User Manual

MUSCOD-II Release 6.0

Christian Hoffmann Christian Kirches Andreas Potschka
Sebastian Sager Leonard Wirsching

based on the previous version by
Moritz Diehl, Daniel B. Leineweber, Andreas A. S. Schäfer

Simulation & Optimization Group
Hans Georg Bock and Johannes P. Schlöder
Interdisciplinary Center for Scientific Computing (IWR)
University of Heidelberg, Germany

March 28, 2011

Contents

1 Introduction

MUSCOD-II is a robust and efficient optimization tool that allows to quickly implement and solve very general optimal control problems in differential-algebraic equations (DAE).

The manual is organized as follows:

1. In Section 2 we present class of problems which can be solved, already introducing the problem syntax used in MUSCOD-II.

2. In Section 3 a brief introduction to the solution algorithm – the direct multiple shooting method – is given. Some understanding of the underlying method helps in learning the specific way of formulating problems for MUSCOD-II.

3. The installation of MUSCOD-II on a Unix-workstation is described in detail in Section 4. You will find information on necessary and useful third-party software, on obtaining the source code, on the file structure of the software, on the installation process, and on details about the location of the algorithmic modules.

4. How to set up and run a problem is described in Section 5. Setting up a problem involves creating a model source file and a data file. A description of the model source file is given in Section 6. In Section 7, a description of the data file as well as command line options are given.

5. An overview of the available dynamically loadable algorithmic modules is given in Section 8.

6. Screen and file output of MUSCOD-II is explained in Section 9. The interactive mode, which allows to view and log data during the optimization run, is described in Section 10.

7. The mixed-integer extension of MUSCOD-II, which is called MS MINTOC, is described in Section 11.

8. An example (source and data file) for standard MUSCOD-II is given in Section 13. An example for MS MINTOC can be found in Section 14.

2 Multistage Optimal Control Problems in DAE

Many dynamic process optimization problems of practical relevance can be expressed as multistage optimal control problems in DAE. MUSCOD-II is able to treat the following general class of multistage optimal control problems, where the time horizon of interest $[t_0, t_M]$ is divided into M model stages corresponding to the subintervals $[t_i, t_{i+1}]$, $i = 0, 1, \ldots, M-1$. On each of these intervals, the corresponding system state is described by the differential and algebraic state vectors $x_i(t) \in \mathbb{R}^{n_i^x}$ and $z_i(t) \in \mathbb{R}^{n_i^z}$. The system behaviour is controlled by the control vectors $u_i(t) \in \mathbb{R}^{n_i^u}$ and the global design parameter vector $p \in \mathbb{R}^{n^p}$.

On each of the model stages the system obeys a differential algebraic equation:

$$\left.\begin{array}{rcl} B_i(t, x_i(t), z_i(t), u_i(t), p) \cdot \dfrac{d}{dt} x_i(t) & = & f_i(t, x_i(t), z_i(t), u_i(t), p) \\ 0 & = & g_i(t, x_i(t), z_i(t), u_i(t), p) \end{array}\right\}, \quad t \in [t_i, t_{i+1}]$$

with the matrix function B_i in $I\!R^{n_i^x \times n_i^x}$ and the derivative $\dfrac{\partial g_i}{\partial z_i} \in I\!R^{n_i^z \times n_i^z}$ invertible, such that the linear-implicit DAE is of semi-explicit type and of index one.

The duration $h_i := t_{i+1} - t_i$ of model stage i may be variable. The end value of the differential state on stage i, $x_i(t_{i+1})$, is determined completely by the initial value $x_i(t_i)$, the control trajectory $u_i(\cdot)$ and the global parameters p and the duration h_i.

2.1 Transition between Model Stages

Between model stages continuity of the differential states is required by default:

$$x_{i+1}(t_{i+1}) = x_i(t_{i+1})$$

Therefore the differential dimensions do not change: $n_{i+1}^x = n_i^x$.

Jumps in the differential states and even dimension changes can be implemented by a special type of model stage, called *transition stage*. A transition stage with index j replaces the DAE integration for the determination of the final state value $x_j(t_{j+1})$ by the simple evaluation of a transition function c_j that may even change the differential state dimensions (n_{j+1}^x is not necessarily equal to n_j^x). Usually, the duration of a transition stage is set to zero, i.e. $t_{j+1} = t_j$. The continuity condition after the transition stage j provides an initial value for the following model stage $j+1$:

$$x_{j+1}(t_{j+1}) = c_j(t_j, x_j(t_j), z_j(t_j), u_j(t_j), p)$$

Here the transition function c_j has the same syntax as the right-hand side function f. [1]

2.2 Interior Point and Path Constraints

For all variables, i.e. states, controls, parameters, and durations, upper and lower bounds can and in fact have to be given. Additionally, general *decoupled* constraint vector functions r_k^d (with dimension n_k^{rd}) can be specified that require at single points $t = t_k$ in time or on complete model stages (i.e. $\forall_{t \in [t_k, t_{k+1}]}$):

$$r_k^d(t, x(t), z(t), u(t), p, p_k^r) \left\{ \begin{array}{c} = \\ \geq \end{array} \right\} 0$$

[1] Please note that allowing the *point* control value $u_j(t_j)$ to enter the transition function amounts to giving it the status of a parameter. If algebraic variables $z_j(t_j)$ are used on the transition stage they have to be defined by declaring an appropriate algebraic equation at time t_j

$$0 = g_j(t_j, x_j(t_j), z_j(t_j), u_j(t_j), p)$$

Note that a pointwise influence of the control values $u_j(t_j)$ on the transition function as above can also occur indirectly via the algebraic states.

Here, the first n_k^{rde} components are equalities and the remaining ones (of altogether n_k^{rd}) inequalities. These decoupled constraints can be formulated either only at the start or end points of a stage or on the whole interior of a stage.

For the formulation of *coupled* constraints, MUSCOD-II employs a specific formulation (for reasons of numerical efficiency) – it allows to couple different time points linearly in the following way: the user specifies vector functions r_k^c (at time points t_k) all of equal dimension n^{rc}. The vector sum of these functions is then required to satisfy:

$$\sum_{k=0}^{K} r_k^c(t_k, x(t_k), z(t_k), u(t_k), p, p_k^r) \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0$$

Again, the first n^{rce} components are taken as equalities, the rest as inequalities.

In both, decoupled and coupled constraints, *local* parameters p_k^r can be employed in addition to the global parameters p – they are preferable to global parameters for reasons of numerical efficiency (if they can replace them) [2].

2.3 The Objective Function

The objective function is of generalized Bolza type, containing Lagrange and Mayer terms for each model stage:

$$\sum_{i=0}^{M-1} \left(\int_{t_i}^{t_{i+1}} L_i(t, x_i(t), z_i(t), u_i(t), p)\, dt \quad + \quad \Phi_i(t_{i+1}, x_i(t_{i+1}), z_i(t_{i+1}), p) \right) \tag{1}$$

Note that no Lagrange term can be defined for transition stages.

2.4 Least Squares Objective Contributions

The objective function of Bolza type may be extended by an additional contribution that contains pointwise defined least squares terms of the form

$$\sum_{k=0}^{K} \| l_k^p(t_k, x(t_k), z(t_k), u(t_k), p, p_k^r) \|_2^2$$

where the time points t_k are specified as for the interior point constraints $r_k^d()$.

Though this special form of an objective contribution could also be formulated by use of Mayer terms as in (1), this explicit formulation allows to exploit the structure of the least squares terms in the numerical solution procedure.

As an additional feature, a continuous least squares function may be defined on each differential modelstage, so that a further contribution of the following form is added to the objective.

$$\sum_{i=0}^{M-1} \int_{t_i}^{t_{i+1}} \| l_k^c(t, x(t), z(t), u(t), p) \|_2^2 \, dt,$$

[2]Please note that a possible use of controls and algebraic states in the *coupled* interior point constraints allows some *point* control values to enter the problem and gives them the effective influence of parameters.

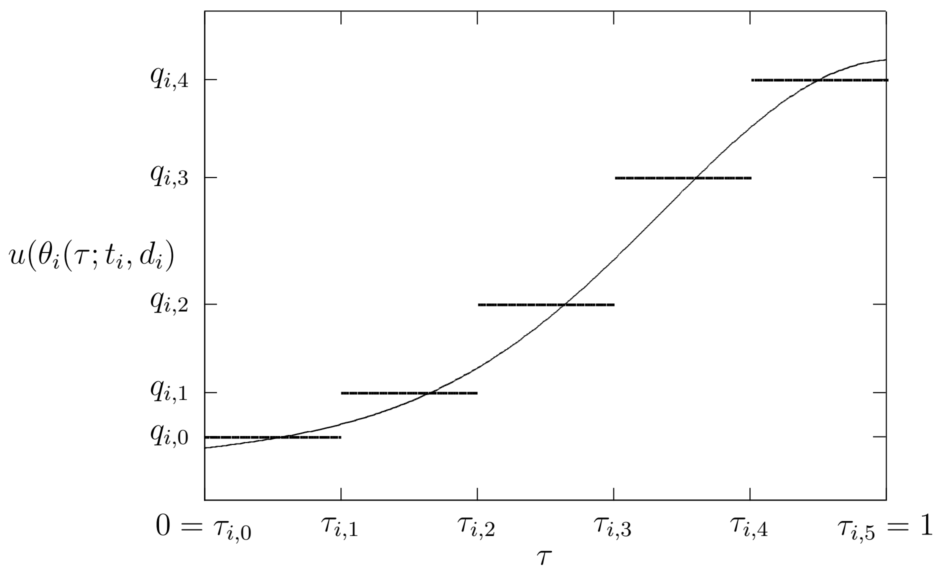


Figure 1: Piecewise constant representation of a control ($m_i = 5$).

which again could in principle be covered by Lagrange terms as in (1), but allows a favourable numerical treatment if explicitly formulated in least-squares form.

Remark:

The relevant dimensions of the problem and all functions mentioned in this section have to be provided by the user in the model source file described in section 6. A correspondence between the customary notation in the model-source file and the notation used in this section is given in Table 1.

3 The Direct Multiple Shooting Method

In the direct multiple shooting method the original continuous optimal control problem is reformulated as an NLP problem which is then solved by an iterative solution procedure, a specially tailored *sequential quadratic programming (SQP)* algorithm. A far more complete description of the methods employed is given by Leineweber, 1999 [Lei99].

3.1 Piecewise Control Discretization

In order to reformulate the original continuous problem as an NLP problem, first the control functions are approximated by a *piecewise* representation using only a finite set of control parameters. This is done by first dividing each model stage i into a number of m_i subintervals, called *multiple shooting intervals* $I_{i,j} := [t_{i,j}, t_{i,j+1}]$, $j \in \{0, 1, \ldots, m_i - 1\}$, with intermediate time points $t_{i,j}$. Then a piecewise approximation $\hat{u}_i$ of the control functions u_i on this grid is defined by

$$\hat{u}_i(t) := \varphi_{i,j}(t, q_{i,j}), \quad t \in I_{i,j} \quad j = 0, 1, \ldots, m_i - 1, \tag{2}$$

using "local" control parameters q_{ij}. The functions φ_{ij} are typically vectors of polynomials. If for example piecewise constant approximations are used for all control functions, we simply have $\varphi_{ij}(t, q_{i,j}) = q_{i,j}$ for $t \in I_{i,j}$, see the scalar example shown in Figure 1.

In MUSCOD-II, five possibilities are implemented: piecewise constant controls; piecewise linear with continuitiy on the corresponding stages, but not between different stages; linear with *overall* continuity; cubic with continuous differentiability, again stagewide or overall.

The user can explicitly specify the locations of the multiple shooting grid points relative to the model stage duration or instead use a uniform grid. If the model stage duration varies, the multiple shooting (sub-)intervals are scaled proportionally (and accordingly the piecewise control representations).

3.2 Multiple Shooting State Parametrization

The basic concept of the multiple shooting method is to solve the differential (algebraic) equation independently on each of the multiple shooting intervals. On interval j of the ith model stage ($j \in \{0, 1, \ldots, n_i^{ms}\}$) the initial value for the DAE solution is given by the so called *node values* $s^x_{i,j}$, $s^z_{i,j}$ for differential and algebraic states[3]. Consistency of the algebraic equations

$$g(t_{i,j}, s^x_{i,j}, s^z_{i,j}, \hat{u}_i(t_{i,j}), p) = 0, \tag{3}$$

and, particularly, continuity of the state trajectory at the multiple shooting grid points

$$s^x_{i,j+1} = x_{i,j}(t_{i,j+1}) \tag{4}$$

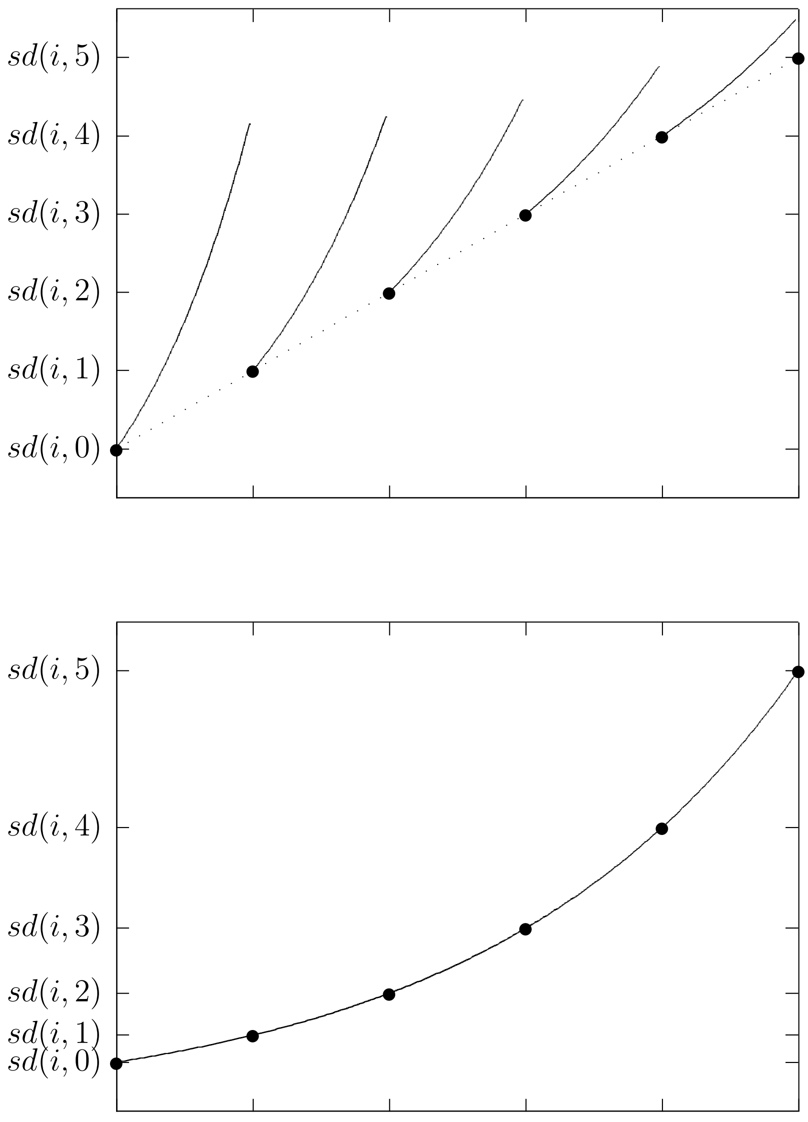
(where $x_{i,j}(t)$ denotes the differential part of the DAE solution on Interval $t \in I_{i,j}$ with initial values $s^x_{i,j}, s^z_{i,j}$) are incorporated as constraints into the NLP. They are required to be satisfied only at the solution of the problem, not during the SQP iterations. This allows to easily incorporate information about the trajectory behaviour into the initial guess, and it leads to good convergence properties of the multiple shooting method.

