are reformulated in discretized form, and some issues related to problem scaling. Section 2 briefly reviews the optimality conditions for the general NLP problem. These conditions will be used repeatedly in what follows (especially in Sections 3 and 6).

A first outline of the solution process by SQP is presented in Section 3, which motivates the basic steps of the algorithm and introduces the QP subproblem in its specific form. This section also discusses the line search procedure as it is implemented in MUSCOD. Although much remains to be

refined in later sections, the major components of the solution method will hopefully already be clear by this point.

Section 4 shows how the required gradients can be computed both efficiently and reliably by internal numerical differentiation (IND). These are very important considerations, since in many realistic problems the main computational burden is due to gradient and function evaluations (and quite often numerical problems can be traced back to poor gradient estimates).

The approximation of the Hessian matrix of the Lagrangian function using variable metric updates is absolutely central to every SQP technique. Section 5 explains the strategies employed by MUSCOD to find a suitable Hessian approximation. The special structure of the optimization problem is reflected in a known, block diagonal structure of the Hessian matrix. (Basically, this property is induced by the specific way in which the piecewise control parameterization and the multiple shooting state discretization are coupled in MUSCOD.) Standard update procedures for the Hessian would destroy this sparse structure. Therefore, MUSCOD uses a partitioned update formula which does not produce any fill-in outside the block diagonal, while at the same time leading to a considerably improved asymptotic convergence rate.

In Section 6, the solution of the QP subproblem is worked out in detail, and it turns out that the special structure of the quadratic program can be exploited by a condensing algorithm, effectively reducing the problem to the same size as for simple shooting.

In Section 7, the "building blocks" of MUSCOD are discussed from an implementational point of view, and an overview of the complete algorithm is given. Also, a number of extensions to the basic solution method are addressed. Finally, the performance of the existing algorithm is summarized, and some ideas on the future directions for the development of MUSCOD are presented (with emphasis on applications in chemical engineering).

Acknowledgements. I wish to thank my advisors Prof. Hans-Georg Bock and Dr. Johannes Schlöder for many inspiring discussions. I would also like to thank Ray Spiteri and Dr. Philipp Rosenau for carefully reading the manuscript and suggesting many improvements. The Interdisciplinary Center for Scientific Computing at the University of Heidelberg (*Interdisziplıäres Zentrum für Wissenschaftliches Rechnen, IWR*) has provided a great environment for doing research, especially through its graduate program "Modellierung und Wissenschaftliches Rechnen in Mathematik und Naturwissenschaften". Financial support by the *Deutsche Forschungsgemeinschaft (DFG)* is gratefully acknowledged.

1 The Optimal Control Problem

1.1 Continuous Form

Many optimal control problems of practical importance—for example, the optimization of dynamic processes in chemical engineering [Ray81, Cuth89, Bock95], or the optimal path planning for robots [Schu94, Stei95a]—can be expressed in the common form,

$$\min_{x,u,p} \; J[x,u,p] = \phi(t_b, x(t_b), p) + \int_{t_a}^{t_b} \Phi(t, x(t), u(t), p)\, dt \tag{1.1a}$$

subject to

$$\begin{aligned} A(t,x(t),u(t),p)\;\dot{x}(t) &= f(t,x(t),u(t),p), \quad t \in [t_a,t_b] && \text{(1.1b)}\\ r(x(t_a),x(t_1),\ldots,x(t_b),p) &= 0 && \text{(1.1c)}\\ g(t,x(t),u(t),p) &\geq 0, \quad t \in [t_a,t_b]. && \text{(1.1d)} \end{aligned}$$

The performance index J in (1.1a) may depend on the state profiles x, the control profiles u, and a finite-dimensional parameter vector p. Often a function ϕ of the final states $x(t_b)$ is included in the minimization. While the initial time t_a is usually assumed to be fixed, the end time t_b may be either fixed or free. The dynamic model is given in terms of a quasilinear-implicit nonlinear system of differential equations as shown in (1.1b), where the matrix A may be singular. In practice, this system is frequently written in the (equivalent) form of *differential-algebraic equations (DAEs)*

$$\begin{aligned} A^d(t,x^d,x^a,u,p)\;\dot{x}^d &= f^d(t,x^d,x^a,u,p), \quad A^d \text{ nonsingular} && \text{(1.2a)}\\ 0 &= f^a(t,x^d,x^a,u,p), && \text{(1.2b)} \end{aligned}$$

where a distinction is made between "differential" states x^d and "algebraic" states x^a, the latter being determined implicitly through the algebraic equations (1.2b). The boundary conditions as well as additional interior point constraints at fixed intermediate times t_j ($t_a < t_j < t_b$) are contained in equation (1.1c), which can be often written in separated form,

$$r_a(x(t_a),p) + r_1(x(t_1),p) + \ldots + r_b(x(t_b),p) = 0, \tag{1.3}$$

i.e., without nonlinear coupling of the states at different points of time. The inequality constraints (1.1d) in many cases just represent (possibly time-dependent) bounds on the state and control profiles,

$$\begin{aligned} \underline{x}(t) &\leq x(t) \leq \overline{x}(t)\\ \underline{u}(t) &\leq u(t) \leq \overline{u}(t). \end{aligned}$$

In order to keep the presentation simple, we will not use the general problem formulation (1.1) as our starting point, but temporarily restrict ourselves to the following specialized class of constrained optimal control problems [Bock84], which will be referred to as (P1),

$$\min_{x,u} \quad J[x,u] = \int_{t_a}^{t_b} \Phi(t, x(t), u(t))\, dt \tag{1.4a}$$

subject to

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u(t)), \quad t \in [t_a, t_b] && \text{(1.4b)} \\ r_a(x(t_a)) + r_b(x(t_b)) &= 0 && \text{(1.4c)} \\ g(t, u(t)) &\geq 0, \quad t \in [t_a, t_b], && \text{(1.4d)} \end{aligned}$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^k$ represent the state and control vectors, respectively, and the time interval $I = [t_a, t_b]$ is assumed to be fixed. The functions $\Phi : I \times \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$, $g : I \times \mathbb{R}^k \to \mathbb{R}^l$, and $r_a, r_b : \mathbb{R}^n \to \mathbb{R}^{l_r}$ shall be piecewise continuous and continuously differentiable, and for the function $f : I \times \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^n$ the usual assumptions shall hold that ensure the local existence and uniqueness of a solution x to the state differential equations for given initial conditions $x(t_a) = x_0$ and controls u (e.g., f must be continuous and continuously differentiable with respect to x). If the boundary equality conditions (1.4c) are replaced by the inequalities

$$r_a(x(t_a)) + r_b(x(t_b)) \;\geq\; 0, \tag{1.5}$$

only minor changes are necessary in the way the QP subproblem is assembled, so this modified problem can be handled as well.

Problem (P1) differs from the original problem given by (1.1) in several respects: the performance index J in (1.4a) no longer includes the function ϕ, the DAEs have been replaced by ordinary differential equations (ODEs), instead of general interior point constraints (1.1c) there are only (separated) two-point boundary conditions (1.4c), the inequality constraints (1.4d) do no longer depend on the states x, and there are no free parameters p. However, in spite of these simplifications, the given problem formulation (P1) is still fairly general. Quite a number of features of the more general problem (1.1) can easily be represented in the restricted form (P1) using suitable reformulations and can therefore be handled by a method capable of solving (P1); other features may be included by straightforward extensions to the basic solution method.

For instance, if the problem has free end time t_b, we introduce an additional state variable $x_{n+1} = t_b$ and use the transformation

$$\theta(\tau) = t_a + \tau(x_{n+1} - t_a), \quad \tau \in [0, 1]$$

to obtain the new system of $n+1$ differential equations defined on $[0, 1]$,

$$\begin{aligned} dx/d\tau &= f(\,.\,, x, u)\,(x_{n+1} - t_a) \\ dx_{n+1}/d\tau &= 0, \end{aligned}$$

where the argument t has been replaced by $\theta(\tau)$. The performance index (1.4a), the boundary conditions (1.4c) or (1.5), and the control inequalities (1.4d) are transformed similarly, resulting in a problem of form (P1), which is defined on the constant time interval $[0, 1]$. If the problem depends on a vector of free parameters $p \in \mathbb{R}^{n_p}$, these parameters can be treated simply as additional constant states $x_{n+i} = p_i$ with $\dot{x}_{n+i} = 0$ for $i = 1, \ldots, n_p$.[‡] Finally, if the performance index J is not of Langrange type as in (1.4a) but rather of Mayer type

$$J = \phi(t_b, x(t_b))$$

or a combination of these two forms (Bolza type) as in (1.1a), it is possible to derive an equivalent Lagrange form.

In Section 7, we will return to the general problem formulation and discuss the extensions necessary to include *separated* interior point constraints[§] (as equalities or inequalities) and DAE models.

1.2 Problem Reformulation

1.2.1 Control Parameterization

As mentioned in the introduction, the continuous problem (P1) has to be replaced by a discretized one, where the control functions (corresponding to "infinitely many" parameters) are approximated by a suitable parameterization using only a finite set of parameters, whose optimal values can then be found by NLP techniques. In principle, one could think of a global representation of the controls on the whole interval $[t_a, t_b]$, but this approach is

[‡] Although conceptually simple, this strategy is not very practical, because it leads to an unnecessary integration overhead. A solution method which explicitly supports free parameters will rather treat them as special, "constant" controls.

[§] Nonlinearly coupled interior point constraints as in (1.1c) cannot be handled directly, because they would destroy the block diagonal structure of the Hessian.

almost never used in practice because of its restricted flexibility (one would have to know too much about how the solution should look) and because it makes it very difficult to deal with discontinuities (which frequently occur in the exact optimal solution—think of "bang-bang" control). In later sections, it will be shown that it is also advantageous from a numerical point of view to have a *partially separable* Lagrangian function, which can be written as a sum of simpler functions, each depending only on a distinct set of "local" parameters.

Therefore, a *piecewise representation* of the control functions is sought. For a suitably chosen mesh

$$t_a = t_0 < t_1 < \ldots < t_{m-1} < t_m = t_b, \tag{1.6}$$

the control vector is approximated on every *subinterval* by a finite set of parameters using given basis functions φ_j,

$$u(t) := \varphi_j(t, q_j), \quad q_j \in \mathbb{R}^{k_j}, \quad t \in I_j = [t_j, t_{j+1}] \quad \text{for} \quad j = 0, 1, \ldots, m-1. \tag{1.7}$$

Most often the φ_j are chosen to be vectors of polynomials, resulting in a piecewise polynomial representation of u (note that φ_j is only defined on the subinterval I_j). The two simplest cases are to take $\varphi_j = q_j$, $q_j \in \mathbb{R}^k$ for a piecewise constant approximation, see Figure 1.1, and

$$\varphi_j = q_j^1 + \frac{t - t_j}{t_{j+1} - t_j}\,(q_j^2 - q_j^1), \quad q_j = \begin{pmatrix} q_j^1 \\ q_j^2 \end{pmatrix} \in \mathbb{R}^{2k} \tag{1.8}$$

for a piecewise linear approximation (linear interpolation between the values q_j^1 and q_j^2 at the endpoints of subinterval I_j).

Piecewise representations according to (1.7) will in general not be continuous at the inner mesh points t_j, $j = 1, 2, \ldots, m-1$. This does not pose a problem for the numerical integration of (1.4b), because the location of these discontinuities is known. If it is desired to obtain a continuous approximation, then the general discontinuous form (1.7) should still be used together with the additional continuity conditions,

$$\varphi_j(t_{j+1}, q_j) - \varphi_{j+1}(t_{j+1}, q_{j+1}) = 0, \quad j = 0, 1, \ldots, m-2. \tag{1.9}$$

By using an inherently continuous form instead, neighboring intervals would no longer be decoupled, which would partly destroy the sparse structure of

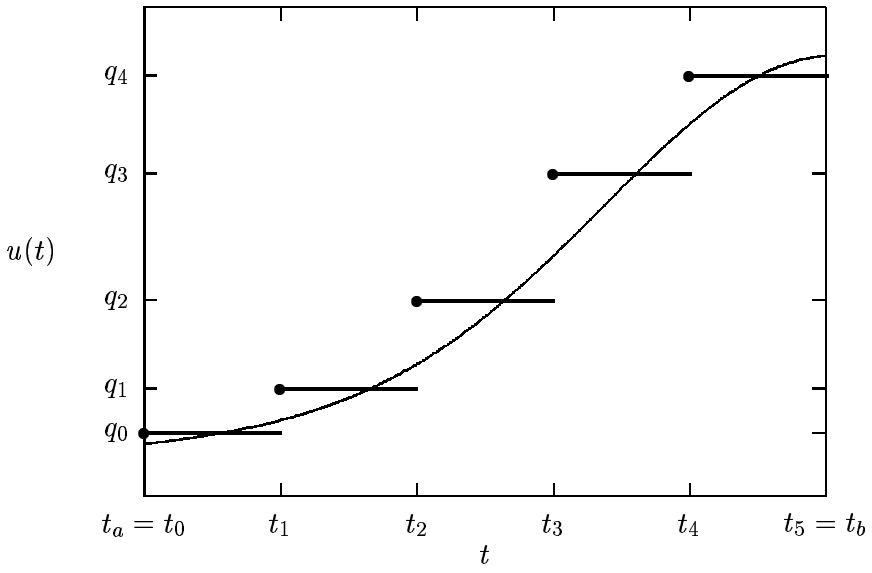

Figure 1.1: Piecewise constant approximation of an optimal control ($m = 5$).

the resulting optimization problem. In case of the piecewise linear approximation (1.8), one has to impose the trivial conditions

$$q_j^2 - q_{j+1}^1 = 0, \quad j = 0, 1, \ldots, m-2 \tag{1.10}$$

for continuity at the inner mesh points.

It should be stressed that the quality of the parameterization is strongly dependent on the number and location of the mesh points in (1.6), and that finding a "good" mesh is generally no trivial task. This can be seen by the example of "bang-bang" control, where the solution of the optimal control problem more or less reduces to finding the optimal switching points (which need not be the same for all controls). In practice, one will often start with a rather coarse mesh and then try successively finer ones, until a satisfactory approximation is obtained.

Finally, the control inequality conditions (1.4d) have to be discretized, leading to

$$g_j(q_j) \geq 0, \quad g_j : \mathbb{R}^{k_j} \to \mathbb{R}^{l_j}, \quad j = 0, 1, \ldots, m-1. \tag{1.11}$$

This will in general only be an approximation to the continuous conditions (1.4d), especially for time dependent g, but in many practical cases (1.11) and (1.4d) are equivalent for the given control parameterization. Consider for instance constant bounds on the vector of control functions,

$$\underline{u} \leq u(t) \leq \overline{u}$$

("box constraints"), which frequently occur in applications. Then the continuous conditions in (P1) have the form

$$g(u(t)) \equiv \begin{pmatrix} u(t) - \underline{u} \\ \overline{u} - u(t) \end{pmatrix} \geq 0, \quad t \in [t_a, t_b],$$

and for the piecewise constant parameterization $\varphi_j = q_j$ one has the equivalent discretized conditions

$$g_j(q_j) \equiv g(q_j) \equiv \begin{pmatrix} q_j - \underline{u} \\ \overline{u} - q_j \end{pmatrix} \geq 0, \quad j = 0, 1, \ldots, m-1.$$

Similarly, for the piecewise linear parameterization defined by (1.8) one obtains

$$g_j(q_j) \equiv \begin{pmatrix} q_j^1 - \underline{u} \\ q_j^2 - \underline{u} \\ \overline{u} - q_j^1 \\ \overline{u} - q_j^2 \end{pmatrix} \geq 0, \quad j = 0, 1, \ldots, m-1,$$

which is again equivalent to the continuous form of the conditions.

This concludes the first step of the problem transformation. In the current form, the problem could already be solved using simple shooting combined with some NLP technique. However, much can be gained by taking another step to *explicitly* discretize the state differential equations.

1.2.2 Multiple Shooting State Discretization

Consider the boundary value problem (BVP) contained in (P1) for some given control parameterization according to (1.7),

$$\dot{x} = f(t, x, u), \quad r_a(x(t_a)) + r_b(x(t_b)) = 0, \tag{1.12}$$

i.e., u is determined by the parameters $q_j \in \mathbb{R}^{k_j}$, $j = 0, 1, \ldots, m-1$ and may be discontinuous at the mesh points t_j. Note that there are l_r boundary conditions ($r_a, r_b : \mathbb{R}^n \to \mathbb{R}^{l_r}$), where unlike the usual two-point boundary value problem, l_r need not be equal to the number of states n. In fact, one has $0 \leq l_r \leq 2n$, where the extreme cases occur if both the initial state $x(t_a)$ and the end point $x(t_b)$ are fixed ($l_r = 2n$), or if they are both left free ($l_r = 0$). In practical applications, $x(t_a)$ very often is fixed, and $x(t_b)$ is either free (then we have $l_r = n$, and (1.12) is reduced to an initial value problem), or it has to be on some given manifold of dimension $d < n$ (thus $l_r = n + d$). These examples show that the boundary conditions in (1.12) are already quite general; in many cases, linear conditions

$$R_a\, x(t_a) + R_b\, x(t_b) = c, \quad R_a, R_b \in \mathbb{R}^{l_r \times n}, \quad c \in \mathbb{R}^{l_r}$$

suffice [Stoer92], and often it is even possible to arrive at a pair of separated conditions like

$$\begin{aligned} \bar{R}_a\, x(t_a) &= c_a, \quad \bar{R}_a \in \mathbb{R}^{l_r^a \times n}, \quad c_a \in \mathbb{R}^{l_r^a} \\ \bar{R}_b\, x(t_b) &= c_b, \quad \bar{R}_b \in \mathbb{R}^{l_r^b \times n}, \quad c_b \in \mathbb{R}^{l_r^b}, \end{aligned}$$

where $l_r = l_r^a + l_r^b$. Therefore, it is no serious limitation that MUSCOD cannot handle general nonlinearly coupled boundary conditions of the form $r(x(t_a), x(t_b)) = 0$ directly. (If desired, such conditions can nevertheless be implemented by introducing additional parameters and constraints.)

In order to solve the boundary value problem (1.12), one has to determine a vector of control parameters

$$q = \begin{pmatrix} q_0 \\ \vdots \\ q_{m-1} \end{pmatrix} \in \mathbb{R}^{\hat{k}}, \quad \hat{k} = \sum_{j=0}^{m-1} k_j,$$

and a starting vector $s \in \mathbb{R}^n$ for the initial value problem (IVP)

$$\dot{x} = f(t, x, u), \quad u = u(t, q), \quad x(t_a) = s \tag{1.13}$$

in such a way that the solution $x(t) = x(t; s, q)$ satisfies the boundary conditions of (1.12),

$$r_a(x(t_a; s, q)) + r_b(x(t_b; s, q)) \equiv r_a(s) + r_b(x(t_b; s, q)) = 0. \tag{1.14}$$

Thus, one has to find a solution[‡] $y = (s, q)$ of the equation

$$h(y) = 0, \quad h(y) := r_a(s) + r_b(x(t_b; s, q)). \tag{1.15}$$

It will be shown in Section 4 that $x(t_b; s, q)$, and hence $h(y)$ are, in general, continuously differentiable functions of s and q. In the context of an "isolated" boundary value problem (without controls and with n boundary conditions) one could therefore use Newton's method to solve (1.15) iteratively for s starting with an initial approximation $s^{(0)}$ [Stoer92]. This is the idea of *simple shooting*, and it requires the repeated integration of the initial value problem (1.13) between t_a and t_b. In the optimization context of problem (P1) a more general approach has to be followed: the equations (1.15) serve as equality constraints, and the parameter vector y is adjusted by the NLP method to satisfy these equalities as well as the control inequalities and at the same time minimize the performance index.

However, this approach of simple shooting plus nonlinear programming frequently fails even when very good initial estimates $y^{(0)}$ are available: the method does not converge, or sometimes it may not even be possible to evaluate $h(y)$ for a given parameter vector y, because $x(t_b; s, q)$ does not exist numerically (the initial value problem cannot be integrated to the end point t_b). The key reason for these difficulties is that (1.13) has to be integrated over the whole interval $[t_a, t_b]$. Thus the error introduced by poor initial data, discretization, or roundoff may be propagated by inherent instabilities of the ODE system and grow very large.

This situation can be considerably improved by using the *multiple shooting method* instead of simple shooting. The central idea is to simultaneously compute the values

$$s_j = x(t_j), \quad j = 0, 1, \ldots, m$$

[‡]Here and in the following we do not use the "proper" but cumbersome notation $y = (s^T, q^T)^T$ for composite column vectors.

of the solution of the BVP (1.12) at several points

$$t_a = t_0 < t_1 < \ldots < t_m = t_b$$

by iteration (and not just the value $s_0 = x(t_0)$). For simplicity of notation, we have assumed here that the same mesh (1.6) as for the control parameterization is used. More generally, the multiple shooting points could be chosen also as a sub- or superset of the control nodes in (1.6).[‡] Let $x(t; s_j, q_j)$ denote the solution of the IVP

$$\dot{x} = f(t, x, \varphi_j(t, q_j)), \quad x(t_j) = s_j, \quad t \in I_j = [t_j, t_{j+1}]. \tag{1.16}$$

The problem now consists in determining the vectors s_j, $j = 0, 1, \ldots, m$, and q_j, $j = 0, 1, \ldots, m-1$ in such a way that the function x pieced together by these IVP solutions,

$$\begin{aligned} x(t) &:= x(t; s_j, q_j) \text{ for } t \in [t_j, t_{j+1}[, \quad j = 0, 1, \ldots, m-1, \\ x(t_m) &:= s_m, \end{aligned}$$

is continuous and hence represents a solution of the differential equation $\dot{x} = f(t, x, u)$ on the whole interval $[t_a, t_b]$, see Figure 1.2. In addition, x must satisfy the boundary conditions $r_a(x(t_a)) + r_b(x(t_b)) = 0$. Defining the augmented parameter vector y by

$$y := (s_0, q_0, \ldots, s_{m-1}, q_{m-1}, s_m) \in \mathbb{R}^{\hat{n}}, \quad \hat{n} = n(m+1) + \sum_{j=0}^{m-1} k_j, \tag{1.17}$$

we therefore obtain the following set of $\hat{m} = nm + l_r$ conditions for the $\hat{n}$ components of y,

$$h(y) := \begin{pmatrix} x(t_1; s_0, q_0) - s_1 \\ x(t_2; s_1, q_1) - s_2 \\ \vdots \\ x(t_m; , s_{m-1}, q_{m-1}) - s_m \\ r_a(s_0) + r_b(s_m) \end{pmatrix} = 0. \tag{1.18}$$

It is important to realize that now the evaluation of $h(y)$ no longer requires integration over the whole interval $[t_a, t_b]$ at once: the IVPs (1.16) only

[‡] In Section 4, we will consider the details of introducing additional control nodes within a given multiple shooting interval $[t_j, t_{j+1}]$. (This may be desirable for the successive refinement of the control mesh.)

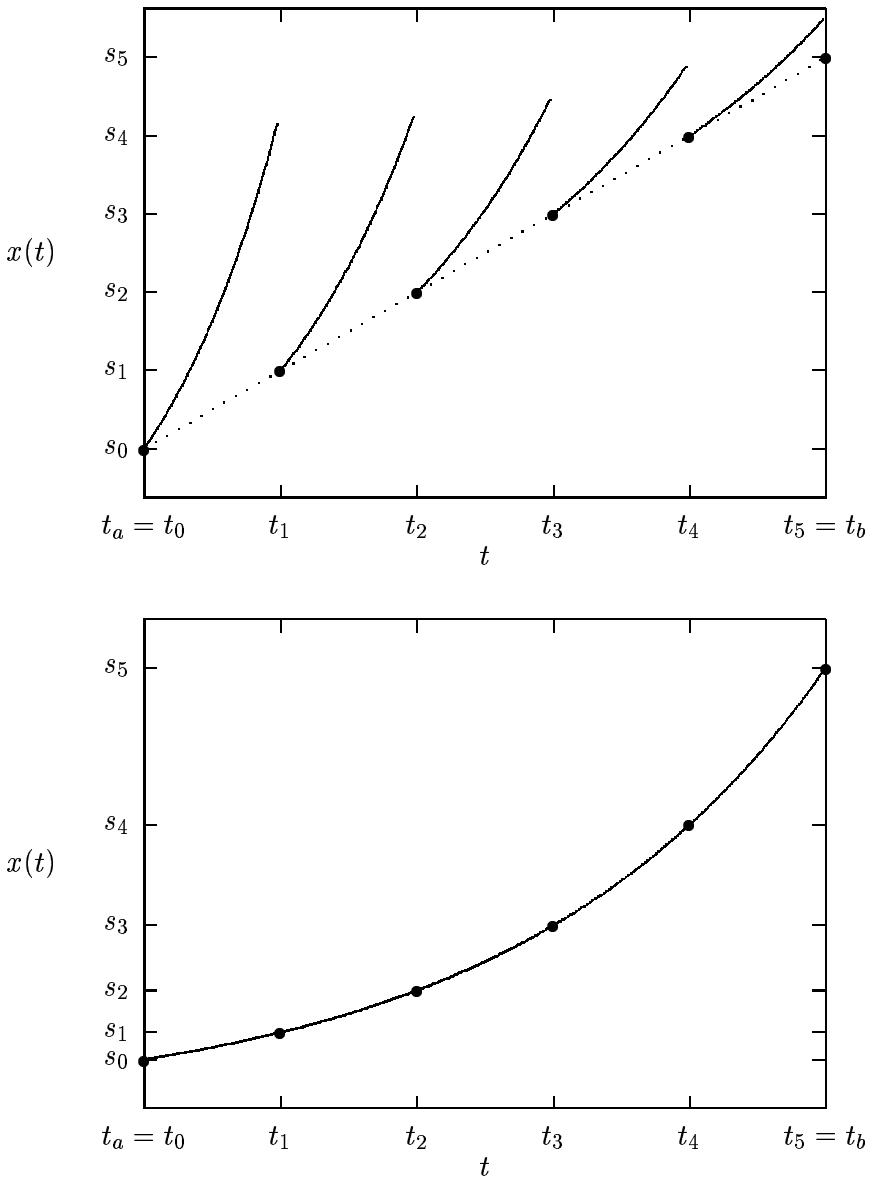

Figure 1.2: Multiple shooting method—initial and final trajectory ($m = 5$). In this example, initial guesses for s_j were obtained by linear interpolation between fixed boundary values.

need to be integrated over the subintervals $I_j = [t_j, t_{j+1}]$, and this very effectively limits the growth of the error in case of instabilities. Finding a solution to problem (P1) requires, of course, not only $h(y) = 0$, but at the same time satisfaction of the inequality constraints and minimization of the performance index. Compared to simple shooting, the multiple shooting method shows greatly improved convergence and numerical stability. For a detailed discussion in the "classical" context of two-point boundary value problems, see e.g. [Stoer92]; for a discussion in an optimization context closely related to direct optimal control (parameter identification in ODEs), see [Bock83, Bock87].

Increasing the size of the optimization problem by introducing the extra parameters s_j, $j = 1, 2, \ldots, m$, may at first seem awkward, but it will be shown that this does not significantly increase the complexity of the problem because the state discretization imposes a very simple, decoupled structure on it. In fact, the condensed QP subproblem actually being solved is of the same size as for simple shooting. Therefore, the additional computational cost of using multiple shooting just amounts to the cost of the condensing algorithm (for first reducing the size of the problem and then recovering the full solution, see Section 6).

1.3 Discretized Form (NLP Problem)

Now the optimal control problem can be restated as a general NLP problem. For a given mesh (1.6) and control parameterization (1.7), we arrive at the following formulation, which will be denoted by (P2),

$$\begin{aligned} \min_y \quad F(y) \;&=\; \sum_{j=0}^{m-1} \int_{t_j}^{t_{j+1}} \Phi(t, x(t; s_j, q_j), \varphi_j(t, q_j))\, dt \\ &=:\; \sum_{j=0}^{m-1} F_j(s_j, q_j) \end{aligned} \tag{1.19a}$$

subject to

$$h_j(s_j, s_{j+1}, q_j) \;:=\; x(t_{j+1}; s_j, q_j) - s_{j+1} \;=\; 0 \tag{1.19b}$$

$$h_m(s_0, s_m) \;:=\; r_a(s_0) + r_b(s_m) \;=\; 0 \tag{1.19c}$$

$$g_j(q_j) \;\geq\; 0, \tag{1.19d}$$

where $j = 0, 1, \ldots, m-1$ in (1.19b) and (1.19d), y is the augmented parameter vector defined in (1.17), and $x(t; s_j, q_j)$ denotes the solution of

the IVP (1.16). If continuity of the control parameterization is desired at the mesh points t_j, $j = 1, 2, \ldots, m-1$, then the additional equality constraints (1.9), namely,

$$\varphi_j(t_{j+1}, q_j) - \varphi_{j+1}(t_{j+1}, q_{j+1}) = 0, \quad j = 0, 1, \ldots, m-2,$$

must be included. The values $F_j(s_j, q_j)$ can be conveniently obtained along with $x(t_{j+1}; s_j, q_j)$ by defining an additional scalar variable

$$z(t) := \int_{t_j}^{t} \Phi(t, x(t; s_j, q_j), \varphi_j(t, q_j))\, dt, \quad t \in I_j = [t_j, t_{j+1}], \tag{1.20}$$

and solving the composite IVP

$$\begin{pmatrix} \dot{z} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} \Phi(t, x, \varphi_j(t, q_j)) \\ f(t, x, \varphi_j(t, q_j)) \end{pmatrix}, \quad \begin{pmatrix} z(t_j) \\ x(t_j) \end{pmatrix} = \begin{pmatrix} 0 \\ s_j \end{pmatrix}, \quad t \in I_j \tag{1.21}$$

instead of (1.16). Then we simply have

$$F_j(s_j, q_j) \equiv z(t_{j+1}; s_j, q_j), \quad j = 0, 1, \ldots, m-1. \tag{1.22}$$

By virtue of being a solution of the IVP (1.21), $F_j(s_j, q_j)$ and $x(t_{j+1}; s_j, q_j)$ are continuously differentiable functions of the initial values s_j and the control parameters q_j (under the usual smoothness assumptions on the right-hand side functions). An integration algorithm and a technique for calculating (approximate) derivatives of $F_j(s_j, q_j)$ and $x(t_{j+1}; s_j, q_j)$ with respect to s_j and q_j will have to be provided for the solution of (P2)—these issues are further discussed in Section 4.

Note that the performance index J of (P1), a functional of x and u, has been replaced by the objective function $F(y)$ in (P2). As indicated by the notation in (1.19a), F is partially separable. In addition, we observe that the inequality constraints g_j in (1.19d) are completely decoupled and the equality conditions h_j in (1.19b)–(1.19c) are only linearly coupled. These properties of (P2) will lead to a special block structure of the corresponding QP subproblem, and can be exploited in order to greatly accelerate the solution process.

1.4 Problem Scaling

The scaling of nonlinear programming problems still seems to be a rather poorly understood subject: it almost invariably involves some kind of heuristics. The relative merits of different scaling techniques can be assessed only

on empirical grounds (i.e., through extensive numerical experimentation). Such comparative studies, however, are rarely found in the literature.

Many NLP methods—including the SQP method used in MUSCOD—are theoretically scale-invariant at least in their central parts and should therefore be relatively insensitive to poor scaling (see Section 3), but problem scaling has nevertheless been found to have a very significant impact on the performance of these methods in practice [Chen84, Bieg85]. This is mainly due to the fact that the ill-conditioning introduced by poor scaling can lead to an extreme amplification of the roundoff errors in finite-precision arithmetic [Stoer92]. In addition, practical NLP methods almost always contain certain scale-dependent parts (e.g., the choice of the initial approximation for the Hessian matrix). Hence, it is important to ensure that the discretized optimal control problem (P2) is properly scaled. We now briefly discuss some promising strategies.

Any scaling procedure should be regarded as an integral part of the modeling itself and should ideally be based on physical insight about the problem at hand. Notice that scaling can (and actually should) already be applied at the level of the continuous problem (P1): especially in formulating the dynamic model (1.4b), it is advisable to follow the standard guidelines for modeling physical systems. That is, one should choose physically meaningful characteristic scales for all the variables involved (including time) and reduce the problem to dimensionless form. Thereby, it is usually possible to ensure that the values of the state and control variables are approximately of order unity, and this is likely to avoid ill-conditioning of the discretized optimal control problem (P2) derived from (P1), and to reduce the negative impact of roundoff errors on the numerical solution process. Clearly, scaling at the level of (P1) has to be done by the modeler, and hence this kind of scaling will be termed *external scaling*. Some useful comments regarding the interplay of model building, dimensional analysis, and scaling can be found in the text [Logan87].

However, it turns out that in practice one cannot rely solely on external scaling because

- the modeler may have done a poor job,
- one may wish to interface the optimal control code with automatically generated large models which are often unscaled and cannot be easily modified to achieve proper scaling, or
- the problem might involve different scales at different points of time.

Therefore, some kind of *internal scaling* is usually applied at the level of problem (P2), simply having the effect that the optimization algorithm treats a rescaled version of the model while the original model formulation remains unchanged. It should be stressed that it is still desirable to have the user provide reasonable characteristic scales for the problem variables because estimating these scales automatically (e.g., from given start data) can be dangerous. In particular, this is true for values which happen to be close to zero.[‡] The scale factors should always be chosen as integer powers of the floating-point base (normally 2) to avoid roundoff [Toml75].

In order to apply internal scaling to problem (P2), we assume that a characteristic scale for each variable in the parameter vector y is available, and that these scales are represented by the diagonal matrices $\check{D}_j^s$ and $\check{D}_j^q$, respectively. Then we can define the scaled components $\check{s}_j$, $\check{q}_j$ of $\check{y}$ by

$$\check{s}_j := (\check{D}_j^s)^{-1}\, s_j, \quad \check{q}_j := (\check{D}_j^q)^{-1}\, q_j, \tag{1.23}$$

thus normalizing the variables to order unity. Note that we can have different characteristic scales on each multiple shooting interval (as already mentioned, there exist applications where the characteristic scales significantly change in time, but most often the same scaling matrices $\check{D}^s$ and $\check{D}^q$ can be used for all j). Substituting $s_j = \check{D}_j^s\, \check{s}_j$ and $q_j = \check{D}_j^q\, \check{q}_j$ into (1.19), we obtain a first scaled version of problem (P2), namely

$$\min_{\check{y}} \quad F(\check{y}) = \sum_{j=0}^{m-1} F_j(\check{D}_j^s\, \check{s}_j, \check{D}_j^q\, \check{q}_j) \tag{1.24a}$$

subject to

$$\begin{aligned} x(t_{j+1}; \check{D}_j^s\, \check{s}_j, \check{D}_j^q\, \check{q}_j) - \check{D}_{j+1}^s\, \check{s}_{j+1} &= 0 && \text{(1.24b)} \\ r_a(\check{D}_0^s\, \check{s}_0) + r_b(\check{D}_m^s\, \check{s}_m) &= 0 && \text{(1.24c)} \\ g_j(\check{D}_j^q\, \check{q}_j) &\geq 0. && \text{(1.24d)} \end{aligned}$$

There is empirical evidence that, for the numerical solution by SQP, it is advantageous to scale not only the variables, but also the objective function [Chen84] and the constraint functions [Bieg85]. It is easily seen

[‡]From a practical point of view, the orders of magnitude lie too close to each other in the interval $[0, 1]$ to be clearly distinguished—for instance, a value of $10^{-4} \approx 0$ could well be assumed by quantities with orders of magnitude ranging from 10^{-4} to 10^{0} or even higher. In the absence of additional knowledge, it is best to leave such "small" quantities unscaled, i.e., to choose a scale factor of unity.

that for the equality constraints (1.24b), an appropriate scale is represented by the matrix $\check{D}^s_{j+1}$, while for the remaining constraints (1.24c) and (1.24d), additional characteristic scales have to be provided by means of the diagonal matrices $\check{D}^r$ and $\check{D}^g_j$, respectively. Similarly, a scalar factor $\check{d}^F$ representing the magnitude of the objective function must be chosen. If we now apply these additional scaling operations to (1.24), we arrive at the final scaled version of (P2), which will be denoted by (P2^s),

$$\min_{\check{y}} \quad \check{F}(\check{y}) = (\check{d}^F)^{-1} \sum_{j=0}^{m-1} F_j(\check{D}^s_j\, \check{s}_j, \check{D}^q_j\, \check{q}_j) \tag{1.25a}$$

subject to

$$\check{h}_j(\check{s}_j, \check{s}_{j+1}, \check{q}_j) \;:=\; (\check{D}^s_{j+1})^{-1}\, x(t_{j+1}; \check{D}^s_j\, \check{s}_j, \check{D}^q_j\, \check{q}_j) - \check{s}_{j+1} \;=\; 0 \tag{1.25b}$$

$$\check{h}_m(\check{s}_0, \check{s}_m) \;:=\; (\check{D}^r)^{-1} \left(r_a(\check{D}^s_0\, \check{s}_0) + r_b(\check{D}^s_m\, \check{s}_m) \right) \;=\; 0 \tag{1.25c}$$

$$\check{g}_j(\check{q}_j) \;:=\; (\check{D}^g_j)^{-1}\, g_j(\check{D}^q_j\, \check{q}_j) \;\geq\; 0. \tag{1.25d}$$

Throughout most of the following discussion, we will return to the original formulation (P2) for notational convenience, but we will always assume proper scaling of the discretized problem. Some details regarding the implementation of the internal scaling procedure used in MUSCOD will be addressed in Section 7.

2 Optimality Conditions

In this section, we consider the general NLP problem of the form

$$\min_y \; F(y), \quad y \in \mathbb{R}^n \tag{2.1a}$$

subject to

$$h(y) = 0 \tag{2.1b}$$
$$g(y) \geq 0, \tag{2.1c}$$

where the functions $F : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, and $g : \mathbb{R}^n \to \mathbb{R}^l$ are at least once continuously differentiable (for second order conditions, we must assume continuous second partial derivatives). The gradients $\nabla F(y)$, $\nabla h_i(y)$, and $\nabla g_j(y)$ of the scalar functions F, h_i, and g_j are defined to be column vectors as usual, and we will also use the convenient notation

$$\nabla h = (\nabla h_1, \ldots, \nabla h_m), \quad \nabla g = (\nabla g_1, \ldots, \nabla g_l),$$

i.e., the Jacobian matrices of h and g are given by $\nabla h(y)^T$ and $\nabla g(y)^T$, respectively. The Hessian matrices of F, h_i, and g_j will be denoted by $\nabla^2 F(y)$, $\nabla^2 h_i(y)$, and $\nabla^2 g_j(y)$.

Definition 2.1 (Feasibility and Optimality)

1. *A point $\hat{y} \in \mathbb{R}^n$ is* feasible *with respect to the constraint $h_i(y) = 0$ if $h_i(\hat{y}) = 0$ (and we say that the constraint is satisfied at $\hat{y}$). Otherwise, $\hat{y}$ is* infeasible*, and we say that the constraint is violated at $\hat{y}$.*

2. *The point $\hat{y}$ is said to be* feasible *with respect to the inequality constraint $g_j(y) \geq 0$ if $g_j(\hat{y}) \geq 0$. (Equivalently, the constraint is satisfied at $\hat{y}$.) The constraint $g_j(y) \geq 0$ is said to be* active *at $\hat{y}$ if $g_j(\hat{y}) = 0$ and* inactive *if $g_j(\hat{y}) > 0$. If $g_j(\hat{y}) < 0$, $\hat{y}$ is* infeasible*, and the constraint is said to be violated at $\hat{y}$.*

3. *The point y^* is a* local minimizer *(or local solution) of the NLP problem (2.1) if:*

   - *y^* is feasible with respect to all the constraints;*
   - *there exists a neighborhood $N(\delta, y^*)$ such that*

$$F(y^*) \leq F(y) \;\; \text{for all feasible} \;\; y \in N(\delta, y^*). \tag{2.2}$$

If inequality (2.2) is strict for all feasible $y \in N(\delta, y^*)$, $y \neq y^*$, then y^* is said to be a strong *or* strict *local minimizer. Otherwise, y^* is called a* weak *local minimizer.*

In the formulation of first and second order optimality conditions for (2.1), a central role is played by the *Lagrangian function*, which is defined by

$$\begin{aligned} L(y, \lambda, \mu) &= F(y) - \lambda^T h(y) - \mu^T g(y) \\ &= F(y) - \sum_{i=1}^{m} \lambda_i \, h_i(y) - \sum_{j=1}^{l} \mu_j \, g_j(y), \end{aligned} \tag{2.3}$$

where $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^l$ are called *Lagrange multipliers*.

2.1 Karush-Kuhn-Tucker Conditions

In order to verify that a feasible point y^* is a local minimizer according to (2.2), a characterization of neighboring feasible points is needed. In general, feasibility can be retained with respect to a set of nonlinear equality and inequality constraints only by moving along a *nonlinear* path in $\mathbb{R}^n$, which is called a *feasible arc*. While an equality constraint $h_i(y) = 0$ allows only *binding* perturbations (which remain on the constraint), an active inequality constraint $g_j(y) = 0$ allows also *non-binding* perturbations (which move off the constraint, i.e., the inequality becomes inactive at the new point). An inactive inequality constraint allows any sufficiently small perturbation without becoming violated, and hence does not impose any restriction on "local" feasible movements.

Let $\alpha(\theta)$ denote a feasible arc parameterized by a single variable θ, with $\alpha(0) = y^*$, and let p be the tangent to the arc at y^*. Then, to remain feasible, for each equality constraint we have the condition $h_i(\alpha(\theta)) \equiv 0$, $0 \leq \theta \leq \bar{\theta}$ (for some small, positive $\bar{\theta}$), which implies that

$$\frac{d}{d\theta} h_i(\alpha(\theta)) \Big|_{\theta=0} = \nabla h_i(\alpha(0))^T \alpha'(0) = \nabla h_i(y^*)^T p = 0, \quad i = 1, 2, \ldots, m,$$

and we observe that the feasible direction p must be in the nullspace of the Jacobian matrix of h,

$$\nabla h(y^*)^T p = 0. \tag{2.4}$$

Similarly, since for each active inequality we have the condition $g_j(\alpha(\theta)) \geq 0$, $0 \leq \theta \leq \bar{\theta}$ (g_j must be non-decreasing along the arc), it follows that

$$\nabla g_j(y^*)^T p \geq 0, \quad j \in \mathcal{J}^* = \{j \mid g_j(y^*) = 0\}. \tag{2.5}$$

Unfortunately, (2.4) and (2.5) only provide a complete characterization of the feasible directions p for *linear* constraint functions h and g—then each feasible direction must satisfy these relationships, and each vector p for which (2.4) and (2.5) hold is a feasible direction. In the case of nonlinear constraints, (2.4) and (2.5) are still necessary conditions for p to be a feasible direction, but then these conditions do not guarantee that a feasible arc with tangent p exists. Therefore, certain *additional* conditions must be imposed on the constraint functions to ensure existence of a feasible arc with tangent p. These additional conditions are usually termed *constraint qualifications*. Here we state one of the various possible definitions which suits our presentation.

Definition 2.2 (First Order Constraint Qualification)
The first order constraint qualification with respect to the equality constraints $h(y) = 0$ *and the inequality constraints* $g(y) \geq 0$ *holds at the feasible point* $\hat{y}$ *if every nonzero vector* $p \in \mathbb{R}^n$ *satisfying*

$$\begin{aligned} \nabla h_i(\hat{y})^T p &= 0, \quad i = 1, 2, \ldots, m, \\ \nabla g_j(\hat{y})^T p &\geq 0, \quad j \in \mathcal{J} = \{j \mid g_j(\hat{y}) = 0\} \end{aligned}$$

is tangent to a continuously differentiable feasible arc $\alpha(\theta)$ *emanating from* $\hat{y}$ *(which means that* $\alpha(0) = \hat{y}$ *and* $\alpha'(0) = p$*).*

Note that the constraint qualification always holds for linear constraints, as indicated above. For nonlinear constraints, the following theorem provides a computationally practical test for verifying that the first order constraint qualification is satisfied. (See [Fiac68] for a proof.)

Theorem 2.3 (First Order Constraint Qualification Condition)
If $\nabla h_i(\hat{y})$, $i = 1, 2, \ldots, m$, *and* $\nabla g_j(\hat{y})$, $j \in \mathcal{J} = \{j \mid g_j(\hat{y}) = 0\}$, *are linearly independent at a feasible point* $\hat{y}$*, then the first order constraint qualification holds.*

Definition 2.4 (Regular Point)
A feasible point which satisfies the condition of Theorem 2.3 is said to be a regular point.

It should be emphasized that the first order constraint qualification may hold at a point which is not a regular point, i.e., Theorem 2.3 provides only a sufficient condition, but not a necessary one. In practical problems, the optimal solution y^* will frequently be a regular point.

We now state the well-known first order Karush-Kuhn-Tucker conditions which have been first derived by Karush in 1939 (and independently by Kuhn and Tucker in 1951) as a generalization of the Lagrange multiplier rule for equality constrained optimization problems. (A proof of the following theorem can be found in virtually any text on nonlinear programming, e.g., [Baza79].)

Theorem 2.5 (Karush-Kuhn-Tucker Necessary Condition)
If the first order constraint qualification holds at y^*, a necessary condition for y^* to be a local minimizer of (2.1) is that there exist multiplier vectors $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^l$, such that (y^*, λ^*, μ^*) satisfies the following set of conditions (commonly known as Karush-Kuhn-Tucker conditions*),*

$$\begin{aligned} \nabla_y L(y^*, \lambda^*, \mu^*) &= 0 && \text{(2.6a)} \\ h(y^*) &= 0 && \text{(2.6b)} \\ g(y^*) &\geq 0 && \text{(2.6c)} \\ \mu^* &\geq 0 && \text{(2.6d)} \\ \mu_j^* \, g_j(y^*) &= 0, \quad j = 1, 2, \ldots, l, && \text{(2.6e)} \end{aligned}$$

where L is the Lagrangian function defined in (2.3), hence

$$\begin{aligned} \nabla_y L(y^*, \lambda^*, \mu^*) &= \nabla F(y^*) - \nabla h(y^*)\, \lambda^* - \nabla g(y^*)\, \mu^* \\ &= \nabla F(y^*) - \sum_{i=1}^{m} \lambda_i^* \, \nabla h_i(y^*) - \sum_{j=1}^{l} \mu_j^* \, \nabla g_j(y^*). \end{aligned}$$

Definition 2.6 (Karush-Kuhn-Tucker Point)
A vector (y^*, λ^*, μ^*) which satisfies conditions (2.6) is called a Karush-Kuhn-Tucker point (KKT point).

We observe that y^* is a *stationary point* of $L(y, \lambda, \mu)$ with respect to y when $\lambda = \lambda^*$ and $\mu = \mu^*$. Another way of interpreting condition (2.6a) is that $\nabla F(y^*)$, the gradient of the objective function, can be written as a linear combination of the constraint gradients $\nabla h_i(y^*)$ and $\nabla g_j(y^*)$.

While the Lagrange multipliers λ^* corresponding to equality constraints are not restricted in sign, the multipliers μ^* corresponding to inequality constraints must be *nonnegative* by (2.6d), and inactive inequalities are assigned *zero* multipliers via the *complementarity condition* (2.6e). The Lagrange multipliers corresponding to equalities and active inequalities have an economic interpretation as *shadow prices*: if the equality $h_i(y) = 0$ is

replaced by $h_i(y) = \delta_i$ and the optimal values of F for the original and the perturbed problem are denoted by F^* and $F^*_{\delta_i}$, respectively, then it can be easily shown that

$$\lim_{\delta_i \to 0} \frac{F^*_{\delta_i} - F^*}{\delta_i} = \lambda^*_i.$$

Therefore, the sign of λ^*_i determines whether the optimal value of F increases or decreases as the right-hand side constant δ_i is increased from zero. For active inequalities, increasing the right-hand side constant implies restricting the original feasible region (because any point $\hat{y}$ with $g_j(\hat{y}) \geq \delta_j$ will automatically be feasible with respect to $g_j(\hat{y}) \geq 0$), and hence the optimal value of F must be nondecreasing. This gives a simple motivation for the nonnegativity condition (2.6d).

Since a KKT point is a *possible candidate* for a local solution of problem (2.1), many NLP methods attempt to converge to a KKT point. In order to check the optimality of such a point, second order sufficient conditions can be used (see Theorems 2.11 and 2.13). However, under certain (quite restrictive) assumptions a KKT point automatically becomes the global optimal solution to the NLP problem. This is stated in the following theorem (a proof of which can be found in [Mang69]).

Theorem 2.7 (Karush-Kuhn-Tucker Sufficient Condition)
Let F be convex, h_i be linear for $i = 1, 2, \ldots, m$, and g_j be concave for $j = 1, 2, \ldots, l$. If there exists a KKT point (y^*, λ^*, μ^*), then y^* is a local minimizer of (2.1). Furthermore, y^* represents the global *optimal solution.*

Notice that no further constraint qualification is needed. As we will see in Section 6, this first order sufficient condition applies to our QP subproblem which is a convex quadratic program.

2.2 Second Order Conditions

If the problem functions F, h_i, and g_j are twice continuously differentiable (which shall be assumed in the following), second order necessary and sufficient conditions for optimality can be derived. By using second order derivative information, the necessary condition allows a "sharper" characterization of candidate solutions, and sufficient conditions can be stated for general problem functions (without the convexity and linearity assumptions of Theorem 2.7). The following results have been developed by McCormick and Fiacco during the 1960s; for details and proofs see [Fiac68].

The second order necessary condition (Theorem 2.10 below) requires not only the first order constraint qualification of Definition 2.2 to hold at a candidate point y^*, but also a second order constraint qualification—again, we state only one of various possible forms.

Definition 2.8 (Second Order Constraint Qualification)
The second order constraint qualification with respect to the equality constraints $h(y) = 0$ and the inequality constraints $g(y) \geq 0$ holds at the feasible point $\hat{y}$ if every nonzero vector $p \in \mathbb{R}^n$ satisfying

$$\begin{aligned} \nabla h_i(\hat{y})^T p &= 0, \quad i = 1, 2, \ldots, m, \\ \nabla g_j(\hat{y})^T p &= 0, \quad j \in \mathcal{J} = \{j \mid g_j(\hat{y}) = 0\} \end{aligned}$$

is tangent to a twice continuously differentiable feasible arc $\alpha(\theta)$ emanating from $\hat{y}$, along which $g_j(\alpha(\theta)) \equiv 0$, $j \in \mathcal{J}$.

For equality constraints, Definition 2.8 implies Definition 2.2, but this is not true for inequality constraints. Fortunately, the same simple condition which ensures satisfaction of the first order constraint qualification (Theorem 2.3) also implies the second order constraint qualification.

Theorem 2.9 (Second Order Constraint Qualification Condition)
At a regular point (see Definition 2.4), the second order constraint qualification holds.

In most cases, Theorems 2.3 and 2.9 provide the only practical way of checking the constraint qualifications at some given point, but it is important to remember that this is only a sufficiency test—a "negative" result does not imply that the constraint qualifications are violated.

We now state the second order necessary condition for optimality. This condition allows a better characterization of candidate solutions than the first order Karush-Kuhn-Tucker conditions of Theorem 2.5 by imposing additional second order restrictions. One could compare with the simple case of unconstrained, univariate minimization, where we have $f'(x^*) = 0$ as the first order necessary condition for a local minimum at x^*, and the combination of this first order condition with $f''(x^*) \geq 0$ constitutes the second order necessary condition.

Theorem 2.10 (Second Order Necessary Condition)
If the first and second order constraint qualifications hold at y^*, the following is a necessary condition for y^* to be a local minimizer of (2.1): there

exist multiplier vectors $\lambda^* \in \mathbb{R}^m$ *and* $\mu^* \in \mathbb{R}^l$, *such that* (y^*, λ^*, μ^*) *is a KKT point, and such that for every nonzero vector* $p \in \mathbb{R}^n$ *satisfying*

$$\begin{aligned} \nabla h_i(y^*)^T p &= 0, \quad i = 1, 2, \ldots, m \\ \nabla g_j(y^*)^T p &= 0, \quad j \in \mathcal{J}^* = \{j \mid g_j(y^*) = 0\}, \end{aligned}$$

it follows that

$$p^T \, \nabla_y^2 L(y^*, \lambda^*, \mu^*) \, p \; \geq \; 0, \tag{2.7}$$

where $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$, *the Hessian matrix of* L *with respect to* y *evaluated at* (y^*, λ^*, μ^*), *is given by*

$$\nabla_y^2 L(y^*, \lambda^*, \mu^*) = \nabla^2 F(y^*) - \sum_{i=1}^{m} \lambda_i^* \, \nabla^2 h_i(y^*) - \sum_{j=1}^{l} \mu_j^* \, \nabla^2 g_j(y^*).$$

This second order necessary condition can be useful to prove that a point which satisfies Theorem 2.5 is *not* a local minimizer of the NLP problem (for instance, both local maxima and saddlepoints satisfy Theorem 2.5 but not Theorem 2.10).

The condition (2.7) involving the Hessian matrix of L is often stated in a slightly different form: let $A(y^*)$ denote a matrix whose rows contain the transposed gradients of the equalities and active inequalities at y^*, and let $Z(y^*)$ be a matrix whose columns form a basis of the nullspace of $A(y^*)$, hence $A(y^*) \, Z(y^*) = 0$. Then each vector p in the nullspace of $A(y^*)$ can be written as a linear combination of the column vectors of $Z(y^*)$ in the form $p = Z(y^*) \, p_z$. Using this notation, condition (2.7) is transformed into

$$p_z^T \; Z(y^*)^T \, \nabla_y^2 L(y^*, \lambda^*, \mu^*) \, Z(y^*) \, p_z \; \geq \; 0, \tag{2.8}$$

where $Z(y^*)^T \nabla_y^2 L(y^*, \lambda^*, \mu^*) Z(y^*)$ is called the *reduced* or *projected* Hessian matrix of L. Since (2.8) has to be satisfied for arbitrary p_z, it follows that the projected Hessian matrix has to be positive semi-definite. However, observe that the Hessian matrix itself is *not* required to be positive semi-definite.

Now we turn to second order sufficient conditions which are of great practical importance—they allow us to prove that a point is a local solution to the NLP without being forced to use the rigid assumptions of Theorem 2.7. One of various possible conditions is given in the following theorem.

Theorem 2.11 (Second Order Sufficient Condition)
A sufficient condition for y^* *to be a* strict local minimizer *of (2.1) is that*

there exist multiplier vectors $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^l$, such that (y^*, λ^*, μ^*) is a KKT point, and such that for every nonzero vector $p \in \mathbb{R}^n$ satisfying

$$\begin{aligned} \nabla h_i(y^*)^T p &= 0, \quad i = 1, 2, \ldots, m && \text{(2.9a)} \\ \nabla g_j(y^*)^T p &= 0, \quad j \in \mathcal{J}_+^* = \{j \mid g_j(y^*) = 0 \wedge \mu_j^* > 0\} && \text{(2.9b)} \\ \nabla g_j(y^*)^T p &\geq 0, \quad j \in \mathcal{J}_0^* = \{j \mid g_j(y^*) = 0 \wedge \mu_j^* = 0\}, && \text{(2.9c)} \end{aligned}$$

it follows that

$$p^T \nabla_y^2 L(y^*, \lambda^*, \mu^*)\, p > 0. \tag{2.10}$$

(Note that $\mathcal{J}_+^* \cup \mathcal{J}_0^* = \mathcal{J}^*$, the index set of all active inequality constraints.)

A comparison of Theorems 2.10 and 2.11 shows that very little has to be added to the necessary condition to obtain a sufficient one: the active inequality constraints with zero Lagrange multipliers have to be treated differently, and condition (2.7) must be satisfied as a strict inequality. Note that no constraint qualification is needed.

As in case of Theorem 2.10, an alternative formulation of condition (2.10) can be given in terms of the projected Hessian: the matrix $A(y^*)$ now contains only the transposed gradients of the equalities and the active inequalities *with nonzero multipliers*, and the columns of $Z(y^*)$ again form a basis of the nullspace of $A(y^*)$. Of course this does not account for the additional restrictions (2.9c) originally imposed on p. Hence the requirement that the projected Hessian $Z(y^*)^T \nabla_y^2 L(y^*, \lambda^*, \mu^*)\, Z(y^*)$ be positive definite is *not* equivalent to condition (2.10), but rather it *implies* this condition. Therefore, the original formulation of Theorem 2.11 is "sharper" in the sense that there may be points that can be proved to be optimal by the original condition (2.10) together with restrictions (2.9) but not by the alternative condition which uses the projected Hessian. Theorem 2.11 can also be extended to weak local minima; for details see [Fiac68].

Finally, we state another, more restrictive set of conditions which together provide not only a sufficient condition for optimality but also ensure the uniqueness of the Lagrange multipliers λ^*, μ^*.

Definition 2.12 (Jacobi Uniqueness Condition)
A KKT point (y^*, λ^*, μ^*) satisfies the Jacobi uniqueness condition if the following conditions are all satisfied:

- *y^* is a regular point (see Definition 2.4)*

- $\mu_j^* > 0$ *if* $j \in \mathcal{J}^* = \{j \mid g_j(y^*) = 0\}$, *i.e. the Lagrange multipliers corresponding to active inequality constraints are strictly positive (this is usually termed* strict complementarity*)*

- *for any nonzero vector* $p \in \mathbb{R}^n$ *satisfying*

$$\begin{aligned} \nabla h_i(y^*)^T p &= 0, \quad i = 1, 2, \ldots, m \\ \nabla g_j(y^*)^T p &= 0, \quad j \in \mathcal{J}^*, \end{aligned}$$

it follows that

$$p^T \nabla_y^2 L(y^*, \lambda^*, \mu^*)\, p > 0.$$

Theorem 2.13 (Sufficiency and Uniqueness of Multipliers)
If a KKT point (y^*, λ^*, μ^*) *satisfies the Jacobi uniqueness condition, then* y^* *is a strict local minimizer of (2.1), and the Lagrange multipliers* $\lambda^* \in \mathbb{R}^m$, $\mu^* \in \mathbb{R}^l$ *are uniquely determined. The Jacobian matrix* J *corresponding to the set of nonlinear equations (2.6a), (2.6b), and (2.6e),*

$$\begin{aligned} \nabla_y L(y, \lambda, \mu) &= 0 \\ h(y) &= 0 \\ \mu_j\, g_j(y) &= 0, \quad j = 1, 2, \ldots, l, \end{aligned}$$

which is given by

$$J = \begin{pmatrix} \nabla_y^2 L(y, \lambda, \mu) & -\nabla h(y) & -\nabla g(y) \\ \nabla h(y)^T & 0 & 0 \\ \operatorname{diag}(\mu_j)\, \nabla g(y)^T & 0 & \operatorname{diag}(g_j(y)) \end{pmatrix}, \tag{2.11}$$

is nonsingular at the point (y^*, λ^*, μ^*).

The nonsingularity of J at (y^*, λ^*, μ^*) implies for instance that it is possible to solve equations (2.6a), (2.6b), and (2.6e) for (y^*, λ^*, μ^*) by Newton's method (if the initial estimate is sufficiently close to the solution).

In this section, we have dealt only with optimality conditions that are necessary *or* sufficient, and there is a considerable "gap" between these two (i.e., there are points which qualify as candidate solutions but cannot be proved to be optimal by the sufficient conditions). However, sets of conditions that are simultaneously necessary *and* sufficient become relatively complicated. Since these conditions can seldom be verified, they are of little practical value and therefore have not been included here.

3 Outline of the SQP Solution Process

3.1 Motivation of SQP Methods

A basic principle used in solving the general NLP problem is that of replacing a difficult problem by an easier one. Application of this principle leads to methods that formulate and solve a *sequence of subproblems*, where each subproblem is related in a certain way to the original problem. For instance, in the simplest case of unconstrained minimization, quite frequently a *local quadratic model* is developed of the nonlinear (but smooth) function to be minimized. A step is then made from the current iterate in the direction toward the minimizer of the model function.[‡] However, since higher-order terms are neglected, the model may be valid only in a small neighborhood of the current point, and a line search or trust region method has to be applied in order to determine a suitable step. By these means, it is in general possible to ensure sufficient "progress" at every iteration, thus obtaining globally convergent methods.

SQP methods for solving the general NLP problem are directly based on the optimality conditions from Section 2, where we found that the optimality of a point y^* is determined by the curvature of the *Lagrangian function* L rather than by that of the objective function F. This suggests using a quadratic model of the Lagrangian function. However, recall that y^* is in general only a *stationary point* of $L(y, \lambda^*, \mu^*)$ with respect to y, and not an unconstrained minimizer (Theorem 2.5). Even when the sufficient conditions of Theorem 2.13 hold, y^* is a minimizer of $L(y, \lambda^*, \mu^*)$ only within the subspace of vectors p satisfying $A(y^*)p = 0$, where the matrix $A(y^*)$ contains the transposed gradients of the equalities and the active inequalities (which all have nonzero multipliers because of the assumed strict complementarity). This restriction to a subspace suggests that linear constraints should be imposed on the quadratic model of the Lagrangian function.

With these ideas in mind, let us now consider the development of an algorithm of the form

$$y_{k+1} = y_k + \alpha_k p_k, \tag{3.1}$$

where p_k is a search direction and α_k is a nonnegative relaxation factor that will be determined by a line search procedure (usually $0 < \alpha_k \leq 1$). Since p_k is intended to be an *estimate* of the step from the current iterate y_k to

[‡]For finding minimum points, a *linear model* is usually inappropriate, since a general linear function is unbounded below. This is in contrast to root-finding, where local linear approximations are widely applied.

y^*, the optimality conditions at y^* should guide the definition of p_k. First of all, y^* must be *feasible*, that is, the conditions $h(y^*) = 0$ and $g(y^*) \geq 0$ must be satisfied. However, remaining feasible or even near-feasible at every iterate is quite inefficient if the constraints are nonlinear—the algorithm would have to take very small steps along the curved surface defined by the constraints, and the effort for maintaining strict feasibility would be more or less wasted at points that are far from optimal. Hence for the equality constraints, instead of enforcing $h(y_k + p_k) = 0$, we impose the condition $h(y_k) + \nabla h(y_k)^T p_k = 0$ or

$$\nabla h(y_k)^T p_k = -h(y_k), \tag{3.2}$$

i.e., the search direction p_k will be a step to a zero of a local *linear approximation* to h. Similarly, for the inequality constraints, we arrive at the linearization

$$\nabla g(y_k)^T p_k \geq -g(y_k). \tag{3.3}$$

If the gradients $\nabla h_i(y_k)$ for $i = 1, 2, \ldots, m$ and $\nabla g_j(y_k)$ for all those j with $\nabla g_j(y_k)^T p_k = -g_j(y_k)$ are linearly independent, these linearized constraints are always consistent; otherwise it may occur that there is no solution. In the limit $y_k \to y^*$, the right-hand side of (3.2) becomes zero, and in (3.3) the vector $-g(y^*)$ has zero components for all inequality constraints active at y^*. Thus at y^*, the search directions p will be entirely in the nullspace of $A(y^*)$, as required above.

Beyond satisfying the linear constraints (3.2)–(3.3), the search direction p_k in (3.1) should be defined by minimization of a quadratic model of the Lagrangian function. Such a model could be obtained from the first three terms of the Taylor series of L around y_k (for fixed values λ_k, μ_k),

$$M_L(y_k + p, \lambda_k, \mu_k) := L_k + \nabla_y L_k^T \, p + \frac{1}{2} p^T \, \nabla_y^2 L_k \, p,$$

where the index k in L_k, $\nabla_y L_k$, and $\nabla_y^2 L_k$ indicates that these quantities are evaluated at (y_k, λ_k, μ_k). Minimizing with respect to p, the constant term L_k can be eliminated from M_L. Furthermore, it can be easily verified that

$$\nabla_y L(y^*, \lambda, \mu)^T \, p \equiv \nabla F(y^*)^T \, p \;\text{ if } p \text{ satisfies } \; A(y^*) \, p = 0,$$

provided $\mu_j = 0$ for $j \notin \mathcal{J}^*$, where $\mathcal{J}^*$ denotes the index set of inequalities active at y^*. Based on this observation, it is justified to replace $\nabla_y L_k$ by ∇F_k in M_L, since they are equal in the limit $y_k \to y^*$. As will be shown below, this modification has a certain advantage.

We can now formulate a first version of the QP subproblem by combining the modified quadratic model of L with the linearized constraints (3.2)–(3.3),

$$\min_p \ \nabla F_k^T p + \frac{1}{2} p^T \nabla_y^2 L_k \, p \tag{3.4a}$$

subject to

$$\begin{aligned} h_k + \nabla h_k^T p &= 0 && \text{(3.4b)} \\ g_k + \nabla g_k^T p &\geq 0, && \text{(3.4c)} \end{aligned}$$

where we have again used abbreviated notation. Note that the formulation of the subproblem depends not only on y_k, but also on the current estimates λ_k, μ_k of the Lagrange multipliers of the original problem. The solution of the subproblem provides the search direction p_k for the algorithm (3.1), and the corresponding subproblem multipliers will be denoted by $\tilde{\lambda}_k$, $\tilde{\mu}_k$ in order to distinguish them from λ_k, μ_k.

We now consider the Karush-Kuhn-Tucker conditions for problem (3.4) which are given by

$$\begin{aligned} \nabla_y^2 L_k \, p_k + \nabla F_k - \nabla h_k \tilde{\lambda}_k - \nabla g_k \tilde{\mu}_k &= 0 && \text{(3.5a)} \\ h_k + \nabla h_k^T p_k &= 0 && \text{(3.5b)} \\ g_k + \nabla g_k^T p_k &\geq 0 && \text{(3.5c)} \\ \tilde{\mu}_k &\geq 0 && \text{(3.5d)} \\ \tilde{\mu}_{k,j} \, (g_k + \nabla g_k^T p_k)_j &= 0, \quad j = 1, 2, \ldots, l, && \text{(3.5e)} \end{aligned}$$

where $\tilde{\mu}_{k,j}$ denotes component j of $\tilde{\mu}_k$. We are interested in the limiting case $p_k \to 0$, since this implies that the algorithm (3.1) converges to some point $\bar{y}$. Let us assume that $p^* = 0$ is a local solution of the subproblem formulated at $(\bar{y}, \bar{\lambda}, \bar{\mu})$. Then there exist Lagrange multipliers $\tilde{\lambda}^*$ and $\tilde{\mu}^*$ such that $(0, \tilde{\lambda}^*, \tilde{\mu}^*)$ is a KKT point satisfying the conditions (3.5).[‡] It immediately follows that then $(\bar{y}, \tilde{\lambda}^*, \tilde{\mu}^*)$ must be a KKT point of the *original problem*, hence we have $y^* = \bar{y}$. Note also that $\lambda^* = \tilde{\lambda}^*$ and $\mu^* = \tilde{\mu}^*$: the multipliers of the original problem are given by the multipliers of the subproblem. As can be seen from equation (3.5a), the subproblem would not have this desirable feature if we had used $\nabla_y L_k$ instead of ∇F_k in the quadratic approximation of the Lagrangian function; this explains why it is advantageous to make the

[‡]Since the constraints are linear, the constraint qualification is always satisfied, and according to Theorem 2.5 a KKT point *must* exist for the subproblem.

modification mentioned above. Another interesting observation can be made after reformulating the subproblem at the new point (y^*, λ^*, μ^*): clearly, this will leave the KKT point of the subproblem unchanged—the conditions (3.5) are still satisfied by $(0, \tilde{\lambda}^*, \tilde{\mu}^*)$. Now suppose that for this newly formulated subproblem, the KKT point $(0, \tilde{\lambda}^*, \tilde{\mu}^*)$ additionally satisfies the second order sufficient condition of Theorem 2.11. Then it follows that the same holds for the original problem and its KKT point (y^*, λ^*, μ^*) (the Hessian matrix of the subproblem is the same as that of the original problem, as the constraints are linear).

Our discussion indicates that the subproblem (3.4) has the following very important features:

1. If no further corrections can be found (i.e., $p = 0$), then a KKT point of the original problem will have been obtained.

2. At each iteration k, the multipliers $\tilde{\lambda}_k$, $\tilde{\mu}_k$ of the subproblem can be used conveniently as the multipliers λ_{k+1}, μ_{k+1} to formulate the next subproblem.

Up to this point, however, we have just *assumed* convergence of the sequence of subproblems such that $p_k \to 0$. It remains to be shown under which conditions convergence of the method can actually be established. Here we consider only *local* convergence from points that are sufficently close to the solution (global convergence will be discussed later). In a sufficiently small neighborhood of the solution, the subproblem can be interpreted as a slight *perturbation* of the original problem, and certain properties hold. In order to establish local convergence to a KKT point (y^*, λ^*, μ^*) of the original problem, we have to assume that (y^*, λ^*, μ^*) satisfies the Jacobi uniqueness condition (Definition 2.12). Then the same is true for the KKT point of the subproblem (3.4) formulated at a point sufficiently close to (y^*, λ^*, μ^*). In addition, the subproblem will identify the correct set $\mathcal{J}^*$ of constraints active at y^* (for a proof of these results, see [Robi74]).

Because of the latter fact, the multipliers μ_j^* corresponding to inactive inequalities ($j \notin \mathcal{J}^*$) are known to be zero, and the solution of the original problem (including the remaining multipliers) can be found by solving an equivalent equality-constrained problem (where the constraints correspond to the equalities and the active inequalities of the original problem). For notational convenience, we write the extended set of equality constraints again in the form $h(y) = 0$ and use the old symbol λ for the corresponding extended multiplier vector. Then the Karush-Kuhn-Tucker conditions for

the equality-constrained problem are given by

$$\nabla_y L(y,\lambda) = \nabla F(y) - \nabla h(y)\lambda = 0 \tag{3.6a}$$

$$h(y) = 0. \tag{3.6b}$$

We now consider solving this set of nonlinear equations by Newton's method. At each iteration k, we have to solve the linear system

$$\begin{pmatrix} \nabla_y^2 L(y_k,\lambda_k) & -\nabla h(y_k) \\ \nabla h(y_k)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta y_k \\ \Delta\lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla_y L(y_k,\lambda_k) \\ h(y_k) \end{pmatrix} \tag{3.7}$$

for the corrections $\Delta y_k = y_{k+1} - y_k =: p_k$ and $\Delta\lambda_k = \lambda_{k+1} - \lambda_k$. Convergence is assured, because we have assumed the initial estimate (y_0, λ_0) to be sufficiently close to (y^*, λ^*), and the Jacobian matrix in (3.7) is nonsingular by Theorem 2.13 (the original problem satisfies the Jacobi uniqueness condition, and it immediately follows that the same is true for the equivalent equality-constrained problem). By substituting $\Delta y_k = p_k$ and $\Delta\lambda_k = \lambda_{k+1} - \lambda_k$ in (3.7) and simplifying, we obtain

$$\begin{pmatrix} \nabla_y^2 L(y_k,\lambda_k) & -\nabla h(y_k) \\ \nabla h(y_k)^T & 0 \end{pmatrix} \begin{pmatrix} p_k \\ \lambda_{k+1} \end{pmatrix} = - \begin{pmatrix} \nabla F(y_k) \\ h(y_k) \end{pmatrix}. \tag{3.8}$$

That is, the new multiplier vector λ_{k+1} can be found directly by solving (3.8) instead of (3.7). It is interesting to note that p_k and λ_{k+1} depend on λ_k only through the Hessian $\nabla_y^2 L(y_k, \lambda_k)$.

Not too surprisingly, we can now establish a close relationship between Newton's method to solve (3.6) and the SQP method discussed above. Consider the equality-constrained QP subproblem

$$\min_p \ \nabla F_k^T p + \frac{1}{2} p^T \nabla_y^2 L_k \, p \tag{3.9a}$$

subject to

$$h_k + \nabla h_k^T p = 0 \tag{3.9b}$$

formulated at (y_k, λ_k). The solution p_k and the multiplier $\tilde{\lambda}_k = \lambda_{k+1}$ of the subproblem (3.9) are found by solving the corresponding Karush-Kuhn-Tucker equations, and it can easily be verified that these equations are given by the system (3.8). Therefore, the solution of (3.9) correponds exactly to one Newton step in the solution of (3.6), and by this equivalence, local convergence of the SQP method is proved for constant steplength $\alpha_k = 1$.

In addition, it is apparent that the sequence (y_k, λ_k) generated by the SQP method converges *quadratically* to (y^*, λ^*) (since Newton's method does). Observe, however, that the equivalence between Newton's method and SQP holds only for equality-constrained problems and thus requires the correct active set $\mathcal{J}^*$ to be known (although the Jacobi uniqueness condition ensures that Newton's method could also be applied directly to the Karush-Kuhn-Tucker equations of the original problem, see Theorem 2.13).

In order to globalize convergence of the SQP method, a line search has to be performed. A relaxation factor α_k is chosen such that at each iteration a sufficient decrease is ensured in a so-called *merit function* P which measures progress toward the solution. Both objective function improvement and reduction of constraint violations have to be taken into account, and this suggests using some type of penalty function. The merit function P should have certain properties. First of all, satisfaction of the sufficient decrease condition should always be possible for the search direction p_k defined by the QP subproblem. Furthermore, at all iterations sufficiently near the solution, the steplength $\alpha_k \equiv 1$ should be accepted in order to retain the local quadratic convergence of the method. We will discuss the choice of a merit function and the details of the line search procedure below.

The SQP method in the form presented above was first proposed by Wilson in 1963. As we have shown, the method has quite interesting features, but it has never had practical success in its original form. This is mainly due to the fact that it is very expensive (if not impossible) in practice to calculate the exact Hessian matrix $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$ on every iteration. Moreover, although the projected Hessian matrix is known to be positive (semi-)definite at the solution (y^*, λ^*, μ^*), this is not necessarily true at points far from the solution: the projected Hessian may become indefinite, and in this case the corresponding subproblem has an unbounded solution. Apart from these difficulties, the information in $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$ has little relevance in defining a good search direction p_k unless the multiplier estimates λ_k and μ_k are reasonably close to λ^* and μ^*. This implies that the whole effort of evaluating all second derivatives may be more or less wasted during the initial block of iterations. These shortcomings of the original method have been largely overcome by combining the SQP idea with the variable metric approach known from unconstrained optimization.

Before continuing our discussion of SQP methods we state more precisely what is meant by *convergence* of a sequence of iterates, and we define several *rates of convergence* that we will encounter in what follows.

Definition 3.1 (Convergence and Rates of Convergence)
The sequence $(y_k)_{k=0}^{\infty}$, $y_k \in \mathbb{R}^n$, *is said to* converge *to a point* $y^* \in \mathbb{R}^n$ *if for every* i, *the* i*-th component of* $y_k - y^*$ *satisfies*

$$\lim_{k\to\infty} (y_k - y^*)_i = 0.$$

If, for some vector norm $\|\cdot\|$, *there exist* $K \geq 0$ *and* $\alpha \in [0, 1[$ *such that for all* $k \geq K$,

$$\|y_{k+1} - y^*\| \leq \alpha \|y_k - y^*\|,$$

then (y_k) *is said to converge* q-linearly *to* y^* *in the norm* $\|\cdot\|$. *If there exists a sequence of scalars* (α_k) *converging to zero such that for all* $k \geq 0$,

$$\|y_{k+1} - y^*\| \leq \alpha_k \|y_k - y^*\|,$$

then (y_k) *is said to converge* q-superlinearly *to* y^*. *If* (y_k) *converges to* y^* *and there exist* $K \geq 0$, $\alpha \geq 0$, *and* $\beta > 0$ *such that for all* $k \geq K$

$$\|y_{k+1} - y^*\| \leq \alpha \|y_k - y^*\|^{\beta},$$

(y_k) *is said to converge to* y^* *with* q-order at least β. *If* $\beta = 2$, *then the convergence is called* q-quadratic.

The prefix q stands for "quotient" rates of convergence. There is a set of similar definitions for r ("root") rates of convergence, which are a weaker form of convergence. For instance, a sequence (y_k) is said to be *r-superlinearly* convergent if $\lim_{k\to\infty} \|y_k - y^*\|^{1/k} = 0$. Note that a sequence may be q-linearly convergent in one norm but not another, but that q-superlinear and q-order $\beta > 1$ convergence are independent of the choice of norm. Newton's method is locally q-quadratically convergent, and many other practical methods are locally q-superlinearly convergent (q-linearly convergent methods tend to be very slow and are usually avoided, unless the convergence parameter α is known to be acceptably small). All r-rates of convergence are independent of the choice of norm. For further detail, see [Orte70] or [Denn89].

3.2 Variable Metric Methods for Constrained Optimization

3.2.1 Some Basic Facts on Variable Metric Methods

Variable metric (or quasi-Newton) methods have been successfully applied in unconstrained minimization, where the desirable local convergence properties of Newton's method can be approached by using suitable update formulas to approximate the Hessian matrix of the function to be minimized,

thus eliminating the need to calculate the second derivatives. We will now try to highlight the major points of the original ("unconstrained") approach before moving on to the constrained case. Let $F(x)$ be the function to be minimized, and let x^* be a local minimizer of F with $\nabla F(x^*) = 0$ and $\nabla^2 F(x^*) > 0$. Suppose that an initial estimate x_0 sufficiently close to x^* is known. At each point x_k, $k = 0, 1, \ldots$, we define a *local quadratic model* of F by

$$M_F^k(x_k + \delta) := F(x_k) + \nabla F(x_k)^T \delta + \frac{1}{2} \delta^T B_k \, \delta,$$

where the symmetric, positive definite matrix B_k is intended to be an approximation to the Hessian $\nabla^2 F(x_k)$. The correction $\delta_k = x_{k+1} - x_k$ is chosen as the step to the minimizer of M_F^k, or, equivalently, as the step to the zero of the *local linear model* of ∇F, which is given by

$$m_{\nabla F}^k(x_k + \delta) := \nabla F(x_k) + B_k \, \delta.$$

Hence δ_k is calculated by solving the linear system

$$B_k \, \delta_k = -\nabla F(x_k). \tag{3.10}$$

Then the key idea is to use the condition

$$\nabla F(x_k) = m_{\nabla F}^{k+1}(x_{k+1} - \delta_k) \equiv \nabla F(x_{k+1}) + B_{k+1}(-\delta_k)$$

in determining the approximate derivative matrix B_{k+1} for the new local models $m_{\nabla F}^{k+1}$ and M_F^{k+1}. This generalized secant rule leads to

$$B_{k+1}\delta_k = \nabla F(x_{k+1}) - \nabla F(x_k) =: \gamma_k, \tag{3.11}$$

which is called the quasi-Newton (or secant) condition. For $n = 1$, (3.11) uniquely determines B_{k+1}. For $n > 1$, further conditions must be imposed on B_{k+1}. We could use, for instance,

$$B_{k+1} z = B_k z \;\text{ for all } z \text{ such that } \; z^T \delta_k = 0,$$

that is, require that B_{k+1} should not differ from B_k on the orthogonal complement of δ_k, and obtain the simple rank-one correction formula

$$B_{k+1} = B_k + \frac{(\gamma_k - B_k \delta_k) \, \delta_k^T}{\delta_k^T \delta_k},$$

which is *Broyden's update*, the most commonly used secant method for solving general systems of nonlinear equations (where B_k cannot be interpreted

as the Hessian matrix of a scalar function). Broyden's method is called a *least-change secant update*, because B_k is changed as little as possible (in the Frobenius norm) consistent with satisfying the secant condition (3.11).