For more details, see e.g. Bock and Plitt [BP84] or Leineweber [Lei99].

3.3 Discretization of Bounds, Interior Point and Path Constraints

Upper and lower bounds for all multiple shooting variables, i.e. node state values $s_{i,j+1}$, control parameters q_{ij}, local and global parameters p^r_k and p, as well as the stage durations, can be specified. Note that this means a slight modification of the original problem, as state and control bounds may be violated *between* multiple shooting nodes in the solution. The same applies to the decoupled path constraints described by functions r^d_k. It should be noted, however, that in the important case of a piecewise constant or linear control representation, upper and lower control bounds are satisfied on the whole interval, if and only if they are satisfied at the multiple shooting nodes.

[3]Potential inconsistency of the algebraic equations at the m.s. nodes is dealt with a specific relaxed DAE-formulation on the m.s. intervals. See e.g. Leineweber [Lei99]



Multiple shooting method – discontinuous initial trajectory and continuous solution ($m_i = 5$). In this example, initial guesses for $s^x_{i,j}$ were obtained by linear interpolation between known boundary values.

3.4 Discretization of Least Squares Terms

The pointwise defined least squares functions l_k^p can be evaluated at all specified multiple shooting nodes, analogously to the constraint functions r_k^c, without any discretization errors. The advantage of an explicit formulation of these least squares terms – compared to formulating them as general Mayer objective contributions – is that this allows to obtain a Gauss-Newton approximation of the second derivative, e.g.

$$\frac{\partial^2 \|l_k^p\|_2^2}{\partial s_k^2} \approx 2 \frac{\partial l_k^p}{\partial s_k}^T \frac{\partial l_k^p}{\partial s_k},$$

which is good for small residuals $\|l_k^p\|_2^2$.

If a *continuous* least squares function l_k^c has to be integrated on a multiple shooting stage, this integral is in the current version of MUSCOD-II approximated by a sum using the trapezoidal rule as follows:

$$\int_{t_{i,j}}^{t_{i,j+1}} \|l_k^c(t, x_{i,j}(t), z_{i,j}(t), \hat{u}_i(t), p)\|^2 \, dt \approx \sum_{k=0}^{n_{i,j}^{stop}} w_{i,j,k} \|l_k^c(t_{i,j,k}, x_{i,j}(t_{i,j,k}), z_{i,j}(t_{i,j,k}), \hat{u}_i(t_{i,j,k}), p)\|^2$$

where the grid points $t_{i,j,k}$ are equally spaced between $t_{i,j,0} = t_{i,j}$ and $t_{i,j,n_{i,j}^{stop}} = t_{i,j+1}$ and the weights $w_{i,j,k}$ are set to $w_{i,j,k} = (t_{i,j+1} - t_{i,j})/n_{i,j}^{stop}$ for $k = 1, \ldots, n_{i,j}^{stop} - 1$ and half this value for $k = 0, n_{i,j}^{stop}$: $w_{i,j,0} = w_{i,j,n_{i,j}^{stop}} = \frac{1}{2}(t_{i,j+1} - t_{i,j})/n_{i,j}^{stop}$. The integrator has to stop at the grid points $t_{i,j,k}$ to evaluate the objective contribution and its derivatives[4].

Note that the approximation of the integral least-squares terms by a sum of intermediate points allows to compute a Gauss-Newton approximation of the second derivatives analogously to the case of point wise defined least-squares terms.

Remark:

All features specific to the multiple shooting method, i.e. the numbers m_i of multiple shooting intervals on the stages, upper and lower bounds, scales and initial guesses for the multiple shooting variables, and some output specifications have to be provided by the user in the data file described in section 7.1. For correspondence of the data file notation to the notation used in this section see also Table 2.

3.5 The resulting Nonlinear Programming Problem

If we subsume all multiple shooting variables (i.e. $s_{i,j}^x, s_{i,j}^z, q_{i,j}, h_i, p$, and p_k^r) to a single vector w of (large) dimension n, we can write the objective function as $F(w) : \mathbb{R}^n \to \mathbb{R}$. Similarly, we can subsume all equality constraints (in particular the continuity and consistency conditions (4) and (3)) to a vector valued function $G(w)$ and the inequality constraints in a vector valued function $H(w)$. Then, the parametrized optimal control problem can be written as a finite

[4]Note that this feature is so far only implemented in the integrators DAESOL and adfDAESOL.

dimensional Nonlinear Program:

$$\begin{array}{rcl} \min\limits_{w} & F(w) & \\ \\ \text{subject to} & & \\ G(w) & = & 0 \\ H(w) & \geq & 0 \end{array} \tag{5}$$

where the inequalities hold componentwise.

3.6 The SQP Algorithm

The SQP algorithm deals with the NLP problem where all functions are explicitly or implicitly defined as functions of the multiple shooting variables only. The numerical DAE solution on the multiple shooting intervals is performed in an underlying evaluation module and has to be carried out with sufficiently high integration tolerance.

Starting with an initial guess w_0 provided by the user, the SQP algorithm iterates

$$w_{k+1} = w_k + \alpha_k \Delta w_k$$

with step directions Δw_k (and relaxation factors $\alpha_k \in (0,1]$), until a prespecified convergence criterion is satisfied.

At the k-th SQP iteration with multiple shooting variables w_k, the algorithm evaluates the NLP functions (i.e. $F(w_k)$, $G(w_k)$, and $H(w_k)$) and their derivatives ($\nabla_w F(w_k)$, $\nabla_w G(w_k)$, and $\nabla_w H(w_k)$) with respect to w. In this way, linearizations of the originally nonlinear NLP functions are obtained that are used to build a quadratic programming (QP) subproblem. Furthermore, an approximation H_k of the Hessian matrix of the Lagrangian function is calculated.

The quadratic programming subproblem solved at the k-th SQP iteration can be written as:

$$\begin{array}{rcl} \min\limits_{\Delta w_k \in \Omega} & \nabla F(w_k)^T \Delta w_k & + \quad \frac{1}{2} \Delta w_k^T H_k w_k \\ \\ \text{subject to} & & \\ G(w_k) + \nabla_w G(w_k)^T \Delta w_k & = & 0 \\ H(w_k) + \nabla_w H(w_k)^T \Delta w_k & \geq & 0 \end{array} \tag{6}$$

where Ω is either the full Euclidean space $\mathbb{R}^n$ or a suitably chosen box in $\mathbb{R}^n$ (that contains $\Delta w_k = 0$) in the trust region approach.

The QP problem is then solved and results in a direction Δw_k that helps to determine the next iterate $w_{k+1} = w_k + \alpha_k \Delta w_k$. Different line search strategies are implemented that determine the relaxation factor α_k ; they are described in section 8.4.

For the new values of the multiple shooting variables all NLP functions and derivatives are again evaluated, a new Hessian matrix approximation H_{k+1} is provided and a new QP problem is solved for the next SQP iteration.

The iterations stop when the solution accuracy, measured by the so called KKT-tolerance, has reached a prespecified value `acc`. It indicates, roughly spoken, to how many digits the objective value is expected to be correct.

In MUSCOD-II, the approximation of the Hessian matrix is either chosen as an initially diagonal matrix[5] which is then revised during the SQP iterations by appropriate update procedures (described in section 8.3) that keep H_k positive definite. Alternatively, an "exact" Hessian matrix can be calculated numerically in each iteration – as positive definiteness of H_k is not guaranteed in this case, a trust region (i.e. a bounded Ω in Equation 6) has to be specified to have a well defined QP.

Remark:

Some specifications concerning the SQP algorithm (as warm starts, final accuracy, maximum number of iterations,...) can be given as optional arguments to the executable. See the explanations in 7.

4 The software package MUSCOD-II

The MUSCOD-II package is delivered together with the linear algebra library LIBLAC (Leineweber and Jost, 1996 [LJ96]) and with our ODE/DAE solver DAESOL (Bauer *et al.*, 1997 [BFD+97, BBKS00]). However, for MUSCOD-II to be fully functionable, some extra software is required:

- Linear algebra packages and subroutines:
  - BLAS routines (Lawson *et al.*, 1979 [LHKK79]; Dongarra *et al.*, 1988 [DCHH88] and 1990 [DCDH90]). The ATLAS library (Whaley, Petitet, Dongarra, 2005) of optimized BLAS routines may be used.
  - LAPACK routines DGETRF, DGETRS (Anderson *et al.*, 1995 [ABB+95])
  - *Harwell MA48* direct linear solvers (Reid and Duff, 1993 [RD93] and 1996 [DR96])
  - Harwell TD12 (HSL archive) sparse numerical derivative subroutine (Reid, 1972 [Rei72])
- Visualization packages:
  - PGPLOT 5.2 (Pearson, 1997 [Pea97]) is a graphics package that may be used for online graphics. It is not essential, but online graphics helps a lot to better understand possible difficulties.
  - MATLAB (The Mathworks, Inc.) may be used for online graphics as well.

Furthermore, MUSCOD-II contains interfaces to some external software modules not distributed with it. You may use these interfaces if you have the software available and hold an appropriate license. user (commercial products in italics):

- ODE/DAE solvers:
  - *DDASAC* (Caracotsios and Stewart, 1985 [CS85])

[5] By default, according to Plitt [BP84], an initial scaling factor is determined that bounds the first QP solution to be roughly twice as big as the minimum norm step satisfying the linearized constraints.

  - *MBSNAT*
  - *METANB*

- standard QP solvers, of which at least *one* has to be licensed, preferably QPQPT:
  - *BQPD*
  - OOQP
  - *QPOPT* (Gill *et al.*)
  - *NAG E04NFF / QPOPT* (Gill *et al.*)
  - *NAG E04NAF / QPSOL* (Gill *et al.*, 1983 [GMSW83])

Installation of MUSCOD-II under Windows is possible, but will require additional software to provide a Linux-like working environment.

- MSYS, a Linux-like shell for Windows.
- MinGW, a collection of the C/C++ and Fortran development tools for Windows.
- GrWin, which provides *PGPLOT* support for Windows.

The installation of MUSCOD-II under Windows is currently not covered by this manual.

4.1 Installation Steps for a UNIX/Linux System

here we describe the installation of MUSCOD-II within a so-called *suite*, a structured collection of packages used in the context of MUSCOD-II. We highly recommend to use the suite when working with MUSCOD-II. Using a manual installation of all required packages is possible, but requires expert knowledge and is likely to cause problems. You have been warned!

4.1.1 Overview

The MUSCOD-II Suite is a structured collection of software packages which are somehow related to MUSCOD-II. Major aims of this bundle are

- reduce the effort required for setting up a MUSCOD-II work environment to a minimum,
- simplify migrating MUSCOD-II to new platforms and operating systems and
- provide standardized environment for easy debugging.

4.1.2 Getting it

The MUSCOD-II suite is checked in to SimOpt's Subversion revision control system. The MUSCOD-II suite repository has a sub-folder for each target platform. Currently, only the Linux version is supported. Like every Subversion repository, the suite also comes in different flavors: `tags/*`, a collection of fixed versions that are not subject to change and `trunk`, the developers' version (unstable). To check out a local working copy of MUSCOD-II to your system, call

```
  name@machine:~> svn checkout \
https://liz.iwr.uni-heidelberg.de/MUSCOD/muscod_suite/linux/<flavor> ~/MUSCOD_SUITE
```

You will have to enter your svn password several times during the checkout, once for every new sub-repository that is accessed, see section Technical Stuff. The name of the target dir (here: `MUSCOD_SUITE`) can be chosen arbitrarily.

If you cannot access the SimOpt subversion server, either contact our system administrator (in the case you are a group member) or your SimOpt cooperation partner (if you are not a group member).

4.1.3 Structure

In its minimal version, the suite has the following directory structure

- `MUSCOD_SUITE/`: Root folder of the MUSCOD-II suite (name arbitrary)
  - `Apps/`: MUSCOD-II applications, **problem-dependent** code and data
  - `AuxScripts/`: Auxiliary shell scripts
  - `Packages/`: Software packages used by MUSCOD-II
  - `MC2/`: The package MUSCOD-II itself, **problem-independent** code
  - `bootstrap`: Installation script for the whole MUSCOD-II Suite
  - `cleanup`: Convenience script for removing files created by bootstrap ("make clean")
  - `LICENSE`: License
  - `README`: Basic information
  - `VERSION`: Version

The directories `MC2`, `Packages/*`, `Apps/*` all have a similar structure to facilitate the separation of source files (which are also in the repository) and configuration and compiled files, also called binary files (which only exist on the local machines). Thus, one can have one source but several differently configured or compiled versions of MUSCOD-II, generated from the same source. The source files are all contained in the `Src` subdirectory, e.g., `~\MUSCOD_SUITE/MC2/Src`. Typically, binary files are in directories `Debug` (in the case of a compilation with debug information), or `Release` (in the case of a compilation with high compiler optimization and no debug information). The bootstrap script creates these directories automatically as described below.

The subdirectories of `~\MUSCOD_SUITE/MC2/Src` encapsulate different modules of MUSCOD-II which are explained in section 4.2.1. The applications' subdirectories, e.g., `MUSCOD_SUITE/Apps/TEST`, have the following structure:

- `Src`: Source directory including problem-dependent configuration files, program code, and problem description data

  - `SRC` Subdirectory containing program code
  - `DAT` Subdirectory containing problem description data

- `Debug, Release, ...`: Binary directories containing compiled libraries
  - `DAT` Subdirectory containing problem description data (usually a symbolic link to `../Src/DAT`)
  - `RES` Subdirectory where the results of computations are stored

4.1.4 Functionality

The suite comes with some bash-scripts which automate common tasks. All scripts that are usually invoked by the user are located in the suite root folder and have executable file permissions. All scripts provide usage information if invoked with the `--help` command-line option. Here is a brief overview:

bootstrap Configures and installs all packages of the MUSCOD-II suite and some applications. Installations with different "build types" (`Debug`, `Release` etc.) are supported. The binaries of each build type have their own directories (with the same names as the build type) and can hence reside in the suite in parallel. This script extensively uses resources from folder `AuxScripts`. The whole installation procedure is described below.

cleanup Removes the binary directories of a selected build type from all packages in the suite. This is essentially a `make clean` for the whole suite and reverts the actions of bootstrap. Note: currently only packages built using CMake as install tool are affected, which are essentially all packages developed by the SimOpt group. Binaries from third-party packages currently have to be removed manually. When using the `-C` option, it only removes the CMake cache of all packages using CMake, effectively re-setting all CMake options to their default values.

4.1.5 Technical Stuff

The repository physically only contains the scripts. Packages belonging to the suite are linked via Subversion's `externals` mechanism. On checkout, update or export, subversion recursively checks out or updates the suite itself and all referenced packages and applications. If you have a revision-controlled working copy of the suite, you can query the current list of references with

```
user@machine:~/MUSCOD_SUITE> svn propget svn:externals .
```

Lines in the output that start with a `#` are commented out.

4.1.6 Installation

The MUSCOD-II suite should be easily installed by checking it out from the Subversion repository and running the bootstrap script.

Prerequisites MUSCOD-II uses many different software packages. The following list of software should be installed by the system's native package management system (e.g., YaST, aptitude, etc.):

- boost-dev
- cmake $\geq$ V2.6.0
- gcc
- gfortran
- subversion
- tcl
- make

Building You can now build and install the whole suite with the bootstrap script. By typing

```
./bootstrap --help
```

you get a list of supported options. They are related to build target (Release/Debug), 32/64 bit compilation, number of parallel build jobs, and more. The default settings should be reasonable for most target platforms. If performance is not an issue for you, it is recommended to build the suite in `Debug` mode, which is the default. After a successful installation, you may want to run a sample MUSCOD-II application to check if everything was built properly.

If you encounter problems during install which you cannot solve, ask for help on the MUSCOD-II mailing list[6] or the SimOpt Wiki[7].

Local Installation The files installed by the command

```
name@machine:~/MUSCOD_SUITE> ./bootstrap
```

stay within the suite folder as long as no `-ipref` argument is specified. This is the desirable behavior for developers. It is also possible to install to a system folder, which is described in the next section.

After successfully running bootstrap, the executables provided by MUSCOD-II can be found in `~/MUSCOD_SUITE/MC2/Debug/bin`. You might want to add that path to your PATH environment variable (however, be careful if you have several build types in parallel!) by adding

```
export PATH=$PATH:$HOME/MUSCOD_SUITE/MC2/Debug/bin
```

to your `~/.bashrc`.

[6]`mailto:agbock_mc2_developer@iwr.uni-heidelberg.de`

[7]`http://ginger.iwr.uni-heidelberg.de/wiki/index.php5/Category:MUSCOD-II`

To remove the files of a `<build_type>` installation, just type

```
name@machine:~/MUSCOD_SUITE> ./cleanup --btype=<build_type>
```

Install To System It is possible to install the packages to subfolders of a central location, from where it can be used by every regular user. The install procedure is identical to the one described above, except that the installation path prefix (e.g. /use/local) has to be specified:

```
name@machine:~/MUSCOD_SUITE> ./bootstrap --ipref=<prefix>
```

Depending on their type, files are installed to different subfolders:

→ `<prefix>/bin/`: executables

→ `<prefix>/lib`: static (*.a) and shared (*.so) libraries (on 32 bit platform)

→ `<prefix>/lib64`: static (*.a) and shared (*.so) libraries (on 64 bit platform)

→ `<prefix>/share/<package>/`: other data related to `<package>`

Attention: Applications for MUSCOD-II and MUSCOD-II itself are never installed to a dir other than `~/MUSCOD_SUITE/Apps/<app>/<build_type>/`! After a successful installation, the suite dir could be removed. But mind the note above and do not delete your applications! Deinstallation has to be done manually. Installation logs are provided at least by the CMake packages in their respective binary dirs.

Usage with other Software Suites In case you use other software from the SimOpt repository which uses common packages (e.g. both MUSCOD-II and VPLAN both use COMMON_CODE), it may be advisable to have the common packages installed only once on your system. This can be achieved by creating a symlink called `Packages` (with exactly that name!) in the `MUSCOD_SUITE` dir to the single common Packages directory before the checkout, e.g.:

```
user@machine:~/MUSCOD_SUITE> ln -s ../Packages
```

Then the packages will actually be checked out to the packages directory on the “suites” level. Using the same technique on your other projects ensures that you only have one actual location for the sources of common packages. You are expected to encounter problems with this approach if you try to mix branches/tags/trunk versions of different suites, e.g. a special MUSCOD-II/tag with MUSCOD-II/trunk. In this case, MUSCOD-II will try to check out a tagged version of the package and MUSCOD-II will try to check out the trunk version of the package to the same directory. In this case the use of suite packages is not advisable and you are encouraged to perform the recursive checkout manually.

Since revision 6, there is only one executable, which links dynamically shared object files at runtime, corresponding to algorithmic settings.

4.2 Making Changes to the Installation

4.2.1 Organization of the Source Code

The directory `MC2` contains a number of subdirectories organizing the source code into specific algorithmic parts of MUSCOD-II. Some subdirectories offer alternative source files that are compiled into different flavors of the same algorithm.

COND Condensing of the block sparse multiple shooting QP into a dense unstructured QP. Comes in two flavors: Standard condensing and minimal condensing. The latter only deals with the control discretization and with separability of global optimization variables such as model parameters, but does not do any condensing.

EVAL Evaluation of all model functions, and computation of derivatives using finite-difference approximation.

HESS Computation of the Hessian of the Lagrangian of the multiple shooting NLP. Comes in various flavors: Constant hessian, Gauß-Newton approximation, BFGS approximation, limited-memory BFGS approximation, and exact Hessian using finite difference stars.

IND Internal Numerical Differentiation. Holds a broad selection of integrators for the solution of ODE/DAE system with various properties, all equipped with *internal numerical differentiation.*

MSSQP Multiple Shooting Sequential Quadratic Programming.

PLOT Plotting. Visualization of state trajectories, control profiles, and the history of objective function values, model parameters, and stage lengths using different backends. Currently implemented are PGPLOT, Matlab, and no plotting.

QPS Quadratic Programming. Solution of the condensed quadratic programs using a broad range of commercial and open–source QP solvers such as QPSOL, QPOPT, OOQP, and BQPD.

SOLVE Globalization of the MSSQP step. Holds line search, trust region, and watchdog approaches.

TCHK Termination Checks for the MSSQP algorithms.

Besides those algorithmic parts, a number of subdirectories hold integral parts of MUSCOD-II that are always present.

ADCAUX ADOL-C auxiliary functions for computation of derivatives of model functions.

DOC Holds the LaTeX source code of this documentation.

INCLUDE Holds C++ header files for all algorithmic parts.

INOUT Input and Output of MUSCOD-II data structures. Ultimately responsible for loading the DAT file.

LINALG Selected linear algebra routines from the HSL archive, used by PRSQP and some of the integrators found in IND.

MAIN Different flavors of the main executable of MUSCOD-II. Currently maintained is `main_dynamic.cpp` only.

MINTOC Mixed-Integer Optimal Control package, see section 11.

MODEL Description of the optimization problem.

NMPC Nonlinear Model Predictive Control.

PDAUX Problem Data Auxiliary functions.

PRSQP Partially Reduced Sequential Quadratic Programming, for optimizationm problems with DAE models. Comes in two flavors: A sparse variant using HSL MA48, and a dense variant using LAPACK.

SCALE Scaling of problem variables.

TD12AUX Support for approximation of sparse jacobians using finite differences and CPR seed matrix compression, uses HSL archive routine TD12.

UTIL Utility functions for MUSCOD-II for things like error handling, logging, interactive mode, stack traces, etc.

Some directories hold currently deprecated or non-maintained code: `QUICKAUX`, `ROBUST`.
The following directories belong to the CMake build system: `CMake`, `helperscripts`.

4.2.2 CMake Compilation Flags

In the main makefile of the MUSCOD-II sources, `~/MUSCOD-II/MC2/makefile` several optional compilation flags can be set.

```
#  user-defined parts of CFLAGS
#    NDEBUG    ...  generate non-debug version
#    MSPLOT    ...  include online graphics
#    CSTATS    ...  include computational statistics
#    PRSQP     ...  use partially reduced SQP strategy
#    FEASIMP   ...  use feasibility improvement for PRSQP
#    NDIRDER   ...  do not use directional derivatives for PRSQP
#    CENTDIFF  ...  use central difference gradient approximations
#    REGOBJ    ...  use regularized objective
#    LSQ       ...  allow least squares terms
#    CLSQ      ...  allow continuous least squares terms
```

As an example, your compilation flags could be:

```
CCUFLAGS = -O2 -DNDEBUG -DCSTATS -DPRSQP -DMSPLOT -DLSQ -DCLSQ

#  user-defined parts of FFLAGS
FCUFLAGS = -O2
```

Here, `-O2` stands for the desired optimization level of your compiler.

Please note that after changing one of these flags *no* automatic compiling is performed after calling `make` without arguments. Instead, one has to use:

```
make FRC=force_rebuild
```

4.3 Compiler Warnings

As a MUSCOD-II distribution contains also third-party software, the developers cannot guarantee that every of the source files will compile without an compiler warning.
These warnings are not of importance neither for the developers nor for the users and can sometimes be quite disturbing. Therefor the compiler warnings for these (individually checked) files can be deactitaved by setting the `MC2_FORGET_KNOWN_COMPILER_WARNINGS` in the file `user.mk` to YES. This is also the default behavior.

4.4 Compatibility of Compilation Flags

Here are listed some known imcompatibilities between different compilation flags:

- (NC): Are incompatible and will not compile together.
- (PW): Partially working. Will compile together, but some functionality of either of one is not provided in some cases.

	List of Imcompatibilities:
(NC)	`PRSQP` and `EXTPRSQP`
(PW)	`CLSQ` and `EXTPRSQP`: `eval_clsq` not evaluated in `rkfXX` and `DAESOL-II`.

5 How to set up a problem

In order to solve an optimal control problem with the stand-alone version of MUSCOD-II, two files have to be prepared by the user: a C or Fortran 77 file which defines the model equations (objective, differential equations, constraints), and an ASCII file which contains the correponding problem data (e.g., initial guesses, scaling factors, bounds).

1. *Model Source File.* Here, the model equations are defined either as ANSI C functions or as Fortran 77 subroutines. In addition to these routines, a function or subroutine `def_model()` must be provided in which the multistage optimal control problem is formally defined.

   The model file must be compiled as a dynamic library as described below.

2. *Data File.* The contents of this ASCII keyword file and its syntax are described below.

5.1 Running an existing problem from the test set

The directories `MUSCOD_SUITE/Apps/TEST` and `MUSCOD_SUITE/Apps/MIP` contain sample problems which lend themselves to being used as reference and templates. To launch one of these problems, change to a binary directory (e.g., `Debug`) and inspect the files in the `DAT` subdirectory. Each of these files describes a combination of a dynamic optimization problem and the variant of the direct multiple shooting SQP algorithm to be used for the solution. To launch one of these, e.g., `DAT/reentry.dat`, create a symbolic link in `MUSCOD_SUITE/Apps/TEST/Debug` by entering

```
user@machine:~/.../TEST/Debug> ln -s ~/MUSCOD_SUITE/MC2/Debug/bin/muscod
```

(of course, this has to be done only once) and call

```
user@machine:~/.../TEST/Debug> ./muscod reentry
```

Alternatively, one may create an `alias` or to add the `MC2/bin` directory to the `PATH` environment variable (cf. your shell's manual, e.g., `man bash`) to avoid the need of typing the prefix `~/MUSCOD_SUITE/MC2/Debug/bin/` of the `muscod` executable. Please note, that when using a different MUSCOD-II binary, e.g. from a Release build, you have to adapt the symbolic link (or any other of the described shortcuts).

5.2 Setting up a new problem

We recommend to add new problems to the existing `MUSCOD_SUITE/Apps/TEST/Src` (or `MIP/Src`) directory. The user has to provide the model source file and the data file, say `<problem>.cpp` and `<problem>.dat` (a detailed description of the structure of these files is given below). The files have to be treated as follows:

- The model source file `<problem>.cpp` has to be added to the `TEST/Src/SRC` directory. Then, edit the `CMakeLists.txt` in this directory. Search for the first occurrence of the CMake variable `TEST_LIBS` in `CMakeLists.txt`. This should be looking like `SET ( TEST_LIBS` followed by a (long) list of names. Add the name of your source file (i.e. `<problem>`) to the list. Take care that you put the name before the end of the list (the right parenthesis) and that you skip the file extension `.cpp`. Save your changes to `CMakeLists.txt`.

- The data file `<problem>.dat` has to be added to the `TEST/Src/DAT` directory.

In the desired binary directory, say `MUSCOD_SUITE/Apps/TEST/Debug`, run `make`. The make process deals automatically with the changes in `CMakeLists.txt` by calling CMake before starting the build. After a successful build, one may execute the problem as described above.

6 The Model Source File

In the model source file, the dynamical model equations, the constraints, and the objective functions of the optimization problem are defined. Furthermore, the user has to provide a function for the setup of the optimization problem. It is highly recommended for the unversed user to use an existing model source file as a template to create new problem source files.

6.1 The parts of the model - dynamic equations, constraints, and objective functions

All functions explained below are defined in `def_usrmod.hpp`, so this file must be included in every new model source file. To relate the function and variable names used in this section to the nomenclature of the theoretical part of the manual the user may refer to Table 1.

`ffcn(t, xd, xa, u, p, rhs, rwh, iwh, info)`

This function describes the differential right-hand side of an ODE or DAE system. Arguments `t`, `xd`, `xa`, `u`, and `p` are double pointers to the current time, differential and algebraic states, controls, and model parameters respectively. Upon return of the function call, the double pointer `res` should have been filled with the requested function values. Double pointer `rwh` and long pointer `iwh` are auxiliary real and integer working arrays, and long pointer `info` should return an error code indicating the status of the function evaluation.

`gfcn(t, xd, xa, u, p, rhs, rwh, iwh, info)`

This function describes the algebraic right-hand side of a DAE system. The arguments are exactly the same as for the differential right-hand side.

`afcn(t, xd, xa, u, p, amat, lda, rwh, iwh, info)`

This function describes the left-hand side of a DAE system. Double pointer `amat` should be filled with the left-hand side matrix upon return of the function call. Long pointer `lda` holds the leading dimension of `amat` as input. The other arguments are the same as for the differential right-hand side.

source file	manual	mathematical content
`NMOS`	M	number of model stages
`NP`	n^p	number of global parameters
`NRC`	n^{rc}	number of (global) coupled constraints
`NRCE`	n^{rce}	number of (global) coupled *equality* constraints. The first `NRCE` of the total number `NRC` of coupled constraints are defined to be equality constraints.
`rcfcn`	r_i^c	coupled multi point constraint function
`NXD`	n_i^x	number of differential states on model stage i
`xd`	x_i	differential state vector on model stage i
`sd`	$s_{i,j}^x$	differential node value on model stage i at node j
`NXA`	n_i^z	number of algebraic states on model stage i
`xa`	z_i	algebraic state vector
`sa`	$s_{i,j}^z$	algebraic node value on model stage i at node j
`NU`	n_i^u	number of controls on model stage i
`NRD`	n_k^{rd}	number of decoupled constraints for specific constraint function at time t_k.
`NRDE`	n_k^{rde}	number of decoupled *equality* constraints at time t_k. The first `NRDE` of the total number `NRD` of decoupled constraints are defined to be equality constraints.
`rdfcn`	r_i^d	decoupled interior point constraint function
`NPR`	n_k^{pr}	number of local constraint parameters for constraint point t_k
`pr`	p_k^r	local constraint parameter vector
`afcn`	B_i	invertible matrix in semi-explicit DAE formulation on model stage i. By default: $B_i \equiv I\!I$
`ffcn`	f_i	differential right hand side function on model stage i
`ftrans`	c_i	transition function on model stage i. Same syntax as `ffcn`.
`gfcn`	g_i	algebraic right hand side function on model stage i
`mfcn`	Φ	Mayer term of objective
`lfcn`	L	Lagrange term of objective
`lsqfcn`	l_k^p,l_k^c	least squares residual vector function

Table 1: Correspondence between customary MUSCOD-II source file notation and mathematical notation

The `RKFSWT` integrator supports the detection of implicitly defined switching events within a model stage, thus eliminating the need for a multi-stage modelling approach and the introduction of transition stages under certain circumstances.

Implicit switches are realized by an implicit switching function, which enables the detection of a switching event by the change of sign in the switching function residuals and a state jump function, which allows for an update of the differential states once a switching event has occurred. The interfaces of the implicit switching function and the state jump function are as follows:

`swtdtcfcn(imos, tau, xd, xa, u, p, nstep, iswt, nswsta, res, rwh, iwh, info)`

This function is called to detect implicitly defined switch conditions. `imos` is a long pointer to the current model stage index. `tau`, `xd`, `xa`, `u`, and `p` are double pointers to the current physical time, differential and algebraic states, controls, and model parameters respectively. The parameter `nstep` is undocumented. `iswt` is a long pointer to a list of indices of the switches to be evaluated by the current function call. `nswsta` is a long pointer to the length of the index list pointed to by `iswt`. Upon return of the function call, `res` should have been filled with the requested switch function residuals.