However, in the context of minimization, there exist more efficient alternatives to Broyden's method, since it is possible to exploit the special properties of the Hessian matrix which we are trying to approximate. Specifically, the update should preserve symmetry (B_k symmetric $\Rightarrow$ B_{k+1} symmetric). Also, it should preserve positive definiteness (B_k positive definite $\Rightarrow$ B_{k+1} positive definite) whenever

$$\delta_k^T \gamma_k = \delta_k^T B_{k+1} \delta_k > 0 \tag{3.12}$$

is satisfied, which is the necessary condition for (3.11) to have a positive definite solution B_{k+1}. The most successful update having these desirable features is the *Broyden-Fletcher-Goldfarb-Shanno (BFGS) update* which is usually written in the form

$$B_{k+1} = B_k + \frac{\gamma_k \gamma_k^T}{\gamma_k^T \delta_k} - \frac{B_k \delta_k \delta_k^T B_k}{\delta_k^T B_k \delta_k}. \tag{3.13}$$

Another rank-two correction formula closely related to (3.13) is the *Davidon-Fletcher-Powell (DFP) update*[‡]

$$B_{k+1} = B_k + \frac{(\gamma_k - B_k \delta_k)\gamma_k^T + \gamma_k(\gamma_k - B_k \delta_k)^T}{\gamma_k^T \delta_k} - \frac{\delta_k^T (\gamma_k - B_k \delta_k)\gamma_k \gamma_k^T}{(\gamma_k^T \delta_k)^2}. \tag{3.14}$$

There exist other methods that do not guarantee that positive definiteness is preserved whenever (3.12) holds, such as the *Powell symmetric Broyden (PSB) update*. It can be shown that under suitable assumptions quasi-Newton methods are *locally q-superlinearly convergent*. Convergence can be globalized by including a line search or trust region method. If a line search is used, it is always possible to satisfy condition (3.12) by choosing a suitable positive steplength (for a proof, see [Denn89]), and thus positive definiteness of B_k can be preserved at each iteration without skipping updates even at points far from the solution, where the true Hessian matrix may or may not be positive definite.

[‡]The BFGS and DFP updates are called *complementary* or *dual* updates, because the BFGS formula is actually derived as a symmetric positive definite update for the inverse $H := B^{-1}$, and the resulting formula (which is related to the DFP update by the transformation $\delta \leftrightarrow \gamma$, $B \leftrightarrow H$) is then "inverted" to obtain (3.13).

We briefly remark that it is unnecessary to solve the linear system (3.10) "from scratch" on each iteration, because a given matrix factorization of B_k can be updated very efficiently with an operation count of just $O(n^2)$. Therefore, the use of quasi-Newton updates not only eliminates the need to calculate second derivatives, but also considerably reduces the amount of work necessary to calculate the correction δ_k from (3.10). Still another possibility is to implement the update directly in terms of $H_k := B_k^{-1}$; the corresponding correction formulas for H_k can be easily derived using the Sherman-Morrison formula [Denn77]. However, the former method is preferable because of better numerical stability. A further discussion of quasi-Newton methods for unconstrained minimization and general root-finding is beyond the scope of our presentation, and the reader is referred to one of the excellent introductions available, for instance [Denn77] or [Denn89].

3.2.2 The SQP Methods of Han and Powell

Quite similar constructions can be applied to approximate the quadratic portion of the SQP subproblem, as first suggested by Garcia-Palomares and Mangasarian [Garc76]. The idea was further developed by Han, who showed in [Han76] that if the Hessian $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$ in the QP subproblem (3.4) is replaced by an approximation B_k, where B_k is calculated using either the DFP update (3.14), the PSB update, or a scaled version of the latter, with

$$\begin{aligned} \delta_k &:= y_{k+1} - y_k \equiv \alpha_k p_k && (3.15) \\ \gamma_k &:= \nabla_y L(y_{k+1}, \lambda_{k+1}, \mu_{k+1}) - \nabla_y L(y_k, \lambda_{k+1}, \mu_{k+1}), && (3.16) \end{aligned}$$

then the resulting SQP method is locally q-superlinearly convergent in all variables (y, λ, μ) simultaneously and with constant steplength $\alpha_k \equiv 1$, provided the KKT point (y^*, λ^*, μ^*) satisfies the Jacobi uniqueness condition and the initial estimate B_0 is sufficiently close to the true Hessian $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ at the solution. For the DFP update, it must be assumed in addition that $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ is positive definite, but this requirement can (at least in theory) be removed by solving an equivalent convexified problem instead of the original problem. In the follow-up paper [Han77a], Han proved global convergence if a line search using the well-known l_1-penalty function

$$P_1(y, \rho) = F(y) + \rho \left(\sum_{i=1}^{m} |h_i(y)| + \sum_{j=1}^{l} |\min(0, g_j(y))| \right) \qquad (3.17)$$

(ρ sufficiently large, but finite) is incorporated into the method, provided however, that the Lagrange multipliers λ_k and μ_k remain bounded for all k (by ρ in the ∞-norm) and that there exist two positive numbers β_1 and β_2 such that

$$\beta_1 \, p^T p \leq p^T B_k \, p \leq \beta_2 \, p^T p$$

for all k and any $p \in \mathbb{R}^n$ (which implies boundedness of B_k and B_k^{-1}). It should be pointed out that for global convergence alone, it is not necessary that the matrices B_k closely approximate the true Hessian matrix; one could use an arbitrary positive definite matrix throughout all iterations (e.g. $B_k \equiv I$), but then the method would converge very slowly (compare with the method of steepest descent in the unconstrained case). From a practical point of view, Han's results leave open many questions—it is not clear, for instance, how to choose a suitable value for the penalty parameter ρ, how to guarantee the boundedness of B_k and B_k^{-1} for a sequence of quasi-Newton updates, and whether B_k will eventually be "sufficiently close" to $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ if one starts the algorithm far from y^* with some arbitrary initial approximation B_0. Han himself did not present any numerical results.

It was Powell who, in 1977, presented an ingenious working version of an SQP method based on these ideas [Powe78a]. The numerical performance of this particular algorithm was so impressive that it has sparked an increasing interest in SQP methods. Powell suggested the use of a modified BFGS update which keeps the Hessian approximations B_k positive definite, although $\nabla_y^2 L_k$ and even $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ may be indefinite. We will discuss the justification and the implications of this strategy below, but first an outline of Powell's method shall be given.

With the step δ_k defined in (3.15) and the change in gradient of the Lagrangian function γ_k given in (3.16), it is not always possible to ensure that the condition $\delta_k^T \gamma_k > 0$ holds—it may happen that $\delta_k^T \gamma_k$ is negative for *all* values $\alpha_k > 0$ (recall that unlike in the unconstrained case, where (3.12) could always be satisfied, we now face the difficulty that y^* is in general *not* an unconstrained minimizer of the Lagrangian function $L(y, \lambda^*, \mu^*)$, and hence p_k is not necessarily a descent direction for L). In order to cope with this situation, Powell replaced γ_k by a vector η_k of the form

$$\eta_k := \Theta_k \gamma_k + (1 - \Theta_k) \, B_k \delta_k, \quad 0 < \Theta_k \leq 1, \tag{3.18}$$

where the parameter Θ_k is determined such that η_k is the vector closest to γ_k that satisfies the condition

$$\delta_k^T \eta_k \geq \epsilon_\Theta \, \delta_k^T B_k \delta_k > 0 \tag{3.19}$$

with an empirically chosen factor $\epsilon_\Theta \in [0.1, 0.2]$, see [Powe84]. Therefore, Θ_k has the value

$$\Theta_k = \begin{cases} 1 & \text{if } \delta_k^T \gamma_k \geq \epsilon_\Theta \, \delta_k^T B_k \delta_k \\ \dfrac{(1-\epsilon_\Theta)\, \delta_k^T B_k \delta_k}{\delta_k^T B_k \delta_k - \delta_k^T \gamma_k} & \text{otherwise} \end{cases} \tag{3.20}$$

(note that this definition implies that $\delta_k^T \eta_k = \epsilon_\Theta \, \delta_k^T B_k \delta_k$ in the case $\delta_k^T \gamma_k < \epsilon_\Theta \, \delta_k^T B_k \delta_k$). The vector η_k is now used instead of γ_k in the BFGS formula (3.13), and condition (3.19) ensures that this update preserves positive definiteness. However, the original secant condition (3.11) is lost whenever $\eta_k \neq \gamma_k$. On the basis of empirical testing, Powell proposed that the line search be carried out using the modified l_1-penalty function

$$P_P(y, \sigma, \tau) = F(y) + \sum_{i=1}^{m} \sigma_i \, |h_i(y)| + \sum_{j=1}^{l} \tau_j \, |\min(0, g_j(y))|, \tag{3.21}$$

where on the first iteration $k = 0$ the weights $\sigma_{0,i} = |\tilde{\lambda}_{0,i}|$ and $\tau_{0,j} = \tilde{\mu}_{0,j}$ are used, and on all subsequent iterations $k = 1, 2, \ldots,$

$$\sigma_{k,i} = \max\left(|\tilde{\lambda}_{k,i}|, \, \frac{1}{2}(\sigma_{k-1,i} + |\tilde{\lambda}_{k,i}|) \right) \tag{3.22a}$$

$$\tau_{k,j} = \max\left(\tilde{\mu}_{k,j}, \, \frac{1}{2}(\tau_{k-1,j} + \tilde{\mu}_{k,j}) \right) \tag{3.22b}$$

(Recall that $\tilde{\lambda}_k$, $\tilde{\mu}_k$ denote the multiplier vectors of the QP subproblem that defines the search direction p_k.) This construction is motivated by the observation that using one large, constant weight for all constraints on all iterations as in Han's l_1-penalty function (3.17) is often inefficient, because most of the time this gives too much emphasis to reducing the constraint infeasibilities, thus resulting in very small steps along the curved constraint surface. Therefore, the single constant ρ has been replaced by individual weights $\sigma_{k,i}$, $\tau_{k,j}$ which are allowed to vary from iteration to iteration.[‡] The line search could now in principle be performed by minimizing the univariate function $T(\alpha)$ defined by

$$T(\alpha) := P_P(y_k + \alpha p_k, \sigma_k, \tau_k), \tag{3.23}$$

[‡]Note that $\sigma_{k,i} \geq |\tilde{\lambda}_{k,i}|$ and $\tau_{k,j} \geq \tilde{\mu}_{k,j}$ always hold. The reduction of the weights for decreasing multiplier values is "delayed", which allows the method to include positive contributions in the function (3.21) from constraints that have been previously active, but are not in the active set at the solution of the current QP subproblem. This seems to be an advantage in a number of cases, as has been observed in numerical experimentation.

but an exact minimization is usually not justified because it requires too many function evaluations. Powell suggests the use of quadratic interpolation to generate a decreasing sequence of trial values $\hat{\alpha}_\ell$ (starting with $\hat{\alpha}_0 = 1$) until the condition

$$T(\hat{\alpha}_\ell) \;\leq\; T(0) + 0.1\,\hat{\alpha}_\ell \frac{dT}{d\alpha}(0) \tag{3.24}$$

is satisfied, and then the steplength α_k is set to the corresponding value $\hat{\alpha}_\ell$. We will discuss the details of the line search procedure used in MUSCOD at the end of this section.

In order to check whether convergence to a local optimum has been achieved (within some acceptable tolerance), a convergence criterion must be defined. One popular choice is the so-called KKT tolerance, i.e., the algorithm is terminated with y_k as solution if

$$|\nabla F(y_k)^T p_k| + \sum_{i=1}^{m} |\tilde{\lambda}_{k,i}\, h_i(y_k)| + \sum_{j=1}^{l} |\tilde{\mu}_{k,j}\, g_j(y_k)| \;\leq\; \epsilon_t, \tag{3.25}$$

taking into account the weighted sum of possible objective function improvement and constraint violations [Chen84]. The convergence parameter ϵ_t has the same units as the objective function. Note that this termination check can be applied directly after the solution of the current QP subproblem (before the line search is performed).

Keeping the matrices B_k positive definite even when the true Hessian matrix $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ is not has the great practical advantage that the resulting QP subproblems are *convex* and hence easier to solve. However, we must give some explanation for the suitability of this strategy. When y_k is near the solution y^*, then the linear approximations to the constraints will usually cause the constraints to be satisfied quite well, and the final search directions lie almost wholly in the null space of $A(y^*)$. This is due to the fact that the range space components of p_k are completely determined by the linearized constraints (3.2)–(3.3) and are therefore independent of B_k, while the determination of the null space components of p_k is based on the accumulation of approximate second derivative information in B_k, i.e., there is a disparity in the quality of information which causes the active constraints to converge to zero faster than the reduced gradient [Gill89]. It follows that on the final block of iterations (during the "local" phase of the algorithm) the purpose of the approximations B_k is mainly to control the choice of a suitable search direction within the null space of $A(y^*)$, and within this

subspace, $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ is known to be positive definite. Therefore, it may be expected that a positive definite matrix B_k allows the "important" part of the Hessian matrix to be approximated well, even if $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$ has some negative eigenvalues [Powe78c]. Furthermore, once the steps δ_k are more or less entirely in the null space of $A(y^*)$, the corresponding gradient differences γ_k will usually be consistent with a positive definite approximation, i.e. the condition $\delta_k^T \gamma_k > 0$ will be satisfied, and the standard BFGS update is used.

3.2.3 Strengths and Weaknesses of Powell's SQP Method

As mentioned, Powell's method was the first "working" SQP algorithm, and its great practical success has sparked an on-going effort to develop even more efficient and robust implementations. Today, SQP is considered one of the most promising general techniques for solving the NLP problem in the differentiable case. The primary advantage over other gradient-based methods is the very low number of *function evaluations* needed to converge to a solution, which is particularly favorable when function evaluations are expensive, as in case of the parameterized optimal control problem (P2).[‡]

Basically two facts contribute to the excellent performance of the method on a wide range of problems. On one hand, the need to maintain strict feasibility at each iteration is eliminated through the use of constraint linearizations, and hence relatively large steps can be taken toward the solution without having to solve (iteratively) the original nonlinear constraints (*infeasible path* strategy). On the other hand, the linearized constraints provide a sensible restriction of the search directions in a way that does not depend on the Hessian approximation. Thus, when many constraints are active at the final solution, there is less dependence on second derivative information, and it becomes easier to find a Hessian approximation that is suitable.

Powell's constrained variable metric algorithm retains one important feature of its unconstrained counterparts: it is invariant under linear transformations of the variables, except for the choice of the initial Hessian approximation B_0 [Powe78a]. The method itself should therefore be insensitve to poor scaling of variables. However, since this is not necessarily true for practical implementations using finite-precision arithmetic, the variables should nevertheless be properly scaled. As we have already discussed in Section 1, a one-time scaling of the problem variables—if possible based on some charac-

[‡] There is, of course, a tradeoff between the amount of *additional* work per iteration (i.e., work not related to function evaluations), which is relatively high for SQP methods, and the required total number of iterations.

teristic, physically meaningful order of magnitude for each variable—before starting the calculations is usually appropriate, although there might be cases justifying more complicated *adaptive* scaling strategies. In addition, it seems to be advisable to scale the constraint functions and the objective function by their respective magnitudes, see [Bieg85, Chen84].

One disadvantage of Powell's SQP method is that it is not directly applicable to large problems: for an efficient QP solution, there is a certain limit on the size of the QP subproblems (they should have no more than about 500 variables if a standard non-sparse QP solver is used). However, for structured problems, it is possible to employ efficient elimination schemes in order to reduce the size of the QP subproblems to be solved. An interesting example of such a strategy is reported by Berna et al. in the context of equation-based chemical process optimization [Berna80]: their large-scale modification of Powell's algorithm found the *optimal* solution to a steady state problem using about the same number of function and gradient evaluations as required by available nonoptimizing simulation packages to simply obtain a single *feasible* solution. Similarly, the condensing algorithm of MUSCOD takes advantage of the special structure of problem (P2) to greatly reduce the size of the QP subproblem (see Section 6). An important consideration in large-scale applications is the possible occurrence of inconsistent linearized constraint sets, which has to be dealt with by a suitable constraint relaxation. Relatively simple strategies have been proposed by Powell [Powe78a] and Biegler [Bieg85]. Further discussion of issues arising in the adaptation of QP-based methods to larger scale problems is given by Gill et al. [Gill81b].

It is important to note that much of the practical success of Powell's algorithm depends on *heuristic elements* (for instance the details of the modified BFGS update and the weight selection scheme for the penalty function P_P), and that these heuristics can lead to failure in certain cases. Neither the local nor the global convergence results of Han are applicable to Powell's method. Powell himself proved the following result on the order of convergence of his method [Powe78b]: assuming that (y^*, λ^*, μ^*) satisfies the Jacobi uniqueness condition, and that the sequence of iterates y_k generated with steplength $\alpha_k \equiv 1$ converges to y^*, and also that the matrices B_k remain bounded for all k, then the rate of convergence is locally *r-superlinear*. However, some of these assumptions are frequently not valid, thus impeding the superlinear rate of convergence. In addition, *global* convergence of the method cannot be guaranteed because of the modifications in the penalty function (variable weights), as was already mentioned by Powell in his defin-

ing paper [Powe78a]. In fact, the following difficulties with Powell's method have been reported in the literature:

1. *Maratos effect*: It may happen that $\alpha_k < 1$ has to be chosen as steplength, even when y_k is very close to y^* and B_k is an excellent approximation to $\nabla_y^2 L(y^*, \lambda^*, \mu^*)$. Hence the rate of convergence is only linear. Possible remedies have been proposed by Mayne [Mayne80] and by Chamberlain et al. [Cham82].

2. *Cycling behaviour*: Examples have been found where the method cycles [Cham79]. This failure of global convergence is attributed to the weight selection scheme (3.22); an alternative strategy ("watchdog technique") is discussed in [Cham82, Powe82, Hoza93].

3. *Unboundedness of B_k*: The matrices B_k may not remain bounded for $k \to \infty$ as shown by a simple example of Chamberlain [Cham79].

4. *Ill-conditioning of B_k*: The modified BFGS update can give highly ill-conditioned Hessian approximations B_k, which invalidates the search directions p_k [Plitt81, Powe84].

The version of Powell's method defined in [Powe78a] is available as subroutine VF12 (originally VF02) in the Harwell Library, and the modifications from [Cham82] and [Powe82] have been implemented in subroutine VF13 (originally VF03), which also uses a more efficient QP solver than VF12. These improvements have led to significantly enhanced robustness and performance of the method, and subroutine VF12 is now considered obsolete (see also [Chen84, Bieg85]).[‡]

Another very successful modification of Powell's method has been developed by Schittkowski [Schi81, Schi85]. This method, available as subroutine NLPQL in the International Mathematical and Statistical Libraries (IMSL), uses an *augmented Lagrangian function* instead of a l_1-penalty function as merit function and has been shown to be considerably more efficient and robust than VF12 [Schi85]; it also performs better than VF13 on many test problems [Powe84].

There is still no generally accepted solution to the ill-conditioning problem mentioned under point 4 above. A quite promising approach has been recently presented by Byrd, Tapia and Zhangh [Byrd92]. In certain cases,

[‡]While the original version of MUSCOD is based on the SQP method described in [Powe78a], there is work under way to test alternative approaches as well.

it could also be beneficial to use a *reduced SQP method* [Gabay82, Noce85, Byrd91], where only an approximation to the reduced Hessian is updated. However, this becomes complicated when inequalities are present (in practice, it is advantageous to employ only *partially* reduced methods [Locke83, Schu95]). In MUSCOD, a *restart* is made when ill-conditioning of the Hessian approximation is detected. Although unsatisfactory from a theoretical point of view, this strategy works quite well in practice. Another approach along the same lines would be to work with *limited memory updates* at least during the initial block of iterations. There the multiplier estimates λ_k and μ_k are likely to be far from λ^* and μ^*, and hence the accumulated curvature information is of little relevance anyway.

3.3 The QP Subproblem $Q(y_k, B_k)$

We now return to our discretized optimal control problem (P2) (see equations (1.19a)–(1.19d)) and introduce the corresponding QP subproblem in its specific form. The Lagrangian function of (P2) is given by

$$\begin{aligned} L(y,\lambda,\mu) &= F(y) - \lambda^T h(y) - \mu^T g(y) \\ &= \sum_{j=0}^{m-1} F_j(s_j,q_j) - \sum_{j=0}^{m-1} \lambda_j^T (x(t_{j+1};s_j,q_j) - s_{j+1}) \\ &\quad - \lambda_m^T (r_a(s_0) + r_b(s_m)) - \sum_{j=0}^{m-1} \mu_j^T g_j(q_j), \end{aligned} \tag{3.26}$$

where $\lambda_m \in \mathbb{R}^{l_r}$, $\lambda_j \in \mathbb{R}^n$, $\mu_j \in \mathbb{R}^{l_j}$, $s_j \in \mathbb{R}^n$, and $q_j \in \mathbb{R}^{k_j}$ for $j = 0, 1, \ldots, m-1$. The composite parameter vector

$$y = (s_0, q_0, \ldots, s_{m-1}, q_{m-1}, s_m) \in \mathbb{R}^{\hat{n}}, \quad \hat{n} = n(m+1) + \sum_{j=0}^{m-1} k_j$$

has been already defined in (1.17), and the composite multiplier vectors λ, μ are of the form

$$\lambda := (\lambda_0, \ldots, \lambda_m) \in \mathbb{R}^{\hat{m}}, \quad \hat{m} = mn + l_r \tag{3.27a}$$

$$\mu := (\mu_0, \ldots, \mu_{m-1}) \in \mathbb{R}^{\hat{l}}, \quad \hat{l} = \sum_{j=0}^{m-1} l_j. \tag{3.27b}$$

For a given point $y_k \in \mathbb{R}^{\hat{n}}$ and a given symmetric, positive definite matrix $B_k \in \mathbb{R}^{\hat{n}\times\hat{n}}$, which is intended to be a suitable approximation to the Hessian matrix $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$, we define the following quadratic programming problem $Q(y_k, B_k)$,

$$\min_p \ \nabla F(y_k)^T p + \frac{1}{2} p^T B_k \, p \tag{3.28a}$$

subject to

$$h(y_k) + \nabla h(y_k)^T p \ = \ 0 \tag{3.28b}$$

$$g(y_k) + \nabla g(y_k)^T p \ \geq \ 0, \tag{3.28c}$$

where the original constraints of (P2) have been linearized at y_k. Therefore, dropping the iteration index k, we have

$$\nabla F(y) = \begin{pmatrix} \nabla_{s_0} F_0(s_0, q_0) \\ \nabla_{q_0} F_0(s_0, q_0) \\ \vdots \\ \nabla_{s_{m-1}} F_{m-1}(s_{m-1}, q_{m-1}) \\ \nabla_{q_{m-1}} F_{m-1}(s_{m-1}, q_{m-1}) \\ 0 \end{pmatrix} = \begin{pmatrix} z_0^s \\ z_0^q \\ \vdots \\ z_{m-1}^s \\ z_{m-1}^q \\ 0 \end{pmatrix} \in \mathbb{R}^{\hat{n}} \tag{3.29}$$

$$h(y) = \begin{pmatrix} x(t_1; s_0, q_0) - s_1 \\ \vdots \\ x(t_m; s_{m-1}, q_{m-1}) - s_m \\ r_a(s_0) + r_b(s_m) \end{pmatrix} \in \mathbb{R}^{\hat{m}} \tag{3.30}$$

$$\nabla h(y)^T = \begin{pmatrix} X_0^s & X_0^q & -I & & & & \\ & & X_1^s & X_1^q & -I & & \\ & & & & \ddots & & \\ & & & & X_{m-1}^s & X_{m-1}^q & -I \\ R_a & & & & & & R_b \end{pmatrix} \in \mathbb{R}^{\hat{m}\times\hat{n}} \tag{3.31}$$

$$g(y) = \begin{pmatrix} g_0(q_0) \\ \vdots \\ g_{m-1}(q_{m-1}) \end{pmatrix} \in \mathbb{R}^{\hat{l}} \tag{3.32}$$

$$\nabla g(y)^T = \begin{pmatrix} 0 & G_0 & 0 & & & \\ & & 0 & G_1 & 0 & \\ & & & & \ddots & \\ & & & & 0 & G_{m-1} & 0 \end{pmatrix} \in \mathbb{R}^{\hat{l}\times\hat{n}} \tag{3.33}$$

with the abbreviations

$$\begin{aligned} z_j^s &:= \nabla_{s_j} z(t_{j+1}; s_j, q_j) \in \mathbb{R}^n \\ z_j^q &:= \nabla_{q_j} z(t_{j+1}; s_j, q_j) \in \mathbb{R}^{k_j} \\ X_j^s &:= \nabla_{s_j} x(t_{j+1}; s_j, q_j)^T \in \mathbb{R}^{n \times n} \\ X_j^q &:= \nabla_{q_j} x(t_{j+1}; s_j, q_j)^T \in \mathbb{R}^{n \times k_j} \\ R_a &:= \nabla r_a(s_0)^T \in \mathbb{R}^{l_r \times n} \\ R_b &:= \nabla r_b(s_m)^T \in \mathbb{R}^{l_r \times n} \\ G_j &:= \nabla g_j(q_j)^T \in \mathbb{R}^{l_j \times k_j} \end{aligned} \tag{3.34}$$

for $j = 1, 2, \ldots, m-1$. (Recall from Section 2 that $\nabla h(y)^T$ denotes the Jacobian matrix of the vector function $h(y)$.)

If the boundary equality conditions (1.4c) are replaced by the inequalities (1.5) in problem (P1), the corresponding modifications to $Q(y_k, B_k)$ are straightforward: the boundary terms $r_a(s_0) + r_b(s_m)$ are included in $g(y)$ instead of $h(y)$, and the block row containing R_a, R_b is removed from $\nabla h(y)^T$ and appended to $\nabla g(y)^T$.

Remember that evaluating $z(t_{j+1}; s_j, q_j)$, $x(t_{j+1}; s_j, q_j)$ and the derivatives z_j^s, z_j^q, X_j^s, X_j^q requires numerical integration and can be quite expensive. An efficient and reliable way to calculate the required derivatives will be discussed in Section 4. In principle, the subproblem $Q(y_k, B_k)$ could be solved directly in the form stated above, but we will see in Section 6 that the special structure of $\nabla h(y)^T$ allows the elimination of the variables s_j, $j = 1, 2, \ldots, m$, by an efficient condensing procedure, and in effect only a much smaller quadratic programming problem has to be solved.

Note that we have not yet included the additional linear equality constraints (1.9)—which have to be imposed for continuity of the control parameterization at the inner mesh points—in the subproblem formulation. The efficient treatment of this special type of constraints will be addressed in Section 7 in the context of interior point constraints.

3.4 Basic Steps of the Algorithm

Having made these preparations, we now give a simplified model algorithm for the solution of problem (P2), based on Powell's SQP method [Powe78a]. However, we shall see that this algorithm has to be refined in several respects in order to be successful in practice, and we will discuss these modifications

in subsequent sections. We also assume that all variables and problem functions have been properly scaled as discussed in Section 1. A complete version of the algorithm will be presented in Section 7.

Given an initial point y_0, set $k = 0$ and execute the following steps:

M1. *Evaluate the functions and gradients at the initial point.* Calculate $F(y_0)$, $h(y_0)$, $g(y_0)$, $\nabla F(y_0)$, $\nabla h(y_0)^T$, and $\nabla g(y_0)^T$.

M2. *Choose an initial Hessian approximation.* Choose a suitable symmetric, positive definite matrix B_0 as initial estimate for the Hessian. This completes the formulation of $Q(y_0, B_0)$.

M3. *Solve the current QP subproblem.* Calculate a KKT point $(p_k, \tilde{\lambda}_k, \tilde{\mu}_k)$ of $Q(y_k, B_k)$.

M4. *Check the termination criterion.* If a given convergence criterion is satisfied, e.g.,

$$|\nabla F(y_k)^T p_k| + \sum_{i=1}^{\hat{m}} |\tilde{\lambda}_{k,i}\, h_i(y_k)| + \sum_{j=1}^{\hat{l}} |\tilde{\mu}_{k,j}\, g_j(y_k)| \;\leq\; \epsilon_t,$$

terminate with y_k as the solution.

M5. *Update the estimate of the solution.* Set

$$y_{k+1} = y_k + \alpha_k p_k,$$

where α_k $(0 < \alpha_k \leq 1)$ is a steplength parameter determined by a line search procedure, and set

$$\lambda_{k+1} = \tilde{\lambda}_k, \;\; \mu_{k+1} = \tilde{\mu}_k.$$

M6. *Calculate the gradient of the Lagrangian function at the old point.* Calculate $\nabla_y L(y_k, \lambda_{k+1}, \mu_{k+1})$ from the gradients $\nabla F(y_k)$, $\nabla h(y_k)^T$, and $\nabla g(y_k)^T$ (but using the best currently available multiplier estimates λ_{k+1}, μ_{k+1}).

M7. *Evaluate the functions and gradients at the new point.* Calculate $F(y_{k+1})$, $h(y_{k+1})$, $g(y_{k+1})$, $\nabla F(y_{k+1})$, $\nabla h(y_{k+1})^T$, and $\nabla g(y_{k+1})^T$.

M8. *Calculate the "new" gradient of the Lagrangian function.* Calculate $\nabla_y L(y_{k+1}, \lambda_{k+1}, \mu_{k+1})$ from the gradients $\nabla F(y_{k+1})$, $\nabla h(y_{k+1})^T$, and $\nabla g(y_{k+1})^T$.

M9. *Revise the Hessian approximation.* Calculate a symmetric, positive definite matrix B_{k+1} by a suitable update procedure,

$$B_{k+1} = B_k + U(B_k, \delta_k, \gamma_k),$$

using $\delta_k := y_{k+1} - y_k$ and the gradient difference

$$\gamma_k := \nabla_y L(y_{k+1}, \lambda_{k+1}, \mu_{k+1}) - \nabla_y L(y_k, \lambda_{k+1}, \mu_{k+1}).$$

This completes the formulation of $Q(y_{k+1}, B_{k+1})$.

M10. *Start a new major iteration.* Increment k by one, and return to step M3.

Note that this sequence of steps properly reflects the fact that the gradients at the *new* point y_{k+1} must be known in order to be able to calculate γ_k and to revise the Hessian (since the old function and derivative values are no longer needed after step M6, they can simply be overwritten by the new ones in step M7).

From an implementational point of view, the following major blocks could be used to build the algorithm:

- Function and gradient evaluation (for steps M1 and M7)
- Hessian initialization (for step M2)
- QP solution (for step M3)
- Termination check (for step M4)
- Line search (for step M5)
- Lagrangian gradient calculation (for steps M6 and M8)
- Hessian update (for step M9)

In Section 7, we will discuss the functionality and the interrelations of these blocks (and some additional ones not mentioned here) for the complete version of the algorithm. Now we turn to the details of the line search procedure as implemented in the original version of MUSCOD.

3.5 The Line Search Strategy

The line search strategy proposed by Powell in [Powe78a] to find an acceptable steplength α_k is based on the well-known Armijo-Goldstein criteria. Methods of this type are frequently used in unconstrained optimization (for details on the theory, see for instance [Denn89]). Like Powell's method, MUSCOD uses the univariate function $T(\alpha)$ defined in (3.23) (based on the penalty function P_P in (3.21)), and the descent condition (3.24). However, the procedure for choosing α_k is slightly different: instead of starting each sequence of trial steplengths $\hat{\alpha}_\ell$, $\ell = 0, 1, \ldots$, with the value $\hat{\alpha}_0 = 1$, a predictor-corrector strategy is used.[‡] On the first major iteration ($k = 0$), a special *start relaxation* is used instead of the line search [Plitt81].

The start relaxation simply ensures that the norm of the first step taken, which is given by $\|\delta_0\| \equiv \alpha_0 \|p_0\|$, does not become too large compared to the norm of the start vector $\|y_0\|$ (the line search function $T(\alpha)$ is *not* used on the first iteration). This heuristic strategy usually gives a reasonable first step, provided a suitable choice has been made for the initial Hessian estimate B_0, as will be discussed in Section 5. If $p_0 \neq 0$ (otherwise, the algorithm terminates), the first steplength is chosen to be

$$\alpha_0 = \min(1,\ \kappa_0 \|y_0\| / \|p_0\|), \tag{3.35}$$

where κ_0 is a given constant. This implies that $\|\delta_0\| \leq \kappa_0 \|y_0\|$. In MUSCOD, the value $\kappa_0 = 0.2$ is used ($\|\cdot\|$ stands for the l_2-norm).[§]

On all subsequent iterations $k = 1, 2, \ldots$, a simple predictor strategy is employed in order to determine a good value for the first trial steplength $\hat{\alpha}_0$, and if necessary, a decreasing sequence of further trial steplengths $\hat{\alpha}_\ell$, $\ell = 1, 2, \ldots$, is generated by the corrector procedure until condition (3.24) is satisfied. The following algorithm is used to calculate α_k on the iterations $k = 1, 2, \ldots$:

LS1. *Predict a suitable starting value for the sequence of trial steplengths.* If on the previous iteration $k - 1$, no correction has been applied to the steplength (which is trivially true for the start relaxation), and if at the same time the condition

$$\|\delta_{k-1}\| \leq \kappa_1 \|y_{k-1}\|$$

[‡] The rationale behind this predictor-corrector strategy is to minimize the number of line search iterations. There is, however, empirical evidence that this strategy may degrade the overall convergence behaviour, and therefore, its use cannot be generally recommended.

[§] It has been found that in many cases, it is actually better *not* to use this kind of start relaxation (i.e., always set $\alpha_0 = 1$).

has been satisfied with a given constant κ_1 ($\kappa_1 = 0.2$ in MUSCOD), then try an increased starting value and set

$$\hat{\alpha}_0 = \min(2\,\alpha_{k-1},\ 1),$$

otherwise just use the steplength from the previous iteration and set

$$\hat{\alpha}_0 = \alpha_{k-1}.$$

LS2. *If necessary, correct the trial values until the descent condition holds.* Set $\ell = 0$ and repeat the following two substeps:

LS2.1. *Test the descent condition.* If

$$T(\hat{\alpha}_\ell) \le T(0) + 0.1\,\hat{\alpha}_\ell \frac{dT}{d\alpha}(0)$$

is satisfied, then the current trial value $\hat{\alpha}_\ell$ can be accepted; go to step LS3.

LS2.2. *Generate a new trial value.* Build the quadratic approximation $M_T^\ell(\alpha)$ to $T(\alpha)$ which is defined by the equations

$$M_T^\ell(0) = T(0),\ \ \frac{d}{d\alpha}M_T^\ell(0) = \frac{d}{d\alpha}T(0),\ \ M_T^\ell(\hat{\alpha}_\ell) = T(\hat{\alpha}_\ell),$$

and let $\hat{\alpha}_{\ell+1}$ be the greater of $0.1\,\hat{\alpha}_\ell$ and the value of α which minimizes $M_T^\ell(\alpha)$, i.e.,

$$\hat{\alpha}_{\ell+1} = \max\left(0.1\,\hat{\alpha}_\ell,\ \arg\min M_T^\ell(\alpha)\right).$$

Increment ℓ by one, and return to step LS2.1.

LS3. *Terminate the line search.* Set $\alpha_k = \hat{\alpha}_\ell$ and stop.

The start relaxation (3.35) and the predictor LS1 guarantee that $\alpha_k \le 1$ for all k (the sequence of trial values generated by the corrector LS2 is decreasing). Step LS2.2 contains a safeguard against unreasonably small steplengths by requiring the new trial value to be at least 10% of the current one. This is important in case $M_T^\ell(\alpha)$ is a poor approximation to $T(\alpha)$. For further details on the quadratic approximation procedure and on the calculation of $dT(0)/d\alpha$, see [Powe78a].

4 Efficient Gradient Generation

4.1 Some ODE Theory

For future reference, we list a few general results from the theory of ordinary differential equations. In particular, we consider initial value problems (IVPs) of the form

$$\dot{x} = f(t, x, q), \quad x(t_0) = s, \quad t_0, t \in [a, b] \tag{4.1}$$

where $x \in \mathbb{R}^n$ is the state vector, $s \in \mathbb{R}^n$ is the vector of initial values, $q \in \mathbb{R}^k$ is a constant parameter vector, and $f : [a, b] \times \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^n$ is a given vector function. We are interested in conditions that ensure the existence and uniqueness of a solution $x(t; s, q)$, and we will also examine how this solution depends on the initial values s and on the parameters q. To this end, let us first state a fundamental existence and uniqueness theorem for the IVP (4.1). (A proof of this theorem can be found in the classical text [Henr62].)

Theorem 4.1 (Existence and Uniqueness of IVP Solution)
For some fixed parameter vector $q \in \mathbb{R}^k$, let f be continuous on the strip $S := \{(t, x) \mid a \le t \le b,\ x \in \mathbb{R}^n\}$ with a, b finite. Furthermore, let there be a constant L such that

$$\|f(t, x_1, q) - f(t, x_2, q)\| \le L \|x_1 - x_2\| \tag{4.2}$$

*for all $t \in [a, b]$ and all $x_1, x_2 \in \mathbb{R}^n$ (*Lipschitz condition*).*[‡] *Then for every $t_0 \in [a, b]$ and for every $s \in \mathbb{R}^n$, there exists exactly one function $x(t)$ such that $x(t)$ is continuous and continuously differentiable for all $t \in [a, b]$ and solves the IVP (4.1) on the whole interval $[a, b]$.*

The value of the Lipschitz constant L may depend on the chosen parameter vector q. Using the mean-value theorem, it can be easily shown that the Lipschitz condition (4.2) is satisfied if the partial derivatives $\partial f_i / \partial x_j$, $i, j = 1, \ldots, n$, exist, are continuous, and are bounded on the strip S. In practice, f is usually continuous and also continuously differentiable on S, but the partial derivatives are often unbounded. This implies that the solution to the IVP (4.1) may be defined only in a certain neighborhood $N(\delta, t_0)$ of the initial point and not on all of $[a, b]$.

[‡]Here $\|\cdot\|$ denotes an arbitrary norm on $\mathbb{R}^n$. In the following, we will also use the notation $\|A\|$ for an associated consistent matrix norm with $\|I\| = 1$.

Next we cite a theorem which shows that the solution of an IVP depends continuously on the initial values (a proof can be found, e.g., in [Stoer92]).