`swtexecfcn(imos, tau, xd, xa, u, p, iswt, rwh, iwh, info)`

This function is called when an implicitly defined switch condition has been detected. `iswt` is a long pointer to the index of the flipped switch, while the meaning of all other parameters is identical to `swtdtcfcn`. Upon return of the function, the differential states `xd` should have been updated if desired.

The switching functions described above are assigned to the model stages by using the `def_swt` function within the model definition function, see below.

The ODE model's right-hand side `ffcn` is provided with the current switch structure by way of the following mechanism, that had to favour backward-compatibility over clarity. Although still declared a `long *`, the `info` pointer of `ffcn` no longer points to a single long integer value, but may instead be safely typecast into a pointer to the structure `rkfXXswt_info_t` defined in `MC2/IND/RKFSWT/ind_rkfXXswt.hpp` with the following declaration:

```
typedef struct {
    long    info;       // return code, as usual
    long*   swt;        // sign structure of the NSWT switches on the stage
} rkfXXswt_info_t;
```

It is up to the model implementor to respect the `swt` vector within `ffcn` so as to evaluate the proper model. Using this mechanism, implicit discontinuities in the model's right-hand side may be covered. In order to cover discontinuities in the differential states themselves, modify them from within `swtexecfcn`.

`rfcn(ts, sd, sa, u, p, pr, res, dpnd, info)`

This function describes the decoupled and coupled interior point constraints. Double pointer `ts` contains the time point at the multiple shooting node where the function is evaluated. Double pointers `sd`, `sa`, `u`, and `p` hold the differential and algebraic node values, the control node values, and the parameter values, respectively. Interior point constraints may use separate local parameter values, which are provided by double pointer `pr`. Upon return of the function call, double pointer `res` should be filled with the interior point constraint residuals. Double pointer `dpnd` holds information about the dependencies of the functions w.r.t. the input variables. The user should start each implementation of an interior point constraint with

```
if (*dpnd) { *dpnd = RFCN_DPND(*ts, *sd, *sa, *u, *p, *pr); return;},
```

where arguments of `RFCN_DPND` which are not used in the function should be set to `NULL`. Long pointer `info` should contain the error code after evaluation, as usual.

`mfcn(ts, sd, sa, p, mval, dpnd, info)`

This function represents a Mayer-type objective function. Double pointer `mval` should contain the Mayer objective value after evaluation. The other arguments have the same meaning and usage as the corresponding arguments in the interior point constraints.

`lfcn(t, xd, xa, u, p, lval, rwh, iwh, info)`

This function represents a Lagrange-type objective function. Note, that the actual objective function value is the integral over time of this function. Double pointer `lval` should return the value of the Lagrange term at time `t`. All other arguments are the same as in the differential right-hand side.

6.2 Putting the model together - the model definition function

The essential function in the model source file is

`def_model()` *(without arguments)*

that formally defines the optimization problem by calling internal MUSCOD-II functions (declared and documented in `def_usrmod.hpp` or `def_usrmod_f77.hpp`). Here the previously defined *model* functions are assigned their role in the optimization problem. If any of the model functions does not exist for a given problem, the `NULL` pointer must be passed instead of the function pointer.

In `def_model()` the following internal MUSCOD-II functions must be called appropriately:

`def_mdims(NMOS, NP, NRC, NRCE)`

must be used to specify the global model dimensions, where

`NMOS` is the number of model stages,

`NP` the number of global parameters,

`NRC` the total number of *coupled* interior point constraints (i.p.c.), and

`NRCE` is the number of those that are equality constraints (thus `NRCE` $\leq$ `NRC`). By convention, the *first* `NRCE` components of the `res` vector in the coupled i.p.c. functions `rcfcnXX` are interpreted as equalities, the remaining ones are required to be greater than zero. Cf. section 2.2.

```
def_mstage(I, NXD, NXA, NU, mfcn, lfcn, jacmlo, jacmup, astruc,
           afcn, ffcn, gfcn, rwh, iwh)
```

Call to define a model stage with index `I`, where

`NXD` is the differential state dimension, `NXA` the algebraic state dimension, and

`NU` is the control dimension.

`mfcn` is a pointer to a Mayer term function (or `NULL`) to be evaluated at the end of the stage, and

`lfcn` a pointer to a Lagrange term (or `NULL`).

For documentation of the left-hand side matrix function `afcn`, and of the integers `jacmlo`, `jacmup`, and `astruc` that provide structural matrix information please consult the DAESOL-manual [BBS99]; setting the integers to zero is equivalent to not defining any structural information.

`ffcn` is a pointer to the differential right hand side function,

`gfcn` the pointer to the algebraic right hand side function (or `NULL`).

`rwh`, `iwh` are real and integer work arrays which can be used to pass a common workspace to the stage functions.

`def_mpc(I, SCOPE, NPR, NRD, NRDE, rdfcn, rcfcn)` *(optional)*

Call `def_mpc` to define interior point constraints (i.p.c.) on a stage.

`I` is the stage index,

`SCOPE` is a string whose first character should be one of `"s","i","e"` or `"*"` *(case insensitive)*, indicating if the following constraint functions shall be evaluated at the start point of the stage only (s), at the interior multiple shooting nodes (i), at the end point (e), or at all multiple shooting nodes of the stage together (*).

Note: The end point can only appear in the final model stage, otherwise the start point of the following stage must be used.

`PR` is the number of *local* parameters `pr` used in `rdfcn()` and `rcfcn()`,

`NRD` defines the dimension of the *decoupled* residual vector `res` in `rdfcn`.

`NRDE` leading componenents of this vector are required to be zero, while the remaining ones are required to be greater than zero. It should be noted here that the *coupled* `rcfcn`-functions at different points have to agree in the dimension `NRC` of the residuals `res`, cf. section 2.2.

`def_lsq(I, SCOPE, NPR, NLSQ, lsqfcn)` *(optional)*

may be called to define least squares terms contributing to the objective on a stage.

`I` is the stage index,

`SCOPE` selects the contributing nodes. The string should contain `"s","i","e"` or `"*"` *(case insensitive)* as first character to define least squares terms $l^p()$ at the corresponding multiple shooting nodes, as described in `def_mpc()`, or alternatively `"c"` to define a continous least squares term $l^c()$ that is integrated on the model stage analogously to a Lagrange term.

Attention: The continuous least squares terms can so far only be treated by the integrators DAESOL, DDASAC, RKF45ADJ, and RKFSWT. Also, this command is only allowed if the CMake flags `LSQ` resp. `CLSQ` are set.

`NPR` indicates the dimension of the vector of local parameters.

`NLSQ` defines the dimension of the residual vector `res` in `lsqfcn()`. The function `lsqfcn()` obeys the same syntax as `rdfcn()` and may depend also on the *local* parameters `pr`, the dimensions of which are specified in `def_mpc()`[8].

`def_mio(minp, mout, mplo)` *(optional)*

allows for an *(optional)* definition of input, output and external plot functions. The input function `minp` is called by MUSCOD-II immediately after the problem data file (see subsection 7.1) has been read. (Data passed through `minp` supersedes the data read from the data file.) The output function `mout` gets is called by MUSCOD-II with the final results (as standard arrays) and thus allows to implement a user defined output, i.e. by printing some results into an external file.

`def_plotoptions(min, max, f1, f2, f3, f4)` *(optional)*

allows to adjust plot options. The first two arguments set the plot granularity of the above `mplo` function, and also of the online graphics. The values specify the lower and upper bound for the number of calls to `mplo` per multiple-shooting interval. The default values are `min=5` and `max=200`. If you require more precise plot data, you should increase the lower bound. Excessively large values of the upper bound will damage the online graphics performance.

The flags `f1, f2, f3, f4` turn on or off the plotting of `f1` vertical lines to indicate stage transition times, `f2` plotting of MS nodes, `f3` plotting of state bounds and `f4` plotting of control bounds. The default value is 1 (yes) for all of them.

`def_swt(I, NSWT, swtdtcfcn, swtexecfcn)` *(optional)*

Attention: Implicit switches are currently supported by the `RKFSWT` integrator only.

`def_swt` defines `NSWT` implicit switches (discontinuities in the states and/or right-hand side) on model stage `I`. `swtdtcfcn` is a mandatory pointer to the switch detection function called to detect switch conditions during integration, while `swtexecfcn` is a mandatory pointer to the switch execution function called only if a switch actually happens.

7 Data and Options

MUSCOD-II uses a keyword based data file for problem data input. Furthermore, this file is also used to override default values for algorithmic settings and options (which in turn can again be overridden by command line options). The order of data items in the problem data

[8]For the continuous least squares terms $l^c()$ the local parameters mean an additional argument, in contrast to the formulation used in subsections 2.4 and 3.4: now, the local parameters are treated like piecewise constant controls, each one used on the multiple shooting interval *after* the point it is originally defined for.

file is relevant only in the sense that the first keyword match from the beginning of the file determines the data item which is read, i.e., possible further keyword matches are ignored.

7.1 Data and the DAT File

Table 2: Correspondence between MUSCOD-II data file notation and manual notation

data file	manual	mathematical content
`nshoot`	m_i	number of multiple shooting nodes on model stage i
`nstop(i,j)`	$n_{i,j}^{stop}$	number of integrator stopping points on m.s. interval $I_{i,j}$
`sd(i,j)`	$s_{i,j}^x$	differential multiple shooting node value at time point $t_{i,j}$ on model stage i.
`sa(i,j)`	$s_{i,j}^z$	algebraic multiple shooting node value at time point $t_{i,j}$ on model stage i.

7.1.1 Data items in DAT Files

Each data item has the form:

```
key
.
.  (associated data)
.
```

The keyword must start at the beginning of a line and must be terminated by a white-space character (the rest of the line is ignored and can be used, e.g., for comments). The following line(s) must then contain the data associated with the keyword. Six different types of data may occur, namely long scalar (`long`), long vector (`LVec`), double scalar (`double`), double vector (`DVec`), string (`Str`), and string vector (`StrVec`). Comments can follow any line exept those containing a string or an element of a string vector. Note that vectors are written one element a line with element indices starting from zero.

Example (data items)

```
s_spec   ! long
2        ! start integration

nshoot    ! LVec with three elements
0: 4      ! initial stage
1: 6      ! intermediate stage
2: 4      ! final stage
```

```
of_sca    ! double
1.0       ! unscaled

p_sca     ! DVec with one element
0: 1.0E-8 ! catalyst concentration

of_unit   ! Str
g/h

xd_name   ! StrVec with three elements
0: Substrate Concentration S
1: Product Concentration P
2: Volume V

sd(1,1)   ! DVec with arbitrary number of elements
ALL: 1.0
```

All keywords are explained in the next section.

7.1.2 Keywords in DAT Files

There are keywords containing one argument

```
key(arg1)
```

and keywords containing two arguments

```
key(arg1,arg2)
```

where `arg1` specifies the model stage and `arg2` in addition specifies one or more multiple shooting points on this model stage. Hence, `arg1` may be

- a valid model stage index
- an asterisk `*` ("all model stages")

and `arg2` may be

- a valid multiple shooting point index for the model stage specified by `arg1`
- the letter `S` or `s` ("start point")
- the letter `I` or `i` ("interior points")
- the letter `E` or `e` ("end point", valid only for final model stage)
- an asterisk `*` ("all multiple shooting points", including "end point" on final model stage)

Note that model stage indices and multiple shooting point indices start from zero. (Therefore, `arg2` = `S` and `arg2` = `0` are equivalent.)

Example (keywords with arguments)

```
sd_sca(*,*)    ! all multiple shooting points on all model stages

sd(*,S)        ! ``start point'' on all model stages

u(0,*)         ! all multiple shooting points on first model stage

rd_sca(0,S)    ! ``start point'' on first model stage

d_sca(0,I)   ! ``interior points'' on first model stage

rd_sca(0,E)    ! ``end point'' (assuming there is only one model stage)
```

Some of the data items are optional – if not explicitly specified, an internal default is used. In the following description, optional data items are indicated by keywords in square brackets [ ], and the corresponding default is given.

7.2 Keywords Defining the Optimal Control Problem

`nshoot`
: numbers of multiple shooting intervals on model stages (`LVec`)

[`grid(*)`]
: multiple shooting grids on model stages (`DVec`)
: default: equal spacing of grid points

[`mos_start`], [`msn_start`]
: model stage start index and multiple shooting point start index for partial reoptimization (`long`)
: defaults: `0`

`p`, `p_sca`, `p_min`, `p_max`
: global model parameter start values, scale factors, and bounds (`DVec`)

[`p_fix`]
: global model parameter fixed value flags (`LVec`) (if `p_fix[i]` is `1`, then parameter `p[i]` is fixed at its start value)
: default: no global model parameters fixed

`h`, `h_sca`, `h_min`, `h_max`
: model stage duration start values, scale factors, and bounds (`DVec`)

[`h_fix`]
: model stage duration fixed value flags (`LVec`) (if `h_fix[i]` is `1`, then duration `h[i]` is fixed at its start value)
default: no model stage durations fixed

[`s_spec`]
: specification mode for state variable start values (`long`)

  - `0` : all values `sd(*,*)`, `sa(*,*)` specified in data file
  - `1` : only values `sd(*,S)`, `sa(*,S)` and `sd(`$M-1$`,E)`, `sa(`$M-1$`,E)` specified, other values automatically generated by linear interpolation (M denotes the number of model stages)
  - `2` : only values `sd(0,S)`, `sa(0,S)` specified, other values automatically generated by integration
  - `3` : only values `sd(0,S)`, `sa(0,S)` specified, other values automatically generated by integration. The integration will be repeated, with the initial values replaced by the values at the end of the time horizon, until a steady state is reached or the maximum number of `simulation_maxiter` iterations has been performed. To cope with special cases with shifted variables, see `simulate_shift_select`.

  default: `0`

[`simulation_maxiter`]
: Maximum number for repeated integration to reach steady state (`long`) (relevant only if `s_spec` is `3`)
default: `100`

[`simulate_shift_select`]
: For every differential variable an index is given. Any value less or equal to -2 for variables with fixed initial value, the value -1 for all variables for which the initial value is replaced with the value at the end of the time horizon (after integration), and the index $0 \leq i \leq n_x$ of another differential variable, if for variable j a periodic shift is performed (as in the SMB example), $x_j^{k+1}(t_0) = x_i^k(t_f)$ in iteration $1 \leq k \leq$`simulation_maxiter` (`LVec`) (relevant only if `s_spec` is `3`, works only if number of differential states is constant)
default: `-1`

[`s_itol`], [`s_pert`]
: start integration tolerance, state perturbation factor (`double`) (relevant only if `s_spec` is `2` or `3`)
defaults: `1.0E-6`, `0.0`

[`nstop(*,*)`]
: number of integrator stopping points on corresponding multiple shooting interval(s) – only needed for continuous least squares terms. (`long`) .

`sd(*,*)`, `sd_sca(*,*)`, `sd_min(*,*)`, `sd_max(*,*)`
: differential state start values, scale factors, and bounds (`DVec`)

[`sd_fix(*,*)`]
: differential state fixed value flags (`LVec`) (if `sd_fix(arg1,arg2)[i]` is `1`, then the corresponding differential state is fixed at its start value by internally setting the lower and upper bounds to `sd(arg1,arg2)[i]`)
default: no differential states fixed

`sa(*,*)`, `sa_sca(*,*)`, `sa_min(*,*)`, `sa_max(*,*)`
: algebraic state start values, scale factors, and bounds (`DVec`)

[`u_type(*)`]
: control parametrization types (`LVec`)

`0` :	piecewise constant
`1` :	piecewise linear (continuous on model stages only, not between)
`2` :	piecewise continuous linear with matching across model stage boundaries (“external” matching)
`3` :	piecewise cubic (continuously differentiable on model stages)
`4` :	piecewise cubic with matching across model stage boundaries (“external” matching)

default: all controls piecewise constant

[`u_midx(*)`]
: “external” matching indices for controls (`LVec`) (relevant only at model stage boundaries and if `u_type` is `2` or `4`)
default: matching of controls with same index

`u(*,*)`, `u_sca(*,*)`, `u_min(*,*)`, `u_max(*,*)`
: control start values, scale factors, and bounds (`DVec`)

[`u_fix(*,*)`]
: control fixed value flags (`LVec`) (if `u_fix(arg1,arg2)[i]` is `1`, then the corresponding control is fixed at its start value by internally setting the lower and upper bounds to `u(arg1,arg2)[i]`)
default: no controls fixed

[`ue(*)`], [`ue_sca(*)`], [`ue_min(*)`], [`ue_max(*)`]
: “end of model stage” control start values, scale factors, and bounds (`DVec`) (relevant only if `u_type` is `1` or `3`)
default: same values as at previous multiple shooting point

[`udot(*,*)`], [`udot_sca(*,*)`], [`udot_min(*,*)`], [`udot_max(*,*)`]

control slope start values, scale factors, and bounds (`DVec`) (relevant only if `u_type` is `3` and `4`)
defaults: all elements `udot(*,*)`, `udot_sca(*,*)`, `udot_min(*,*)`, and `udot_max(*,*)` set to values `0.0`, `1.0`, `-100.0`, and `100.0`, respectively

[`uedot(*)`], [`uedot_sca(*)`], [`uedot_min(*)`], [`uedot_max(*)`]
"end of model stage" control slope start values, scale factors, and bounds (`DVec`) (relevant only if `u_type` is `3`)
default: same values as at previous multiple shooting point

`pr(*,*)`, `pr_sca(*,*)`, `pr_min(*,*)`, `pr_max(*,*)`
local interior point constraint parameter start values, scale factors, and bounds (`DVec`)

[`pr_fix(*,*)`]
local i.p.c. parameter fixed value flags (`LVec`) (if `pr_fix(*,*)[i]` is `1`, parameter `pr(*,*)[i]` is fixed at its start value)
default: no local i.p.c. parameters fixed

`g_sca(*,*)`
algebraic right-hand side scale factors (`DVec`)

`rd_sca(*,*)`
decoupled i.p.c. scale factors (`DVec`)

`rc_sca`
coupled i.p.c. scale factors (`DVec`)

`of_sca`, `of_min`, `of_max`
objective scale and expected range (`double`)

7.2.1 Keywords for Robust Optimal Control

Some additional keywords allow to automatically create a robustified version of an optimal control problem. For theory and algorithms, and to understand the implications of the approximate linearization robustification approach used in MUSCOD-II, please refer to [DBK06].

`rob_sd0`, `rob_p`
Select the initial values and model parameters that are considered uncertain. Set a 1 for each uncertain value, a 0 for each conventional value.

`rob_may(*)`
Selects the Mayer type objective functions (one per model stage) to be robustified against uncertainty of the selected initial values and/or parameters. Set a 1 if the Mayer type objective on the selected stage is to be robustified, a 0 if not. Default: 0.

`rob_rd(*,*)`

Selects the decoupled point constraints to be robustified against uncertainty of the selected initial values and/or parameters. Set a 1 for each constraint on the selected stage and node that is to be robustified. Default: all 0.

`rob_cov_sd0`, `rob_cov_sd0_p`, `rob_cov_p`
Sets the covariance submatrices $\Sigma_{x0,x_0} \in \mathbb{R}^{n_x \times n_x}$, $\Sigma_{x0,p} \in \mathbb{R}^{n_x \times n_p}$, and $\Sigma_{p,p} \in \mathbb{R}^{n_p,n_p}$ that form the overall covariance matrix

$$\Sigma = \begin{pmatrix} \Sigma_{x_0,x_0} & \Sigma_{x_0,p} \\ \Sigma_{x_0,p}^T & \Sigma_{p,p} \end{pmatrix}.$$

Default: $\Sigma = Id$, i.e., $\Sigma_{x_0,x_0} = Id$, $\Sigma_{p,p} = Id$, $\Sigma_{x_0,p} = 0$.

`rob_gamma`
Set the overal scale factor for the penalties on objective and residuals computed from the linearizations (`double`). Defaults to 1.0

7.3 Keywords Selecting Dynamically Loadable Modules

The following keywords define the algorithm used to solve the solve the optimal control problem. For the names and a description of all available modules, refer to section 8. All settings may be overridden on the command line, see section 7.7

`libmodel` Selects the model library compiled from the model source file. Defaults to the name of the DAT file.

`libhessian` Selects the hessian algorithm module. Defaults to `hess_update`, the BFGS update with Powell modification.

`libsolve` Selects the globalization strategy module for the SQP algorothm. Defaults to `solve_slse`, the standard line search.

`libcond` Selects the condensing algorithm module for the condensing of the block sparse QP. Defaults to `cond_std` and should not normally need to be changed.

`libtchk` Selects the termination check module for the SQP algorithm. Defaults to `tchk` using the KKT-tolerance as termination criterion.

`libmssqp` Selects the multiple shooting SQP algorithm. Defaults to `mssqp_standard`.

`libeval` Selects the EVAL module variant. Defaults to `eval_ind`.

`libind` Selects the ODE/DAE solver module to use per stage of the multi stage optimal control problem. All stages default to `ind_rkf45`, the 4th/5th order Runge–Kutta–Fehlberg solver. You will want to set this to `ind_daesol` for DAE and stiff ODE problems.

Unlike all other library selection options, this option takes a vector of strings as a value. You need to specify an IND module per stage, like in the following example:

```
0: ind_daesol        ! DAE or stiff ODE on first stage
1: ind_strans        ! transition stage
2: ind_rkf45         ! non-stiff ODE on second stage
```

`libqps` Selects the QP solver module to use for the solution of the condensed QP. Defaults to `qps_qpopt`, the C-converted solver QPOPT.

`libplot` Selects the visualization module. Defaults to `plot_pgplot`. You'll want to set this to `plot_noplot` if you take timings.

`libmintoc` Selects the mixed-integer optimal control module. Defaults to `mintoc` and does not need to be changed.

7.4 Keywords Setting Algorithmic Options

The following keywords set algoritmic options. All settings have default values and may also be overridden on the command line, see section 7.7. There is no need to specify any of the keywords of this section in the DAT file. It is however good practice to do so, because it's a standard way to document the ideal algorithmic options for the efficient solution of the problem you're working on.

`options_acc` Sets the termination criterion's acceptable KKT tolerance *kktacc* of the solution. The default value is 10^{-6}. The value can be overridden on the command line using the `-a` option.

`options_ftol` Sets the final integration tolerance *ftol*. Should be lower than the termination accuracy *kktacc* to ensure validity of the termination criterion. By default, its value is *acc*/10. The value can be overridden on the command line using the `-t` option.

`options_itol` Sets the initial integration tolerance *itol*. By default, the value is *ftol*. The value can be overridden on the command line using the `-h` option.

`options_rfac` Regularization factor *rfac* to make some non unique problems solvable. If the original problem has a flat minimum this option may be useful, as it adds a tiny quadratic term of all problem variables to the objective function to define the minimum uniquely; the weighting factor is measured as a multiple of *acc*: $1/2\ rfac\ acc\ PVars^2$. Defaults to 0.0 The value can be overridden on the command line using the `-r` option.

`options_levmar` Sets the Levenberg–Maquardt regularization factor λ for the Hessian. Use this to add $\lambda \cdot Id$ onto a Gauß-Newton approximation of the Hessian matrix. Defaults to 0.0. The value can be overridden on the command line using the `-l` option.

`options_qp_featol` Sets the feasibility tolerance of the QP solver QPOPT. Defaults to 10^{-8}. The value can be overridden on the command line using the `--qp-featol` option.

`options_qp_relax` Sets the constraint relaxation factor for infeasible QPs. When an infeasible QP is detected, all infeasible constraints are shifted by *rel* times the amount of infeasibility, to make the next QP guaranteedly feasible. Defaults to 1.1. Useful values are greater than 1.0. The value can be overridden on the command line using the `--qp-relax` option.

`options_nhtopy` Allows to employ a homotopy strategy: beginning with a lower initial integration tolerance *itol* (and thus faster SQP iterations) set via `options_itol`, the current integration tolerance *ctol* is tightened until – after *nhtopy* steps – *ftol* (set via `options_ftol`) is attained. The steps in *ctol* interpolate logarithmically between *itol* and *ftol*. A homotopy step is performed during the SQP iterations whenever the current accuracy *cacc* is attained. It is determined as $cacc = ctol \cdot acc/ftol$. By default, no homotopy is employed. The value can be overridden on the command line using the `-h` option.

`options_frstart` **and** `options_frmax` Allows to freeze the integrator discretization after *frstart* SQP iterations to possibly enable better convergence to the solution. The value can be overridden on the command line using the `-f` option.

If the termination criterion is *not* satisfied after the *mf* following steps, the integrator is once again given full freedom to adapt the discretization, to be maintained for the following *mf* SQP iterations, etc. Currently works only for RKF integrators. Default: no freezing (*sf*=0, *mf*=0). Syntax: `-f`*sf*,*mf* . Example: `-f10,5`.

`options_bflag` Sets the MS-MINTOC strategy code, see section 11. The value can be overridden on the command line using the `-b` option.

`options_cflag` Cold start after a previous run of the problem. Uses only the attained state and control variable values. Note that it is possible to perform small changes in the DAT file between restarts, e.g. concerning bounds or output specifications. Dimensional changes are *not* allowed, however. Default: no cold start. The value can be overridden on the command line using the `-c` option.

`options_itmax` Sets the maximum number of SQP iterations. The default is 100. The value can be overridden on the command line using the `-i` option.

`options_qp_expand` Sets the expansion factor of the EXPAND strategy of the QP solver QPOPT. EXPAND is a strategy to avoid cycling in the active set. Useful values range from about 5 to 100, values greater than 9999999 by design of QPOPT disable the EXPAND strategy. This option only has an effect if the QP solver module `qps_qpopt` is loaded.

`options_qp_itmax` Sets the maximum number of iteration of the QP solver per SQP iteration. Defaults to 10000. The value can be overridden on the command line using the `--qp-itmax` option.

`options_sflag` Stop after each SQP iteration and wait for a keystroke. The value can be overridden on the command line using the `-s` option.

`options_wflag` Warm start after previous run of same problem. Warm start uses *all* information from the previous `./RES/*.bin`-file, including the Hessian approximation. Note that it is possible to perform small changes in the DAT file between restarts, e.g. concerning bounds or output specifications. Dimensional changes are *not* allowed, however. Default: no warm start. The value can be overridden on the command line using the `-w` option.

`options_iest_hess_plitt` Another choice to be taken by the user is the initial scaling of the Hessian matrix: either according to Plitt by setting the value to true, or as a unit matrix by setting it to false. Default: `true`.

7.5 Keywords Setting Output Options

Keywords from this section control how much information is written to the console and to text files in the `./RES/` directory during the solution process. There is no need to specify any of the keywords of this section in the DAT file if you don't want to.

`options_plevel_screen` Sets the print level for printout to the screen or console. Defaults to 0.

`options_plevel_file` Sets the print level for printout to the text files. Defaults to 1. The value can be overridden on the command line using the `-p` option.

`options_plevel_matlab` Sets the print level for printout to the MATLAB log file. Defaults to 0. The value can be overridden on the command line using the `--pmatlab` option.

7.6 Keywords for Visualization

The following keywords influence the visualization of the optimal control problem's solution and of the solution progress. There is no need to specify any of the keywords of this section in the DAT file if you don't want to.

`options_output_ps` Enables or disables the output of PGPLOT graphics to PostScript files.

`options_output_gif` Enables or disables the output of PGPLOT graphics to GIF files.

[`nhist`]
number of values in objective/parameter history plots (`long`)
default: `0` (i.e., no objective/parameter history plots)

[`plot_first`], [`plot_last`]
index of first and last model stage to be visualized (`long`)
defaults: `0` and number of model stages minus one (i.e., all model stages are visualized)

[`xd_name`], [`xa_name`], [`u_name`], [`h_name`], [`p_name`]
state, control, duration, and parameter name strings (`StrVec`)
note: if the first nonspace character of the string is `!`, the corresponding variable will not be plotted, if it is `>`, a new graphics window will be opened; a leading `#` switches to logarithmic plotting.
Greek letters can be obtained with the prefix \g, e.g., ϵ = \ge. Sub- and upperscripts can

be obtained by switching the mode with \d and \u, e.g., A^0 = A\u0\d.
If Matlab online graphics are used, the name strings should be in Matlab syntax
defaults: `Differential State Function`, `Algebraic State Function`, `Control Function`, `Model Stage Duration`, `Global Model Parameter`

[`of_name`]
objective name string (`Str`)
note: if Matlab online graphics are used, the name strings should be in Matlab syntax
default: `Objective`

[`xd_unit`], [`xa_unit`], [`u_unit`], [`h_unit`], [`p_unit`]
state, control, duration, and parameter unit strings (`StrVec`)
note: if Matlab online graphics are used the unit strings should be in Matlab syntax
default: empty string

[`of_unit`]
objective unit string (`Str`)
note: if Matlab online graphics are used, the unit strings should be in Matlab syntax
default: empty string

[`t_unit`]
time unit string (`Str`)
note: if Matlab online graphics are used, the unit strings should be in Matlab syntax
default: empty string

7.7 The Command Line

After compilation and linking, the executable (e.g. target `muscod`) can be called with a number of options. The syntax is as follows:

```
muscod [-switch[=value] ... | --option[=value] ... ] problem
```

Square backets indicate optional parts of the command line. Any number of switches (named with single characters) and options (long names) may be specified. Some of them may take values, which may be appended immediately or after an assignment sign ('`=`'). Only the last argument `problem` is mandatory as it determines the files to be used:

- `./DAT/<problem>.dat` is the DAT file, see section 7.1.
- `./RES/<problem>.txt` output files for detailed solution information.
- `./RES/<problem>.log` chronological history of solution process.
- `./RES/<problem>.bin` binary solution information for warm starts, potentially incompatible across machines.

The list of possible options depends on the global compilation flags (cf. appendix 4.2.2). When calling the executable `muscod` without any argument a brief list of possible options is shown. The command line option `--help` prints a detailed list of switches and options, together with a quick reference and the default values.

Furthermore, option values specified on the command line have precedence over those set in the DAT file.

`-a`***kktacc*** Sets the termination criterion's acceptable KKT tolerance *kktacc* of the solution. Example: `-a1e-6` (which gives the default $acc = 10^{-6}$).

`-b`***code*** Sets the MS-MINTOC strategy code, see 11.

`-c`, `-w` Warm/Cold start after previous run of same problem. Warm start uses *all* information from the previous `./RES/*.bin`-file, including the Hessian approximation. The cold start uses only the attained variable values. Note that it is possible to perform small changes in the DAT file between restarts, e.g. concerning bounds or output specifications. Dimensional changes are *not* allowed, however. Example: either `-w` or `-c`.

`-e` "Evaluate twice". Use of two gradient evaluations per SQP iteration to better approximate the partially reduced Hessian. There exists a proof for asymptotically superlinear convergence [Sch96, Sch98]. Example: `-e`.

This option only appears if `PRSQP` is employed.

`-f`***sf,mf*** Allows to freeze the integrator discretization after *sf* SQP iterations to possibly enable better convergence to the solution. If the termination criterion is *not* satisfied after the *mf* following steps, the integrator is once again given full freedom to adapt the discretization, to be maintained for the following *mf* SQP iterations, etc. Currently works only for RKF integrators. Default: no freezing (*sf*=0, *mf*=0). Syntax: `-f`*sf*,*mf* . Example: `-f10,5`.