Theorem 4.2 (Continuous Dependence on Initial Values)
For some fixed parameter vector $q \in \mathbb{R}^k$, let the function $f : S \to \mathbb{R}^n$ be continuous on the strip S and satisfy the Lipschitz condition (4.2) for all $(t, x_i) \in S$, $i = 1, 2$. Let $a \leq t_0 \leq b$. Then for the solution $x(t; s, q)$ of the IVP (4.1) we have the estimate

$$\|x(t; s_1, q) - x(t; s_2, q)\| \leq e^{L\,|t-t_0|}\, \|s_1 - s_2\| \tag{4.3}$$

for all $t \in [a, b]$, where L is the Lipschitz constant from (4.2).

Observe from condition (4.3) that the solution $x(t; s, q)$ may be very sensitive to the initial data if $\exp(L\,|t - t_0|)$ is large (this problem can be avoided by restricting the length $|t - t_0|$ of the integration interval).

It is possible to sharpen the previous theorem: under slightly more restrictive assumptions, the solution of the IVP actually depends on the initial values in a continuously differentiable manner. We have the following result (which is proved, e.g., in [Codd55]).

Theorem 4.3 (Derivatives with respect to Initial Values)
For some fixed parameter vector $q \in \mathbb{R}^k$, let f be continuous on the strip S. If, in addition, the Jacobian matrix $\nabla_x f(t, x, q)^T = (\partial f_i / \partial x_j)_{i,j}$ exists on S and is continuous and bounded there,

$$\|\nabla_x f(t, x, q)^T\| \leq L \;\; \textit{for all} \;\; (t, x) \in S$$

(implying satisfaction of the Lipschitz condition (4.2)), then the solution $x(t; s, q)$ of the IVP (4.1) is continously differentiable with respect to the initial values s for all $t \in [a, b]$ and all $s \in \mathbb{R}^n$. The derivative

$$Z(t; s, q) := \nabla_s x(t; s, q)^T = \left(\frac{\partial x(t; s, q)}{\partial \sigma_1}, \ldots, \frac{\partial x(t; s, q)}{\partial \sigma_n} \right)$$

(with $s = (\sigma_1, \ldots, \sigma_n)^T$) is the solution of the sensitivity equation

$$\dot{Z} = \nabla_x f(t, x(t; s, q), q)^T\, Z, \quad Z(t_0; s, q) = I. \tag{4.4}$$

Note that $\dot{Z} = \partial Z / \partial t$, Z, and $\nabla_x f(t, x(t; s, q), q)^T$ are $n \times n$ matrices. That is, (4.4) represents an IVP for a system of n^2 linear differential equations. We can formally obtain equation (4.4) by differentiating the identities

$$\dot{x}(t; s, q) = f(t, x(t; s, q), q), \quad x(t_0; s, q) = s$$

with respect to the initial values s. Like the solution $x(t;s,q)$ itself, the derivative $Z(t;s,q)$ may only be defined in a certain neighborhood $N(\delta,t_0)$ of the initial point if $\nabla_x f(t,x,q)^T$ is unbounded on S.

Using Theorem 4.3 it is now a simple matter to establish the corresponding result for the derivatives of the solution with respect to the parameters.

Theorem 4.4 (Derivatives with respect to Parameters)
Let the function f be continuous on the extended strip

$$\tilde{S} := \{(t,x,q) \mid a \le t \le b,\ x \in I\!R^n,\ q \in I\!R^k\}.$$

If, in addition, the Jacobian matrices $\nabla_x f(t,x,q)^T$ and $\nabla_q f(t,x,q)^T$ exist on $\tilde{S}$ and are continuous and bounded there, then the solution $x(t;s,q)$ is continuously differentiable also with respect to the parameters q for all $(t,x,q) \in \tilde{S}$. The derivative

$$W(t;s,q) := \nabla_q x(t;s,q)^T = \left(\frac{\partial x(t;s,q)}{\partial \rho_1}, \ldots, \frac{\partial x(t;s,q)}{\partial \rho_k}\right)$$

(with $q = (\rho_1, \ldots, \rho_k)^T$) is the solution of the parametric sensitivity equation

$$\dot{W} = \nabla_x f(t, x(t;s,q), q)^T\, W + \nabla_q f(t, x(t;s,q), q)^T, \quad W(t_0;s,q) = 0. \quad (4.5)$$

Here (4.5) describes an IVP for a system of nk linear inhomogeneous differential equations ($\dot{W}$, W, and $\nabla_q f(t,x(t;s,q),q)^T$ are $n \times k$ matrices). In order to prove Theorem 4.4, we treat the parameters q simply as additional constant states and define the extended IVP

$$\dot{\tilde{x}} := \begin{pmatrix} \dot{x} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} f(t,x,q) \\ 0 \end{pmatrix} =: \tilde{f}(t,\tilde{x}), \quad \tilde{x}(t_0) = \begin{pmatrix} s \\ q \end{pmatrix} =: \tilde{s}. \qquad (4.6)$$

Due to the assumptions on f made in Theorem 4.4, the newly defined $\tilde{f}$ satisfies the hypotheses of Theorem 4.3, and hence the solution $\tilde{x}(t;\tilde{s})$ of (4.6) is differentiable with respect to the extended vector of initial values $\tilde{s}$. The derivative $\tilde{Z}(t;\tilde{s}) := \nabla_{\tilde{s}} \tilde{x}(t;\tilde{s})^T$ is the solution of (4.4), that is

$$\dot{\tilde{Z}} = \nabla_{\tilde{x}} \tilde{f}(t, \tilde{x}(t;\tilde{s}))^T\, \tilde{Z}, \quad \tilde{Z}(t_0;\tilde{s}) = I.$$

Writing the same equation with partitioned matrices, we obtain

$$\underbrace{\begin{pmatrix} \dot{Z} & \dot{W} \\ 0 & 0 \end{pmatrix}}_{\dot{\tilde{Z}}} = \begin{pmatrix} \nabla_x f(t,x,q)^T & \nabla_q f(t,x,q)^T \\ 0 & 0 \end{pmatrix} \underbrace{\begin{pmatrix} Z & W \\ 0 & I \end{pmatrix}}_{\tilde{Z}}, \qquad (4.7)$$

$$\tilde{Z}(t_0;\tilde{s}) = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix},$$

where in the arguments of $\nabla_x f^T$ and $\nabla_q f^T$ we have used the shorthand x for $x(t;s,q)$. From (4.7), the result (4.5) for W—as well as (4.4) for Z—can be directly obtained, and Theorem 4.4 is proved.

We conclude this general discussion with two remarks. Up to this point, we have only considered the properties of the *exact* solution $x(t;s,q)$ to the IVP (4.1). Unfortunately, this exact solution is generally not available. The best we can do in practice is to determine *approximate* values $\eta_i := \eta(t_i;s,q)$ for the exact values $x_i := x(t_i;s,q)$ at certain discrete abscissas t_i, $i = 0,1,\ldots$, and we will have to examine carefully in which way the results valid for the exact solution x carry over to the approximate solution η. Our second remark is concerned with the somewhat different nature of *boundary value problems (BVPs)*: while IVPs are normally uniquely solvable (at least locally) according to Theorem 4.1, BVPs may also have no solution or several solutions (there is no fundamental existence and uniqueness theorem comparable to Theorem 4.1). However, for the BVPs appearing in the context of optimal control the existence of a solution can usually be assumed based on physical insight about the underlying real-life problem.

4.2 Derivatives of the Discretized Optimal Control Problem

We have seen in Section 3.3 that in order to formulate the QP subproblem $Q(y_k, B_k)$, we need the function values $h(y_k)$, $g(y_k)$ and the gradient values $\nabla F(y_k)$, $\nabla h(y_k)^T$, and $\nabla g(y_k)^T$. Now we will show that these function and derivative values exist and how they can be obtained in principle. We will also discuss the case of addititional control nodes within the multiple shooting intervals. The issues related to the practical computation of suitable approximations are deferred to the last part of this section.

Let us start by briefly revisiting the relevant definitions from Section 3.3 and listing more specifically which values have to be provided (again dropping the iteration index k):

- The evaluation of $h(y)$ requires $x(t_{j+1};s_j,q_j)$, $j = 0,1,\ldots,m-1$, where $x(t;s_j,q_j)$ is the the second part of the solution of the composite IVP (1.21). In addition, $r_a(s_0) + r_b(s_m)$ is needed, which can be calculated without integration.

- The components of $g(y)$ can be directly evaluated without integration.

- For $\nabla F(y)$ the derivatives

$$z_j^s \;:=\; \nabla_{s_j} z(t_{j+1};s_j,q_j) \in \mathbb{R}^n$$

$$z_j^q := \nabla_{q_j} z(t_{j+1}; s_j, q_j) \in \mathbb{R}^{k_j},$$

$j = 0, 1, \ldots, m-1$, are needed, where the scalar function $z(t; s_j, q_j)$ is the first component of the solution of (1.21).

- The evaluation of $\nabla h(y)^T$ requires

$$\begin{aligned} X_j^s &:= \nabla_{s_j} x(t_{j+1}; s_j, q_j)^T \in \mathbb{R}^{n \times n} \\ X_j^q &:= \nabla_{q_j} x(t_{j+1}; s_j, q_j)^T \in \mathbb{R}^{n \times k_j}, \end{aligned}$$

$j = 0, 1, \ldots, m-1$, to be found by integration, while the derivatives

$$\begin{aligned} R_a &:= \nabla r_a(s_0)^T \in \mathbb{R}^{l_r \times n} \\ R_b &:= \nabla r_b(s_m)^T \in \mathbb{R}^{l_r \times n}, \end{aligned}$$

are calculated directly either from user-provided analytic formulas or by a suitable difference approximation.

- The components $G_j := \nabla g_j(q_j)^T \in \mathbb{R}^{l_j \times k_j}$, $j = 0, 1, \ldots, m-1$, of $\nabla g(y)^T$ are calculated in the same way as R_a, R_b.

In the following, we are primarily concerned with the determination of the derivatives z_j^s, z_j^q, X_j^s, and X_j^q of the solution of (1.21) at the end point of the multiple shooting interval I_j with respect to the initial values s_j and the parameters q_j. Note that the IVPs (1.21) on different intervals I_j are completely decoupled, hence the functions and gradients can be determined independently for each interval. We will first consider the simple case without additional control nodes (i.e., using exactly the same mesh for control and state parameterization).

Assuming that the functions $\Phi(t, x, u)$ and $f(t, x, u)$ have continuous partial derivatives in x and u, and that the control parameterization $u(t) := \varphi_j(t, q_j)$, $t \in I_j$, is continuously differentiable with respect to q_j, a unique solution $(z(t; s_j, q_j),\ x(t; s_j, q_j))$ of (1.21) exists at least in some neighborhood $N(\delta_j, t_j)$ of the initial point t_j, according to Theorem 4.1. The same is also true for the partial derivatives of this solution with respect to s_j, q_j, as shown by Theorems 4.3 and 4.4. Therefore, by choosing a sufficiently fine multiple shooting mesh such that $t_{j+1} \in N(\delta_j, t_j)$, we can ensure that all the required functions and derivatives exist at the end point t_{j+1}. If for the composite problem (1.21), we define

$$\bar{x}(t) := \begin{pmatrix} z(t) \\ x(t) \end{pmatrix}, \quad \bar{s}_j := \begin{pmatrix} 0 \\ s_j \end{pmatrix}, \quad \bar{f}_j(t, \bar{x}, q_j) := \begin{pmatrix} \Phi(t, x, \varphi_j(t, q_j)) \\ f(t, x, \varphi_j(t, q_j)) \end{pmatrix}$$

and determine the corresponding derivative matrices $\bar{Z}(t;\bar{s}_j,q_j)$ with dimension $(n+1)\times(n+1)$ and $\bar{W}(t;\bar{s}_j,q_j)$ with dimension $(n+1)\times k_j$ by solving the sensitivity equations (4.4) and (4.5), respectively, we obtain

$$\bar{Z}(t_{j+1};\bar{s}_j,q_j) \equiv \begin{pmatrix} 1 & z_j^{sT} \\ 0 & X_j^s \end{pmatrix}, \quad \bar{W}(t_{j+1};\bar{s}_j,q_j) \equiv \begin{pmatrix} z_j^{qT} \\ X_j^q \end{pmatrix} \tag{4.8}$$

(the first column of $\bar{Z}$ is trivial and need not be calculated). Hence on each interval I_j, we have to integrate $1+n+k_j$ systems of differential equations of dimension $n+1$: one integration for (1.21), n integrations for (4.4), and k_j integrations for (4.5).

When there are additional control nodes $\tau_{j,\ell}$ within the multiple shooting interval $I_j = [t_j, t_{j+1}]$, say

$$t_j = \tau_{j,0} < \tau_{j,1} < \ldots < \tau_{j,m_j} = t_{j+1}, \tag{4.9}$$

the analysis becomes slightly more complicated. As before, we are interested in the function and derivative values at the multiple shooting point t_{j+1}, but now we have to solve a *sequence* of IVPs defined on the subintervals $I_{j,\ell} = [\tau_{j,\ell}, \tau_{j,\ell+1}]$, $\ell = 0,1,\ldots,m_j-1$, in order to arrive at these values. On each subinterval $I_{j,\ell}$, the control parameterization is given by $u(t) := \varphi_{j,\ell}(t,q_{j,\ell})$ with the local parameter vector $q_{j,\ell}$, hence $q_j = (q_{j,0},\ldots,q_{j,m_j-1})$. The solution $\bar{x}(t_{j+1};\bar{s}_j,q_j)$ is obtained by determining recursively the values

$$\begin{aligned} \bar{x}_{j,1} &:= \bar{x}(\tau_{j,1};\bar{s}_j,q_{j,0}) \\ \bar{x}_{j,2} &:= \bar{x}(\tau_{j,2};\bar{x}_{j,1},q_{j,1}) \\ &\vdots \\ \bar{x}_{j,m_j} &:= \bar{x}(\tau_{j,m_j};\bar{x}_{j,m_j-1},q_{j,m_j-1}) \equiv \bar{x}(t_{j+1};\bar{s}_j,q_j), \end{aligned}$$

where $\bar{x}_{j,1}$ is the end value of the solution of an IVP of type (1.21) defined on $I_{j,0}$, which is then used as the initial value for the next problem defined on $I_{j,1}$, and so on. For each of these values $\bar{x}_{j,\ell}$, the derivatives $\bar{Z}_{j,\ell}$, $\bar{W}_{j,\ell}$ with respect to the preceding value $\bar{x}_{j,\ell-1}$ (or $\bar{s}_j$ if $\ell=1$) and the parameter vector $q_{j,\ell-1}$ can be determined by solving the sensitivity equations (4.4) and (4.5) on the corresponding subintervals. Therefore, using the notation

$$\frac{\partial \bar{x}_{j,1}}{\partial \bar{s}_j} := \bar{Z}_{j,1}, \quad \frac{\partial \bar{x}_{j,l}}{\partial \bar{x}_{j,\ell-1}} := \bar{Z}_{j,\ell}, \quad \frac{\partial \bar{x}_{j,\ell}}{\partial q_{j,\ell-1}} := \bar{W}_{j,\ell},$$

we obtain the required derivatives at the multiple shooting end point t_{j+1} in a straightforward manner by applying the chain rule. For

$$\bar{Z}(t_{j+1}; \bar{s}_j, q_j) = \nabla_{\bar{s}_j} \bar{x}(t_{j+1}; \bar{s}_j, q_j)^T$$

we get the formula

$$\bar{Z}(t_{j+1}; \bar{s}_j, q_j) = \frac{\partial \bar{x}_{j,m_j}}{\partial \bar{x}_{j,m_j-1}} \cdots \frac{\partial \bar{x}_{j,2}}{\partial \bar{x}_{j,1}} \cdot \frac{\partial \bar{x}_{j,1}}{\partial \bar{s}_j} = \prod_{k=0}^{m_j-1} \bar{Z}_{j,m_j-k}, \tag{4.10}$$

and for the components

$$\bar{W}_{q_{j,\ell}}(t_{j+1}; \bar{s}_j, q_j) = \nabla_{q_{j,\ell}} \bar{x}(t_{j+1}; \bar{s}_j, q_j)^T$$

of $\bar{W}(t_{j+1}; \bar{s}_j, q_j) := (\bar{W}_{q_{j,0}}, \ldots, \bar{W}_{q_{j,m_j-1}})$, we have

$$\begin{aligned} \bar{W}_{q_{j,\ell}}(t_{j+1}; \bar{s}_j, q_j) &= \frac{\partial \bar{x}_{j,m_j}}{\partial \bar{x}_{j,m_j-1}} \cdots \frac{\partial \bar{x}_{j,\ell+2}}{\partial \bar{x}_{j,\ell+1}} \cdot \frac{\partial \bar{x}_{j,\ell+1}}{\partial q_{j,\ell}} \\ &= \left(\prod_{k=0}^{m_j-\ell-2} \bar{Z}_{j,m_j-k} \right) \bar{W}_{j,\ell+1} \end{aligned} \tag{4.11}$$

for $\ell = 0, 1, \ldots, m_j - 2$, and $\bar{W}_{q_{j,m_j-1}}(t_{j+1}; \bar{s}_j, q_j) = \bar{W}_{j,m_j}$. On each subinterval $I_{j,\ell}$, we have to integrate $1 + n + k_{j,\ell}$ systems of differential equations of dimension $n+1$ (one system for $\bar{x}_{j,\ell}$, n systems for $\bar{Z}_{j,\ell}$, and $k_{j,\ell}$ systems for $\bar{W}_{j,\ell}$). Hence in the usual case of $k_{j,\ell} = k_j$, $\ell = 0, 1, \ldots, m_j - 1$, the number of integrations required in total on the composite interval $I_j = [t_j, t_{j+1}]$ is completely independent of the number of additional control nodes (note, however, that the number of matrix multiplications in (4.10) and (4.11) increases with the number of control nodes).

4.3 Internal Numerical Differentiation (IND)

Now we address a number of important issues regarding the *practical computation* of suitable approximations to the required function and derivative values. For simplicity of presentation, we temporarily assume again that just one mesh is used for both control and state parameterization—the extension to the more general case with additional control nodes is straightforward and hence has only been included in the formal statement of the IND algorithm at the end of this section. We also restrict ourselves to a discussion of IND within the context of explicit one-step integration methods.

In order to determine an approximation $\bar{\eta}(t_{j+1}; \bar{s}_j, q_j)$ to $\bar{x}(t_{j+1}; \bar{s}_j, q_j)$, we employ a numerical integration method to solve the IVP (1.21),

$$\dot{\bar{x}} = \bar{f}_j(t, \bar{x}, q_j), \quad \bar{x}(t_j) = \bar{s}_j, \quad t \in I_j = [t_j, t_{j+1}].$$

An *explicit one-step method* would typically proceed as follows: starting with

$$\bar{\eta}_0 := \bar{s}_j, \quad \vartheta_0 := t_j,$$

the approximate values $\bar{\eta}_i := \bar{\eta}(\vartheta_i; \bar{s}_j, q_j)$ for the exact solution values $\bar{x}_i := \bar{x}(\vartheta_i; \bar{s}_j, q_j)$ at the discrete abscissas ϑ_i are calculated successively by

$$\begin{aligned} \bar{\eta}_{i+1} &= \bar{\eta}_i + h_i \, \Psi(\vartheta_i, \bar{\eta}_i, q_j; h_i; \bar{f}_j) \\ \vartheta_{i+1} &= \vartheta_i + h_i, \end{aligned} \tag{4.12}$$

$i = 0, 1, \ldots, N-1$, where $\Psi(t, \bar{x}, q_j; h; \bar{f}_j)$ is a function characterizing the specific one-step method. In the method of forward Euler, we have

$$\Psi(t, \bar{x}, q_j; h; \bar{f}_j) := \bar{f}_j(t, \bar{x}, q_j),$$

i.e., here Ψ is independent of h. The stepsizes h_i, $i = 0, 1, \ldots, N-1$, satisfy

$$\sum_{i=0}^{N-1} h_i = t_{j+1} - t_j,$$

and in the simplest case, one could just use the fixed stepsize

$$h := (t_{j+1} - t_j)/N$$

on all steps i. However, due to the unpredictable behaviour of the solutions of differential equations, the numerical integration is in general forced to proceed with stepsizes which vary from point to point if a prescribed error bound ("tolerance") is to be maintained on each step. For further detail on the practical implementation of one-step methods with automatic stepsize control, see for instance [Stoer92].

From our discussion in Section 4.1, it is clear that approximations to the derivative matrices $\bar{Z}(t_{j+1}; \bar{s}_j, q_j)$ and $\bar{W}(t_{j+1}; \bar{s}_j, q_j)$ can be determined through the direct numerical solution of the sensitivity equations (4.4) and (4.5), respectively, along with the state equations (4.1). If accurate partial derivatives of the right-hand side function $\bar{f}_j(t, \bar{x}, q_j)$ with respect to $\bar{x}$ and q_j can be readily made available (e.g., by using automatic differentiation,

or if $\bar{f}_j$ is linear), this approach is actually quite attractive, since it allows to determine very accurate approximations to $\bar{Z}$ and $\bar{W}$. In the following, however, we will describe an alternative approach which does not require the local Jacobian of the state equations and can be more easily implemented in our context of explicit one-step methods.

Instead of calculating the matrices $\bar{Z}$ and $\bar{W}$ by solving the sensitivity equations, each column of $\bar{Z}$ and $\bar{W}$ is approximated by a difference quotient. Writing $\bar{s}_j = (0, \sigma^1, \ldots, \sigma^n)_j$, the approximate derivative of $\bar{x}(t_{j+1}; \bar{s}_j, q_j)$ with respect to σ_j^ν is thus calculated through

$$\begin{aligned} \frac{\partial \bar{x}(t_{j+1}; \bar{s}_j, q_j)}{\partial \sigma_j^\nu} \approx \frac{1}{\Delta\sigma_j^\nu} \Big(& \bar{\eta}(t_{j+1}; (0, \sigma^1, \ldots, \sigma^\nu + \Delta\sigma^\nu, \ldots, \sigma^n)_j, q_j) \\ & - \bar{\eta}(t_{j+1}; (0, \sigma^1, \ldots, \sigma^\nu, \ldots, \sigma^n)_j, q_j) \Big), \end{aligned} \tag{4.13}$$

where $\Delta\sigma_j^\nu$ is some suitably chosen small number. Note that in addition to $\bar{\eta}(t_{j+1}; \bar{s}_j, q_j)$ (which has to be determined anyway) this requires the calculation of $\bar{\eta}(t_{j+1}; (0, \sigma^1, \ldots, \sigma^\nu + \Delta\sigma^\nu, \ldots, \sigma^n)_j, q_j)$ by solving (1.21) with a slightly perturbed vector of initial values. Similar formulas are obtained for the approximate partial derivatives of $\bar{x}(t_{j+1}; \bar{s}_j, q_j)$ with respect to $q_j = (\rho^1, \ldots, \rho^k)_j$.

Some care has to be exercised in choosing the "small" numbers $\Delta\sigma_j^\nu$. If $\Delta\sigma_j^\nu$ is chosen too large, the difference quotient no longer provides a good approximation to the derivative, since the truncated higher order terms of the Taylor series expansion become significant. On the other hand, if $\Delta\sigma_j^\nu$ is chosen too small, then

$$\bar{\eta}(t_{j+1}; (0, \sigma^1, \ldots, \sigma^\nu + \Delta\sigma^\nu, \ldots, \sigma^n)_j, q_j) \approx \bar{\eta}(t_{j+1}; \bar{s}_j, q_j)$$

and the subtraction in (4.13) is subject to loss of significance which of course may invalidate the derivative approximation. An acceptable balance between these adverse effects can be found, for instance, by choosing $\Delta\sigma_j^\nu$ such that, in n-digit computation, corresponding vector elements of $\bar{\eta}(t_{j+1}; (0, \sigma^1, \ldots, \sigma^\nu + \Delta\sigma^\nu, \ldots, \sigma^n)_j, q_j)$ and $\bar{\eta}(t_{j+1}; \bar{s}_j, q_j)$ have approximately the first $n/2$ digits in common. (Then the loss of significant digits is still tolerable.) As a rule of thumb, this is the case for

$$\Delta\sigma_j^\nu = \sqrt{\epsilon_m}\, |\sigma_j^\nu|, \tag{4.14}$$

where ϵ_m denotes the machine precision, see e.g. [Stoer92]. However, one would normally expect the remaining $n/2$ digits of the perturbed and unperturbed $\bar{\eta}$-values to carry useful information only if these values are calculated with an accuracy comparable to ϵ_m. Although such high accuracy in

the solution of the IVP (1.21) can be attained with extrapolation methods [Stoer92], this is in general prohibitively expensive. In addition, it should be noted that with the choice (4.14) for $\Delta\sigma_j^\nu$, the fractional error of the finite-difference approximation (4.13) is *at least* of order $\sqrt{\epsilon_m}$. That is, in spite of the high accuracy integration, only a low accuracy result is obtained.

Let us take a second, closer look at the calculation of the derivative approximations to see how things may be improved. It is a well-known fact that the output of modern, automatic integrators is usually a *discontinuous* function of the initial values and parameters—if one of these inputs is varied, jumps of the order of the integrator tolerance have to be expected in the output, which can cause severe errors in the numerical estimation of derivatives [Gear83]. These discontinuities are due to the discrete nature of the stepsize (and possibly order) selection mechanisms; they arise when a change of an input parameter causes the algorithm to follow a different path of execution (e.g., employ another sequence of stepsizes).

It is obvious that using such an automatic integrator simply as a black-box is likely to produce very poor derivative approximations, unless the integrator tolerance is set to a value close to the machine precision. In the latter case, however, the efficiency of this so-called *external numerical differentiation (END)* scheme is extremely low. In order to motivate the alternative approach of *internal numerical differentiation (IND)* [Bock81b, Bock83, Bock87], which is much more efficient than END, we now address the question of how the situation changes when the discontinuities in the integrator output are not present.

The major points are best illustrated by looking at the following elementary example: consider the scalar IVP

$$\dot{x} = f(x), \quad x(0) = s, \tag{4.15}$$

where $x,\ s \in \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$ is continuously differentiable at least once. We are interested in the values

$$x(T;s) \ \text{ and } \ z(T;s) := \frac{\partial x}{\partial s}(T;s)$$

at some fixed end time T. According to (4.4), the derivative z solves the sensitivity equation

$$\dot{z} = \frac{df}{dx}(x(t;s))\, z, \quad z(0) = 1. \tag{4.16}$$

We use the forward Euler method with the *fixed* stepsize

$$h := T/N \quad \text{for some given } N \in \mathbb{N}$$

to calculate the approximate solution values

$$\eta_{i+1} = \eta_i + h\, f(\eta_i), \quad \eta_0 = s \tag{4.17a}$$

$$\zeta_{i+1} = \zeta_i + h\, \frac{df}{dx}(\eta_i)\, \zeta_i, \quad \zeta_0 = 1 \tag{4.17b}$$

for $i = 0, 1, \ldots, N-1$. At the end time $T = Nh$, we then have

$$\eta_N - x(T; s) = O(h) \tag{4.18a}$$

$$\zeta_N - z(T; s) = O(h), \tag{4.18b}$$

since Euler's method is of order 1. Instead of directly solving (4.16) along with (4.15), the derivative z can also be obtained through a finite difference approximation as discussed above. We repeat the integration of (4.15) using the (suitably) perturbed initial value $s + \Delta s$ to determine

$$\tilde{\eta}_{i+1} = \tilde{\eta}_i + h\, f(\tilde{\eta}_i), \quad \tilde{\eta}_0 = s + \Delta s \tag{4.19}$$

for $i = 0, 1, \ldots, N-1$, and of course we have again

$$\tilde{\eta}_N - x(T; s + \Delta s) = O(h). \tag{4.20}$$

Now we see if anything can be said about the order of

$$\frac{\tilde{\eta}_N - \eta_N}{\Delta s} - z(T; s) =: \Delta_s \eta_N - z(T; s),$$

i.e., the order of the discretization error of the finite difference approximation $\Delta_s \eta_N$ which is based on $\tilde{\eta}_N$ and η_N. In fact, there is a very close relationship between $\Delta_s \eta_N$ and ζ_N. To derive it, we subtract (4.17a) from (4.19) and divide by Δs to obtain the scheme

$$\begin{aligned}
\Delta_s \eta_{i+1} &= \Delta_s \eta_i + h\, \frac{f(\tilde{\eta}_i) - f(\eta_i)}{\Delta s} \\
&= \Delta_s \eta_i + h\, \frac{f(\eta_i) + \dfrac{df}{dx}(\eta_i)\, \Delta_s \eta_i\, \Delta s + O(\Delta s^2) - f(\eta_i)}{\Delta s} \\
&= \Delta_s \eta_i + h\, \frac{df}{dx}(\eta_i)\, \Delta_s \eta_i + O(h \Delta s), \quad \Delta_s \eta_0 = 1,
\end{aligned} \tag{4.21}$$

for $i = 0, 1, \ldots, N-1$, where in the Taylor expansion of $f(\tilde{\eta})$ we have immediately replaced the unknown derivative $\partial\eta_i/\partial s$ by $\Delta_s\eta_i + O(\Delta s)$. If the stepsize h is chosen such that

$$h \approx \Delta s, \tag{4.22}$$

then, since the two schemes (4.21) and (4.17b) agree except for a term of second order in h, it is apparent that

$$\Delta_s\eta_N - \zeta_N = O(h). \tag{4.23}$$

From (4.23) and (4.18b), we finally obtain the result

$$\Delta_s\eta_N - z(T;s) = O(h), \tag{4.24}$$

which shows that the discretization error of $\Delta_s\eta_N$ is of the same order as that of ζ_N. Thus we can conclude that, in our fixed-stepsize case, calculating the finite difference $\Delta_s\eta_N$ from $\tilde{\eta}_N$ and η_N (obtained by evaluating the schemes (4.19) and (4.17a), respectively) is *numerically equivalent* to directly calculating the approximate derivative ζ_N by evaluating (4.17).

It can be easily verified that this conclusion holds as well for general variable-stepsize algorithms, as long as the *same sequence of stepsizes* is used in the two numerical solutions differenced to estimate the derivative. This is the basic principle of IND, which has been first introduced by Bock in [Bock81b]: if the same path of execution is followed in the calculation of both base and perturbed trajectories, then it is no longer necessary to do the integrations with an accuracy close to the machine precision ϵ_m in order to obtain reliable finite-difference approximations. For instance, if $\Delta\sigma_j^\nu$ is chosen according to (4.14), an integrator tolerance of $O(\sqrt{\epsilon_m})$ can be used instead of $O(\epsilon_m)$ which would be required for END. Observe that IND computes an approximation to the well-defined, smooth derivative of *one* single (variable order and step) discretization scheme, while END combines *two* (generally different) discretizations to obtain a finite-difference approximation which is meaningful only if the discretization errors involved are very small.

In the context of explicit one-step methods, the easiest way to implement the IND strategy is to perform the integrations needed for the base trajectory and all of the perturbed trajectories *simultaneously* and use the estimated local discretization error of the base solution for stepsize control. Of course, it would be also possible to use instead the sum of the estimated errors

of all trajectories, but since normally the errors of the base solution and the perturbed solutions are roughly the same, the less complicated first possibility is often preferred.

We can now give a formal statement of the IND algorithm based on an explicit one-step integrator as used in the original version of MUSCOD. For each multiple shooting interval $I_j = [t_j, t_{j+1}]$, $j = 0, 1, \ldots, m-1$, repeat the following steps in order to (approximately) determine $\bar{x}(t_{j+1}; \bar{s}_j, q_j)$ and $\bar{Z}(t_{j+1}; \bar{s}_j, q_j)$, $\bar{W}(t_{j+1}; \bar{s}_j, q_j)$.[‡]

IND1. *Set the initial values for the first subinterval.* For the base trajectory use the initial value $\bar{\eta}^0_{j,0} = \bar{s}_j = (0, \sigma^1, \ldots, \sigma^n)^T_j$, and for the first block of n perturbed trajectories use the perturbed initial values

$$\bar{\eta}^\nu_{j,0} = \bar{s}_j + \Delta\sigma^\nu_j e_{\nu+1} = (0, \sigma^1, \ldots, \sigma^\nu + \Delta\sigma^\nu, \ldots, \sigma^n)^T_j,$$

$\nu = 1, 2, \ldots, n$, where $\Delta\sigma^\nu_j$ are suitably chosen small numbers. For the second block of $k_{j,0}$ perturbed trajectories, use the same initial value as for the base trajectory, that is, $\bar{\eta}^{n+\mu}_{j,0} = \bar{s}_j$, $\mu = 1, 2, \ldots, k_{j,0}$. Furthermore, define the set of perturbed parameter vectors

$$q^\mu_{j,0} = q_{j,0} + \Delta\rho^\mu_{j,0} e_\mu = (\rho^1, \ldots, \rho^\mu + \Delta\rho^\mu, \ldots, \rho^k)^T_{j,0},$$

$\mu = 1, 2, \ldots, k_{j,0}$, for suitably chosen $\Delta\rho^\mu_{j,0}$. Set $\ell = 0$.

IND2. *Solve the IVPs on the current subinterval.* Find approximate solutions to

$$\dot{\bar{x}} = \bar{f}_{j,\ell}(t, \bar{x}, q_{j,\ell}), \quad \bar{x}(\tau_{j,\ell}) = \bar{\eta}^\nu_{j,\ell}, \quad t \in I_{j,\ell} = [\tau_{j,\ell}, \tau_{j,\ell+1}],$$

$\nu = 0, 1, \ldots, n$, as well as to

$$\dot{\bar{x}} = \bar{f}_{j,\ell}(t, \bar{x}, q^\mu_{j,\ell}), \quad \bar{x}(\tau_{j,\ell}) = \bar{\eta}^{n+\mu}_{j,\ell}, \quad t \in I_{j,\ell},$$

$\mu = 1, 2, \ldots, k_{j,l}$. To this end, define the initial abscissa $\vartheta_0 = \tau_{j,\ell}$ and the integration end point $\vartheta_e = \tau_{j,\ell+1}$, set $i = 0$, and repeat the following three substeps (where the interval indices j, ℓ have been dropped from all variables, and the index i is used to denote the steps of the numerical integration procedure).

[‡]We will use the term *subinterval* to denote the control intervals $I_{j,\ell} = [\tau_{j,\ell}, \tau_{j,\ell+1}]$, $\ell = 0, 1, \ldots, m_j - 1$.

IND2.1. *Perform one integration step for the base trajectory.* Using automatic stepsize control, calculate

$$\bar{\eta}_{i+1}^0 \;=\; \bar{\eta}_i^0 + h_i \Psi(\vartheta_i, \bar{\eta}_i^0, q; h_i; \bar{f}),$$

where Ψ is the function characterizing the one-step integration method, and h_i is the chosen stepsize ($h_i \leq \vartheta_e - \vartheta_i$).

IND2.2. *Apply the same step to the perturbed trajectories.* With the stepsize h_i from above, determine

$$\begin{aligned} \bar{\eta}_{i+1}^{\nu} &= \bar{\eta}_i^{\nu} + h_i \Psi(\vartheta_i, \bar{\eta}_i^{\nu}, q; h_i; \bar{f}), \quad \nu = 1, 2, \ldots, n \\ \bar{\eta}_{i+1}^{n+\mu} &= \bar{\eta}_i^{n+\mu} + h_i \Psi(\vartheta_i, \bar{\eta}_i^{n+\mu}, q^{\mu}; h_i; \bar{f}), \quad \mu = 1, 2, \ldots, k, \end{aligned}$$

set $\vartheta_{i+1} = \vartheta_i + h_i$, and increment i by one.

IND2.3. *Check for the end point of the subinterval.* If $\vartheta_i < \vartheta_e$ go back to step IND2.1, otherwise continue with step IND4.

IND4. *Calculate approximations to the current derivative matrices.* From the final approximate solution values determined in step IND2, calculate the vectors

$$\Delta\bar{\eta}_{j,\ell}^{\nu} = \frac{1}{\Delta\xi_{j,\ell}^{\nu}}(\bar{\eta}_{j,\ell}^{\nu} - \bar{\eta}_{j,\ell}^0), \quad \nu = 1, 2, \ldots, n$$

(for $\ell = 0$ replace $\Delta\xi$ by $\Delta\sigma$ in this formula), and set

$$\bar{Z}_{j,\ell+1} = (e_1, \Delta\bar{\eta}^1, \ldots, \Delta\bar{\eta}^n)_{j,\ell},$$

where e_1 is the first canonical basis vector in $\mathbb{R}^{n+1}$. Similarly, determine the vectors

$$\Delta\bar{\eta}_{j,\ell}^{n+\mu} = \frac{1}{\Delta\rho_{j,\ell}^{\mu}}(\bar{\eta}_{j,\ell}^{n+\mu} - \bar{\eta}_{j,\ell}^0), \quad \mu = 1, 2, \ldots, k_{j,\ell},$$

and set

$$\bar{W}_{j,\ell+1} = (\Delta\bar{\eta}^{n+1}, \ldots, \Delta\bar{\eta}^{n+k})_{j,\ell}.$$

In addition, set the current function value $\bar{x}_{j,\ell+1} = \bar{\eta}_{j,\ell}^0$.

IND5. *Check for the last subinterval.* If the solution on the last subinterval has been determined ($\ell = m_j - 1$) go to step IND7, otherwise continue with step IND6.

IND6. *Set the initial values for the next subinterval.* Increment ℓ by one, then for the base trajectory set $\bar{\eta}^0_{j,\ell} = \bar{x}_{j,\ell} = (\zeta, \xi^1, \ldots, \xi^n)^T_{j,\ell}$, and for the first block of perturbed trajectories set

$$\bar{\eta}^\nu_{j,\ell} = \bar{x}_{j,\ell} + \Delta\xi^\nu_{j,\ell} e_{\nu+1} = (\zeta, \xi^1, \ldots, \xi^\nu + \Delta\xi^\nu, \ldots, \xi^n)^T_{j,\ell},$$

$\nu = 1, 2, \ldots, n$, where $\Delta\xi^\nu_{j,\ell}$ are again suitably chosen small numbers. For the second block of perturbed trajectories set $\bar{\eta}^{n+\mu}_{j,\ell} = \bar{x}_{j,\ell}$ (same as for base trajectory), and define the new set of perturbed parameter vectors

$$q^\mu_{j,\ell} = q_{j,\ell} + \Delta\rho^\mu_{j,\ell} e_\mu = (\rho^1, \ldots, \rho^\mu + \Delta\rho^\mu, \ldots, \rho^k)^T_{j,\ell},$$

$\mu = 1, 2, \ldots, k_{j,\ell}$. Go back to step IND2.