`-h`***itol,N*** Allows to employ a homotopy strategy: beginning with a lower initial integration tolerance *itol* (and thus faster SQP iterations), the current integration tolerance *ctol* is tightened until – after N steps – *ftol* is attained. The steps in *ctol* interpolate logarithmically between *itol* and *ftol*. A homotopy step is performed during the SQP iterations whenever the current accuracy *cacc* is attained. It is determined as $cacc = ctol \cdot acc/ftol$. By default, no homotopy is employed. Example: `-h1e-3,1`.

`-i`***itmax*** Maximum number of SQP iterations. Example: `-i100` (default).

`-j`***cores*** Sets the number of CPU cores on multicore machines. Currently without effect.

`-l`***lmreg*** Sets the Levenberg–Maquardt regularization factor λ for the Hessian. Use this to add $\lambda \cdot Id$ onto a Gauß-Newton approximation of the Hessian matrix.

`-p`***level*** Print level for `./RES/*.log`-file, ranging from 0 to 3. A print level of 0 corresponds to printing only the visible standard output into the log file whereas 3 is the maximum output. If Matlab online graphics are used, the Matlab output buffer is printed into the log file in case print level is greater or equal to 1.
Example: `-p0` (default).

`-r`***rfac*** Regularization factor *rfac* to make some non unique problems solvable. If the original problem has a flat minimum this option may be useful, as it adds a tiny quadratic term of all problem variables to the objective function to define the minimum uniquely; the weighting factor is measured as a multiple of *acc*: $1/2\ rfac\ acc\ PVars^2$ Example: `-r1.5` (default: 0.0).

This option appears only if the global compilation flag `REGOBJ` is set in the `MC2/makefile`.

`-s` Stop after each SQP iteration and wait for a keystroke. Example: `-s`.

`-t`***ftol*** Sets the final integration tolerance *ftol*. Should be lower than the termination accuracy *kktacc* to ensure validity of the termination criterion. By default, its value is *acc*/10. Example: `-t1e-7`.

`--gif` Enables output of the graphical visualization to GIF files in the `./RES/` directory. This is only possible if the `plot_pgplot` visualization module is loaded.

`--help` Prints an extended list of all command line options, their effect, and their default value.

`--libcond`***name*** Override the IND module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libdir`***path*** Override the search path for dynamically loadable modules. Defaults to the CMake installation directory.

`--libeval`***name*** Override the EVAL module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libhessian`***name*** Override the hessian module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libind`***name*** Override the IND module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libmssqp`***name*** Override the MSSQP module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libplot`***name*** Override the visualization module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libqps`***name*** Override the QP solver module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libsolve`***name*** Override the globalization strategy module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--libtchk`***name*** Override the termination check module specified in the DAT file. For a list of possible values of *name*, refer to section 8.

`--plotadjoints`***name*** Opens a new window in which the multipliers for the continuity conditions of the differential equations at the multiple shooting nodes are plotted. Note that these values can be seen as an approximation to the adjoint variables $\lambda(t)$, however, they are only defined and calculated on the MS nodes.

`--pmatlab`***level*** Sets the MATLAB logging level. Possible values range from 0 to 3 and increase the amount of data written to the MATLAB log file.

This option only has an effect if the MATLAB header files and libraries were found at compile time (when the bootstrap script was last run).

`--ps` Enables output of the graphical visualization to PostScript (PS) files in the `./RES/` directory. This is only possible if the `plot_pgplot` visualization module is loaded.

`--qp-expand`***N*** Sets the expansion factor of the EXPAND strategy of the QP solver QPOPT. EXPAND is a strategy to avoid cycling in the active set. Useful values range from about 5 to 100, values greater than 9999999 by design of QPOPT disable the EXPAND strategy. Example: `--qp-expand50` (default).

This option only has an effect if the QP solver module `qps_qpopt` is loaded.

`--qp-featol`***tol*** Sets the feasibility tolerance of the QP solver QPOPT.
Example: `--qp-featol1e-8` (default).

This option only has an effect if the QP solver module `qps_qpopt` is loaded.

`--qp-itmax`***itmax*** Sets the maximum number of QP solver iterations per SQP iteration. Example: `--qp-itmax10000` (default).

`--qp-relax`***rel*** Sets the constraint relaxation factor for infeasible QPs. Useful values are greater than 1.0. When an infeasible QP is detected, all infeasible constraints are shifted by *rel* times the amount of infeasibility, to make the next QP guaranteedly feasible. Example: `--qp-relax1.1` (default).

`--plotadjoints` Opens an additional output window in which the adjoint trajectories of the ODE are drawn.

`--lagmulreg`***reg*** Regularization of the multipliers resulting from solution of the condensed QP. Applies a singular value decomposition to the matrix of active constraints and determines the multipliers with minimum ℓ_2 norm from the set of multipliers satisfying stationarity. This helps for convergence problems that are caused by LICQ violation. The parameter *reg* gives the cutoff threshold for the smallest accepted singular value. Setting *reg* to zero disables this feature (default). Example: `--lagmulreg1e-08`

8 Dynamically Loadable Modules

MUSCOD-II incorporates a variety of different algorithms, as a combination of different dynamically loadable libraries. The choice between these libraries is performed on three levels:

default values that are overridden by settings in the DAT file of the problem, Section 7.1, which in turn can be overridden by command line options. The shared object files (dynamic link libraries) are loaded on startup of MUSCOD-II.

8.1 SQP Algorithms

For overall control of SQP solution process, the following shared objects are provided:

- `mssqp_standard`: The standard multiple shooting SQP algorithm.
- `mssqp_nmpc_gn`: Nonlinear model-predictive control with a Gauß-Newton approximation of the Hessian.
- `mssqp_mhe_gn`: Moving horizon estimation with a Gauß-Newton approximation of the Hessian.
- `mssqp_picard`: A Newton-Picard algorithm. Alpha stage, usage strongly discouraged.

8.2 ODE/DAE Solvers

To control the solution of the ODE/DAE system, the following shared objects are provided. Their individual capabilities are summarized in table 3.

- `ind_daesol`: Our preferred variable order variable step size BDF method for stiff and non-stiff ODE and DAE systems. Use `ind_daesol_spa` for the sparse variante of DAESOL. Use `ind_daesol_prsqp` or `ind_daesol_spa_prsqp` if you use PRSQP.
- `ind_rkf12`: The 1st/2nd order Runge-Kutta-Fehlberg method.
- `ind_rkf23`: The 2nd/3rd order Runge-Kutta-Fehlberg method.
- `ind_rkf45`: Our preferred Runge–Kutta solver, the 4th/5th order Runge-Kutta-Fehlberg method.
- `ind_rkf45adj`: The 4th/5th order Runge-Kutta-Fehlberg method, currently the only solver capable of computing adjoint sensitivities. Use `ind_rkf45adj_spa` for a sparse variant using TD12.
- `ind_rkf78`: The 7th/8th order Runge-Kutta-Fehlberg method.
- `ind_rkf7b`: A 7th/8th order Runge-Kutta method by H.G.Bock.
- `ind_rkfswt`: The 4th/5th order Runge-Kutta-Fehlberg method, currently the only solver capable of treating implicitly discontinuous ODE systems.
- `ind_strans`: Stage transition solver for multistage optimal control problems.
- `ind_nullslv`: Null solver, does nothing. It can be used to define NLP-Problems without any underlying dynamic system.

In addition, the following solvers are available:

- `ind_ddasac`: Another ODE/SAE solver. Use `ind_ddasac_adf` for models with derivative functions generated by ADIFOR. Use `ind_ddasac_prsqp` or `ind_ddasac_adf_prsqp` if you use PRSQP.
- `ind_mbsnat`: The multibody system solver MBSSIM. Usage discourages as MBSSIM is currently unmaintained.
- `ind_metanb`: The METANB solver from PARFIT.

Solver	ODE	Stiff ODE	DAE	CLSQ	SPA	ADJ	ADF	Switches
`ind_rkf12`	●							
`ind_rkf23`	●							
`ind_rkf45`	●							
`ind_rkf78`	●							
`ind_rkf7b`	●							
`ind_rkf45adj`	●			●	●	●		
`ind_rkfswt`	●			●				●
`ind_daesol`	●	●	●	●	●			
`ind_ddasac`	●	●	●	●			●	
`ind_mbsnat`	●	●	?	?				
`ind_metanb`	●	●	●					

Table 3: Available ODE/DAE solvers and their capabilities.

8.3 Computation of the Hessian

For the computation or approximation of the Hessian, the following shared objects are provided:

- `hess_const`: Constant Hessian matrix for simplified Newton method. Good for refinement of almost converged solutions.
- `hess_finitediff`: Calculation of Hessian approximation from finite differences. Very efficient for systems with few state variables and many multiple shooting intervals. Needs module `solve_tbox` to cope with possibly non-positive-definite Hessian approximations.
- `hess_gaussnewton`: Hessian approximation for least-squares objective functionals. Very efficient for problems with a solution with small objective value.
- `hess_update`: variable metric Hessian approximation. By default, BFGS updates with Powell modification are selected.

  For expert users: *By manually changing the source code in* ***`hess_update.cpp`****, the following alternatives may be realized:*

  – *Update formula:*

    * *BFGS updates by defining `BFGS_UPDATE`.*
    * *DFP updates by defining `PSB_UPDATE`.*
    * *PSB updates by defining `PSB_UPDATE`.*
    * *SR1 updates by defining `SR1_UPDATE`.*
  - *For the BFGS update there exist modifications to keep the Hessian matrices positive definite which may be selected by manually changing the source code in `hess_update.cpp`:*
    * *Powell modification of the BFGS update by defining `MOD_BFGS_POWELL`.*
    * *Swaney modification of the BFGS update by defining `MOD_BFGS_SWANEY`.*
    * *Nocedal modification of the BFGS update by defining `MOD_BFGS_NOCEDAL`.*

- `hess_limitedmemoryupdate`: Limited memory version of `hess_update`. Only BFGS updates are realized for this module.

  For expert users: *You can choose the update modification, and the initial scaling of the Hessian matrix by modifying the file `hess_limitedmemoryupdate.cpp` in the same way as described for the module `hess_update`. Here you have the additional option whether the number of update vectors is set adaptively via the flag `VARIABLE_MEMORY`, or if a fixed maximum number is used. The macros `HAP_LLIMIT` and `HAP_ULIMIT` provide lower and upper limits.*

8.4 Globalization Strategy

For the calculation of SQP correction step, the following shared objects are provided:

- `solve_alf` for the Schittkowski augmented line search.
- `solve_fullstep` to disable globalization and always perform the full SQP step.
- `solve_lsq` for a line search via natural level functions, suited for least-squares problems.
- `solve_slse` for a standard line search. For NMPC, the variants `solve_slse_nmpc` and `solve_slse_fullstep_nmpc` exists. In contrast to the standard line search, they both also includes simple bounds in the line search. The fullstep variant always attempts to do a full step first.
- `solve_tbox` for a boxstep trust region technique. This variant is especially required if you work with Hessians that are not guaranteed to be positive definite (SR1 or PSB updates, BFGS updates without modification, exact Hessians by finite differences).
- `solve_vmcwd` for a line search with a watchdog technique.
- `solve_vmcwd_nmpc` for a line search with a watchdog technique. For NMPC, the variant `solve_vmcwd_nmpc` exists that also includes simple bounds in the line search.
- `solve_wdog` A watchdog technique using Powell's penalty function.
- **For expert users:** *By manually changing the source code, the following alternatives may be realized:*

  - *In the slse and tbox algorithms you have the choice of using a second order correction step via the flag* ***SOC_STEP*** *to be set in* ***`solve_slse_common.cpp`*** *or* ***`solve_tbox.cpp`****.*
  - *The flag* `LARGER_ALPHA` *can be used in all modules to enforce the calculation of a maximum line search stepwidth.*
  - *The flag* `UPHILL_MOD` *sets an optional uphill modification that introduces a new weighting parameter for the objective function in the line search function to enforce descent of the line search function.*
  - *The value* `MAX_ALPHA` *holds the maximum step length and defaults to* 1.0*.*

8.5 Condensing of the Block Sparse QP

For the condensing of the block sparse multiple shooting QP, the following shared objects are provided:

- `cond_std`: The standard condensing algorithm reduces the large amount of independent variables of the multiple shooting method by exploiting the linearized continuity conditions. Only the *condensed* quadratic program is then solved by a dense QP solver.

- `cond_min`: The minimal condensing algorithm is normally not used. It only cares about the control discretization and a separabile formulation for the global unknowns. It does not eliminate state variables using the linearized continuity conditions. This variant may be useful in conjunction with a sparse QP solver (such as OOQP) which is able to exploit the sparsity in the uncondensed QP. For dense QP solvers (the majority of supported QP codes), using this variant will result in inferior performance.

- **For expert users:** *By manually changing the source code, the following alternatives may be realized:*

  - *You have the option to truncate the step to avoid violation of bounds via the flag* `TRUNC_STP` *(inactive by default). This may be advantageous if no feasible QP solution was available (only a* relaxed *one), and bounds should be given priority over linearized continuity conditions.*
  - *Not all state bounds are given to the lower level QP solver: only* potentially active *ones, i.e. those that have previously been active are condensed and passed on. By inactivating the flags* `SD_BOUNDS` *and* `SA_BOUNDS` *(default: active) for differential and algebraic states the corresponding bounds are neglected and not even checked.*
  - *The flag* `RECALC_QP` *(default: active) enforces a recalculation the QP problem in the same SQP iteration, if the potentially active bounds have changed.*

8.6 Solution of the Condensed QP

For the solution of the condensed QP, the following shared objects are provided:

- `qps_qpopt` is an interface to our in–house C-converted version of the QP solver QPOPT 1.0-10 by Gill, Murray and Saunders. This is by far the most important QP solver available. The solver itself is included in `Packages/QPOPT` and is licensed to IWR for academic purposes.

- `qps_ooqp` is an interface to the open–source sparse interior–point QP solver OOQP, available from the COIN-OR project web site. It is up to now the only publicly available sparse QP solver interfaced with MUSCOD-II and should be used in conjunction with the minimal condensing module `cond_min`.

  Using this solver inside an SQP algorithm is usually ineffective. It has however been successfully used for highly sensitive multibody ODE systems in robotics for which condensing leads to ill-conditioned QPs.

- `qps_bqpd` is an interface to the QP solver BQPD by R. Fletcher. The solver itself is not included and has to be separately licensed and obtained from the University of Dundee.

- `qps_old_qpopt` is an interface to the QP solver QPOPT (Fortran version 1.0-9), part of the NAG library. The interface is deprecated, it is recommended to use the `qps_qpopt` instead.

- `qps_old_qpsol` is an interface to the QP solver QPSOL (now superseeded by QPOPT), part of the NAG library. The interface is deprecated, it is recommended to use the `qps_qpopt` instead.

- `qps_old_lssol` is an interface to the linear least-squares solver LSSOL, part of the NAG library. Using this solver is recommeded for QPs which have only few active constraints in the solution. The interface is experimental.

- The solvers `ve02` and `ve17` available in older releases of MUSCOD-II have been removed as these codes are nowadays considered obsolete.

- **For expert users:** *By manually changing the source code, the following alternatives may be realized:*

  - *The active set is stabilized by setting the flag* `ACTSTAB` *(default: active) that introduces a relaxed tolerance for previously inactive constraints.*
  - *The QPOPT solver writes a file* `qpopt.txt` *providing detailed information about the QP solution process if you set* `msglvl` *to one of the values* 5*,* 10*,* 20*, or* 30 *in the file* `qps_qpopt_new.cpp`*.*
  - *The options* `BIGBND` *and* `BIGSTP` *define the magnitudes of bounds or steps that are considered infinite. The default values are* 10^6*, and it may be necessary to rise them to* 10^{16} *for ill-conditioned QPs.*

8.7 Termination Check for the SQP Algorithms

For the termination check of the SQP algorithm, there are two possible termination criteria, namely

- `tchk` using the KKT-Tolerance and
- `tchk_loccont` which is based on the local contraction theorem. You might need the `-lagmulreg` option for a regularization of the Lagrange Multipliers.

For a detailed description of both see: [Sch09].

8.8 Graphical Visualization of the Results

Graphical visualization of the convergence process and the optimal control problem's solution is realized by using one of the following shared objects:

- `plot_pgplot`: Visualization using the PGPLOT library.
- `plot_matlab`: Visualization using MATLAB. This is variant is currently not maintained.
- `plot_noplot`: No visualization. This is especially useful if you're going for minimum computation times (many runs, publication, etc.)

8.9 Restrictions in the Choice of Modules

Some combinations of algorithms are not sane from a mathematical point of view, or are unsupported in the current implementation. The resulting restrictions are listed in this section.

- With the SR1 and PSB update, and with the unmodified BFGS update, the hessian matrices do not need to be positive definite, so that you have to use a trust region strategy.

  $\longrightarrow$ Use `solve_tbox` if you use `hess_update` with internally enabled SR1 or PSB update, or BFGS update without a modification guaranteeing positive definiteness.

- The exact hessian computed using finite differences does not need to be positive definite, so that you have to use a trust region strategy.

  $\longrightarrow$ Use `solve_tbox` if you use `hess_finitediff`.

- If your model uses a continuous least-squares objective (CLSQ), you must use an ODE/-DAE solver that is capable of evaluating it.

  $\longrightarrow$ Use `ind_daesol`, `ind_rkf45adj`, or `ind_rkf45swt` if you use CLSQ.

- If you use the sparse interior point QP solver OOQP, performance is usually improved by leaving out condensing.

  $\longrightarrow$ Try using `cond_min` if you use `qps_ooqp`.

9 MUSCOD-II output

MUSCOD-II provides information on algorithmic settings, on convergence behavior, and the results on different levels, described in the following subsections. An additional possibility to obtain information at runtime is to use the interactive version of MUSCOD-II, see Section 10.

9.1 Terminal Output

The terminal output of MUSCOD-II will start with some general information on the chosen algorithmic settings,the current version of the software, and problem dimensions. Then SQP iterations are described, e.g.,

```
>>>>  SQP iterations

it qp    qptol aset   kkttol      sobj   |sinf|  |vstep| |vstpr| |mulstp|       |lgrd| alpha    merit crit stat
 0                            9.402310 1.48E-05
 1 42 1.48E-07  NEW 8.78E-02  9.314528 2.49E-04 2.96E-01 0.00E+0 1.18E+01 1.000000E+00  1.00 9.315597  REL SUCC
 2  5 2.49E-06  NEW 9.08E-01  8.406353 2.53E-02 3.03E+00 0.00E+0 2.99E+00 3.010900E-01  1.00 8.521348  STD SUCC
 3 33 1.00E-04  NEW 1.33E+00  7.078570 5.95E-02 4.08E+00 0.00E+0 1.47E+01 3.589541E-01  1.00 7.507059  STD SUCC
...
```

The explanation of the columns is given in Table 4.

column	**description**
`it`	Number of current SQP iteration
`qp`	Number of QP iterations needed
`qptol`	Tolerance given to the QP solver
`aset`	Has the active set changed compared to last SQP iteration?
`kkttol`	Karush-Kuhn-Tucker Tolerance used for termination check
`sobj`	Objective function value
`|sinf|`	Norm of infeasibility
`|vstep|`	Norm of variable step
`|vstpr|`	Norm of step in the primal variables
`|mulstp|`	Norm of step in the dual variables
`|lgrd|`	Norm of Lagrange gradient
`alpha`	Step length deduced by line search
`trad` `tact`	Trust region radius Is trust region bound active?
`merit`	Value of merit function
`crit`	Relaxation of constraints? (`STD` or `REL`)
`stat`	Status of iteration (`SUCC`, `PEND`, or `BACK`)

Table 4: Columns in the terminal output of MUSCOD-II

Finally, statistics on function evaluations and CPU time usage are displayed.

9.2 Online Graphics

MUSCOD-II uses a powerful `PLOT` module that contains interfaces to different online graphic software. Currently these are `PGPLOT` and `MATLAB`, a `CairoGraphics` implementation is envisaged. The plotting of differential and algebraic states, of control functions, and of parameters and objective function histories are controlled by setting appropriate names in the dat file, Section 7.1.

The PGPLOT visualization module `plot_pgplot` reads several options from the file `default.plot` as detailed in table 5

Options	Possible Values	Description
`screen_enabled`	on, off	Switch on-screen visualization on or off.
`screen_width`	`double`	Screen width in inch (pixels / dpi). Recommended values are 10.6 for 1024 pixels, 13.3 for 1280 pixels, 16.6 for 1600 pixels.
`screen_aspect`	`double`	Screen's aspect ratio (height / width). Recommended values are 0.75 for a 4:3 screen, 0.625 for a 16:10 widescreen.
`screen_hostname`	`string`	PGPLOT X server host name, e.g., `jim.iwr.uni-heidelberg.de`.
`screen_display`	`int`	X server display number, usually 0.
`screen_screen`	`int`	X server screen number, usually 0.
`ps_enabled`	on, off	Switch generation of PostScript graphics on or off.
`ps_append_file`	on, off	Whether to write all iterations' figures into the same PostScript file on multiple pages
`ps_width`	`double`	PostScript paper width in inches. Recommended values are 7.5 for portrait, 10.6 for landscape
`ps_aspect`	`double`	PostScript paper aspect ratio. Recommended values are 1.4142 for portrait, 0.7071 for landscape
`ps_color`	on, off	Whether to generate a color or a grayscale PostScript file
`gif_enabled`	on, off	Switch generation of GIF graphics on or off.
`gif_width`	`double`	GIF file width in inches
`gif_aspect`	`double`	GIF file aspect ratio

Table 5: Options for the PGPLOT visualization module `plot_pgplot`.

An even more flexible way to plot any function of states and controls, possibly also in a time-dependent movie-like manner, is given by a problem-dependent usage of the `PLOT` module. We refer here to examples that serve best to illustrate the broad applicability.

- `MIP/lotkaindirekt`: additional plotting of a switching function and an implicitely determined control

- `MIP/robotpath`: 2d-spatial and time dependent visualization of robot positions with underlying prescribed trajectories, output to movie
- `MIP/smb_super`: 1d-spatial and time dependent visualization of concentration profiles with text and numbers, output to movie
- `MIP/urethan` and `MIP/vpbimolcat`: plotting of optimum experimental design sampling decisions on a discrete time grid (transition stages only)
- `MIP/oberle`: zoom into small region in time of interest
- `TEST/chain1d`: 1d-spatial and time dependent visualization of controlled chain of masses
- `TEST/parest`: Visualization of fitting a model to measurement data

9.3 Result Files

By default, several files will be written to the `RES` subdirectory.

- `./RES/name.txt`: detailed solution information
- `./RES/name.log`: chronological history of solution process
- `./RES/name.bin`: binary solution information for warm starts
- `./RES/name.ps`: if switched on, a postscript (or gif) file of the online graphics

Furthermore, by using any `-b` option, compare Section 11, for example `-b0`, a new dat-file will be written with the result of the optimization as starting point. This file is generically named `./DAT/name_opt.dat`. Note that also several `_opt` may be concatenated. This file may then in turn be used for restarts, possibly with slightly modified bounds (e.g., in homotopies), for a repetition of plotting the optimal solution in a time efficient manner, and the like.

9.4 Matlab Logging

MUSCOD-II can export internal data to Matlab. This includes not only the results of the computations but also most of the intermediate steps for each iteration of the full computation. The main purpose of the Matlab logging capabilities is to exploit the flexibility of Matlab for investigating bad convergence behavior in the Multiple Shooting SQP method. The full range of numerical tools can be exploited, e.g., to calculate projected Hessians and their eigenvalues, which can lead to insight about violated assumptions like LICQ or positive definiteness of the projected Hessian. This may give rise to clues for a reformulation of the underlying optimization problem. Additionally, the user may also find the flexibility of Matlab's visualization tools helpful. However, MUSCOD-II can slow down considerably when large amounts of data have to be logged which is why Matlab logging is disabled by default in the MUSCOD-II suite.

Reconfiguring the suite. We assume that you have checked out a fresh suite and have it configured and compiled with `bootstrap`. The procedure to enable Matlab logging is the following:

- Locate the Matlab installation on your file system. We assume here that Matlab has been installed to `/usr/local/matlab`.
- Reconfigure the COMMON_CODE package in the suite by calling the CMake-GUI, e.g., with

  ```
  user@machine:~/MUSCOD_SUITE/Packages/COMMON_CODE/Debug> ccmake .
  ```

  Enable the options `PROVIDE_LOGGING_TO_MATLAB` and `PROVIDE_TYPE_CONVERSION_MATLAB`. Press "c" to configure the package. CMake will try to find your Matlab installation. Sometimes, you have to help manually by setting the advanced variable `MATLAB_DIR` to your Matlab installation path. Even that may not be enough: It is a known problem of CMake (at least for version 2.6 and earlier) that libraries ending in a version number (e.g., `.so.1.0`, instead of `.so`) will not be found automatically. In Matlab 7.6, the following needed libraries have that problem:

  ```
  MATLAB_ICUDATA_LIBRARY  /usr/local/matlab/bin/glnxa64/libicudata.so.36
  MATLAB_ICUI18N_LIBRARY  /usr/local/matlab/bin/glnxa64/libicui18n.so.36
  MATLAB_ICUIO_LIBRARY    /usr/local/matlab/bin/glnxa64/libicuio.so.36
  MATLAB_ICUUC_LIBRARY    /usr/local/matlab/bin/glnxa64/libicuuc.so.36
  ```

  You can either create symbolic links, e.g., via

  ```
  user@machine:/usr/local/matlab/bin/glnxa64> ln -s libicuuc.so.36 libicuuc.so
  ```

  which is only possible if you have root access on your machine. Alternatively, you have to type in the proper locations in the CMake-GUI. If you work in a 32bit environment, you will have to exchange `glnxa64` for `glnx86`. Press "c" and "g" to configure and generate the Makefiles. Leave the CMake-GUI.
- Recompile COMMON_CODE by calling

  ```
  user@machine:~/MUSCOD_SUITE/Packages/COMMON_CODE/Debug> make install
  ```

- Call the CMake-GUI in your `MC2` binary directory

  ```
  user@machine:~/MUSCOD_SUITE/MC2/Debug> ccmake .
  ```

- Enable the options `MATLAB_SUPPORT`
- Press "c" and fill in the possibly missing Matlab libraries like above.
- Configure, generate the Makefile, leave the GUI and recompile MUSCOD-II via

  ```
  user@machine:~/MUSCOD_SUITE/MC2/Debug> make install
  ```

- Recompile your application, e.g., by

  ```
  user@machine:~/MUSCOD_SUITE/Apps/TEST/Debug> make
  ```

Using Matlab logging. You can now have MUSCOD-II data logged to a file `RES/problemname.mat` by via the `--pmatlab<n>` option of the MUSCOD-II executable. The print level `<n>` currently knows two thresholds:

`n=2`: Log only primal and dual variables. The Matlab file will contain a single hierarchical cell array called `iterations` which contains a struct for each iteration. The struct consists of the substruct variables `varstep`, `varnew`, and `mulnew`, corresponding to the step in the primal variables, the updated primal variables, and the updated dual variables, respectively. The substructs are structs themselves. E.g., `varnew.sd` is a cell array containing the differential Multiple Shooting states. To examine the differential state at the third shooting node (attention: counted from 1 according to Matlab) at the end of the fifth iteration, one calls

```
>> iterations{5}.newvar.sd{3}
```

in Matlab.

`n=3`: Log most of the intermediate data. The names in the `iterations` cell array will be close to or the same as in the MUSCOD-II source code which shall serve as a reference here. In this print level, there can be a lot of warnings in the output for values which are written several times in one iteration, e.g., when line search damping has to be performed.

10 Interactive MUSCOD-II

MUSCOD-II incorporates an interactive mode that can be initiated by wither the `-s` command line flag, or by pressing simultaneously the `Ctrl` and the `c` key at runtime[9]. The result will be an interactive mode that waits for commands between SQP iterations. The list of possible commands is given in Table 6.

Note that QP data and solution are always unscaled. The order of variables here is:

phf, sd_0, (qls, qc, sa, prf, qle)_j,

where j is the index over all multiple shooting nodes.

[9]note that pressing `Ctrl+c` twice aborts directly without possibility to restart in the current solution point

command	**description**
`h`,`help`	Print list of available commands
`s`,`step`,`[return]`	Calculate next SQP iteration
`c`,`cont`	Continue in noninteractive mode
`q`,`quit`	Quit (may be restarted with -c or -w)
`set`	List all current options

`set` OPTION VAL
`output`
`outputmode`
`scaled, unscaled`
`tol` VAL
	Change option to val:
	screen, log or filename
	col, row, mat, std, rseq or cseq
	Scaled or unscaled output
	Set tolerance for constraint violation

`dd`,`dumpDers`	Dump derivatives to Matlab file
`pd`,`printDers`	Print nonzero objective derivatives
`pcd`,`printConDers`	Print nonzero constraint derivatives
`pf`,`printFuns`	Print objective and constraint function values
`pmc`,`printMatch`	Print matching conditions
`pm`,`printMults`	Print all multipliers
`ps`,`printSens`	Print nonzero Wronskian sensitivities (scaled)
`pv`,`printVars`	Print all variables
`pvc`,`printVConst`	Print all violated constraints
`save`,`saveBin`	Store current variables and Hessian in binary file
`pfp`,`printFixedPars`	Print fixed global parameter values
`eval`,`evaluate`	Reevaluate functions and gradients (necessary after change of parameters)
`sfp`	Set fixed parameter to new value, needs 2 arguments: index and new value
`uif`,`userInteract`	Call def_uif function
`ph`,`printHess`	Print Hessian matrix
`kkt`,`pkkt`	Print details about KKT tolerance
`pqpd`,`printQPData`	Print QP data (hessian, gradient, constraint matrix/vector, lower/upper bound vector)
`pqps`,`printQPSol`	Print QP solution (step, constraint multipliers, bound multipliers)
`dqp`,`dumpQP`	Dump QP data and solution to matlab file

Table 6: Interactive mode commands

11 MS MINTOC

MS MINTOC is an extension of MUSCOD-II written by Sebastian Sager to incorporate integer-valued functions and variables into the problem formulation. Good places to look for a description of theory and algorithms are [Sag09] and [SRB09].

The main difference compared to the MUSCOD-II problem formulation is that in addition to the interior point and path constraints (2.2) integrality conditions on global parameters and/or control functions are imposed.

It is expected that an *outer convexification* [Sag09] is performed by the modeler himself. Hence, all integer controls are expected to be of the form

$$\omega_j(t) \in \{\omega_j^{\min}, \omega_j^{\max}\} \qquad \forall\, t \in [t_i, t_{i+1}] \tag{7}$$

for control function j on model stage i.