IND7. *Apply the chain rule to obtain the final derivative matrices.* Using the local derivatives $\bar{Z}_{j,\ell}$, $\bar{W}_{j,\ell}$, $\ell = 1, 2, \ldots, m_j$, calculate

$$\begin{aligned} \bar{Z}(t_{j+1}; \bar{s}_j, q_j) &= \prod_{k=0}^{m_j-1} \bar{Z}_{j,m_j-k} \\ \bar{W}_{q_{j,\ell}}(t_{j+1}; \bar{s}_j, q_j) &= \prod_{k=0}^{m_j-\ell-2} \bar{Z}_{j,m_j-k}\, \bar{W}_{j,\ell+1}, \quad \ell = 0, 1, \ldots, m_j - 2 \end{aligned}$$

(note that $\bar{W}_{q_{j,m_j-1}}(t_{j+1}; \bar{s}_j, q_j) = \bar{W}_{j,m_j}$ is already known), and set

$$\bar{W}(t_{j+1}; \bar{s}_j, q_j) = (\bar{W}_{q_{j,0}}, \ldots, \bar{W}_{q_{j,m_j-1}}).$$

The function value $\bar{x}(t_{j+1}; \bar{s}_j, q_j)$ is given by $\bar{x}_{j,m_j}$.

In MUSCOD, the small numbers $\Delta\rho^\mu_{j,\ell}$ (and similarly $\Delta\sigma^\nu_j$ and $\Delta\xi^\nu_{j,\ell}$) are chosen as

$$\Delta\rho^\mu_{j,\ell} = \varepsilon_1(|\rho^\mu_{j,\ell}| + \varepsilon_2), \tag{4.25}$$

where for the calculation in double precision the constants $\varepsilon_1 = 0.1 \cdot 10^{-6}$ and $\varepsilon_2 = 0.1$ are used. This strategy is intended to strike a balance between the adverse effects of roundoff and truncation.[‡] Furthermore, $\Delta\rho$ should

[‡] As we have seen, the roundoff error is mainly due to cancellation effects (and becomes significant when $\Delta\rho$ is chosen too small), while the truncation error comes from neglecting the higher order terms of the Taylor series expansion (and becomes significant when $\Delta\rho$ is chosen too large).

be always chosen so that ρ and $\rho + \Delta\rho$ differ by an exactly representable number, i.e., the identity

$$\Delta\rho \equiv (\rho + \Delta\rho) - \rho \tag{4.26}$$

should be satisfied in machine arithmetic. For a discussion of this point and a simple method which can be used to achieve satisfaction of (4.26), see the text [Press92, p. 186]. Observe also that the perturbed initial values and the perturbed parameters should under no circumstances violate any "hard" bounds that may exist for state or parameter values (e.g., a concentration should never become negative), because this could lead to a breakdown of the right-hand side function.

The original version of MUSCOD uses a Runge-Kutta-Fehlberg method of order 7/8 [Fehl69]. However, it is important to note that the basic IND strategy does *not* depend on a specific integration method and can as well be implemented using, for instance, a multistep or extrapolation method. In the context of implicit integration methods, it can be advantageous to solve the sensitivity equations along with the state equations, since an approximation to the local Jacobian of the state equations must be determined anyway.[‡] This leads to a number of alternative approaches to IND, see e.g. [Bock83, Cara85, Bauer94]. In Section 7, we will briefly discuss a BDF method suitable for solving DAE models and stiff ODE models. Further examples of IND algorithms (including the adjoint scheme, the case of discontinuous right-hand sides, and the case of trajectories with jump discontinuities) can be found in [Bock87].

[‡]If an accurate approximation to the Jacobian is available—e.g., through automatic differentiation—it is then possible to obtain a result of higher accuracy than $O(\sqrt{\epsilon_m})$.

5 Approximation of the Hessian Matrix

5.1 Structure of the Hessian

In Section 3.3, we have introduced the QP subproblem $Q(y_k, B_k)$ and discussed the block structure of the constraint Jacobians. However, we have not yet considered the inherent structure of $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$, the actual Hessian matrix. We will see that this structure should be reflected also in the approximations B_k. Let us therefore start by having a closer look at the structure of $\nabla_y^2 L(y_k, \lambda_k, \mu_k)$ before discussing in detail how to derive suitable approximations B_k. (In the following, we will again occasionally drop the iteration index k.)

The Lagrangian function $L(y, \lambda, \mu)$ of problem (P2) has already been defined in equation (3.26),[‡]

$$\begin{aligned} L(y,\lambda,\mu) &= \sum_{j=0}^{m-1} F_j(s_j,q_j) - \sum_{j=0}^{m-1} \lambda_j^T (x(t_{j+1};s_j,q_j) - s_{j+1}) \\ &\quad - \lambda_m^T (r_a(s_0) + r_b(s_m)) - \sum_{j=0}^{m-1} \mu_j^T g_j(q_j). \end{aligned}$$

We observe that L is *partially separable*. If we define the "local" parameters

$$y_j := (s_j, q_j) \in \mathbb{R}^{n+k_j}, \quad j = 0, 1, \ldots, m-1; \quad y_m := s_m \in \mathbb{R}^n, \tag{5.1}$$

then the parameter vector y from (1.17) becomes

$$y = (y_0, y_1, \ldots, y_m),$$

and we can write L in the form

$$L(y, \lambda, \mu) = \sum_{j=0}^{m} L_j(y_j, \lambda, \mu), \tag{5.2}$$

where the functions L_j are given by

$$\begin{aligned} L_0(y_0,\lambda,\mu) &:= F_0(s_0,q_0) - \lambda_0^T x(t_1;s_0,q_0) - \lambda_m^T r_a(s_0) - \mu_0^T g_0(q_0) \\ L_j(y_j,\lambda,\mu) &:= F_j(s_j,q_j) + \lambda_{j-1}^T s_j - \lambda_j^T x(t_{j+1};s_j,q_j) - \mu_j^T g_j(q_j), \\ &\quad j = 1, 2, \ldots, m-1 \\ L_m(y_m,\lambda,\mu) &:= \lambda_{m-1}^T s_m - \lambda_m^T r_b(s_m). \end{aligned}$$

[‡]Note that if the boundary equality conditions are replaced by inequalities we simply have to change λ_m into μ_m in this formula.

This property of L is a direct consequence of the piecewise control parameterization and the multiple shooting state discretization described in Section 1.

By differentiating $L(y, \lambda, \mu)$ in (5.2) with respect to y, we obtain

$$\nabla_y L(y, \lambda, \mu) = \begin{pmatrix} \nabla_{y_0} L_0 \\ \nabla_{y_1} L_1 \\ \vdots \\ \nabla_{y_m} L_m \end{pmatrix}, \tag{5.3}$$

with components $\nabla_{y_j} L_j \in \mathbb{R}^{n+k_j}$, $j = 0, 1, \ldots, m-1$, and $\nabla_{y_m} L_m \in \mathbb{R}^n$, which can be directly calculated using the known derivatives of the discretized optimal control problem,

$$\begin{aligned}
\nabla_{y_0} L_0 &= \begin{pmatrix} z_0^s \\ z_0^q \end{pmatrix} - \begin{pmatrix} X_0^{s\,T} \\ X_0^{q\,T} \end{pmatrix} \lambda_0 - \begin{pmatrix} R_a^T \\ 0 \end{pmatrix} \lambda_m - \begin{pmatrix} 0 \\ G_0^T \end{pmatrix} \mu_0 \\
\nabla_{y_j} L_j &= \begin{pmatrix} z_j^s \\ z_j^q \end{pmatrix} + \begin{pmatrix} \lambda_{j-1} \\ 0 \end{pmatrix} - \begin{pmatrix} X_j^{s\,T} \\ X_j^{q\,T} \end{pmatrix} \lambda_j - \begin{pmatrix} 0 \\ G_j^T \end{pmatrix} \mu_j, \quad j = 1, 2, \ldots, m-1 \\
\nabla_{y_m} L_m &= \lambda_{m-1} - R_b^T \lambda_m
\end{aligned}$$

(the abbreviations z_j^s, z_j^q, X_j^s, X_j^q, R_a, R_b, and G_j are defined in (3.34)). As we have seen in Section 3, the gradient of the Lagrangian function is required for the variable metric update.

Differentiating $\nabla_y L(y, \lambda, \mu)$ in (5.3) again with respect to y, we observe that the Hessian matrix

$$\nabla_y^2 L(y, \lambda, \mu) = \begin{pmatrix} \nabla_{y_0}^2 L_0 & & & \\ & \nabla_{y_1}^2 L_1 & & \\ & & \ddots & \\ & & & \nabla_{y_m}^2 L_m \end{pmatrix} \tag{5.4}$$

has a *block diagonal structure* due to the partial separability of L. We need not derive explicit formulas for the calculation of the blocks $\nabla_{y_j}^2 L_j$ with dimensions $(n + k_j) \times (n + k_j)$, $j = 0, 1, \ldots, m-1$, and $\nabla_{y_m}^2 L_m$ with dimension $n \times n$ from (5.2) because we do not use the exact Hessian $\nabla_y^2 L(y, \lambda, \mu)$ in our algorithm.

For several reasons, it would be desirable to impose this known block diagonal structure also on the Hessian approximation B:

1. *Improved convergence*: One should expect that finding a suitable approximation B is easier if it has the right structure *a priori*.

2. *Memory savings*: Instead of the full matrix B, only the diagonal blocks would have to be stored.

3. *Computational savings*: A certain amount of work could be saved in the condensing algorithm, see Section 6.

However, standard rank-two updates like the modified BFGS-formula of Powell (see Section 3) would immediately destroy this sparse structure, because the corrections which are added to the current approximation are in general full matrices. At the end of this section we will describe an alternative high-rank block update (also based on the modified BFGS formula) which does not produce any fill-in outside the block diagonal.

To be able to employ any update procedure, one needs an initial estimate for the Hessian to start from. Therefore, the important question of how to find a suitable initial estimate will be discussed next.

5.2 Initial Estimate for the Hessian

In many SQP implementations, the Hessian approximation B is initially set to the identity matrix. However, this obvious choice has been found to be quite unsatisfactory even for well-scaled problems. Unless additional safeguards are provided, it may lead to unreasonably large and poorly directed correction steps p on the first few iterations, occasionally even resulting in a failure of the line search procedure [Plitt81, Chen84]. To circumvent this problem, Chen and Stadtherr proposed starting the line search with a trial steplength much smaller than unity whenever $\|p\|$ is large during the first few iterations [Chen84]. Although this helps to avoid line search failures, it does not improve the *direction* of the correction steps p.

A completely different strategy has been suggested by Plitt [Plitt81]. This strategy is applied only on the very first iteration ($k = 0$) and involves determining the initial Hessian estimate B_0 in such a way that the correction step p_0 obtained as the solution of $Q(y_0, B_0)$ is reasonably bounded. Thus, by applying this step to the initial point y_0, one remains in a region around y_0 where the constraint linearizations can be expected to provide an acceptable approximation to the original nonlinear constraints. Plitt's strategy obviously has the flavour of a *trust region method*, because it follows the same basic idea of *first* choosing a suitable bound on the norm of the step (*trust radius*) and *then* determining the optimal step subject to this bound by solving a modified subproblem. Consequently, MUSCOD uses *no* line search on the first major iteration (see Section 3). A detailed discussion

of the trust region approach for the unconstrained case (and a comparison with line search methods) can be found in [Denn89].

We now address the question of how to derive a reasonable bound for the norm of the first step. To this end, we define the following "minimum-norm" QP problem $Q_c(y_0)$,

$$\min_p \; p^T p = \|p\|_2^2 \tag{5.5a}$$

subject to

$$\begin{aligned} h(y_0) + \nabla h(y_0)^T p &= 0 && (5.5b) \\ g(y_0) + \nabla g(y_0)^T p &\geq 0, && (5.5c) \end{aligned}$$

which combines the linearized constraints of the initial QP subproblem $Q(y_0, B_0)$ with a minimum norm objective. Therefore, the solution p_c of (5.5) is a step from y_0 to the "nearest" point (in the l_2-norm) satisfying the initial linearized constraints. From the knowledge of this minimum-norm solution p_c, a meaningful bound on the norm of the initial step p_0 can be established in a straightforward manner. Plitt suggests to use the condition

$$\|p_0\|_2 \leq M_c \, \|p_c\|_2, \tag{5.6}$$

where $M_c > 1$ is an empirical constant (e.g., $M_c = 2$) [Plitt81].

This leaves us with the problem of determining an initial Hessian estimate B_0 such that the solution p_0 of $Q(y_0, B_0)$ satisfies condition (5.6). Using a matrix B_0 of the form

$$B_0 = \frac{1}{\kappa_c} I \tag{5.7}$$

(scaled identity),[‡] Plitt has shown that in the *equality-constrained case*, it is possible to ensure satisfaction of the bound (5.6) on $\|p_0\|_2$ by properly choosing the parameter κ_c [Plitt81]. The resulting formula for κ_c can be used as well in the general case with additional inequality constraints, but it should be noted that this generalization is purely heuristic, since the QP problems $Q_c(y_0)$ and $Q(y_0, B_0)$ need not have the same active set. Nevertheless, this approach has been found to work very well in practice.

In order to derive Plitt's formula for κ_c, let us assume that there are no inequality constraints present in the initial QP subproblem $Q(y_0, B_0)$. Then

[‡]Note that using this simple form for B_0 is justified only if the problem variables are well-scaled.

$Q(y_0, B_0)$ with B_0 according to (5.7) has the form

$$\min_p \ \nabla F(y_0)^T p + \frac{1}{2\kappa_c} p^T p \tag{5.8a}$$

subject to

$$h(y_0) + \nabla h(y_0)^T p \ = \ 0, \tag{5.8b}$$

while the corresponding minimum-norm problem $Q_c(y_0)$ is given by (5.5a)–(5.5b) above. We also assume that the constraint gradients are linearly independent (i.e., the matrix $\nabla h(y_0)^T$ has full row rank). The Karush-Kuhn-Tucker conditions for these equality-constrained QP problems $Q(y_0, B_0)$ and $Q_c(y_0)$ are given by

$$\frac{1}{\kappa_c} p_0 - \nabla h(y_0)\, \tilde{\lambda}_0 \ = \ -\nabla F(y_0) \tag{5.9a}$$

$$\nabla h(y_0)^T p_0 \ = \ -h(y_0) \tag{5.9b}$$

and

$$p_c - \nabla h(y_0)\, \tilde{\lambda}_c \ = \ 0 \tag{5.10a}$$

$$\nabla h(y_0)^T p_c \ = \ -h(y_0), \tag{5.10b}$$

respectively, and we observe that both these linear systems are uniquely solvable, yielding the KKT points $(p_0, \tilde{\lambda}_0)$ and $(p_c, \tilde{\lambda}_c)$. Note also that the minimum-norm solution p_c lies entirely in the range space $\mathcal{R}_c$ of the constraint gradients, as can be seen from (5.10a). We can now decompose p_0 in the form

$$p_0 = p_c + p_p, \tag{5.11}$$

where p_p is the orthogonal projection of $-\kappa_c \nabla F(y_0)$ onto the nullspace $\mathcal{N}_c$ of the constraint gradients,

$$\mathcal{N}_c := \{p \in \mathbb{R}^{\hat{n}} \mid \nabla h(y_0)^T p = 0\}.$$

Hence p_p is obtained by the standard projection formula[‡]

$$p_p = (I - \nabla h_0\, (\nabla h_0^T \nabla h_0)^{-1} \nabla h_0^T)\, (-\kappa_c \nabla F_0), \tag{5.12}$$

[‡]Recall that $\mathcal{N}_c$ is orthogonal to the subspace $\mathcal{R}_c$ spanned by the constraint gradients which form the columns of $\nabla h(y_0)$.

where we have used the abbreviations ∇h_0, ∇F_0 for $\nabla h(y_0)$, $\nabla F(y_0)$. In addition, we have

$$\tilde{\lambda}_0 = \frac{1}{\kappa_c}\,\tilde{\lambda}_c + (\nabla h_0^T \nabla h_0)^{-1} \nabla h_0^T \nabla F_0. \tag{5.13}$$

The correctness of (5.11) and (5.13) can be easily shown by substitution into (5.9) and use of (5.10). Now it is a simple matter to derive the desired formula for κ_c. Using Pythagoras' law, we obtain from (5.11)

$$\begin{aligned} \|p_0\|_2^2 &= \|p_c\|_2^2 + \|p_p\|_2^2 \\ &\leq \|p_c\|_2^2 + \kappa_c^2 \,\|\nabla F_0\|_2^2, \end{aligned}$$

and assuming that $\|p_c\|_2 \neq 0$, we can write

$$\|p_0\|_2^2 \;\leq\; \left(1 + \kappa_c^2 \,\frac{\|\nabla F_0\|_2^2}{\|p_c\|_2^2}\right)\, \|p_c\|_2^2.$$

By comparison with condition (5.6), we set

$$M_c^2 = 1 + \kappa_c^2 \,\frac{\|\nabla F_0\|_2^2}{\|p_c\|_2^2},$$

and if in addition $\|\nabla F_0\|_2 \neq 0$, we can finally solve for κ_c,

$$\kappa_c = \sqrt{M_c^2 - 1}\,\frac{\|p_c\|_2}{\|\nabla F_0\|_2}. \tag{5.14}$$

An implementation of this formula has to provide safeguards against values of $\|p_c\|_2$ and $\|\nabla F_0\|_2$ which are close to zero, see below.

We can now summarize the heuristic procedure used in MUSCOD to find a suitable initial estimate for the Hessian.

HI1. *Determine the "minimal" step to a zero of the linearized constraints.* Find the solution p_c of the minimum-norm QP problem $Q_c(y_0)$ by the methods discussed in Section 6.

HI2. *Calculate the Hessian scale factor.* Determine

$$\kappa_c = \sqrt{M_c^2 - 1}\,\frac{\max(\|p_c\|_2,\; \epsilon_c\,\|y_0\|_2)}{\max(\|\nabla F(y_0)\|_2,\; \epsilon_c)},$$

where M_c and ϵ_c are empirical constants. In MUSCOD, the values $1.5 \leq M_c \leq 3$ (usually $M_c = 2$) and $\epsilon_c = 0.1$ are used.

HI3. *Initialize the Hessian by the scaled identity matrix.* For each diagonal block $B_{0,j}$ of B_0 set

$$B_{0,j} = \frac{1}{\kappa_c} I_j, \quad j = 0, 1, \ldots, m,$$

where I_j denotes an identity matrix of appropriate dimension.

5.3 Variable Metric Block Update

As we have seen at the beginning of this section, application of any of the standard rank-two update formulas for revising the Hessian approximation would destroy its known block diagonal structure. For this reason, Plitt proposed a generalized update procedure which revises each diagonal block $B_{k,j}$ of B_k *separately* using the corresponding components of the step δ_k and the Lagrangian gradient difference γ_k [Plitt81]. If the standard rank-two update formula is given by

$$B_{k+1} = B_k + U(B_k, \delta_k, \gamma_k) \tag{5.15}$$

(where provisions should have been made for proper handling of the degenerate case $\delta_k = 0$, i.e., $U(B, 0, \gamma) \equiv 0$), then the generalized update formula can be written as

$$B_{k+1} = B_k + \sum_{j=0}^{m} U(P_j\, B_k\, P_j,\ P_j\, \delta_k,\ P_j\, \gamma_k), \tag{5.16}$$

using appropriate projection matrices P_j [Bock84]. It is easily seen that the generalized formula is of rank $2\,(m+1)$ in the non-degenerate case and that it preserves the block diagonal structure of the matrices B_k. A similar partitioned update procedure for unconstrained optimization has been proposed by Griewank and Toint [Grie82].

We now give a formal statement of the BFGS block update procedure used in MUSCOD. All vectors are partitioned corresponding to the blocks of the Hessian in exactly the same way as defined in (5.1) for the components $y_{k,j}$ of y_k. Hence we have the partitioning

$$\delta_{k,j} = y_{k+1,j} - y_{k,j} \equiv \alpha_k p_{k,j}, \quad j = 0, 1, \ldots, m, \tag{5.17}$$

of the current step δ_k. Given B_k, δ_k, and the two Lagrangian gradients

$$\nabla L(y_{k+1}, \lambda_{k+1}, \mu_{k+1}) \ \text{ and } \ \nabla L(y_k, \lambda_{k+1}, \mu_{k+1})$$

calculated according to (5.3), we perform the following steps to determine the new Hessian approximation B_{k+1}.

RH1. *Calculate the Lagrangian gradient difference.* Determine

$$\gamma_{k,j} = \nabla_{y_j} L(y_{k+1}, \lambda_{k+1}, \mu_{k+1}) - \nabla_{y_j} L(y_k, \lambda_{k+1}, \mu_{k+1})$$

for $j = 0, 1, \ldots, m$, using the "new" and the "old" gradient of the Lagrangian function.

RH2. *If necessary, modify the gradient difference.* For each $j = 0, 1, \ldots, m$, do the following: if the condition

$$\delta_{k,j}^T \gamma_{k,j} \geq \epsilon_\Theta \, \delta_{k,j}^T B_{k,j} \delta_{k,j}$$

is satisfied (where ϵ_Θ is an empirical constant, e.g., $\epsilon_\Theta = 0.2$), simply set

$$\eta_{k,j} = \gamma_{k,j}$$

(no modification is neccessary to ensure a positive definite update of the corresponding Hessian block). Otherwise, calculate the parameter

$$\Theta_{k,j} = \frac{(1 - \epsilon_\Theta) \, \delta_{k,j}^T B_{k,j} \delta_{k,j}}{\delta_{k,j}^T B_{k,j} \delta_{k,j} - \delta_{k,j}^T \gamma_{k,j}}$$

with $0 < \Theta_{k,j} \leq 1$, and set

$$\eta_{k,j} = \Theta_{k,j} \gamma_{k,j} + (1 - \Theta_{k,j}) \, B_{k,j} \delta_{k,j}.$$

This implies that $\delta_{k,j}^T \eta_{k,j} = \epsilon_\Theta \, \delta_{k,j}^T B_{k,j} \delta_{k,j} > 0$, and hence guarantees a positive definite update.

RH3. *Update the diagonal blocks of the Hessian.* Apply a generalized version of Powell's modified BFGS update separately to each block $B_{k,j}$ of B_k. That is, set

$$B_{k+1,j} = \begin{cases} B_{k,j} + \dfrac{\eta_{k,j} \eta_{k,j}^T}{\eta_{k,j}^T \delta_{k,j}} - \dfrac{B_{k,j} \delta_{k,j} \delta_{k,j}^T B_{k,j}}{\delta_{k,j}^T B_{k,j} \delta_{k,j}} & \text{if } \delta_{k,j} \not\approx 0 \\ B_{k,j} & \text{if } \delta_{k,j} \approx 0 \end{cases}$$

for $j = 0, 1, \ldots, m$. In MUSCOD, $\delta_{k,j} \approx 0$ if $\|\delta_{k,j}\|_2 \leq \epsilon_d$, where $\epsilon_d = 10^{-4}$ is an empirical constant.

Notice that in step RH3, the degenerate case $\delta_{k,j} \approx 0$ has to be dealt with explicitly, because $\delta_k \not\approx 0$ does not necessarily imply $\delta_{k,j} \not\approx 0$ for all j.

Therefore, $\delta_{k,j} \approx 0$ may occur for some j before complete convergence has been achieved. It would be even possible to go one step further and skip the update of $B_{k,j}$ not only for $\delta_{k,j} \approx 0$, but also in cases where $\delta_{k,j}$ and $\gamma_{k,j}$ are such that performing the update would lead to severe ill-conditioning of the new Hessian approximation $B_{k+1,j}$. A strategy of this kind has been suggested by [Noce85] for the standard rank-two update, but it has been found in practice that it is quite dangerous to skip updates since this is likely to impair the superlinear rate of convergence. However, there is a good chance that such a strategy would work much better for the high-rank block update, since one generally expects that, on a given iteration, most of the Hessian blocks $B_{k,j}$ are updated in the normal way, and it seems rather unlikely that an update will be skipped entirely (i.e., none of the blocks will be updated).

6 Solution of the QP Subproblem

We now consider in some detail the solution of the quadratic programming (QP) problem

$$\min_p \; b^T p + \frac{1}{2} p^T B p \tag{6.1a}$$

subject to

$$C p = c \tag{6.1b}$$

$$D p \geq d, \tag{6.1c}$$

where, without loss of generality, B is assumed to be symmetric. For positive definite B, we obtain a *convex* problem which qualifies for the first order sufficiency theorem stated in Section 2. Hence if there exists a KKT point $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$, then p^* is a strong local minimizer of (6.1) and in addition represents the global optimal solution (see Theorem 2.7). Moreover, from the necessary conditions of Theorem 2.5, it follows that if (6.1) has a solution p^*, then a KKT point $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ must exist because the constraint qualification is always satisfied for linear constraints. Furthermore, if the solution p^* is a regular point (Definition 2.4)—which automatically implies strict complementarity for a convex QP problem—then the multipliers $\tilde{\lambda}^*$, $\tilde{\mu}^*$ are uniquely determined. That is, there exists exactly one KKT point. In summary, a convex QP problem has a unique solution p^* whenever the constraints (6.1b)–(6.1c) are consistent, and in the regular case, it has also unique multipliers $\tilde{\lambda}^*$, $\tilde{\mu}^*$.

For positive semidefinite or indefinite B, the situation is more complicated: the QP problem may then have an infinite number of weak solutions, or it may even be unbounded in the indefinite case. We will not consider these other possibilities any further here but restrict ourselves to convex QP problems. (Recall that the QP subproblem $Q(y_k, B_k)$ introduced in Section 3.3 is convex, since B_k is positive definite by construction.)

6.1 Linear Problem Transformations

In the following, we will need two elementary results on the effects of certain linear transformations on the solution of the quadratic programming problem (6.1). Note that similar theorems can be stated as well for the general NLP problem (2.1).

Theorem 6.1 (Linear Transformation of Coordinates)
The Lagrange multipliers $\tilde{\lambda}^*$, $\tilde{\mu}^*$ of (6.1) are invariant under the linear transformation of coordinates $p = S\,s$, where S is a nonsingular matrix. Hence if $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ is a KKT point of the original problem, then

$$(s^*, \tilde{\lambda}^*, \tilde{\mu}^*) \equiv (S^{-1}p^*, \tilde{\lambda}^*, \tilde{\mu}^*) \tag{6.2}$$

is a KKT point of the transformed problem

$$\min_s\ (S^T b)^T s + \frac{1}{2}\, s^T (S^T B\, S)\, s \tag{6.3a}$$

subject to

$$\begin{aligned} (C\,S)\,s &= c & \text{(6.3b)}\\ (D\,S)\,s &\geq d. & \text{(6.3c)} \end{aligned}$$

In particular, a *permutation* of the components of p does not alter the values of $\tilde{\lambda}^*$, $\tilde{\mu}^*$. For a proof of this theorem we observe that $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ satisfies the Karush-Kuhn-Tucker conditions for (6.1),

$$\begin{aligned} B\,p^* + b &= C^T\tilde{\lambda}^* + D^T\tilde{\mu}^*\\ C\,p^* &= c\\ D\,p^* &\geq d\\ \tilde{\mu}^* &\geq 0\\ \tilde{\mu}^{*T}(D\,p^* - d) &= 0. \end{aligned} \tag{6.4}$$

Substituting $p^* = S\,s^*$ and premultiplying the first equation by S^T, we obtain the Karush-Kuhn-Tucker conditions for (6.3),

$$\begin{aligned} (S^T B\,S)\,s^* + (S^T b) &= (C\,S)^T\tilde{\lambda}^* + (D\,S)^T\tilde{\mu}^*\\ (C\,S)\,s^* &= c\\ (D\,S)\,s^* &\geq d\\ \tilde{\mu}^* &\geq 0\\ \tilde{\mu}^{*T}\,((D\,S)\,s^* - d) &= 0, \end{aligned}$$

and hence $(s^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ is a KKT point of the transformed problem.

Theorem 6.2 (Linear Transformation of Equality Constraints)
The solution p^* as well as the multiplier $\tilde{\mu}^*$ of (6.1) do not change if both

sides of the equalities (6.1b) are premultiplied by some nonsingular matrix M. Let $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ be a KKT point of the original problem, then a KKT point of the transformed problem

$$\min_p \; b^T p + \frac{1}{2} p^T B p \tag{6.5a}$$

subject to

$$\begin{aligned} (M\,C)\,p &= (M\,c) && \text{(6.5b)} \\ D\,p &\geq d && \text{(6.5c)} \end{aligned}$$

is given by $(p^*, M^{-T}\tilde{\lambda}^*, \tilde{\mu}^*)$, where M^{-T} denotes the inverse of the transpose of M.

As we will discuss below, such a linear transformation can be used to solve the equalities for some components of p in terms of the remaining ones, so that the former can then be eliminated from the QP problem. The proof of Theorem 6.2 is elementary. Again we start from the Karush-Kuhn-Tucker conditions for the original problem which are given by (6.4) above. Premultiplying the second equation by M and inserting $M^T M^{-T} = I$ between C^T and $\tilde{\lambda}^*$ in the first equation, we obtain the Karush-Kuhn-Tucker conditions for (6.5),

$$\begin{aligned} B\,p^* + b &= (M\,C)^T (M^{-T}\tilde{\lambda}^*) + D^T \tilde{\mu}^* \\ (M\,C)\,p^* &= (M\,c) \\ D\,p^* &\geq d \\ \tilde{\mu}^* &\geq 0 \\ \tilde{\mu}^{*T}(D\,p^* - d) &= 0, \end{aligned}$$

and hence $(p^*, M^{-T}\tilde{\lambda}^*, \tilde{\mu}^*)$ is a KKT point of the transformed problem. A similar result can be proved for linear transformations of the inequality constraints.

6.2 A Condensing Algorithm for $Q(y_k, B_k)$

Let us return to the QP subproblem $Q(y_k, B_k)$ from Section 3.3, which can be written in the above form (6.1) by dropping the iteration index k and introducing the abbreviations

$$b \;:=\; \nabla F(y) \;\in\; \mathbb{R}^{\hat{n}}$$

$$
\begin{aligned}
C &:= \nabla h(y)^T \in \mathbb{R}^{\hat{m} \times \hat{n}} \\
c &:= -h(y) \in \mathbb{R}^{\hat{m}} \\
D &:= \nabla g(y)^T \in \mathbb{R}^{\hat{l} \times \hat{n}} \\
d &:= -g(y) \in \mathbb{R}^{\hat{l}}
\end{aligned}
\tag{6.6}
$$

(note that b, C, c, D, and d depend on the current parameter vector y and that B is positive definite by construction). For an efficient solution of $Q(y, B)$, we will exploit the known block structure of $C = \nabla h(y)^T$ given in (3.31), namely,

$$
C = \begin{pmatrix}
X_0^s & X_0^q & -I & & & & \\
 & & X_1^s & X_1^q & -I & & \\
 & & & & \ddots & & \\
 & & & & X_{m-1}^s & X_{m-1}^q & -I \\
R_a & & & & & & R_b
\end{pmatrix},
$$

in order to eliminate all components of the solution vector p that correspond to variables introduced by multiple shooting. Specifically, if we write p in the form

$$
p := (\Delta s_0, \Delta q_0, \Delta s_1, \Delta q_1, \ldots, \Delta s_{m-1}, \Delta q_{m-1}, \Delta s_m) \tag{6.7}
$$

(compare with the corresponding definition (1.17) for y), then the components Δs_j, $j = 1, 2, \ldots, m$, are to be eliminated. To achieve this goal, we first collect these components at the beginning of the parameter vector by a suitable permutation ("reordering"), and then solve the equality constraints for $(\Delta s_1, \Delta s_2, \ldots, \Delta s_m)$ in terms of the remaining components of p by a block Gaussian elimination procedure. The preceding two steps correspond to linear transformations of the original problem according to Theorems 6.1 and 6.2, respectively, and lead to a modified QP problem which will be denoted by $Q'(y, B')$. After these preparations, it is a simple matter to eliminate the components Δs_j from $Q'(y, B')$, leading to a third QP problem $Q''(y, B'')$ of greatly reduced size. This problem $Q''(y, B'')$ is the one which is actually solved using a standard QP solver, and from its solution the eliminated components as well as the original Lagrange multipliers can be recovered very efficiently by recurrence relations.

Recall that the upper part of $C = \nabla h(y)^T$ (i.e., all block rows except for the last one) comes from the linearized continuity equations (or "matching conditions") at the multiple shooting points. Hence each component Δs_j,

$j = 1, 2, \ldots, m$, is eliminated using the corresponding linearized matching condition. In fact, it makes no difference to the elimination procedure whether the *remaining* constraints are equalities or inequalities. Therefore, the same condensing algorithm can handle the two alternative cases of boundary conditions (equalities or inequalities) which may occur in problem (P1), compare Section 1.

We now consider the elimination procedure in detail and start by defining a suitable permutation matrix $P \in \mathbb{R}^{\hat{n}\times\hat{n}}$ such that

$$P\,p = (\underbrace{\Delta s_1, \Delta s_2, \ldots, \Delta s_m}_{=:\, v \in \mathbb{R}^{mn}}, \underbrace{\Delta s_0, \Delta q_0, \Delta q_1, \ldots, \Delta q_{m-1}}_{=:\, w \in \mathbb{R}^{n+\hat{k}}}) =: p'. \tag{6.8}$$

Then we substitute $p = P^T p'$ into $Q(y, B)$ to arrive at the transformed problem

$$\min_{p'} (P\,b)^T p' + \frac{1}{2}\, p'^T (P\,B\,P^T)\, p' \tag{6.9a}$$

subject to

$$\begin{aligned} (C\,P^T)\,p' &= c & (6.9b)\\ (D\,P^T)\,p' &\geq d, & (6.9c) \end{aligned}$$

where the matrix $(C\,P^T)$ has the form

$$\left(\begin{array}{ccccc|ccccc} -I & & & & & X_0^s & X_0^q & & & \\ X_1^s & -I & & & & & & X_1^q & & \\ & X_2^s & -I & & & & & & X_2^q & \\ & & \ddots & \ddots & & & & & & \ddots \\ & & & X_{m-1}^s & -I & & & & & X_{m-1}^q \\ 0 & & & & R_b & R_a & & & & 0 \end{array}\right)$$

which is obtained by a simple rearrangement of the block columns of C. Similarly, the matrix $(D\,P^T)$ has the form

$$\left(\begin{array}{ccc|ccc} 0 & & & 0 & G_0 & & \\ & 0 & & & & G_1 & \\ & & \ddots & & & & \ddots \\ & & 0 & & & & G_{m-1} \end{array}\right)$$

(compare with the definition of $D = \nabla g(y)^T$ in (3.33)).

The equalities (6.9b) can now be readily solved for v using a block Gaussian elimination procedure analogous to the usual Gaussian elimination method for the triangular decomposition of a matrix (for details on Gaussian elimination see, e.g., [Stoer92]). We define

$$M_j := \begin{pmatrix} I & & & & \\ & \ddots & & & \\ & & I & & \\ & & X_j^s & I & \\ & & & & \ddots \\ & & & & & I \end{pmatrix} \in \mathbb{R}^{\hat{m}\times\hat{m}}, \quad j = 1, 2, \ldots, m-1 \quad (6.10a)$$

$$M_m := \begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & R_b & I \end{pmatrix} \in \mathbb{R}^{\hat{m}\times\hat{m}}, \quad (6.10b)$$

and premultiply both sides of (6.9b) by the matrix

$$M := M_m M_{m-1} \cdots M_1 \quad (6.11)$$

$$= \begin{pmatrix} I & & & & & \\ X_1^s & & I & & & \\ X_2^s X_1^s & & X_2^s & \ddots & & \\ \vdots & & \vdots & & & \\ & & & & I & \\ \prod_{j=1}^{m-1} X_j^s & & \prod_{j=2}^{m-1} X_j^s & & X_{m-1}^s & I \\ R_b \prod_{j=1}^{m-1} X_j^s & & R_b \prod_{j=2}^{m-1} X_j^s & \ldots & R_b X_{m-1}^s & R_b & I \end{pmatrix}$$

(the shorthand $\prod_{j=k}^{m-1} X_j^s$ stands for $X_{m-1}^s \cdots X_k^s$, i.e., $\prod$ is to be interpreted as a "premultiplication operator"). Thus we obtain the transformed equalities

$$\underbrace{M\,(C\,P^T)}_{=:\,C'}\,p' = \underbrace{M\,c}_{=:\,c'}, \quad (6.12)$$

where C' and c' can be written in the form

$$C' = \begin{pmatrix} -I & C'_{12} \\ 0 & C'_{22} \end{pmatrix}, \quad c' = \begin{pmatrix} c'_1 \\ c'_2 \end{pmatrix} \quad (6.13)$$

with $c'_1 \in \mathbb{R}^{mn}$, $c'_2 \in \mathbb{R}^{l_r}$, and the simple blocks $C'_{11} = -I$ and $C'_{21} = 0$ with dimensions $mn \times mn$ and $l_r \times mn$. The remaining blocks C'_{12} and C'_{22} with dimensions $mn \times (n + \hat{k})$ and $l_r \times (n + \hat{k})$ are given by the following formulas:

$$C'_{12} = \begin{pmatrix} X_0^s & X_0^q & & \\ X_1^s X_0^s & X_1^s X_0^q & X_1^q & \\ X_2^s X_1^s X_0^s & X_2^s X_1^s X_0^q & X_2^s X_1^q & \\ \vdots & \vdots & \vdots & \\ \prod_{j=0}^{m-1} X_j^s & (\prod_{j=1}^{m-1} X_j^s)\, X_0^q & (\prod_{j=2}^{m-1} X_j^s)\, X_1^q & \end{pmatrix} \tag{6.14}$$

$$\begin{pmatrix} X_2^q & & \\ \vdots & \ddots & \\ & & X_{m-2}^q \\ (\prod_{j=3}^{m-1} X_j^s)\, X_2^q & \dots & X_{m-1}^s X_{m-2}^q \quad X_{m-1}^q \end{pmatrix}$$

$$C'_{22} = \Big(R_a + R_b\,(\textstyle\prod_{j=0}^{m-1} X_j^s),\ R_b\,(\prod_{j=1}^{m-1} X_j^s)\, X_0^q,\ R_b\,(\prod_{j=2}^{m-1} X_j^s)\, X_1^q, \\ R_b\,(\prod_{j=3}^{m-1} X_j^s)\, X_2^q,\ \dots,\ R_b X_{m-1}^s X_{m-2}^q,\ R_b X_{m-1}^q\Big) \tag{6.15}$$

The calculation of C'_{12} and C'_{22} requires just $m + m\,(m+1)/2 = m\,(m+3)/2$ block multiplications: as indicated by the above notation, each block element below the first nonzero block in each column is obtained by premultiplying the preceding one by an elementary matrix X_j^s (or R_b). Note that this rule extends from C'_{12} to C'_{22}: only the first block element of C'_{22} needs special treatment (there the matrix R_a has to be added to the result of the multiplication).

The right-hand side vector $c' = Mc$ in (6.12) can be calculated efficiently using a recurrence relation, i.e., the matrix vector product Mc need not be explicitly evaluated. In order to derive this recurrence formula, we observe that

$$c = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \\ c_m \end{pmatrix}, \quad c' = \begin{pmatrix} c'_{1,0} \\ c'_{1,1} \\ \vdots \\ c'_{1,m-1} \\ c'_2 \end{pmatrix},$$

where $c_j,\ c'_{1,j} \in \mathbb{R}^n,\ j = 0, 1, \ldots, m-1$, and $c_m,\ c'_2 \in \mathbb{R}^{l_r}$. The components $c_j,\ j = 0, 1, \ldots, m$, correspond to $-h_j$ with the components h_j of $h(y)$ as defined in (3.30). Using the definition of M from (6.11), it can be easily verified that all the components of c' can be computed through the recurrence

$$\begin{aligned} c'_{1,0} &= c_0 \\ c'_{1,j} &= X_j^s\, c'_{1,j-1} + c_j, \quad j = 1, 2, \ldots, m-1 \\ c'_2 &= R_b\, c'_{1,m-1} + c_m. \end{aligned} \tag{6.16}$$

For instance, consider the cases $j = 1,\ 2$, where we obtain

$$\begin{aligned} c'_{1,1} &= X_1^s\, c_0 + c_1 = X_1^s\, c'_{1,0} + c_1 \\ c'_{1,2} &= X_2^s\, X_1^s\, c_0 + X_2^s\, c_1 + c_2 = X_2^s\, (X_1^s\, c_0 + c_1) + c_2 = X_2^s\, c'_{1,1} + c_2. \end{aligned}$$

The recursive calculation of c' by (6.16) is considerably more efficient than the matrix vector multiplication involving M. We will see that M is never actually needed in explicit form throughout the condensing algorithm, i.e., it need not be calculated and stored.

This completes our description of the two successive transformation steps leading from $Q(y, B)$ to $Q'(y, B')$—namely, the reordering of variables and the linear transformation of the equality constraints. In addition to c' and C' we define the vectors and matrices

$$b' = \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix} := P\, b, \quad B' = \begin{pmatrix} B'_{11} & B'_{12} \\ B'^T_{12} & B'_{22} \end{pmatrix} := P\, B\, P^T, \tag{6.17a}$$

$$D' = \begin{pmatrix} 0 & D'_2 \end{pmatrix} := D\, P^T, \quad d' := d, \tag{6.17b}$$

where $b'_1 \in \mathbb{R}^{mn}$, $b'_2 \in \mathbb{R}^{n+\hat{k}}$, and the blocks B'_{11}, B'_{12}, and B'_{22} have the dimensions $mn \times mn$, $mn \times (n + \hat{k})$, and $(n + \hat{k}) \times (n + \hat{k})$, respectively. Figure 6.1 shows in more detail how B' and B are related to each other. The matrix D' has been partitioned into a zero block D'_1 with dimension $\hat{l} \times mn$ and the block

$$D'_2 = \begin{pmatrix} 0 & G_0 & & & \\ & & G_1 & & \\ & & & \ddots & \\ & & & & G_{m-1} \end{pmatrix} \tag{6.18}$$

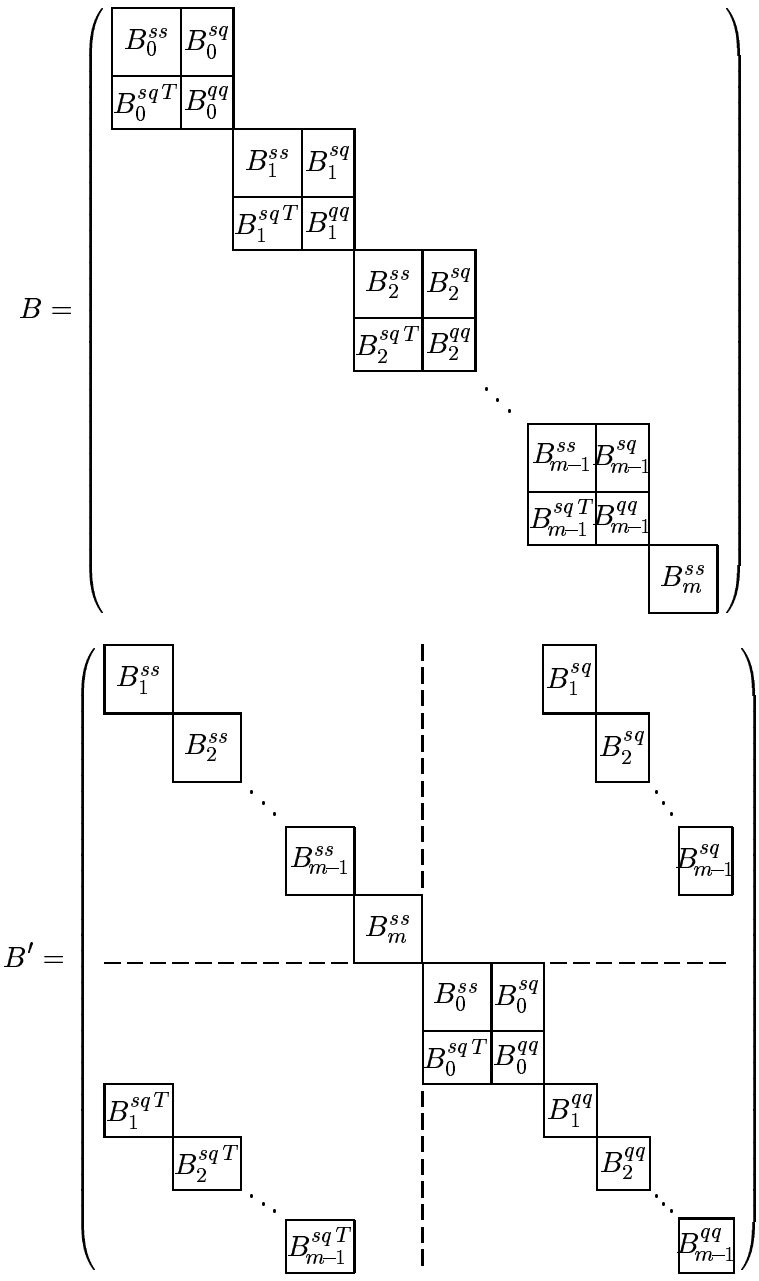

Figure 6.1: Block structure of the Hessian approximations B and B'.

with dimension $\hat{l} \times (n+\hat{k})$. Using this notation, the intermediate QP problem $Q'(y, B')$ can be stated in the form,

$$\min_{(v,w)} \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix}^T \begin{pmatrix} v \\ w \end{pmatrix} + \frac{1}{2} \begin{pmatrix} v \\ w \end{pmatrix}^T \begin{pmatrix} B'_{11} & B'_{12} \\ B'^T_{12} & B'_{22} \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} \tag{6.19a}$$

subject to

$$\begin{pmatrix} -I & C'_{12} \\ 0 & C'_{22} \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} c'_1 \\ c'_2 \end{pmatrix} \tag{6.19b}$$

$$\begin{pmatrix} 0 & D'_2 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} \geq d', \tag{6.19c}$$

which lends itself to the elimination of v. By this elimination step, we arrive at the final problem $Q''(y, B'')$,

$$\min_w \; b''^T w + \frac{1}{2} w^T B'' w \tag{6.20a}$$

subject to

$$C'_{22} w = c'_2 \tag{6.20b}$$

$$D'_2 w \geq d', \tag{6.20c}$$

where

$$B'' = C'^T_{12} B'_{11} C'_{12} + C'^T_{12} B'_{12} + B'^T_{12} C'_{12} + B'_{22} \tag{6.21a}$$

$$b'' = C'^T_{12} b'_1 + b'_2 - C'^T_{12} B'_{11} c'_1 - B'^T_{12} c'_1 \tag{6.21b}$$

are obtained by substituting $v = C'_{12} w - c'_1$ into the objective (6.19a) and simplifying the result (terms which do not involve w can be neglected). For an efficient calculation of B'' and b'' the block sparse structure of B' shown in Figure 6.1 has to be taken into account. The term $C'^T_{12} B'_{11} C'_{12}$ which appears in (6.21a) is given by

$$C'^T_{12} B'_{11} C'_{12} = \begin{pmatrix} \mathcal{B}^{ss}_{0,0} & \mathcal{B}^{sq}_{0,0} & \mathcal{B}^{sq}_{0,1} & \dots & \mathcal{B}^{sq}_{0,m-1} \\ * & \mathcal{B}^{qq}_{0,0} & \mathcal{B}^{qq}_{0,1} & \dots & \mathcal{B}^{qq}_{0,m-1} \\ * & * & \mathcal{B}^{qq}_{1,1} & \dots & \mathcal{B}^{qq}_{1,m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \dots & \mathcal{B}^{qq}_{m-1,m-1} \end{pmatrix}, \tag{6.22}$$

where an asterisk denotes elements which need not be calculated because the resulting matrix is known to be symmetric, and

$$\begin{aligned}\mathcal{B}^{ss}_{0,0} &:= X_0^{s\,T}\, B_1^{ss}\, X_0^s + X_0^{s\,T}\, X_1^{s\,T}\, B_2^{ss}\, X_1^s\, X_0^s + \ldots \\ &\quad + (\textstyle\prod_{j=0}^{m-1} X_j^s)^T\, B_m^{ss}\, (\prod_{j=0}^{m-1} X_j^s) \end{aligned} \tag{6.23a}$$

$$\begin{aligned}\mathcal{B}^{sq}_{0,\ell} &:= (\textstyle\prod_{j=0}^{\ell} X_j^s)^T\, B_{\ell+1}^{ss}\, X_\ell^q + (\prod_{j=0}^{\ell+1} X_j^s)^T\, B_{\ell+2}^{ss}\, X_{\ell+1}^s\, X_\ell^q + \ldots \\ &\quad + (\textstyle\prod_{j=0}^{m-1} X_j^s)^T\, B_m^{ss}\, (\prod_{j=\ell+1}^{m-1} X_j^s)\, X_\ell^q, \\ &\quad \ell = 0, 1, \ldots, m-1 \end{aligned} \tag{6.23b}$$

$$\begin{aligned}\mathcal{B}^{qq}_{k,\ell} &:= X_k^{q\,T}\, (\textstyle\prod_{j=k+1}^{\ell} X_j^s)^T\, B_{\ell+1}^{ss}\, X_\ell^q + \ldots \\ &\quad + X_k^{q\,T}\, (\textstyle\prod_{j=k+1}^{m-1} X_j^s)^T\, B_m^{ss}\, (\prod_{j=\ell+1}^{m-1} X_j^s)\, X_\ell^q, \\ &\quad k = 0, 1, \ldots, m-1 \text{ and } \ell = k, k+1, \ldots, m-1. \end{aligned} \tag{6.23c}$$

The remaining matrix product $C_{12}'^T\, B_{12}' = (B_{12}'^T\, C_{12}')^T$ required for evaluating (6.21a) is given by the following formula:

$$C_{12}'^T\, B_{12}' = \begin{pmatrix} 0 & 0 & X_0^{s\,T} B_1^{sq} & X_0^{s\,T} X_1^{s\,T} B_2^{sq} & \ldots & (\prod_{j=0}^{m-2} X_j^s)^T B_{m-1}^{sq} \\ & & X_0^{q\,T} B_1^{sq} & X_0^{q\,T} X_1^{s\,T} B_2^{sq} & \ldots & X_0^{q\,T} (\prod_{j=1}^{m-2} X_j^s)^T B_{m-1}^{sq} \\ & & & X_1^{q\,T} B_2^{sq} & \ldots & X_1^{q\,T} (\prod_{j=2}^{m-2} X_j^s)^T B_{m-1}^{sq} \\ & & & & \ddots & \vdots \\ & & & & & X_{m-2}^{q\,T} B_{m-1}^{sq} \\ & & & & & 0 \end{pmatrix} \tag{6.24}$$

Note that unlike B and B', the condensed Hessian approximation B'' is a *dense* matrix without any zero blocks. For the calculation of b'' according to (6.21b) only matrix vector multiplications are required since $C_{12}'^T\, B_{11}'\, c_1'$ can be evaluated as $C_{12}'^T\, (B_{11}'\, c_1')$.

We have now discussed how the condensed problem $Q''(y, B'')$ is derived from the original problem $Q(y, B)$ via the intermediate problem $Q'(y, B')$. However, in order to be able to recover the full solution (including the Lagrange multipliers) to problem $Q(y, B)$ from the solution to problem $Q''(y, B'')$, we need to establish the relationships among the KKT points of $Q(y, B)$, $Q'(y, B')$, and $Q''(y, B'')$. These important results are provided by the following theorem.

Theorem 6.3 (Transformation Rules for the KKT Points)
Let $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ be a KKT point of $Q(y, B)$. Then $(p'^*, \tilde{\lambda}'^*, \tilde{\mu}^*)$ with

$$p'^* = \begin{pmatrix} v^* \\ w^* \end{pmatrix} = P\,p^* \tag{6.25a}$$

$$\tilde{\lambda}'^* = \begin{pmatrix} \tilde{\nu}^* \\ \tilde{\omega}^* \end{pmatrix} = M^{-T}\,\tilde{\lambda}^* \tag{6.25b}$$

is a KKT point of $Q'(y, B')$, and $(w^*, \tilde{\omega}^*, \tilde{\mu}^*)$ is a KKT point of $Q''(y, B'')$. The matrices P and M are defined in equations (6.8) and (6.11), respectively. The multipliers $\tilde{\nu}^* \in \mathbb{R}^{mn}$ correspond to the first block of equalities in (6.19b) which involve both v and w, while the multipliers $\tilde{\omega}^* \in \mathbb{R}^{l_r}$ correspond to the second block of equalities which involve only w. Note that the multipliers $\tilde{\mu}^*$ corresponding to the inequalities are not transformed.

Conversely, let a KKT point $(w^*, \tilde{\omega}^*, \tilde{\mu}^*)$ of $Q''(y, B'')$ be given. Then $(p'^*, \tilde{\lambda}'^*, \tilde{\mu}^*)$ with

$$p'^* = \begin{pmatrix} v^* \\ w^* \end{pmatrix} = \begin{pmatrix} C'_{12}\,w^* - c'_1 \\ w^* \end{pmatrix} \tag{6.26a}$$

$$\tilde{\lambda}'^* = \begin{pmatrix} \tilde{\nu}^* \\ \tilde{\omega}^* \end{pmatrix} = \begin{pmatrix} -B'_{11}\,v^* - B'_{12}\,w^* - b'_1 \\ \tilde{\omega}^* \end{pmatrix} \tag{6.26b}$$

is a KKT point of $Q'(y, B')$, and $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ with

$$p^* = P^T\,p'^* \tag{6.27a}$$

$$\tilde{\lambda}^* = M^T\,\tilde{\lambda}'^* \tag{6.27b}$$

is a KKT point of the original problem $Q(y, B)$. For the definitions of C'_{12}, c'_1, B'_{11}, B'_{12}, and b'_1, see equations (6.12)–(6.14) and (6.17a).

If the KKT point of $Q(y, B)$ is uniquely determined, then the same is true for the KKT points of $Q'(y, B')$ and $Q''(y, B'')$.

The rule governing the step from $Q(y, B)$ to $Q'(y, B')$ follows directly from Theorems 6.1 and 6.2 (setting $S = P^{-1} \equiv P^T$ in Theorem 6.1). The corresponding "inverse" rule for the step from $Q'(y, B')$ to $Q(y, B)$ is thereby proved as well (the inverse linear transformations were assumed to exist, and hence Theorems 6.1 and 6.2 can be applied once more). In order to prove the remaining pair of transformation rules, we consider the Karush-Kuhn-Tucker conditions for $Q'(y, B')$, which are satisfied by $(p'^*, \tilde{\lambda}'^*, \tilde{\mu}^*)$:

$$\begin{pmatrix} B'_{11} & B'_{12} \\ B'^T_{12} & B'_{22} \end{pmatrix} \begin{pmatrix} v^* \\ w^* \end{pmatrix} + \begin{pmatrix} b'_1 \\ b'_2 \end{pmatrix} = \begin{pmatrix} -I & 0 \\ C'^T_{12} & C'^T_{22} \end{pmatrix} \begin{pmatrix} \tilde{\nu}^* \\ \tilde{\omega}^* \end{pmatrix} + \begin{pmatrix} 0 \\ D'^T_2 \end{pmatrix} \tilde{\mu}^* \quad (6.28a)$$

$$\begin{pmatrix} -I & C'_{12} \\ 0 & C'_{22} \end{pmatrix} \begin{pmatrix} v^* \\ w^* \end{pmatrix} = \begin{pmatrix} c'_1 \\ c'_2 \end{pmatrix} \quad (6.28b)$$

$$\begin{pmatrix} 0 & D'_2 \end{pmatrix} \begin{pmatrix} v^* \\ w^* \end{pmatrix} \geq d' \quad (6.28c)$$

$$\tilde{\mu}^* \geq 0 \quad (6.28d)$$

$$\tilde{\mu}^{*T} \left(\begin{pmatrix} 0 & D'_2 \end{pmatrix} \begin{pmatrix} v^* \\ w^* \end{pmatrix} - d' \right) = 0. \quad (6.28e)$$

The first block row of (6.28a) provides an explicit relationship for $\tilde{\nu}^*$ in terms of v^* and w^*,

$$\tilde{\nu}^* = -B'_{11}\, v^* - B'_{12}\, w^* - b'_1,$$

while the first block row of (6.28b) gives v^* in terms of w^*,

$$v^* = C'_{12}\, w^* - c'_1.$$

By substituting these two relationships into the remaining equations and thereby eliminating v^* and $\tilde{\nu}^*$ from (6.28), we obtain, after simplification,

$$\begin{aligned} B''\, w^* + b'' &= C'^T_{22}\, \tilde{\omega}^* + D'^T_2\, \tilde{\mu}^* \\ C'_{22}\, w^* &= c'_2 \\ D'_2\, w^* &\geq d' \\ \tilde{\mu}^* &\geq 0 \\ \tilde{\mu}^{*T}\, (D'_2\, w^* - d') &= 0, \end{aligned} \quad (6.29)$$

with B'' and b'' as defined in (6.21a) and (6.21b). Since (6.29) represents the Karush-Kuhn-Tucker conditions for $Q''(y, B'')$, we have thereby shown that $(w^*, \tilde{\omega}^*, \tilde{\mu}^*)$ is a KKT point of $Q''(y, B'')$ if $(p'^*, \tilde{\lambda}'^*, \tilde{\mu}^*)$ is a KKT point of $Q'(y, B')$.

Conversely, if we start from a point $(w^*, \tilde{\omega}^*, \tilde{\mu}^*)$ satisfying the conditions (6.29) (thus being a KKT point of $Q''(y, B'')$), then it is clear that $(p'^*, \tilde{\lambda}'^*, \tilde{\mu}^*)$ with p'^* and $\tilde{\lambda}'^*$ according to (6.26a) and (6.26b) satisfies (6.28), and therefore is a KKT point of $Q'(y, B')$.

It remains to be shown that the KKT points of $Q'(y, B')$ and $Q''(y, B'')$ are uniquely determined when $Q(y, B)$ has a unique KKT point. To this

end, we first observe that all three QP problems are *convex*: the matrix B is positive definite by construction, and hence $B' = P\,B\,P^T$ and B'' as given in (6.21a) are positive definite as well. The latter can be seen from

$$w^T B'' w = \begin{pmatrix} w^T C_{12}'^T & w^T \end{pmatrix} \begin{pmatrix} B'_{11} & B'_{12} \\ B_{12}'^T & B'_{22} \end{pmatrix} \begin{pmatrix} C'_{12}\, w \\ w \end{pmatrix} > 0,$$

which holds for arbitrary w since B' is positive definite. Furthermore, if the rows of C and the rows of D corresponding to active inequalities are linearly independent—that is, the solution p^* of $Q(y, B)$ is a regular point—then the same is true for the transformed matrices C', D', and C'_{22}, D'_2, respectively. Therefore, the solutions p'^* of $Q'(y, B')$ and w^* of $Q''(y, B'')$ are regular points as well. Finally, since convexity and regularity automatically imply strict complementarity for QP problems, it follows that the KKT points of $Q'(y, B')$ and $Q''(y, B'')$ satisfy the Jacobi uniqueness condition whenever this is true for the KKT point of $Q(y, B)$. This completes the proof of Theorem 6.3.

Given a KKT point of the condensed problem $Q''(y, B'')$, the corresponding KKT point of the original problem $Q(y, B)$ could in principle be calculated using the explicit relations stated in Theorem 6.3. However, it is much more efficient to use instead the recurrence relations to be discussed in the following. Once the condensed parameter vector

$$w^* = (\Delta s_0^*, \Delta q_0^*, \Delta q_1^*, \ldots, \Delta q_{m-1}^*)$$

has been determined by solving $Q''(y, B'')$, we can calculate the component vectors of

$$v^* = (\Delta s_1^*, \Delta s_2^*, \ldots, \Delta s_m^*)$$

recursively from equation (6.9b), which leads to

$$\Delta s_{j+1}^* = X_j^s\, \Delta s_j^* + X_j^q\, \Delta q_j^* - c_j, \quad j = 0, 1, \ldots, m-1, \tag{6.30}$$

where $c_j = -h_j = -(x(t_{j+1}; s_j, q_j) - s_{j+1})$. This recursive calculation of previously eliminated components is in close analogy to the "classical" multiple shooting method for two-point boundary value problems as described by, e.g., [Stoer92, Bock81b].

In order to determine the multiplier vector $\tilde{\lambda}^*$ of the original problem from v^*, w^*, and $\tilde{\omega}^*$, we first have to calculate

$$\tilde{\nu}^* = -B'_{11}\, v^* - B'_{12}\, w^* - b'_1, \tag{6.31}$$

and then apply (6.27b),

$$\tilde{\lambda}^* = M^T \begin{pmatrix} \tilde{\nu}^* \\ \tilde{\omega}^* \end{pmatrix},$$

where M is the matrix defined in (6.11). The explicit multiplication by M^T in the last formula can again be avoided through the use of a more efficient recursive calculation procedure. Writing $\tilde{\lambda}^*$ and $\tilde{\nu}^*$ in partitioned form,

$$\tilde{\lambda}^* = \begin{pmatrix} \tilde{\lambda}_0^* \\ \tilde{\lambda}_1^* \\ \vdots \\ \tilde{\lambda}_{m-1}^* \\ \tilde{\lambda}_m^* \end{pmatrix}, \quad \tilde{\nu}^* = \begin{pmatrix} \tilde{\nu}_0^* \\ \tilde{\nu}_1^* \\ \vdots \\ \tilde{\nu}_{m-1}^* \end{pmatrix},$$

where $\tilde{\lambda}_j^*,\ \tilde{\nu}_j^* \in \mathbb{R}^n,\ j = 0, 1, \ldots, m-1$, and $\tilde{\lambda}_m^* \in \mathbb{R}^{l_r}$, it can be easily verified from (6.11) and (6.27b) that

$$\begin{aligned} \tilde{\lambda}_m^* &= \tilde{\omega}^* \\ \tilde{\lambda}_{m-1}^* &= \tilde{\nu}_{m-1}^* + R_b^T \tilde{\lambda}_m^* \\ \tilde{\lambda}_{m-k}^* &= \tilde{\nu}_{m-k}^* + X_{m-k+1}^{sT} \tilde{\lambda}_{m-k+1}^*, \quad k = 2, 3, \ldots, m. \end{aligned} \tag{6.32}$$

For instance, when $k = 2$, we obtain

$$\begin{aligned} \tilde{\lambda}_{m-2}^* &= \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} \tilde{\nu}_{m-1}^* + X_{m-1}^{sT} R_b^T \tilde{\omega}^* \\ &= \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} (\tilde{\nu}_{m-1}^* + R_b^T \tilde{\lambda}_m^*) \\ &= \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} \tilde{\lambda}_{m-1}^*, \end{aligned}$$

which is in agreement with the last formula in (6.32).

We can now summarize the condensing algorithm as it is implemented in MUSCOD. Given the formulation (6.1) of the original QP subproblem $Q(y, B)$ with the abbreviations defined in (6.6), perform the following steps in order to determine a KKT point $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$.

CON1. *Obtain b', B', and D' by rearranging b, B, and D.* The rearrangement according to equations (6.17a) and (6.17b) can be accomplished by just reordering the component vectors of b and the blocks of B and D, respectively. Therefore, no significant work has to be done to obtain b', B', and D'.

CON2. *Calculate the blocks C'_{12} and C'_{22} of C'.* In order to obtain the first block column of C'_{12} and C'_{22}, start with the element X_0^s and successively calculate

$$\prod_{j=0}^{l} X_j^s, \quad l = 1, 2, \ldots, m-1,$$

and

$$R_a + R_b \left(\prod_{j=0}^{m-1} X_j^s\right).$$

For the remaining block columns, start with X_k^q, $k = 0, 1, \ldots, m-1$, and successively calculate

$$\left(\prod_{j=k+1}^{l} X_j^s\right) X_k^q, \quad l = k+1, k+2, \ldots, m-1,$$

and

$$R_b \left(\prod_{j=k+1}^{m-1} X_j^s\right) X_k^q.$$

Recall that $\prod_{j=k}^{l} X_j^s$ stands for $X_l^s \cdots X_k^s$.

CON3. *Recursively calculate the components of c'.* Calculate c'_1 and c'_2 from the components of $h(y)$ using the recurrence

$$\begin{aligned} c'_{1,0} &= -h_0 \\ c'_{1,j} &= X_j^s \, c'_{1,j-1} - h_j, \quad j = 1, 2, \ldots, m-1 \\ c'_2 &= R_b \, c'_{1,m-1} - h_m, \end{aligned}$$

with h_j, $j = 0, 1, \ldots, m$, as defined in (3.30).

CON4. *Calculate B'' and b''.* From the previously determined matrices B', C', and vectors b', c', calculate

$$B'' = C'^T_{12} \, B'_{11} \, C'_{12} + C'^T_{12} \, B'_{12} + B'^T_{12} \, C'_{12} + B'_{22},$$

and

$$b'' = C'^T_{12} \, b'_1 + b'_2 - C'^T_{12} \, B'_{11} \, c'_1 - B'^T_{12} \, c'_1.$$

This completes the formulation of the condensed problem $Q''(y, B'')$.

CON5. *Solve the condensed problem.* Using a standard QP solver for dense problems, calculate a KKT point $(w^*, \tilde{\omega}^*, \tilde{\mu}^*)$ of $Q''(y, B'')$.

CON6. *Recursively calculate the components of* v^*. Calculate the components $\Delta s_1^*, \Delta s_2^*, \ldots, \Delta s_m^*$ from $\Delta s_0^*, \Delta q_0^*, \Delta q_1^*, \ldots, \Delta q_m^*$ and the components of $h(y)$ using the recurrence

$$\Delta s_{j+1}^* = X_j^s \, \Delta s_j^* + X_j^q \, \Delta q_j^* + h_j, \quad j = 0, 1, \ldots, m-1,$$

with h_j, $j = 0, 1, \ldots, m-1$, as defined in (3.30).

CON7. *Determine* $\tilde{\nu}^*$ *and recursively calculate the components of* $\tilde{\lambda}^*$. Determine the multiplier vector $\tilde{\nu}^*$ through

$$\tilde{\nu}^* = -B_{11}' \, v^* - B_{12}' \, w^* - b_1',$$

and partition $\tilde{\nu}^*$ according to

$$\tilde{\nu}^* = (\tilde{\nu}_0^*, \tilde{\nu}_1^*, \ldots, \tilde{\nu}_{m-1}^*).$$

Then calculate the components of the multiplier vector $\tilde{\lambda}^*$ from $\tilde{\omega}^*$ and $\tilde{\nu}_j^*$, $j = 0, 1, \ldots, m-1$, using the recurrence

$$\begin{aligned} \tilde{\lambda}_m^* &= \tilde{\omega}^* \\ \tilde{\lambda}_{m-1}^* &= \tilde{\nu}_{m-1}^* + R_b^T \, \tilde{\lambda}_m^* \\ \tilde{\lambda}_{m-k}^* &= \tilde{\nu}_{m-k}^* + X_{m-k+1}^{sT} \, \tilde{\lambda}_{m-k+1}^*, \quad k = 2, 3, \ldots, m. \end{aligned}$$

CON8. *Obtain* p^* *by reordering the components of* (v^*, w^*). Again, the rearrangement can be accomplished by simple "renumbering" of components. Thus, the determination of a KKT point $(p^*, \tilde{\lambda}^*, \tilde{\mu}^*)$ of the original QP subproblem $Q(y, B)$ is complete.

We conclude this discussion of the "classical" condensing algorithm with an important remark: since we have intended here to exploit as far as possible the *specific* structure of the QP subproblem, it is clear that the resulting condensing algorithm is more or less *tied* to this specific structure—whenever the structure of the problem changes, the condensing procedure has to be changed as well.[‡] In Section 7, we will present a more general condensing algorithm which is capable of handling separated interior point constraints.

[‡]However, recall that trivial changes like replacing the boundary equality conditions by inequalities do not pose a problem.

6.3 The QP Solver

We now briefly discuss the solution of the condensed QP problem $Q''(y, B'')$ using a standard *active-set strategy* (applied to the primal problem). For dense QP problems of moderate size, there exist a number of numerically stable and efficient active-set methods available as library routines [Gill78]. At the end of this section, we will give a brief overview of various non-sparse QP methods, and we will also point out an alternative approach to the solution of the original QP problem $Q(y, B)$ without the use of condensing.

Let us first consider the solution of the following equality-constrained QP problem (EQP),

$$\min_p \; b^T p + \frac{1}{2} p^T B p \tag{6.33a}$$

subject to

$$C p \; = \; c, \tag{6.33b}$$

where we have returned to the generic notation introduced at the beginning of this section (here we assume that $p \in \mathbb{R}^n$, and C is of dimension $m \times n$). We will see that problems of the form (EQP) occur as *subproblems* within the active-set method for the solution of the general QP problem (6.1) which has additional inequalities. The Karush-Kuhn-Tucker conditions for problem (EQP) are given by the linear system

$$\underbrace{\begin{pmatrix} B & C^T \\ C & 0 \end{pmatrix}}_{=: K} \begin{pmatrix} p^* \\ -\tilde{\lambda}^* \end{pmatrix} = \begin{pmatrix} -b \\ c \end{pmatrix}, \tag{6.34}$$

where the matrix K with dimension $(n+m) \times (n+m)$ is called *Karush-Kuhn-Tucker* (or *KKT*) *matrix*. When C has full rank and B is positive definite on the null space of C, K is nonsingular, and the unique solution $(p^*, \tilde{\lambda}^*)$ of the system (6.34) corresponds to a local (and also global) optimum of problem (EQP).[‡] Given an arbitrary point p, the linear relation (6.34) allows a convenient representation of the step Δp from p to p^*. By substituting $p^* = p + \Delta p$ into (6.34) and reversing signs we obtain

$$\begin{pmatrix} B & C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} -\Delta p \\ \tilde{\lambda}^* \end{pmatrix} = \begin{pmatrix} b + B p \\ C p - c \end{pmatrix}. \tag{6.35}$$

[‡]Note that the positive definiteness requirement is always satisfied for *convex* problems.

Since K is symmetric but *indefinite*, the systems (6.34) and (6.35) may be solved directly using a symmetric indefinite factorization (e.g., the Bunch-Parlett factorization, see [Gill89]), which is appropriate especially for sparse problems. Most often for non-sparse problems, however, another approach is used which is based on a representation of the solution Δp of (6.35) in the form

$$\Delta p = Y\,\Delta p_Y + Z\,\Delta p_Z,$$

where Y is a basis for the range space of C^T and Z is a basis for the null space of C (the matrices Y and Z may be obtained, e.g., from a QR factorization of C^T). Thereby it is possible to determine Δp_Y, Δp_Z, and $\tilde{\lambda}^*$ successively by solving a sequence of three simpler linear systems. For further detail on this and other numerically stable methods for the solution of (EQP), see for instance [Gill78, Gill89].

Now we shall outline the steps of a primal-feasible active-set method for the solution of the general QP problem (6.1). Let C^* denote the matrix of all constraints active at the solution p^*, i.e., C^* contains all rows of C and the rows of D corresponding to the active inequalities. Then the central idea is the following: if an arbitrary *feasible* point p (i.e., $C\,p = c$, $D\,p \geq d$) and the correct active set C^* are known, the step Δp from p to p^* can be calculated directly by solving problem (EQP) with C^* instead of C. Of course, C^* is in general not known beforehand and must be determined iteratively. The algorithm thus has to employ a *prediction* C^w of C^* which is called the *working set*.

First, an arbitrary initial feasible point p is required. This can be obtained, for instance, using the phase 1 simplex algorithm (see, e.g., [Rekl83]). The search for an initial feasible point may account for a significant fraction of the total computational work required to solve the problem. All linearly independent constraints active at p make up the initial working set C^w. Then a search direction Δp is determined by solving (6.35) with $C = C^w$. Two situations may arise:

1. *The point $p+\Delta p$ violates one or more inequalities not currently in C^w.* Obviously, C^w is not the correct active set in this case. In order to remain feasible, a nonnegative steplength $\bar{\alpha} < 1$ is determined such that $\bar{\alpha}\,\Delta p$ is the largest step that retains feasibility. The inequality that becomes active at $p + \bar{\alpha}\Delta p$ is then added to the working set C^w, and a new search direction is computed with this modified working set using $p + \bar{\alpha}\,\Delta p$ as starting point. Notice that $\bar{\alpha} = 0$ may occur if there

are one or more so-called *idle constraints* (inequalities already active at p but not yet contained in C^w).

2. *The point $p+\Delta p$ is feasible.* If the Lagrange multipliers corresponding to the *inequality rows* in C^w are all nonnegative, it can be easily seen that $p^* = p + \Delta p$ provides not only the minimizer of problem (EQP) with $C = C^w$ but also the minimizer of the general QP problem (6.1) because the Karush-Kuhn-Tucker conditions (6.4) for the latter are satisfied. This implies that the correct active set C^* has been identified, and the algorithm terminates. Otherwise, at least one of the inequalities has a strictly negative multiplier, and hence there exists a feasible descent direction which moves off this constraint. Consequently such a constraint is removed from the working set C^w and a new search direction is computed using $p + \Delta p$ as the starting point.

Under suitable assumptions, methods of this general structure will converge to a local solution of problem (6.1) in a finite number of iterations. However, for problems with large numbers of inequalities, many iterations may be required to identify the correct active set, and the methods tend to become inefficient. In the context of SQP this difficulty can be largely overcome by using the active set of the last QP subproblem as a prediction for that of the current one, since it has been found that the correct active set is usually identified quite early in the solution process. Hence the QP solver used for SQP should permit "warm starts", i.e., it should be able to exploit user-provided predictions of the active set to enhance efficiency.

Another important consideration for the practical implementation of QP methods is the efficient handling of simple bounds ("box constraints") on the variables: the QP solver should allow specification of these simple bounds separately from the general linear inequality constraints and should fully exploit their structure to enable significant savings in storage and computational work [Chen84].

Finally, we give an overview of three important active-set QP methods which have been extensively used in the context of SQP:

- *Primal active-set method of Fletcher (1971).* Available as Harwell subroutine VE02; for a detailed description see [Flet71]. This is the QP solver used by the original Harwell version of Powell's SQP algorithm (VF02/VF12). Subroutine VE02 is generally regarded as inefficient today [Bieg85].

- *Primal active-set method of Gill et al. (1978/83).* Available as Fortran package QPSOL from the authors; for a description see [Gill78, Gill83].[‡] This method has been used in a number of SQP implementations (e.g., [Locke83, Chen84, Bieg85]) and has been found to be very stable and efficient.

- *Dual active-set method of Goldfarb and Idnani (1983).* Available as Harwell subroutine VE17; the method has been first described in [Gold83] (see also [Powe85]). This QP method is used by the improved Harwell version of Powell's SQP algorithm (VF03/VF13). An advantage of this dual method compared to primal methods is that it does not require the (possibly expensive) search for an initial feasible point, because a dual feasible starting point is trivially known. The Harwell implementation (which is due to Powell) seems to be very stable and efficient [Spel85].

There are, of course, alternative approaches to the solution of quadratic programming problems—for instance the *modified simplex type methods* (for convex problems) first proposed by Beale [Beale55, Beale59], and the *constrained least-squares methods* (for problems where a factorization $B = A^T A$ is available) which are essentially due to Stoer [Stoer71] and have been further developed in [Schi79]. However, a complete review of QP methods is beyond the scope of this discussion.

At this point, we shall just briefly mention one particularly promising new approach to the solution of the *uncondensed* QP problem $Q(y, B)$. As has been recently shown by Steinbach [Stei95a], the m-stage block sparse structure of $Q(y, B)$ can be directly exploited in a very efficient way by a special, recursive $O(m)$ algorithm for the solution of the equality-constrained problem EQP, thus rendering the condensing procedure unnecessary. This recursive KKT solver can be embedded either in an active set strategy or in an *interior point method* (see, e.g., [Vand93]). Particularly the latter approach seems to have great potential for large-scale optimal control problems with many path constraints [Stei95a].

[‡]Essentially the same subroutine is also available in the NAG library as routine E04NAF.

7 Algorithm Summary and Extensions

7.1 The Building Blocks of MUSCOD

Now we summarize the algorithm developed so far from an implementational point of view. Table 7.1 gives an overview of the proposed new modular organization of MUSCOD. (The original version of the code as developed by Plitt [Plitt81] is more or less "monolithic" and hence very difficult to modify or extend.) On the highest level, the algorithm has been structured in a way that allows a natural representation of the basic steps which invariably occur in SQP-type methods. Therefore, the chosen modular structure should provide a reasonably general framework for the implementation of such methods. Alternative realizations for certain parts of the algorithm can be quickly implemented by replacing the corresponding blocks, without changing the basic functionality and the interfaces of the high-level modules. This should make future modifications and extensions much easier.

Table 7.1: Modular organization of MUSCOD.

Module	Driver Routine(s)	Description
MUSCOD	*main()*	pre- and postprocessing
MSSQP	*mssqp()*	control of SQP solution process
EVAL	*eval(), scale_()*	function and gradient evaluation
IND	*ind()*	ODE solver interface
MODEL	*pfcn(), ffcn(), cfcn()*	model interface
HESS	*iesth(), glag(), revh()*	Hessian approximation
SOLVE	*solve()*	generation of the next iterate (QP subproblem solution)
COND	*cond()*	condensing algorithm
QPS	*qps()*	QP solver interface
TCHK	*tchk()*	termination check
LSEA	*lsea()*	line search

It is important to note that, strictly speaking, neither the ODE solver with internal numerical differentiation, nor the QP solver are part of the MUSCOD package. Instead, MUSCOD provides the generic interfaces *ind()* and *qps()*, respectively, to accomodate a variety of existing solvers. The *ind()* interface should be general enough to be useful not only in the context of ODE models but also for DAE models and for the generalized multiple shooting structure (these more general model types will be discussed in Section 7.2).

The implementation of a block-oriented algorithm like MUSCOD critically depends on the basic data structures used. It is very hard, if not impossible, to write readable and maintainable code without structured types which allow to directly represent the underlying hierarchical structure of the data, and without the possibility to dynamically allocate variables of these types. (These are the major problems with Fortran 77.) On the other hand, the introduction of convenient dynamic data structures such as partitioned vectors or block matrices should not be at the expense of reduced efficiency or portability. At the same time, full compatibility with existing Fortran code (especially libraries) should be maintained. One possible way to meet all these requirements is to use the C programming language together with a carefully designed library which provides Fortran-compatible structured types and the corresponding basic operations, see [Lein95b].

We assume that the control of the SQP solution process is centralized in module MSSQP, which in addition provides the facilities for the storage of the current

- function and gradient values,
- Hessian approximation,
- iterate and step,
- multipliers, and
- gradients of the Lagrangian function.

Each other module is explicitly given access to the data items it requires, e.g., through the use of pointers. It is essential that this "central data" is organized in a flexible way, allowing the representation of various forms of the discretized optimal control problem (see Section 7.2 for examples of more general problem formulations).

We have adopted here the traditional procedural-modular programming paradigm to illustrate one possible form of implementation. However, it should be mentioned that an object-oriented approach may significantly ease the programmer's job and is likely to enhance the flexibility and adaptability of the resulting code. These potential advantages are mainly due to the powerful facilities for *data abstraction* (user-defined types) which are offered by modern object-oriented programming (OOP) languages like C++. On the other hand, it is interesting to note that there seems to be comparatively little scope for using the more advanced OOP-concepts such as inheritance in the area of "classical" numerical computation [Stro91]. Note also that inheritance should actually be *excluded* from the parts of a C++ program that must interface directly with code written in other languages like Fortran or C to maintain compatibility with traditional data structures [Stro91].

In the following, we will briefly discuss the modules of Table 7.1 and their interfaces (driver routines). Note that each module may contain additional computational routines as well as local data, but these internal details shall *not* be discussed at this point. The function call tree for the driver routines of MUSCOD is shown in Figure 7.1.

7.1.1 Function and Gradient Evaluation

The evaluation of functions and gradients is implemented by modules EVAL, IND, and MODEL. Module EVAL provides the high-level driver routine *eval()* which returns the function and gradient values for the current iterate. There is an option to calculate only the function values and no gradients (this is necessary for the line search which requires a number of additional function evaluations).

Recall that outside *eval()*, all calculations are done in terms of *scaled* variables and functions, while the model itself is *unscaled*. Hence *eval()* has to perform appropriate scaling operations on input and output. All scaling operations are simple multiplications by the appropriate scale factors (which should always be integer powers of the floating-point base). The scale factors are locally stored in EVAL and must be initialized by calling a special initialization routine. (Note that the scale factors for the gradients are obtained from the variable and function scales by the chain rule.)

Module IND provides the generic driver *ind()* for the model evaluation method (e.g., ODE solver with internal numerical differentiation) which is called once for each control interval, returning the function and gradient values at the interval endpoint.

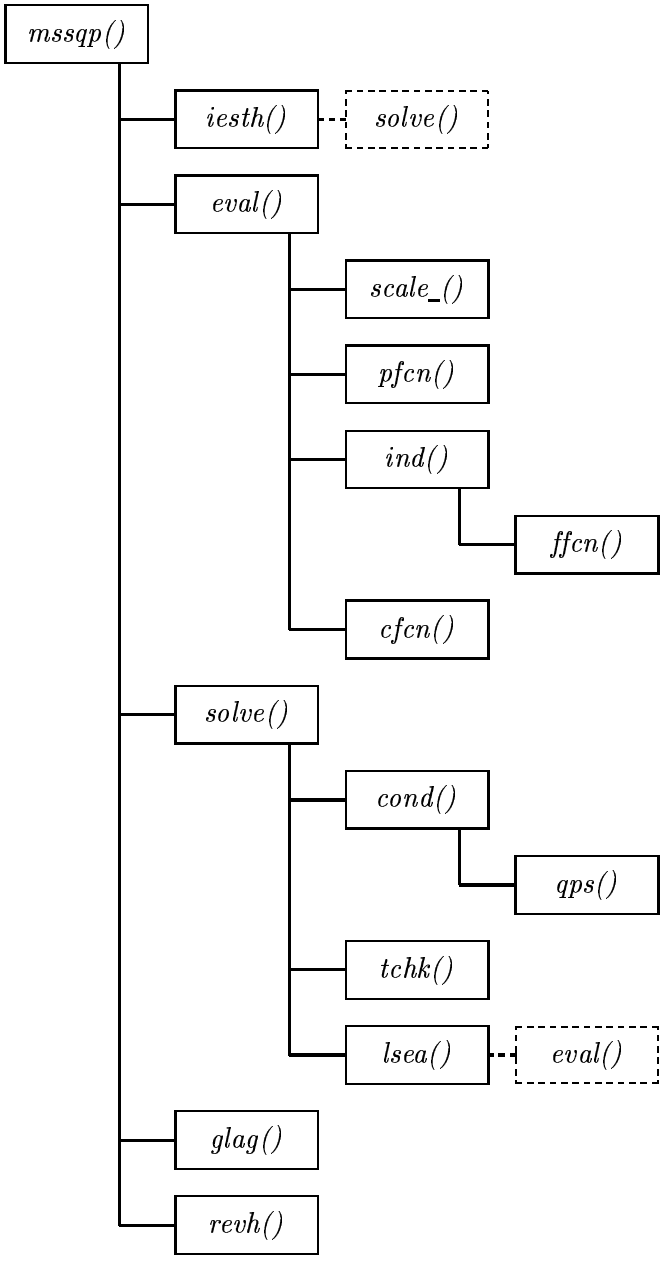

Figure 7.1: Function call tree of MUSCOD (driver routines only).

The model description itself is contained in module MODEL. Three routines must be specified by the user,

- a routine *pfcn()* which returns the relevant model parameters (like dimensions, etc.),
- a routine *ffcn()* which specifies the right-hand sides of the ODE model in terms of the state and control variables, and
- a routine *cfcn()* which contains analytical formulas for the constraint functions (excluding, of course, the continuity conditions, which do not have an analytical representation).

7.1.2 Approximation of the Hessian

The routines for approximating the Hessian matrix are implemented by module HESS. Three driver routines are provided:

- Routine *iesth()* calculates the initial estimate for the Hessian matrix. To find the solution of the minimum-norm QP problem $Q_c(y_0)$, routine *solve()* is called with the option that the solution of the QP problem is directly returned as the current step.
- Routine *glag()* calculates the gradient of the Lagrangian function.
- Routine *revh()* implements the BFGS block update.

7.1.3 Generation of the Next Iterate

The generation of the next iterate is implemented by modules SOLVE, COND, QPS, TCHK, and LSEA. Module SOLVE provides the high-level driver routine *solve()* which returns the current step, the new iterate, the new multipliers, and a flag which indicates whether the convergence criterion is satisfied. The present implementation is based on a line search in the direction of the solution of the current QP subproblem. However, essentially the same interface could be used, e.g., for a trust region method. There is an alternative option to directly return the solution of the QP subproblem as current step (which in our context means bypassing the termination check and the line search).

Module COND implements the condensing algorithm. The driver routine *cond()* returns a KKT point of the current QP subproblem. For the solution

of the condensed problem, a standard QP solver (library routine) is called via the generic interface *qps()* which is defined in module QPS.

The convergence criterion (e.g., KKT-tolerance) used to check whether the algorithm should be terminated is implemented by module TCHK. The driver routine *tchk()* returns a flag indicating the result of the convergence check.

Module LSEA implements both the start relaxation and the line search. The driver routine *lsea()* returns the current steplength (on the first call of the routine, the start relaxation is used instead of the normal line search). For the required function evaluations, routine *eval()* is called with the option to calculate no gradients.

7.1.4 An Overview of the Complete Algorithm

A flow diagram for the central part of the algorithm—the SQP iteration loop in *mssqp()*—is given in Figure 7.2. For each driver routine called, the required input data and the generated output data are shown. The calling sequence is such that the output can always *replace* the old values of the corresponding data items since these old values are no longer needed. (Compare the arrangement of the steps M1–M10 of the simplified model algorithm in Section 3.) Note that only *scaled* variables and functions appear on this level, i.e., the optimization is done for the scaled version ($P2^s$) of the discretized optimal control problem.

The general algorithm structure (including the module interfaces) can be kept essentially independent of the specific implementation of the modules if it is ensured that *only* the parameters shown in Figure 7.2—which are directly related to the optimization problem—are used in the argument lists for the information exchange between modules. Any additional parameters which may be required by a specific module implementation should not be passed each time the corresponding driver routine is called, but rather they should be initialized before starting the calculations. Therefore, each module should provide an *initialization routine* to be called before any other routine of the module can be used (in case of module EVAL, one would thereby initialize, e.g., the internal scale factors). If the modules have been properly designed, it is likely that only this "initialization interface" has to be modified when the implementation of the corresponding module is changed.

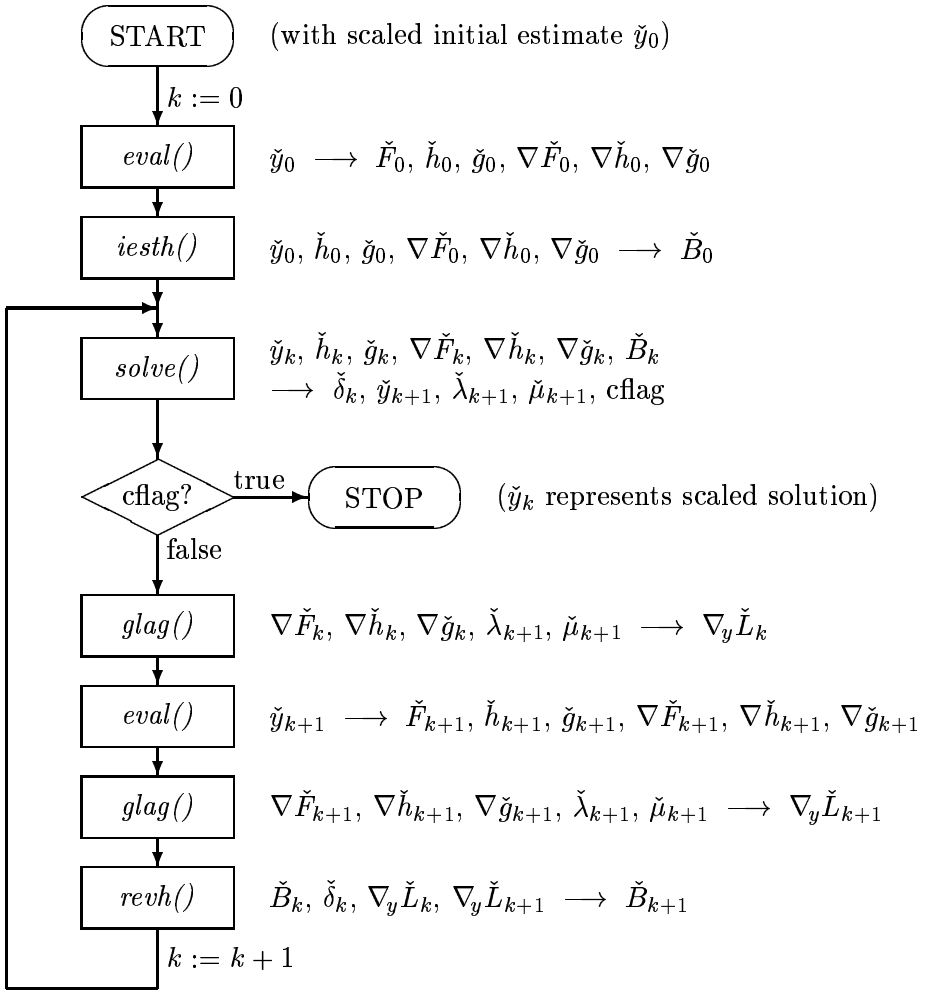

Figure 7.2: Flow diagram for the SQP iteration loop in *mssqp()*.

7.2 Extensions to the Basic Solution Method

7.2.1 Interior Point Constraints

Linearly Coupled Interior Point Constraints. As mentioned in Section 1, only *separated* interior point constraints (those without nonlinear coupling between different points) can be handled directly. We first consider *linearly coupled* interior point constraints as a natural generalization of the two-point boundary conditions (1.4c), and we briefly discuss the necessary modifications of the condensing algorithm and the Lagrangian gradient calculation. For the discretized problem (P2), such linearly coupled constraints can be written in the form

$$r_0(s_0, q_0) + r_1(s_1, q_1) + \ldots + r_{m-1}(s_{m-1}, q_{m-1}) + r_m(s_m) = 0 \qquad (7.1)$$

(note that there may be a dependency on the control parameters q_j). The derivatives (Jacobian matrices) of the functions r_j are given by

$$\begin{aligned} R_j^s &:= \nabla_{s_j} r_j(s_j, q_j)^T, \quad j = 0, 1, \ldots, m-1 \\ R_j^q &:= \nabla_{q_j} r_j(s_j, q_j)^T, \quad j = 0, 1, \ldots, m-1 \\ R_m^s &:= \nabla_{s_m} r_m(s_m)^T. \end{aligned} \qquad (7.2)$$

We obtain the new matrix $C = \nabla h(y)^T$,

$$C = \begin{pmatrix} X_0^s & X_0^q & -I & & & & & & \\ & & X_1^s & X_1^q & -I & & & & \\ & & & & & \ddots & & & \\ & & & & & & X_{m-1}^s & X_{m-1}^q & -I \\ R_0^s & R_0^q & R_1^s & R_1^q & & \ldots & R_{m-1}^s & R_{m-1}^q & R_m^s \end{pmatrix},$$

and the matrix $(C\,P^T)$ becomes

$$\left(\begin{array}{ccccc|cccccc} -I & & & & & X_0^s & X_0^q & & & & \\ X_1^s & -I & & & & & & X_1^q & & & \\ & X_2^s & -I & & & & & & X_2^q & & \\ & & \ddots & \ddots & & & & & & \ddots & \\ & & & X_{m-1}^s & -I & & & & & & X_{m-1}^q \\ R_1^s & R_2^s & \ldots & R_{m-1}^s & R_m^s & R_0^s & R_0^q & R_1^q & R_2^q & \ldots & R_{m-1}^q \end{array}\right).$$

The matrices M_j are modified accordingly,

$$M_j := \begin{pmatrix} I & & & & \\ & \ddots & & & \\ & & I & & \\ & & X_j^s & I & \\ & & & & \ddots \\ & & R_j^s & & & I \end{pmatrix}, \quad j = 1, 2, \ldots, m-1$$

$$M_m := \begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & R_m^s & I \end{pmatrix},$$

and the new matrix M is

$$\begin{aligned} M &:= M_m \, M_{m-1} \cdots M_1 \\ &= \begin{pmatrix} I & & & & & \\ X_1^s & & I & & & \\ X_2^s \, X_1^s & & X_2^s & \ddots & & \\ \vdots & & \vdots & & & \\ & & & & I & \\ \prod_{j=1}^{m-1} X_j^s & \prod_{j=2}^{m-1} X_j^s & & X_{m-1}^s & I & \\ S_1 & & S_2 & \ldots & S_{m-1} & S_m & I \end{pmatrix}, \end{aligned}$$

where the blocks S_j are given by

$$\begin{aligned} S_1 &:= R_1^s + R_2^s \, X_1^s + R_3^s \, X_2^s \, X_1^s + \ldots + R_m^s \, \textstyle\prod_{j=1}^{m-1} X_j^s \\ S_2 &:= R_2^s + R_3^s \, X_2^s + R_4^s \, X_3^s \, X_2^s + \ldots + R_m^s \, \textstyle\prod_{j=2}^{m-1} X_j^s \\ &\vdots \\ S_{m-1} &:= R_{m-1}^s + R_m^s \, X_{m-1}^s \\ S_m &:= R_m^s. \end{aligned} \tag{7.3}$$

Note that only the last block row of M has changed. The matrix C'_{12} defined in (6.14) remains unchanged, while C'_{22} becomes

$$\begin{aligned} C'_{22} = \Big(& R_0^s + S_1 \, X_0^s, \; R_0^q + S_1 \, X_0^q, \; R_1^q + S_2 \, X_1^q, \; R_2^q + S_3 \, X_2^q, \\ & \ldots, \; R_{m-2}^q + S_{m-1} \, X_{m-2}^q, \; R_{m-1}^q + S_m \, X_{m-1}^q \Big). \end{aligned} \tag{7.4}$$

We observe that the matrix products $S_1 X_0^s$ and $S_j X_{j-1}^q$, $j = 1, 2, \ldots, m$, can be conveniently calculated from the entries in the corresponding block columns of C'_{12}. Hence the matrices S_j need not be explicitly calculated. (Recall that no explicit use is made of the matrix M throughout the condensing procedure.) The new recurrence for the right-hand side vector c' is

$$\begin{array}{rcl} c'_{1,0} & = & c_0 \\ c'_{1,j} & = & X_j^s \, c'_{1,j-1} + c_j, \quad j = 1, 2, \ldots, m-1 \\ c'_2 & = & S_1 \, c_0 + S_2 \, c_1 + \ldots + S_m \, c_{m-1} + c_m \\ & = & \sum_{j=1}^m R_j^s \, c'_{1,j-1} + c_m. \end{array} \tag{7.5}$$

(We note that only the formula for c'_2 has changed.) The recurrence (6.30) for the calculation of $v^* = (\Delta s_1^*, \Delta s_2^*, \ldots, \Delta s_m^*)$ remains valid. However, for the calculation of $\tilde{\lambda}^* = (\tilde{\lambda}_0^*, \tilde{\lambda}_1^*, \ldots, \tilde{\lambda}_m^*)$, we obtain the new recurrence

$$\begin{array}{rcl} \tilde{\lambda}_m^* & = & \tilde{\omega}^* \\ \tilde{\lambda}_{m-1}^* & = & \tilde{\nu}_{m-1}^* + R_m^{sT} \, \tilde{\lambda}_m^* \\ \tilde{\lambda}_{m-k}^* & = & \tilde{\nu}_{m-k}^* + X_{m-k+1}^{sT} \tilde{\lambda}_{m-k+1}^* + R_{m-k+1}^{sT} \tilde{\lambda}_m^*, \quad k = 2, 3, \ldots, m, \end{array} \tag{7.6}$$

where an additional term $R_{m-k+1}^{sT} \tilde{\lambda}_m^*$ appears in the last line. For instance, in the case $k = 2$ we have

$$\begin{array}{rcl} \tilde{\lambda}_{m-2}^* & = & \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} \, \tilde{\nu}_{m-1}^* + S_{m-1}^T \, \tilde{\omega}^* \\ & = & \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} \, \tilde{\nu}_{m-1}^* + \left(R_{m-1}^{sT} + X_{m-1}^{sT} \, R_m^{sT} \right) \tilde{\lambda}_m^* \\ & = & \tilde{\nu}_{m-2}^* + X_{m-1}^{sT} \, \tilde{\lambda}_{m-1}^* + R_{m-1}^{sT} \, \tilde{\lambda}_m^*. \end{array}$$

Altogether, the following steps of the condensing algorithm from Section 6 have to be modified in order to account for linearly coupled equality or inequality constraints of the form (7.1),

- CON2: calculation of C'_{22} according to (7.4),
- CON3: new formula for c'_2 according to (7.5), and
- CON7: new recurrence for $\tilde{\lambda}_{m-k}^*$, $k = 2, 3, \ldots, m$ as shown in (7.6).

Observe that the condensing algorithm remains valid (except for minor changes in notation) if the equalities in (7.1) are replaced by inequalities.

In addition, the calculation of the gradient of the Lagrangian function must be modified: instead of the formulas discussed in Section 5, we now

have

$$\begin{aligned}
\nabla_{y_0} L_0 &= \begin{pmatrix} z_0^s \\ z_0^q \end{pmatrix} - \begin{pmatrix} X_0^{s\,T} \\ X_0^{q\,T} \end{pmatrix} \lambda_0 - \begin{pmatrix} R_0^{s\,T} \\ R_0^{q\,T} \end{pmatrix} \lambda_m - \begin{pmatrix} 0 \\ G_0^T \end{pmatrix} \mu_0 \\
\nabla_{y_j} L_j &= \begin{pmatrix} z_j^s \\ z_j^q \end{pmatrix} + \begin{pmatrix} \lambda_{j-1} \\ 0 \end{pmatrix} - \begin{pmatrix} X_j^{s\,T} \\ X_j^{q\,T} \end{pmatrix} \lambda_j - \begin{pmatrix} R_j^{s\,T} \\ R_j^{q\,T} \end{pmatrix} \lambda_m - \begin{pmatrix} 0 \\ G_j^T \end{pmatrix} \mu_j, \\
& \quad j = 1, 2, \ldots, m-1 \\
\nabla_{y_m} L_m &= \lambda_{m-1} - R_m^{s\,T} \lambda_m,
\end{aligned}$$

as can be easily verified from the definition of the Lagrangian function corresponding to the modified problem. If the equalities in (7.1) are replaced by inequalities, we simply change λ_m into μ_m in these formulas.

Uncoupled Interior Point Constraints. Let us next briefly turn to *completely uncoupled* interior point constraints of the form

$$\bar{r}_j(s_j, q_j) = 0, \quad j = 0, 1, \ldots, m-1; \quad \bar{r}_m(s_m) = 0, \tag{7.7}$$

which represent a special case of the linearly coupled constraints (7.1). We note that the vector functions $\bar{r}_j$ may have different dimensions l_r^j ($\bar{r}_j : \mathbb{R}^n \times \mathbb{R}^{k_j} \to \mathbb{R}^{l_r^j}$, $j = 0, 1, \ldots, m-1$, and $\bar{r}_m : \mathbb{R}^n \to \mathbb{R}^{l_r^m}$). The above modified condensing procedure could in principle be used for these uncoupled constraints as well, but this would be inefficient since the additional structure in (7.7) is not exploited. Hence we derive another modification of the condensing algorithm which is specifically tailored to constraints of the form (7.7). Let the Jacobian matrices of the functions $\bar{r}_j$ are denoted by $\bar{R}_j^s$, $\bar{R}_j^q$. Then the matrix $C = \nabla h(y)^T$ becomes

$$C = \begin{pmatrix}
X_0^s & X_0^q & -I & & & & & & \\
 & & X_1^s & X_1^q & -I & & & & \\
 & & & & & \ddots & & & \\
 & & & & & & X_{m-1}^s & X_{m-1}^q & -I \\
\bar{R}_0^s & \bar{R}_0^q & & & & & & & \\
 & & \bar{R}_1^s & \bar{R}_1^q & & & & & \\
 & & & & & \ddots & & & \\
 & & & & & & \bar{R}_{m-1}^s & \bar{R}_{m-1}^q & \\
 & & & & & & & & \bar{R}_m^s
\end{pmatrix},$$

and for $(C\,P^T)$ we obtain

$$\left(\begin{array}{ccccc|cccccc}
-I & & & & & X_0^s & X_0^q & & & & \\
X_1^s & -I & & & & & & X_1^q & & & \\
 & X_2^s & -I & & & & & & X_2^q & & \\
 & & \ddots & \ddots & & & & & & \ddots & \\
 & & & X_{m-1}^s & -I & & & & & & X_{m-1}^q \\
0 & & & & 0 & \bar{R}_0^s & \bar{R}_0^q & & & & \\
\bar{R}_1^s & & & & & & & \bar{R}_1^q & & & \\
 & \bar{R}_2^s & & & & & & & \bar{R}_2^q & & \\
 & & \ddots & & & & & & & \ddots & \\
 & & & \bar{R}_{m-1}^s & & & & & & & \bar{R}_{m-1}^q \\
 & & & & \bar{R}_m^s & 0 & & & & & 0
\end{array}\right).$$

Using accordingly modified matrices M_j, we get

$$\begin{aligned} M &:= M_m\, M_{m-1} \cdots M_1 \\ &= \begin{pmatrix} M_{11} & 0 \\ M_{21} & I \end{pmatrix}, \end{aligned}$$

where the blocks M_{11} and M_{21} are given by

$$M_{11} = \begin{pmatrix}
I & & & & \\
X_1^s & I & & & \\
X_2^s\, X_1^s & X_2^s & \ddots & & \\
\vdots & \vdots & & & \\
 & & & I & \\
\prod_{j=1}^{m-1} X_j^s & \prod_{j=2}^{m-1} X_j^s & \cdots & X_{m-1}^s & I
\end{pmatrix}$$

$$M_{21} = \begin{pmatrix}
0 & & & & & \\
\bar{R}_1^s & & & & & \\
\bar{R}_2^s\, X_1^s & \bar{R}_2^s & & & & \\
\bar{R}_3^s\, X_2^s\, X_1^s & \bar{R}_3^s\, X_2^s & \ddots & & & \\
\vdots & \vdots & & & & \\
 & & & \bar{R}_{m-1}^s & & \\
\bar{R}_m^s \prod_{j=1}^{m-1} X_j^s & \bar{R}_m^s \prod_{j=2}^{m-1} X_j^s & \cdots & \bar{R}_m^s\, X_{m-1}^s & \bar{R}_m^s
\end{pmatrix}.$$

The matrix C'_{12} defined in (6.14) remains unchanged, while C'_{22} becomes

$$C'_{22} = \begin{pmatrix} \bar{R}_0^s & & \bar{R}_0^q & & \\ \bar{R}_1^s X_0^s & & \bar{R}_1^s X_0^q & & \bar{R}_1^q \\ \bar{R}_2^s X_1^s X_0^s & & \bar{R}_2^s X_1^s X_0^q & & \bar{R}_2^s X_1^q \\ \vdots & & \vdots & & \vdots \\ \bar{R}_m^s \prod_{j=0}^{m-1} X_j^s & \bar{R}_m^s (\prod_{j=1}^{m-1} X_j^s) X_0^q & & \bar{R}_m^s (\prod_{j=2}^{m-1} X_j^s) X_1^q & \end{pmatrix}$$

$$\begin{pmatrix} \bar{R}_2^q & & & \\ \vdots & \ddots & \bar{R}_{m-2}^q & \\ & & \bar{R}_{m-1}^s X_{m-2}^q & \bar{R}_{m-1}^q \\ \bar{R}_m^s (\prod_{j=3}^{m-1} X_j^s) X_2^q & \dots & \bar{R}_m^s X_{m-1}^s X_{m-2}^q & \bar{R}_m^s X_{m-1}^q \end{pmatrix}. \tag{7.8}$$

We observe that the matrix C'_{22} can be conveniently calculated from C'_{12} and $\bar{R}_j^s$, $\bar{R}_j^q$ without explicit use of M. Assuming that c_m and c'_2 are partitioned into

$$c_m = \begin{pmatrix} c_{m,0} \\ c_{m,1} \\ \vdots \\ c_{m,m} \end{pmatrix}, \quad c'_2 = \begin{pmatrix} c'_{2,0} \\ c'_{2,1} \\ \vdots \\ c'_{2,m} \end{pmatrix},$$

according to the dimensions l_r^j of the vector functions $\bar{r}_j$ in (7.7), the new recurrence for the right-hand side vector c' is

$$\begin{aligned} c'_{1,0} &= c_0 \\ c'_{1,j} &= X_j^s\, c'_{1,j-1} + c_j, \quad j = 1, 2, \dots, m-1 \\ c'_{2,0} &= c_{m,0} \\ c'_{2,j} &= \bar{R}_j^s\, c'_{1,j-1} + c_{m,j}, \quad j = 1, 2, \dots, m-1, \end{aligned} \tag{7.9}$$

i.e., the $c'_{1,j}$ are unchanged, and there is a new set of formulas for the $c'_{2,j}$. The recurrence (6.30) for the calculation of $v^* = (\Delta s_1^*, \Delta s_2^*, \dots, \Delta s_m^*)$ remains valid. For the calculation of $\tilde{\lambda}^* = (\tilde{\lambda}_0^*, \tilde{\lambda}_1^*, \dots, \tilde{\lambda}_m^*)$, we assume that $\tilde{\lambda}_m^*$ and $\tilde{\omega}^*$ are partitioned into

$$\tilde{\lambda}_m^* = \begin{pmatrix} \tilde{\lambda}_{m,0}^* \\ \tilde{\lambda}_{m,1}^* \\ \vdots \\ \tilde{\lambda}_{m,m}^* \end{pmatrix}, \quad \tilde{\omega}^* = \begin{pmatrix} \tilde{\omega}_0^* \\ \tilde{\omega}_1^* \\ \vdots \\ \tilde{\omega}_m^* \end{pmatrix},$$

again according to the dimensions of the $\bar{r}_j$. Hence, we obtain the modified recurrence

$$\begin{aligned}
\tilde{\lambda}^*_{m,j} &= \tilde{\omega}^*_j, \quad j = 0, 1, \ldots, m \\
\tilde{\lambda}^*_{m-1} &= \tilde{\nu}^*_{m-1} + \bar{R}^{s\,T}_m \tilde{\lambda}^*_{m,m} \\
\tilde{\lambda}^*_{m-k} &= \tilde{\nu}^*_{m-k} + X^{s\,T}_{m-k+1} \tilde{\lambda}^*_{m-k+1} + \bar{R}^{s\,T}_{m-k+1} \tilde{\lambda}^*_{m,m-k+1}, \\
&\quad k = 2, 3, \ldots, m.
\end{aligned} \tag{7.10}$$

Thus, we have to modify the following steps of the condensing algorithm in order to account for completely uncoupled interior point constraints of the form (7.7),

- CON2: calculation of C'_{22} according to (7.8),
- CON3: new set of formulas for $c'_{2,j}$ as given in (7.9), and
- CON7: modified recurrence (7.10) for $\tilde{\lambda}^*$.

Again the condensing algorithm is not affected if the equalities in (7.7) are replaced by inequalities.

The new formulas for the calculation of the gradient of the Lagrangian function are

$$\begin{aligned}
\nabla_{y_0} L_0 &= \begin{pmatrix} z^s_0 \\ z^q_0 \end{pmatrix} - \begin{pmatrix} X^{s\,T}_0 \\ X^{q\,T}_0 \end{pmatrix} \lambda_0 - \begin{pmatrix} \bar{R}^{s\,T}_0 \\ \bar{R}^{q\,T}_0 \end{pmatrix} \lambda_{m,0} - \begin{pmatrix} 0 \\ G^T_0 \end{pmatrix} \mu_0 \\
\nabla_{y_j} L_j &= \begin{pmatrix} z^s_j \\ z^q_j \end{pmatrix} + \begin{pmatrix} \lambda_{j-1} \\ 0 \end{pmatrix} - \begin{pmatrix} X^{s\,T}_j \\ X^{q\,T}_j \end{pmatrix} \lambda_j - \begin{pmatrix} \bar{R}^{s\,T}_j \\ \bar{R}^{q\,T}_j \end{pmatrix} \lambda_{m,j} - \begin{pmatrix} 0 \\ G^T_j \end{pmatrix} \mu_j, \\
&\quad j = 1, 2, \ldots, m-1 \\
\nabla_{y_m} L_m &= \lambda_{m-1} - \bar{R}^{s\,T}_m \lambda_{m,m},
\end{aligned}$$

as can be easily verified. Replacing the equalities in (7.7) by inequalities simply means that $\lambda_{m,j}$ has to be changed into $\mu_{m,j}$, $j = 0, 1, \ldots, m$. Of course, the uncoupled constraints (7.7) and the linearly coupled constraints (7.1) may both occur in the same problem.

Control Continuity Conditions. Finally, we address an efficient way to handle the simple conditions (1.10) on the control parameters,

$$q^2_j - q^1_{j+1} = 0, \quad j = 0, 1, \ldots, m-2,$$

which have to be imposed for continuity of the piecewise linear control approximation (1.8). Here the augmented parameter vector y has the form

$$y := (s_0, q_0^1, q_0^2, s_1, q_1^1, q_1^2, \ldots, s_{m-1}, q_{m-1}^1, q_{m-1}^2, s_m),$$

and assuming that interior point constraints of both types (7.1) and (7.7) are present, the matrix $C = \nabla h(y)^T$ becomes

$$\begin{pmatrix}
X_0^s & X_0^{q^1} & X_0^{q^2} & -I & & & & & & \\
 & & I & & -I & & & & & \\
 & & & X_1^s & X_1^{q^1} & X_1^{q^2} & -I & & & \\
 & & & & & I & & -I & & \\
 & & & & \ddots & & & & & \\
 & & & & & & X_{m-1}^s & X_{m-1}^{q^1} & X_{m-1}^{q^2} & -I \\
\bar{R}_0^s & \bar{R}_0^{q^1} & \bar{R}_0^{q^2} & & & & & & & \\
 & & & \bar{R}_1^s & \bar{R}_1^{q^1} & \bar{R}_1^{q^2} & & & & \\
 & & & & \ddots & & & & & \\
 & & & & & & \bar{R}_{m-1}^s & \bar{R}_{m-1}^{q^1} & \bar{R}_{m-1}^{q^2} & \\
 & & & & & & & & & \bar{R}_m^s \\
R_0^s & R_0^{q^1} & R_0^{q^2} & & \ldots & & R_{m-1}^s & R_{m-1}^{q^1} & R_{m-1}^{q^2} & R_m^s
\end{pmatrix}$$

(note the block rows containing only I and $-I$, corresponding to the conditions (1.10)). Reordering the columns of this matrix, we obtain

$$\left(\begin{array}{ccccc|ccccccc}
-I & & & & & X_0^s & X_0^{q^1} & X_0^{q^2} & & & \\
 & -I & & & & & & I & & & \\
X_1^s & X_1^{q^1} & -I & & & & & & X_1^{q^2} & & \\
 & \ddots & & & & & & & & \ddots & \\
 & & X_{m-1}^s & X_{m-1}^{q^1} & -I & & & & & & X_{m-1}^{q^2} \\
0 & & & & 0 & \bar{R}_0^s & \bar{R}_0^{q^1} & \bar{R}_0^{q^2} & & & \\
\bar{R}_1^s & \bar{R}_1^{q^1} & & & & & & & \bar{R}_1^{q^2} & & \\
 & \ddots & & & & & & & & \ddots & \\
 & & \bar{R}_{m-1}^s & \bar{R}_{m-1}^{q^1} & & & & & & & \bar{R}_{m-1}^{q^2} \\
 & & & & \bar{R}_m^s & 0 & & & & & 0 \\
R_1^s & R_1^{q^1} & \ldots & & R_m^s & R_0^s & R_0^{q^1} & R_0^{q^2} & R_1^{q^2} & \ldots & R_{m-1}^{q^2}
\end{array}\right),$$

which lends itself to a "pre-condensing" step to eliminate the variables Δq_j^1, $j = 1, 2, \ldots, m-1$, from the corresponding reordered solution vector p'. Let

the conditions (1.10) be satisfied at the initial point, and let feasibility with respect to these linear constraints be maintained on every iteration. Then

$$\Delta q_{j+1}^1 = \Delta q_j^2, \quad j = 0, 1, \ldots, m-2, \tag{7.11}$$

and the preceding matrix can be reduced to

$$\left(\begin{array}{cccc|ccccccc}
-I & & & & X_0^s & X_0^{q^1} & X_0^{q^2} & & & & \\
X_1^s & -I & & & & & X_1^{q^1} & X_1^{q^2} & & & \\
 & \ddots & & & & & & \ddots & & & \\
 & & X_{m-1}^s & -I & & & & & X_{m-1}^{q^1} & X_{m-1}^{q^2} & \\
0 & & & 0 & \bar{R}_0^s & \bar{R}_0^{q^1} & \bar{R}_0^{q^2} & & & & \\
\bar{R}_1^s & & & & & & \bar{R}_1^{q^1} & \bar{R}_1^{q^2} & & & \\
 & \ddots & & & & & & \ddots & & & \\
 & & \bar{R}_{m-1}^s & & & & & & \bar{R}_{m-1}^{q^1} & \bar{R}_{m-1}^{q^2} & \\
 & & & \bar{R}_m^s & 0 & & & & & 0 & \\
R_1^s & \ldots & R_{m-1}^s & R_m^s & R_0^s & R_0^{q^1} & R_0^{q^2}+R_1^{q^1} & R_1^{q^2}+R_2^{q^1} & \ldots & R_{m-1}^{q^2} &
\end{array}\right),$$

thus eliminating the components Δq_j^1, $j = 1, 2, \ldots, m-1$, from p'. Observe that the blocks

$$X_j^{q^1}, \quad \bar{R}_j^{q^1}, \quad \text{and} \quad R_j^{q^1} \quad \text{for } j = 1, 2, \ldots, m-1$$

have moved to the right, and that the original block structure is recovered on the left side of the indicated partition. Therefore, the matrix M of the block Gaussian elimination procedure is *not* affected, and the recurrence relations derived for the calculation of the right-hand side vector c' and the multiplier vector $\tilde{\lambda}^*$ remain valid since they both only depend on M.[‡]

From the upper part of the pre-condensed matrix, we can directly read off the new recurrence for the calculation of $v^* = (\Delta s_1^*, \Delta s_2^*, \ldots, \Delta s_m^*)$,

$$\begin{aligned}
\Delta s_1^* &= X_0^s \, \Delta s_0^* + X_0^{q^1} \Delta q_0^{1\,*} + X_0^{q^2} \Delta q_0^{2\,*} - c_0 \\
\Delta s_{j+1}^* &= X_j^s \, \Delta s_j^* + X_j^{q^1} \Delta q_{j-1}^{2\,*} + X_j^{q^2} \Delta q_j^{2\,*} - c_j, \quad j = 1, 2, \ldots, m-1,
\end{aligned} \tag{7.12}$$

where $c_j = -(x(t_{j+1}; s_j, q_j) - s_{j+1})$. The calculation of the matrices C_{12}' and C_{22}' becomes slightly more complicated because of the additional blocks

[‡]We do *not* calculate the additional Lagrange multipliers which correspond to the continuity conditions (1.10).

which now appear on the right side of the partition in the above matrix. For C'_{12}, we obtain the modified formula

$$C'_{12} = \begin{pmatrix} X_0^s & X_0^{q^1} & X_0^{q^2} \\ X_1^s X_0^s & X_1^s X_0^{q^1} & X_1^s X_0^{q^2} + X_1^{q^1} \\ X_2^s X_1^s X_0^s & X_2^s X_1^s X_0^{q^1} & X_2^s X_1^s X_0^{q^2} + X_2^s X_1^{q^1} \\ \vdots & \vdots & \vdots \\ \prod_{j=0}^{m-1} X_j^s & (\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^1} & (\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^2} + (\prod_{j=2}^{m-1} X_j^s)\, X_1^{q^1} \end{pmatrix}$$

$$\begin{pmatrix} X_1^{q^2} & & \\ X_2^s X_1^{q^2} + X_2^{q^1} & & \\ \vdots & \ddots & \\ (\prod_{j=2}^{m-1} X_j^s)\, X_1^{q^2} + (\prod_{j=3}^{m-1} X_j^s)\, X_2^{q^1} & \cdots & X_{m-1}^s X_{m-2}^{q^2} + X_{m-1}^{q^1} \end{pmatrix} \begin{pmatrix} & \\ & \\ & \\ X_{m-2}^{q^2} & \\ X_{m-1}^{q^2} & \end{pmatrix} \tag{7.13}$$

(compare equation (6.14)). The matrix C'_{22}, of course, depends on the type of the corresponding equality or inequality constraints: for uncoupled interior point constraints (7.7), we have

$$C'_{22} = \begin{pmatrix} \bar{R}_0^s & \bar{R}_0^{q^1} & \bar{R}_0^{q^2} \\ \bar{R}_1^s X_0^s & \bar{R}_1^s X_0^{q^1} & \bar{R}_1^s X_0^{q^2} + \bar{R}_1^{q^1} \\ \bar{R}_2^s X_1^s X_0^s & \bar{R}_2^s X_1^s X_0^{q^1} & \bar{R}_2^s X_1^s X_0^{q^2} + \bar{R}_2^s X_1^{q^1} \\ \vdots & \vdots & \vdots \\ \bar{R}_m^s \prod_{j=0}^{m-1} X_j^s & \bar{R}_m^s (\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^1} & \bar{R}_m^s (\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^2} + \\ & & + \bar{R}_m^s (\prod_{j=2}^{m-1} X_j^s)\, X_1^{q^1} \end{pmatrix}$$

$$\begin{pmatrix} \bar{R}_1^{q^2} & & & \\ \bar{R}_2^s X_1^{q^2} + \bar{R}_2^{q^1} & & & \\ \vdots & \ddots & \bar{R}_{m-2}^{q^2} & \\ & & \bar{R}_{m-1}^s X_{m-2}^{q^2} + \bar{R}_{m-1}^{q^1} & \bar{R}_{m-1}^{q^2} \\ \bar{R}_m^s (\prod_{j=2}^{m-1} X_j^s)\, X_1^{q^2} + & \cdots & \bar{R}_m^s X_{m-1}^s X_{m-2}^{q^2} + & \bar{R}_m^s X_{m-1}^{q^2} \\ + \bar{R}_m^s (\prod_{j=3}^{m-1} X_j^s)\, X_2^{q^1} & & + \bar{R}_m^s X_{m-1}^{q^1} & \end{pmatrix} \tag{7.14}$$

(compare the original form in (7.8) above). For linearly coupled interior point constraints (7.1), we obtain the following modified version of (7.4),

$$C'_{22} = \Big(R_0^s + S_1\, X_0^s,\;\; R_0^{q^1} + S_1\, X_0^{q^1},\;\; R_0^{q^2} + R_1^{q^1} + S_1\, X_0^{q^2} + S_2\, X_1^{q^1},\;\; \ldots,$$
$$R_{m-2}^{q^2} + R_{m-1}^{q^1} + S_{m-1}\, X_{m-2}^{q^2} + S_m\, X_{m-1}^{q^1},\;\; R_{m-1}^{q^2} + S_m\, X_{m-1}^{q^2}\Big), \tag{7.15}$$

where the blocks S_j are defined in (7.3). Again the matrix products

$$S_1\, X_0^s,\;\; S_j\, X_{j-1}^{q^1},\;\; \text{and}\;\; S_j\, X_{j-1}^{q^2}\;\; \text{for}\;\; j = 1, 2, \ldots, m$$

can be calculated conveniently along with the blocks of C'_{12}. In the normal case of two-point boundary conditions (1.4c), we obtain

$$C'_{22} = \Big(R_a + R_b\,(\textstyle\prod_{j=0}^{m-1} X_j^s),\;\; R_b\,(\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^1},\;\; R_b\,(\prod_{j=1}^{m-1} X_j^s)\, X_0^{q^2}$$
$$+ R_b\,(\textstyle\prod_{j=2}^{m-1} X_j^s)\, X_1^{q^1},\;\; \ldots,\;\; R_b\, X_{m-1}^s\, X_{m-2}^{q^2} + R_b\, X_{m-1}^{q^1},\;\; R_b X_{m-1}^{q^2}\Big) \tag{7.16}$$

(compare equation (6.15)).

Before B'' and b'' can be calculated from (6.21a) and (6.21b), "pre-condensing" must also be applied to B' and b'. The block diagonal structure of the original matrix B (as shown in Figure 6.1) remains valid if q is interpreted as (q^1, q^2). Again, the blocks of B are reordered according to the permutation of variables. This leads to the matrix B' which is shown in Figure 7.3 for the case $m = 3$. However, to account for the elimination of Δq_j^1, $j = 1, 2, \ldots, m-1$, from the solution vector p' using conditions (7.11), B' must be replaced by $\hat{B}'$ (see Figure 7.3). It is easily verified that, subject to conditions (7.11),

$$\hat{p}'^T \hat{B}'\, \hat{p}' \;\equiv\; p'^T B'\, p',$$

where $\hat{p}'$ denotes the solution vector with Δq_j^1, $j = 1, 2, \ldots, m-1$, being eliminated. Similarly, b' must be replaced by a new vector $\hat{b}'$ such that $\hat{b}'^T \hat{p}' \equiv b'^T p'$. Using $\hat{B}'$, $\hat{b}'$ instead of B', b' along with the modified formula (7.13) for the calculation of C'_{12}, it is straightforward to derive the corresponding new versions of (6.22)–(6.24); hence these relationships are not explicitly given here.

As can be seen from the above discussion, the complexity of the condensing algorithm is only slightly increased if "pre-condensing" is used for the continuity conditions (1.10). Therefore, the option of including these conditions together with the associated additional parameters should definitely be provided to enhance the flexibility in choosing the control parameterizations.

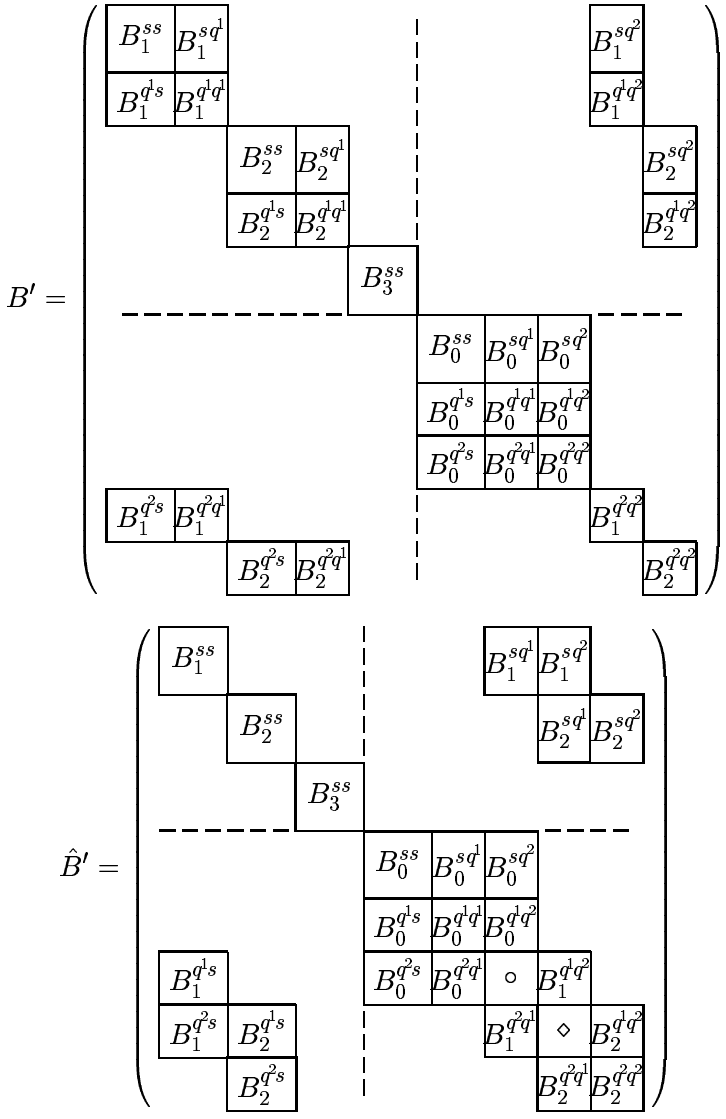

Figure 7.3: "Pre-condensing" of the Hessian approximation ($m = 3$). In $\hat{B}'$, the symbols $\circ$ and $\diamond$ stand for $B_0^{q^2q^2} + B_1^{q^1q^1}$ and $B_1^{q^2q^2} + B_2^{q^1q^1}$, respectively.

7.2.2 DAE Models

Now we briefly sketch various possible strategies for extending the original ODE approach of MUSCOD to DAE models. We restrict ourselves to *quasilinear-implicit nonlinear DAEs of index 1*,

$$\begin{aligned} A^d(t, x^d, x^a, u)\ \dot{x}^d &= f^d(t, x^d, x^a, u) \\ 0 &= f^a(t, x^d, x^a, u), \end{aligned} \tag{7.17}$$

where A^d as well as $\partial f^a / \partial x^a$ are *nonsingular*. As in the case of ODE models, a multiple shooting discretization is used for the states $x = (x^d, x^a)$. On each multiple shooting interval, we integrate the following *relaxed* version of the original DAE model (7.17),

$$\begin{aligned} A^d(t, x^d, x^a, \varphi_j(t, q_j))\ \dot{x}^d &= f^d(t, x^d, x^a, \varphi_j(t, q_j)) \\ 0 &= f^a(t, x^d, x^a, \varphi_j(t, q_j)) \\ &\quad - f^a(t_j, s_j^d, s_j^a, \varphi_j(t_j, q_j)) \end{aligned} \tag{7.18}$$

with $x(t_j) = (s_j^d, s_j^a)$ and $t \in I_j = [t_j, t_{j+1}]$, see [Bock88, Bock95]. For instance, a BDF method can be used to perform these integrations. The recently developed BDF solver DAESOL [Eich87, Bauer94] automatically handles the above DAE relaxation and provides internal numerical differentiation for the efficient generation of sensitivity information.[‡]

Let $x(t; s_j^d, s_j^a, q_j)$ denote the solution of (7.18). We then impose the conditions

$$x^d(t_{j+1}; s_j^d, s_j^a, q_j) - s_{j+1}^d = 0, \quad j = 0, 1, \ldots, m-1 \tag{7.19}$$

$$f^a(t_j, s_j^d, s_j^a, \varphi_j(t_j, q_j)) = 0, \quad j = 0, 1, \ldots, m, \tag{7.20}$$

which require continuity of the *differential* variables x^d at the mesh points t_j for $j = 1, 2, \ldots, m$, as well as consistency of $x(t_j) = (s_j^d, s_j^a)$ with the algebraic equations f^a of the original DAE model (7.17) for $j = 0, 1, \ldots, m$ [Bock88]. Both sets of conditions (7.19) and (7.20) are included as equality constraints in the discretized optimization problem (P2). Use of the relaxed DAE formulation (7.18), together with the additional conditions (7.19) and (7.20), allows the choice of arbitrary initial values for *all* states, while in general, continuity and consistency will hold only for the final solution.

[‡]DAESOL either solves the sensitivity equations along with the state equations or alternatively uses the perturbation approach for the calculation of derivatives. Another DAE solver capable of calculating sensitivities is the BDF code DDASAC [Cara85].

Note that the algebraic variables x^a may have discontinuities at the mesh points t_j, $j = 1, 2, \ldots, m$. Therefore, an additional vector of control parameters q_m must be specified for a complete determination of $x(t_m)$, and the augmented parameter vector y becomes

$$y := (s_0^d, s_0^a, q_0, s_1^d, s_1^a, q_1, \ldots, s_m^d, s_m^a, q_m). \tag{7.21}$$

There are a number of possibilities to handle the conditions (7.19) and (7.20) in the context of SQP-based optimization:

1. The continuity conditions (7.19) can be used to eliminate the variables Δs_j^d, $j = 1, 2, \ldots, m$, from the solution vector p of the QP subproblem $Q(y, B)$ by a *condensing algorithm* as in the ODE case, treating the consistency conditions (7.20) as uncoupled interior point constraints ($\bar{r}_j \equiv f_j^a$). Thereby, the algebraic variables s^a are interpreted as additional control parameters which are completely determined by the constraints $\bar{r}_j = 0$ (note that in this case the variables Δs_j^a are *not* eliminated from the QP subproblem).

2. A more general *direct elimination procedure* (which requires block factorizations) can be applied to the linearizations of (7.19) and (7.20) in order to eliminate both Δs_j^d, $j = 1, 2, \ldots, m$, and Δs_j^a, $j = 0, 1, \ldots, m$, from problem $Q(y, B)$, following the ideas of Berna et al. [Berna80].

3. The QP subproblem can be solved "as is" in full space using the large-scale recursive KKT solver of Steinbach [Stei95a]. This algorithm is specifically tailored to the inherent m-stage block sparse structure of the original QP problem.

4. The natural decomposition of the discretized states s_j into differential and algebraic variables s_j^d and s_j^a, respectively (which is induced by (7.17)) can be exploited using *reduced SQP methods* (see, e.g., [Gabay82, Locke83, Schm93, Schu95]). That is, the problem is projected onto the reduced space of differential variables plus control parameters. Subsequently, a condensing procedure can be applied to eliminate the variables Δs_j^d, $j = 1, 2, \ldots, m$, from the reduced-space QP problem.

The first strategy is easy to implement but has the disadvantage that the dimension of the QP subproblem cannot be reduced as effectively as in the ODE case (note that in many applications the number of algebraic variables by far exceeds the number of differential variables). Strategy 2 allows

a significant further reduction in the problem dimension by eliminating all algebraic variables, but is relatively difficult to implement because the Hessian is approximated in full space, and the transformation which is required on each iteration is considerably more complicated than in strategy 1. The third strategy has the easiest implementation (the QP subproblem need not be transformed at all) and is definitely to be preferred over the first and second one. However, note that the "full-space" strategies 1–3 require setting up the *complete* system of linearized constraints. Calculating all gradients of the continuity conditions (7.19) can become prohibitively expensive for large problems. This can be avoided using the reduced-space approach of strategy 4. Therefore, the last strategy seems to have the greatest potential for future large-scale applications, especially if there is a large fraction of algebraic variables and the Jacobians of the consistency conditions (7.20) are sparse.

7.2.3 Generalization of the Multiple Shooting Structure

We have seen that the multiple shooting discretization of the states (as discussed in Section 1) imposes a specific, decoupled structure on the discretized optimal control problem and that this structure can be exploited in the solution process. Here we show that it is possible to formulate considerably more general discretized optimal control problems without sacrificing any of the advantages of the original problem formulation. This generalization allows, for instance, the introduction of discontinuities of the (differential) states at the discretization mesh points and the possibility to employ different dynamic models (perhaps of different dimension).

The classical multiple shooting structure comes from the continuity conditions

$$x(t_{j+1}; s_j, q_j) - s_{j+1} = 0, \tag{7.22}$$

where $x(t_{j+1}; s_j, q_j)$ represents the solution of the initial value problem

$$\dot{x} = f_j(t, x, q_j), \quad x(t_j) = s_j, \quad t \in I_j = [t_j, t_{j+1}]$$

at the integration end point t_{j+1}. The integration over I_j defines a nonlinear mapping $x_j : \mathbb{R}^n \times \mathbb{R}^{k_j} \to \mathbb{R}^n$ of the initial values and the parameters to the solution end-value. Now the important point is to realize that this special kind of mapping may be replaced by other functional dependencies without affecting the basic structure of the resulting optimization problem. As a first example, we consider

$$s_j + \bar{c}_j(q_j) - s_{j+1} = 0, \quad t_{j+1} = t_j^+, \tag{7.23}$$

where the vector $\bar{c}_j$ is used to introduce "jumps" of the states at time t_j. Such a facility can be very useful in practice for the modeling of (nearly) instantaneous changes of the states due to certain discrete events, e.g., the very quick addition of a finite amount of reactant to a stirred chemical reactor ("impulsive action").

A further generalization is provided by the condition

$$c_j(s_j, q_j) - s_{j+1} = 0, \quad t_{j+1} = t_j^+, \tag{7.24}$$

where the function $c_j : \mathbb{R}^{n_j} \times \mathbb{R}^{k_j} \to \mathbb{R}^{n_{j+1}}$ may be given either explicitly or in the implicit form

$$H_j(s_j, q_j, c_j) = 0, \quad \partial H_j / \partial c_j \text{ nonsingular.} \tag{7.25}$$

Of course, a suitable method must be available to solve (7.25) for c_j. Approximate derivatives of c_j with respect to s_j and q_j are obtained, e.g., by finite differences, and we define the Jacobian matrices

$$\begin{array}{rcl} C_j^s & := & \nabla_{s_j} c_j(s_j, q_j)^T \in \mathbb{R}^{n_{j+1} \times n_j} \\ C_j^q & := & \nabla_{q_j} c_j(s_j, q_j)^T \in \mathbb{R}^{n_{j+1} \times k_j} \end{array} \tag{7.26}$$

which correspond to the matrices X_j^s and X_j^q in the original problem formulation. A generalized multiple shooting structure is then defined by an arbitrary sequence of conditions (7.22) or (7.24).

Observe that a condition of the form (7.24) allows one to couple two dynamic models having different dimensions by mapping the solution end value of the first model at time t_j to a suitable initial value for the second model. This may be useful for modeling certain problems in the field of aerospace engineering (e.g., the rendezvous of two spacecraft). An application in the area of chemical engineering would be the 1D steady-state optimization of reactor networks, where the conditions (7.24) could be used to model lumped elements like mixers and continuously stirred tank reactors (CSTRs) while plug flow reactors and similar equipment would be described by DAE models (using space instead of time as the independent variable).

7.3 Algorithm Performance and Future Developments

Over the years, MUSCOD has been applied to a variety of small to medium scale problems from many different fields of application. Numerical results on a number of test problems are reported in [Plitt81] and [Bock84].

A comparison of MUSCOD with two other optimal control packages can be found in the latter reference. On the so-called *Van der Pol problem*, MUSCOD performed significantly better—both in terms of number of function/gradient evaluations and CPU time—than the direct method of Pouliot et al. [Poul81], which is based on simple shooting. On the other hand, the performance of MUSCOD was found to be quite similar to that of an *indirect* method [Bock78] which uses multiple shooting for the solution of the resulting boundary value problem, provided both methods were started with "comparable" initial data.[‡] MUSCOD has been also successfully applied to important practical problems in the field of mechanical engineering, especially the optimization of robot movements, see e.g. [Schu94, Stei95b]. Currently the method is being used for a relatively large optimal control problem from the area of chemical recation kinetics involving 54 states, 8 parameters, and 2 control functions (for this purpose, MUSCOD has been interfaced with the CHEMKIN-II chemical kinetics software package [Kee91]).

Although MUSCOD has been shown to perform very well on quite a number of problems, it can suffer from the well-known limitations of the original version of Powell's SQP method, that have been discussed in Section 3. These difficulties are now being addressed by

- the implementation of alternative line search strategies based on the "watchdog technique" [Cham82, Hoza93],
- the development of trust region strategies,
- the implementation of limited memory updates for the Hessian,
- the use of more efficient and stable QP solvers [Gill83, Gold83], and
- the implementation of better strategies for dealing with inconsistent constraint linearizations [Bieg85].

It is expected that these algorithmic improvements will further enhance the overall robustness and performance of the method, especially for larger scale applications.

A project has been recently started here at the IWR to tailor MUSCOD to the class of models which arise in the dynamic simulation of chemical processes. Modern equation-oriented flowsheeting packages like SPEEDUP

[‡]Recall that it is difficult to find appropriate initial values for the adjoint variables. Therefore, a direct method is frequently employed to estimate the initial data required by the indirect approach [Bock84].

[Pant88] or DIVA [Krön90] provide a powerful modeling platform and a comprehensive set of facilities for steady-state and dynamic simulation (and frequently also for steady-state optimization). However, optimization in the dynamic case is currently not supported. Therefore, it is planned to introduce the concepts of MUSCOD into this equation-oriented environment for dynamic process simulation, see Figure 7.4. Future work is required mainly in the following areas:

1. *Treatment of implicit model discontinuities.* Process models almost invariably contain state-dependent discontinuities (e.g., piecewise defined thermodynamic functions, and the representations of inherently discontinuous elements like weirs or safety relief valves). Such discontinuities must be properly handled to ensure that the model derivatives remain meaningful [Bock81a, Bock87].

2. *Efficient treatment of very large DAE models with many algebraic variables in the optimization context.* A reduced-space approach is likely to be most practical in this case. Therefore, a suitable specialized reduced SQP strategy must be developed which should fully exploit the sparsity of the Jacobian matrix of f^a in (7.20).

3. *Reduced order modeling.* In many cases, the dimension of a given process model must be considerably reduced to make the optimization problem tractable. However, note that there may be a tradeoff between reducing the model order and destroying the simple structure of the original full model, so some caution has to be exercised.

A second project deals with a similar adaptation of the parameter estimation package PARFIT [Bock81b, Bock83, Bock87], which is based on multiple shooting and a generalized Gauss-Newton method. As shown in Figure 7.4, both MUSCOD and PARFIT require the same type of information from the process model (values of the state and sensitivity functions) and can therefore use the *same* specifically tailored DAE solver (which must provide for instance a means for the efficient generation of sensitivity information, e.g. [Cara85, Bauer94]). Furthermore, in spite of the differences between SQP and the generalized Gauss-Newton method, many similarities can be exploited in the problem structures and the corresponding basic numerical solution methods. Thus, the proposed concepts could considerably enhance the capabilities of existing dynamic process simulators [Lein95a].

Another closely related potential application of MUSCOD would be in the area of *nonlinear model predictive control*, see for instance [Eaton92].

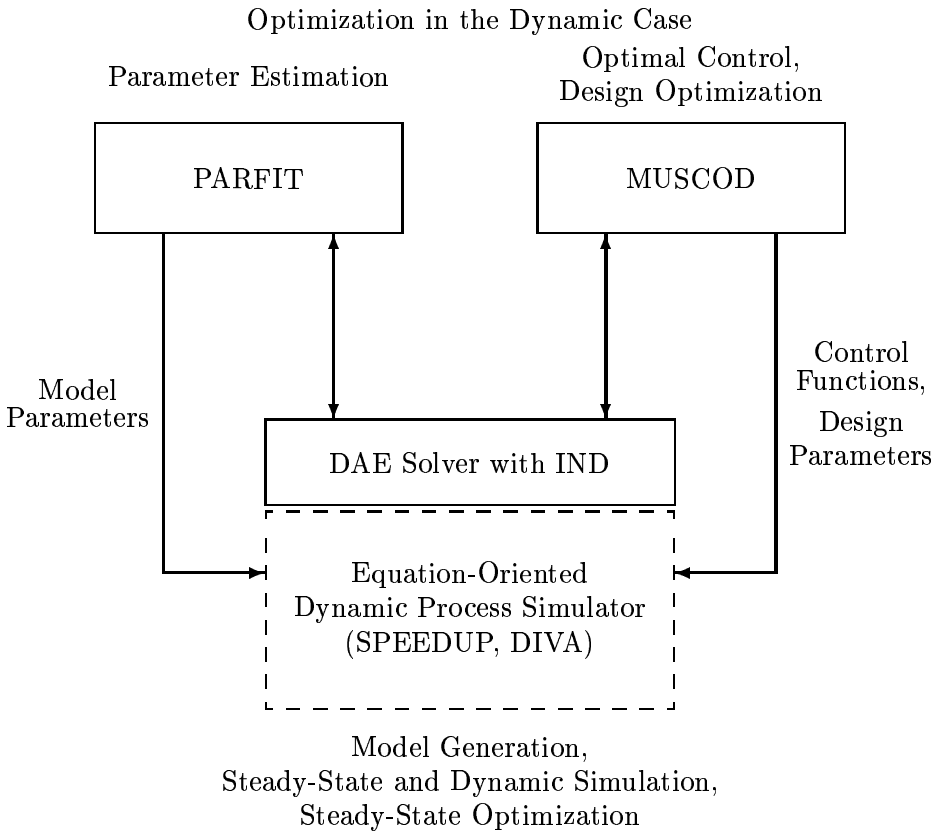

Figure 7.4: Integration of MUSCOD into an equation-oriented environment for dynamic process simulation [Lein95a].

Observe that the algorithm is ideally suited for parallel computation because the integrations on all multiple shooting intervals can be performed simultaneously. These integrations are in general the most expensive part of the method. Hence a significant speedup can be attained through little programming effort (only one loop has to be replaced by an equivalent parallel construct). This should allow the online optimization of moderately sized process models.

References

[Bauer94] Bauer, I.: Numerische Behandlung differentiell-algebraischer Gleichungen mit Anwendungen in der Chemie. Diplomarbeit, Universität Augsburg, 1994.

[Baza79] Bazaara, M. S., and C. M. Shetty: Nonlinear Programming: Theory and Applications. Wiley, New York, 1979.

[Beale55] Beale, E. M. L.: On minimizing a convex function subject to linear inequalities. Journal of the Royal Statistical Society Series B 17, 173–184, 1955.

[Beale59] Beale, E. M. L.: On quadratic programming. Naval Research Logistics Quarterly 6, 227–243, 1959.

[Berna80] Berna, T. J., M. H. Locke, and A. W. Westerberg: A new approach to optimization of chemical processes. AIChE J. 26, 37–43, 1980.

[Bieg84] Biegler, L. T.: Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. Comp. Chem. Eng. 8, 243–248, 1984.

[Bieg85] Biegler, L. T., and J. E. Cuthrell: Improved infeasible path optimization for sequential modular simulators—II: the optimization algorithm. Comp. Chem. Eng. 9, 257–267, 1985.

[Bock78] Bock, H.-G.: Numerical solution of nonlinear multipoint boundary value problems with applications to optimal control. Z. Angew. Math. Mech. 58, 407–409, 1978.

[Bock81a] Bock, H.-G.: Numerische Behandlung von zustandsbeschränkten und Chebyshev-Steuerungsproblemen. Bericht für die Carl-Cranz-Gesellschaft, Oberpfaffenhofen, 1981.

[Bock81b] Bock, H.-G.: Numerical treatment of inverse problems in chemical reaction kinetics. In K. H. Ebert, P. Deuflhard, and W. Jäger (eds.): Modelling of Chemical Reaction Systems (Springer Series in Chemical Physics 18). Springer, Heidelberg, 1981.

[Bock83] Bock, H.-G.: Recent advances in parameter identification techniques for O.D.E. In P. Deuflhard, and E. Hairer (eds.): Numerical Treatment of Inverse Problems in Differential and Integral Equations. Birkhäuser, Boston, 1983.

[Bock84] Bock, H.-G., and K.-J. Plitt: A multiple shooting algorithm for direct solution of optimal control problems. International Federation of Automatic Control, 9th World Congress, Budapest, 1984.

[Bock87] Bock, H.-G.: Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen (Dissertation). Bonner Mathematische Schriften Nr. 183, Bonn, 1987.

[Bock88] Bock, H.-G., E. Eich, and J. P. Schlöder: Numerical solution of constrained least squares boundary value problems in differential-algebraic equations. In K. Strehmel (ed.): Numerical Treatment of Differential Equations. Teubner, Leipzig, 1988.

[Bock95] Bock, H.-G., J. P. Schlöder, and V. H. Schulz: Numerik großer Differentiell-Algebraischer Gleichungen—Simulation und Optimierung. In H. Schuler (ed.): Prozeßsimulation. VCH, Weinheim, 1995.

[Brys76] Bryson, A. E., and Y. C. Ho: Applied Optimal Control. Wiley, New York, 1976.

[Byrd91] Byrd, R. H., and J. Nocedal: An analysis of reduced Hessian methods for constrained optimization. Math. Progr. 49, 285–323, 1991.

[Byrd92] Byrd, R. H., R. A. Tapia, and Y. Zhangh: An SQP augmented Lagrangian BFGS algorithm for constrained optimization. SIAM J. Optimization 2, 210–241, 1992.

[Cara85] Caracotsios, M., and W. E. Stewart: Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations. Comp. Chem. Eng. 9, 359–365, 1985.

[Cham79] Chamberlain, R. M.: Some examples of cycling in variable metric methods for constrained minimization. Math. Progr. 16, 378–383, 1979.

[Cham82] Chamberlain, R. M., C. Lemaréchal, H. C. Pedersen, and M. J. D. Powell: The watchdog technique for forcing convergence in algorithms for constrained optimization. Math. Progr. Study 16, 1–17, 1982.

[Chen84] Chen, H.-S., and M. A. Stadtherr: Enhancements of the Han-Powell method for successive quadratic programming. Comp. Chem. Eng. 8, 229–234, 1984.

[Codd55] Coddington, E. A., and N. Levinson: Theory of Ordinary Differential Equations. McGraw-Hill, New York, 1955.

[Cuth87] Cuthrell, J. E., and L. T. Biegler: On the optimization of differential-algebraic process systems. AIChE J. 33, 1257–1270, 1987.

[Cuth89] Cuthrell, J. E., and L. T. Biegler: Simultaneous optimization and solution methods for batch reactor control profiles. Comp. Chem. Eng. 13, 49–62, 1989.

[Denn74] Dennis, J. E., and J. J. Moré: A characterization of superlinear convergence and its application to Quasi-Newton methods. Math. Comp. 28, 549–560, 1974.

[Denn77] Dennis, J. E., and J. J. Moré: Quasi-Newton methods: motivation and theory. SIAM Review 19, 46–89, 1977.

[Denn89] Dennis, J. E., and R. B. Schnabel: A View of Unconstrained Optimization. In: G. L. Nemhauser et al., eds.: Handbooks in Operations Research and Management Science, Vol. 1 (Optimization), Elsevier, Amsterdam, 1989.

[Eaton92] Eaton, J. W., and J. B. Rawlings: Model-predictive control of chemical processes. Comp. Chem. Eng. 47, 705–720, 1992.

[Eich87] Eich, E.: Numerische Behandlung semi-expliziter differentiell-algebraischer Gleichungssysteme vom Index 1 mit BDF-Verfahren. Diplomarbeit, Universität Bonn, 1987.

[Fehl69] Fehlberg, E.: Klassische Runge-Kutta Formeln fünfter und siebter Ordnung mit Schrittweitenkontrolle. Computing 4, 93–106, 1969.

[Fiac68] Fiacco, A. V., and G. P. McCormick: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. Wiley, New York, 1968.

[Flet71] Fletcher, R.: A general quadratic programming algorithm. J. Inst. Math. Appl. 7, 76–91, 1971.

[Föll88] Föllinger, Otto: Optimierung dynamischer Systeme. Oldenbourg, München, 1988.

[Gabay82] Gabay, D.: Reduced quasi-Newton methods with feasibility improvement for nonlinearly constrained optimization. Math. Progr. Study 16, 18–44, 1982.

[Garc76] Garcia-Palomares, U. M. and O. L. Mangasarian: Superlinearly convergent quasi-Newton algorithms for nonlinearly constrained optimization problems. Math. Progr. 11, 1–13, 1976.

[Gear83] Gear, C. W., and T. Vu: Smooth numerical solutions of ordinary differential equations. In P. Deuflhard, and E. Hairer (eds.): Numerical Treatment of Inverse Problems in Differential and Integral Equations. Birkhäuser, Boston, 1983.

[Gill78] Gill, P. E., and W. Murray: Numerically stable methods for quadratic programming. Math. Progr. 14, 349–372, 1978.

[Gill81a] Gill, P. E., W. Murray, and M. H. Wright: Practical Optimization. Academic Press, London, 1981.

[Gill81b] Gill, P. E., W. Murray, M. A. Saunders, and M. H. Wright: QP-based methods for large scale nonlinearly constrained optimization. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson: Nonlinear Programming 4. Academic Press, New York, 1981.

[Gill83] Gill, P. E., W. Murray, M. A. Saunders, and M. H. Wright: User's guide for SOL/QPSOL: a Fortran package for quadratic programming. Technical Report SOL 83-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1983.

[Gill89] Gill, P. E., W. Murray, and M. H. Wright: Constrained Nonlinear Programming. In: G. L. Nemhauser et al., eds.: Handbooks

in Operations Research and Management Science, Vol. 1 (Optimization), Elsevier, Amsterdam, 1989.

[Gold83] Goldfarb, D., and A. Idnani: A numerically stable dual method for solving strictly convex quadratic programs. Math. Progr. 27, 1–33, 1983.

[Grie82] Griewank, A., and P. L. Toint: Partitioned variable metric updates for large structured optimization problems. Numer. Math. 39, 119–137, 1982.

[Han76] Han, S. P.: Superlinearly convergent variable-metric algorithms for general nonlinear programming problems. Math. Progr. 11, 263–282, 1976.

[Han77a] Han, S. P.: A globally convergent method for nonlinear programming. JOTA 22, 297–310, 1977.

[Han77b] Han, S. P.: Dual variable metric algorithms for constrained optimization. SIAM J. Control and Optimization 15, 546–565, 1977.

[Henr62] Henrici, P.: Discrete Variable Methods in Ordinary Differential Equations. Wiley, New York, 1962.

[Hoza93] Hoza, M., and M. A. Stadtherr: An improved watchdog line search for successive quadratic programming. Comp. Chem. Eng. 17, 943–947, 1993.

[Kee91] Kee, R. J., F. M. Rupley, and J. A. Miller: CHEMKIN-II: A Fortran Chemical Kinetics Package for the Analysis of Gas Phase Chemical Kinetics. Report SAND89-8009B, Sandia National Laboratories, Livermore, 1991.

[Krön90] Kröner, A., P. Holl, W. Marquardt, and E. D. Gilles: DIVA—an open architecture for dynamic simulation. Comp. Chem. Eng. 14, 1289–1295, 1990.

[Lein95a] Leineweber, D. B., and A. C. Ströder: Parameterschätzung und Optimale Steuerung für dynamische Prozesse der Verfahrenstechnik. Talk held at the Workshop "Scientific Computing in der chemischen Verfahrenstechnik", Technische Universität Hamburg-Harburg, March 01–03, 1995.

[Lein95b] Leineweber, D. B.: LIBLAC—structured data types and basic operations for numerical linear algebra in an ANSI C/Fortran 77-environment. To appear 1995.

[Locke83] Locke, M. H., A. W. Westerberg, and R. H. Edahl: Improved successive quadratic programming optimization algorithm for engineering design problems. AIChE J. 29, 871–874, 1983.

[Logan87] Logan, J. D.: Applied Mathematics—A Contemporary Approach. Wiley, New York, 1987.

[Logs89] Logsdon, J. S., and L. T. Biegler: Accurate solution of differential-algebraic optimization problems. I&EC Res. 28, 1628–1639, 1989.

[Logs92] Logsdon, J. S., and L. T. Biegler: Decomposition strategies for large-scale dynamic optimization problems. Chem. Eng. Sci. 47, 851–864, 1992.

[Logs93] Logsdon, J. S., and L. T. Biegler: A relaxed reduced space SQP strategy for dynamic optimization problems. Comp. Chem. Eng. 17, 367–372, 1993.

[Mang69] Mangasarian, O. L.: Nonlinear Programming. McGraw-Hill, New York, 1969.

[Mayne80] Mayne, D. Q.: The use of exact penalty functions to determine steplength in optimization algorithms. In G. A. Watson (ed.): Numerical Analysis, Dundee 1979. Lecture Notes in Mathematics No. 773, Springer, Berlin, 1980.

[Murr82] Murray, W., and M. H. Wright: Computation of the search direction in constrained optimization algorithms. Math. Progr. Study 16, 62–83, 1982.

[Noce85] Nocedal, J., and M. L. Overton: Projected Hessian updating algorithms for nonlinearly constrained optimization. SIAM J. Numer. Anal. 22, 821–850, 1985.

[Orte70] Ortega, J. M. and W. C. Rheinboldt: Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York, 1970.

[Pant88] Pantelides, C. C.: SPEEDUP—recent advances in process simulation. Comp. Chem. Eng. 12, 745–755, 1988.

[Plitt81] Plitt, K.-J.: Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschränkter optimaler Steuerungen. Diplomarbeit, Universität Bonn, 1981.

[Poul81] Pouliot, M. R., B. L. Pierson, and R. G. Brusch: Recursive quadratic programming solutions to minimum-time aircraft trajectory problems. International Federation of Automatic Control, 2nd IFAC Workshop on Control Applications of Nonlinear Programming, 1981.

[Powe78a] Powell, M. J. D.: A fast algorithm for nonlinearly constrained optimization calculations. In G. A. Watson (ed.): Numerical Analysis, Dundee 1977. Lecture Notes in Mathematics No. 630, Springer, Berlin, 1978.

[Powe78b] Powell, M. J. D.: The convergence of variable metric methods for nonlinearly constrained optimization calculations. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson (eds.): Nonlinear Programming 3. Academic Press, New York, 1978.

[Powe78c] Powell, M. J. D.: Algorithms for nonlinear constraints that use Lagrangian functions. Math. Progr. 14, 224–248, 1978.

[Powe80] Powell, M. J. D.: Variable metric methods for constrained optimization. In L. C. W. Dixon, E. Spedicato, and G. P. Szegö (eds.): Nonlinear Optimization: Theory and Algorithms. Birkhäuser, Boston, 1980.

[Powe82] Powell, M. J. D.: Extensions to subroutine VF02. In R. F. Drenick, and F. Kocin (eds.): System Modeling and Optimization. Lecture Notes in Control and Information Sciences 38, Springer 1982.

[Powe84] Powell, M. J. D.: The performance of two subroutines for constrained optimization on some difficult test problems. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel: Numerical Optimization 1984. SIAM, Philadelphia, 1985.

[Powe85] Powell, M. J. D.: On the quadratic programming algorithm of Goldfarb and Idnani. Math. Progr. Study 25, 46–61, 1985.

[Press92] Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery: Numerical Recipes in C—The Art of Scientific Computing. Cambridge University Press, New York, 1992.

[Ray81] Ray, W. H.: Advanced Process Control. McGraw-Hill, New York, 1981.

[Rekl83] Reklaitis, G. V., A. Ravindran, and K. M. Ragsdell: Engineering Optimization—Methods and Applications. Wiley, New York, 1983.

[Renf87] Renfro, J. G., A. M. Morshedi, and O. A. Asbjornsen: Simultaneous optimization and solution of systems described by differential-algebraic equations. Comp. Chem. Eng. 11, 503–517, 1987.

[Robi74] Robinson, S. M.: Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. Math. Progr. 7, 1–16, 1974.

[Sarg78] Sargent, R. W. H. and G. R. Sullivan: The development of an efficient optimal control package. In J. Stoer (ed.): Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2. Springer, 1978.

[Schi79] Schittkowski, K., and J. Stoer: A factorization method for the solution of constrained linear least squares problems allowing subsequent data changes. Numer. Math. 31, 431–463, 1979.

[Schi81] Schittkowski, K.: The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. Numer. Math. 38, 83–114, 1981.

[Schi85] Schittkowski, K.: NLPQL—a Fortran subroutine solving constrained nonlinear programming problems. Annals of Operations Research 5, 485–500, 1985.

[Schm93] Schmid, C., and L. T. Biegler: Acceleration of Reduced Hessian Methods for Large-Scale Nonlinear Programming. Comp. Chem. Eng. 17, 451–463, 1993.

[Schu94] Schulz, V. H., H.-G. Bock, and R. W. Longman: Shortest paths for satellite mounted robot manipulators. In R. Bulirsch, and D. Kraft (eds.): Computational Optimal Control. Birkhäuser, Basel, 1994.

[Schu95] Schulz, V. H.: Reduced SQP-Methods for Large Scale Optimal Control Problems in DAE with Application to Path Planning Problems for Satellite Mounted Robots (Dissertation). Universität Heidelberg, 1995.

[Spel85] Spelucci, P.: Sequential quadratic programming: theory, implementation, problems. Meth. of Operations Science 53, 183–213, 1985.

[Stei95a] Steinbach, M. C.: Fast Recursive SQP Methods for Large-Scale Optimal Control Problems (Dissertation). Universität Heidelberg, 1995.

[Stei95b] Steinbach, M. C., H.-G. Bock, and R. W. Longman: Time-optimal extension or retraction in polar-coordinate robots: a numerical analysis of the switching structure. JOTA, to appear 1995.

[Stoer71] Stoer, J.: On the numerical solution of constrained least squares problems. SIAM J. Num. Anal. 8, 382–411, 1971.

[Stoer92] Stoer, J., and R. Bulirsch: Introduction to Numerical Analysis. Springer, New York, 1992.

[Stro91] Stroustrup, B.: The C++ Programming Language. Second Edition. Addison-Wesley, Reading, MA, 1991.

[Toml75] Tomlin, J. A.: On scaling linear programming problems. Math. Progr. Study 4, 146–166, 1975.

[Vand93] Vanderbei, R. J., and T. J. Carpenter: Symmetric indefinite systems for interior point methods. Math. Progr. 58, 1–32, 1993.