11.1 Defining integer variables

The feasible values for binary controls and parameters are given in the usual way as upper and lower bounds. E.g., $\omega_j^{\min}$ is given by the corresponding value of `u_min`. To define whether a control function is an integer, binary or continuous control, the option `u_int` within the dat-file is used. For every control function a natural number x_j has to be specified with the interpretation

- $x_j = 0$: continuous control function, $\omega_j(t) \in [\omega_j^{\min}, \omega_j^{\max}]$
- $x_j = 1$: binary control function, $\omega_j(t) \in \{\omega_j^{\min}, \omega_j^{\max}\}$
- $x_j = -1$: integer control function, $\omega_j(t) \in \{\omega_j^{\min}, \omega_j^{\min} + 1, \dots, \omega_j^{\max} - 1, \omega_j^{\max}\}$
- $x_j > 1$: binary control function, $\omega_j(t) \in \{\omega_j^{\min}, \omega_j^{\max}\}$, with SOS 1 constraint

$$\sum_{k: x_k = x_j > 1} \omega_k(t) = 1 \qquad \forall\, t \in [t_i, t_{i+1}] \tag{8}$$

  that needs to be taken into account when applying, e.g., rounding algorithms. Note that several independent SOS1 constraints can be specified by choosing different values for x_k.

- $x_j < -1$: as in the case $x_j > 1$, but the modeler replaced ω_{n_ω} by $1 - \sum_k^{n_\omega - 1} \omega_k(t)$.

A similar formulation (however model stage independent) is used for global parameters via the flag `p_int`.

11.2 Available algorithms

Table 7 lists all possibilities to select different algorithms to solve a MIOCP. The general concept is a little uncommon: digits correspond to algorithms, and their order determines the order of their execution, from right to left. For example, 5 corresponds to a switching time optimization, and 8 to the default rounding procedure. Hence, method 58 will first solve a relaxed problem (that will always be done), then apply the rounding procedure 8 to the result, and use the rounded solution as starting value and fixed structure for the switching time optimization.[10]

[10]As there are not enough digits for all algorithms, this system is not always consistent, unfortunately. 586 means first 6, then 8, then 5; however, 581 means first 81, then 5.

column	**description**
0	Relaxed solution (writes result into DAT/..._opt.dat)
1	Enumeration
10	Random integer solution
11	Lower bound solution
12	Upper bound solution
13	Internal use (simulation, modify mintoc.cpp)
2	Branch and Bound [default 241]
20	.. without start heuristics
2x	.. with start heuristics x in 8, 81, 82, 83, 84, 85, 41, 42
21	.. with all start heuristics 8, 9, 11, 12, 41
3	Outer Approximation (not yet implemented)
4	Penalty approach [default 41]
41	.. quadratic penalty function
42	.. exponential penalty function
5	Switching time optimization [default 5816]
5x	.. with initialization x in 1, 10, 2, 4x, 6, 64, 8x
6	Adapt control discretization grid `numadaptiters` times
60	Adapt c.d.g. combined with penalty approach
7	Integral Approximation
8	Rounding [default 83]
81	Rounded relaxed solution
82	Sum Up Rounding with specific offset
83	Sum Up Rounding
84	Sequentially relaxed solution on shrinking horizon with rounding on (moving) 1st interval
85	Sequentially relaxed solution on shrinking horizon with enumeration on (moving) 1st interval
86	Sequentially relaxed solution on shrinking horizon with SUR on (moving) 1st interval

Table 7: Possible values for the `-b` flag and the corresponding algorithms of MS MINTOC

11.3 MS MINTOC specific options

There are several options that are only meaningful in the context of MS MINTOC. They are typically specified by means of the dat file, compare Section 7.1. In Table 8 a list of possible options, their meaning, and the default value is given.

key word `mintoc_`	**description and [default]**
Penalty term parameters	
`eps_init`	Initialization of penalty parameter [0.0001]
`eps_step`	Multiplier of penalty parameter [2.5]
`penTolZero`	Penalty tolerance [0.0001]
`penIntTol`	Penalty integer tolerance [1e-08]
`penConTol`	Penalty convergence tolerance [1e-06]
`penNumQPSteps`	Number of QP iterations in Penalty strategy [100]
`penMaxSteps`	Maximum number of iterations in Penalty strategy [50]
`penMaxStuck`	Maximum number of iterations before stuck in Penalty strategy [20]
Adaptparam parameters	
`minstagelength`	Minimum length of stages [1e-05]
`numadaptiters`	Number of successive adaptations [1]
`adaptmode`	Adaptmode: 0 bisection 1 middle peak 2 adaptive [0]
`adaptPenStart`	Start of Penalty after adaptivity [5]
`adaptPenIter`	What was this again [0.5]
General parameters	
`tolZero`	Zero tolerance [1e-06]
`simIndex`	Index of control to be used for simulation (internal) [7]
`maxIterationsForConvergence`	Maximum number of iterations for NO restart in solution [6]
`restart_s_spec`	What `s_spec` for restarts? [2]
`plotAlgorithmicData`	Open additional plot window? [0]
`milp_solver`	Name of callable (by AMPL) MILP solver [cbc]
Rounding parameters	
`roundOffset`	Offset that has been added to `u_max` to avoid cycling [0]
`simulateOnly`	What after rounding? 0 Optimize 1 Simulate only [0]
`roundedEmbedding`	Rounding: 1 fix only bounds, 0 set also variable value [0]
Sequential parameters	
`seqOptFreq`	Frequency of re-optimization on shrinking horizon in mode 86 [1]
Integral Approximation	
`u_switch_max`(imos)	Vector of maximum number of switches of binary control [6]

Table 8: List of MS MINTOC specific options. The key word in the leftmost column is precedented by `mintoc_` in the dat files.

11.4 Switching Time Optimization

The optimization of switching times within MS MINTOC is implemented as a reformulation towards a multiple model stage problem with model stage lengths subject to optimization. Obviously, the model description provided by the `def_model()` routine needs to be adjusted.

The number of model stages will depend on the algorithm that has been applied. To be as flexible as possible and to allow, e.g., both fixed and free end time formulations of the original problem, a user specific formulation in the form `def_model_ct()` is necessary (ct for continuous time). However, MS MINTOC supplies information on the number of introduced model stages at runtime, allowing for usage of `def_model_ct()` both within the first run and from scratch with an automatically written dat file `DAT/name_ct.dat`. The callback function `getNmos (NMOS)` takes the original number `NMOS` as an argument and returns a `LVec` with the dimension `NMOS` and the number of introduced stages per original stage.

The last entry, `V_EACC( newnmos , NMOS )`, contains the overall new number of model stages.

The usage is best exemplified by an easy example, see Section 14.

12 Win XP

12.0.1 Requirements

Administrator Rights

The installation process will require you to hold administrator's rights.

WinZip

In order to uncompress `tar.gz` files you need to install WinZip. Download a free evaluation version of WinZip from `http://www.winzip.com` if necessary. As of writing this document, the file in question is named `winzip100.exe`.

MinGW and MSYS

The GNU toolsets, headers and libraries, as well as a linux-like shell, are freely available from `http://www.mingw.org`. Download from the following files or their appropriate newer releases from the 'Current' section

1. `binutils-2.15.91-20040904-1.tar.gz`
2. `gcc-core-3.4.2-20040916-1.tar.gz`
3. `gcc-g++-3.4.2-20040916-1.tar.gz`
4. `gcc-g77-3.4.2-20040916-1.tar.gz`
5. `gdb-5.2.1-1.exe`
6. `mingw-runtime-3.9.tar.gz`
7. `mingw32-make-3.80.0-3.exe`
8. `w32api-3.5.tar.gz`
9. `tcltk-8.4.1-1.exe`
10. `MSYS-1.0.11-2004.04.30-1.exe` from the 'Snapshot' section.

GrWin

PGPLOT support for Windows, download from

`http://spdg1.sci.shizuoka.ac.jp/grwinlib/english/download.html` the file

1. `lGrWn0999be-MinGW.exe`

or an appropriate newer release.

LAPACK

Download the LAPACK archive from `http://www.netlib.org/lapack/lapack.tgz`.

MUSCOD-II

MUSCOD-II can be found in the workgroup's subversion repository.

Follow the steps precisely as listed below. Especially make sure to

- properly distinguish slashes ('/') from backslashes ('\'),
- perform all the actions using Windows, unless you're told to launch an MSYS shell and do things from there.

12.0.2 Installing WinZip

1. Download and start the WinZip executable and follow the installation process, if you don't already have a working version of WinZip installed. An open–source alternative to WinZip is `http://sourceforge.net/projects/sevenzip/`.
2. Before you extract the first archive, launch WinZip, click ...., and make sure that the checkbox that reads 'Convert ....' is unchecked.

12.0.3 Installing MinGW, Tcl/Tk, MSYS, and GrWin

1. Create the folder

    `C:\Program Files\MinGW`, or

    `C:\Programme\MinGW`, etc.,

    depending on the language of your Windows XP installation. For the sake of simplicity we'll just use the German path names from now on.

2. Extract the files below to the folder

    `C:\Programme\MinGW`

    in precisely this order:

    (a) `mingw-runtime-3.9.tar.gz`

    (b) `gcc-core-3.4.2-20040916-1.tar.gz`

   (c) `gcc-g++-3.4.2-20040916-1.tar.gz`
   (d) `gcc-g77-3.4.2-20040916-1.tar.gz`
   (e) `w32api-3.5.tar.gz`
   (f) `binutils-2.15.91-20040904-1.tar.gz`

3. Install the GNU Debugger (GDB) by executing

   ```
   gdb-5.2.1-1.exe
   ```

   When it asks for a folder to install to, enter the path to MinGW:

   ```
   C:\Programme\MinGW
   ```

4. Install GNU Make by executing

   `mingw32-make-3.80.0-3.exe`.

   When it asks for a folder to install to, enter the path to MinGW:

   ```
   C:\Programme\MinGW
   ```

5. Install Tcl/Tk by executing

   ```
   tcltk-8.4.1-1.exe
   ```

   When it asks for a folder to install to, enter the path to MinGW:

   ```
   C:\Programme\MinGW
   ```

6. Install MSYS by executing

   ```
   MSYS-1.0.11-2004.04.30-1.exe
   ```

   Don't install it to MinGW's folder, but enter

   ```
   C:\Programme\MSYS
   ```

   as the installation folder instead. A windows command prompt dialog will ask you several questions, enter 'y' twice. When it asks for MinGW's installation folder, enter

   ```
   C:/Programme/MinGW
   ```

   Make sure to use linux-style slashes this time.

   You can ignore any Microsoft Word documents jumping at you.

7. Install GrWin by executing

   ```
   lGrWn0999be-MinGW.exe
   ```

   Enter

   ```
   C:\Programme\GrWin
   ```

   as the installation folder.

12.0.4 Compiling LAPACK, LIBLAC, and MUSCOD-II

1. Extract the LAPACK archive

   ```
   lapack.tgz
   ```

   to your home directory

   ```
   C:\Programme\MSYS\home\<user>
   ```

2. Now open an MSYS shell using the new blue M icon on your desktop. You're now in a linux-like shell and have to use slashes instead of backslashes.

   The Windows folder

   ```
   C:\Programme\MSYS\home\<user>,
   ```

   where `<user>` is your Windows account's user name, will serve as your home directory; within MSYS the command

   ```
   cd ~
   ```

   will bring you there.

   The Windows drives `C:\`, `D:\`, etc. can be found as invisible directories below the root. For example, within MSYS your home folder can also be accessed by typing

   ```
   cd /c/Programme/MSYS/home/<user>
   ```

   MinGW itself can be found in the folder `/mingw`, which corresponds to `/c/Programme/MinGW`.

3. We're now applying some changes to several make files. Should you find that these changes have already been applied in your version of these files, this is fine.

   Change to the LAPACK installation folder by typing

   ```
   cd ~/LAPACK/INSTALL
   ```

   Make sure the correct makefile will be used by typing

   ```
   cp make.inc.linux ../make.inc
   ```

   Change to the parent folder (which is `~/LAPACK`) by typing

   ```
   cd ..
   ```

   Open the file `Makefile` again using your favourite editor, and change the line

   ```
   all: install lib testing blas_testing timing blas_timing
   ```

   to read

   ```
   all: blaslib lapacklib
   ```

   Make LAPACK by typing

   ```
   make
   ```

   This may take some minutes. When make finished, the files `lapack_LINUX.a` and `blas_LINUX.a` should have been created. Move and rename them by typing

```
mv lapack_LINUX.a /mingw/lib/liblapack.a
mv blas_LINUX.a /mingw/lib/libblas.a
```

4. Change to your home directory

   ```
   cd ~
   ```

   and create a new directory `MUSCOD-II` there

   ```
   mkdir MUSCOD-II
   ```

   Check out a current version of MUSCOD-II from the workgroup's subversion repository and put it into that folder.

5. Still using the MSYS shell, change to the LIBLAC folder by typing

   ```
   cd ~/MUSCOD-II/LIBLAC
   ```

   Open the file `makefile` and make sure that the line at the very top reads

   ```
   MACHINE=inc_MINGW32.mk
   ```

   Look for the `LIBS` entry and comment it (put a '`#`' in front of all lines belonging to the `LIBS` entry). Uncomment the currently commented MinGW32 version (remove the appropriate '`#`' signs).

   Leave the editor and make LIBLAC by typing

   ```
   make
   ```

   Again this may take some time.

6. Change to the MUSCOD-II folder by typing

   ```
   cd ../MC2
   ```

   Run `make` once to create the file `user.mk`. Open this file and make sure that

   ```
   MACHINE=inc_MINGW32.mk
   ```

   is set. Make MUSCOD-II by typing

   ```
   make
   ```

   Again this may take some time.

7. Create the file

   ```
   ~/.profile
   ```

   and type the following lines:

   ```
   export PGPLOT_DIR=/c/Programme/GrWin/pgplot
   export PGPLOT_FONTS=/c/Programme/GrWin/pgplot/grfont.dat
   export PGPLOT_RGB=/c/Programme/GrWin/pgplot/rgb.txt
   ```

   Leave the editor again. Apply the changes by typing

   ```
   source .profile
   ```

8. If you checked out the MUSCOD-II test projects, change to the MUSCOD-II test projects folder by typing

   ```
   cd ../MC2_TEST
   ```

   Make the MUSCOD-II test projects by typing

   ```
   make
   ```

   Again this may take some time.

9. The installation process is now complete and MUSCOD-II should be at your services.

13 Example: ODE Test Problem `reentry`

Listing 1: reentry source file

```
/*
 *
 *  MUSCOD-II/Apps/TEST/SRC/reentry.cpp
 *  (c) Daniel B. Leineweber, 1995
 *
 *  reentry of Apollo type vehicle (Plitt, 1981; Stoer/Bulirsch, 1992)
 *
 *  $Id: reentry.cpp 3369 2009-07-14 09:10:40Z chris $
 *
 */

#include <cmath>
#include "def_usrmod.hpp"

#define  NMOS    1
#define  NP      0
#define  NRC     0
#define  NRCE    0
#define  NXD     3
#define  NXA     0
#define  NU      1
#define  NPR     0
#define  NRD_S   3
#define  NRDE_S  3
#define  NRD_E   3
#define  NRDE_E  3


#define PI 3.1415
#define BETA 4.26
#define G 3.2172E-4
#define R 209.0
#define SM 53200.0
#define RHO_0 2.704E-3

static void lfcn(double *t, double *xd, double *xa, double *u,
  double *p, double *lval, double * rwh, long *iwh, long *info)
{
        *lval = 10.0*xd[0]*xd[0]*xd[0]*sqrt(RHO_0*exp(-BETA*R*xd[2]));
}

static void ffcn(double *t, double *xd, double *xa, double *u,
  double *p, double *rhs, double *rwh, long *iwh, long *info)
{
        double rho = RHO_0*exp(-BETA*R*xd[2]);
        double cw  = 1.174 - 0.9*cos(u[0]);
        double ca  = 0.6*sin(u[0]);

        rhs[0] = - 0.5*SM*rho*xd[0]*xd[0]*cw
                 - G*sin(xd[1])/(1.0+xd[2])/(1.0+xd[2]);
        rhs[1] = 0.5*SM*rho*xd[0]*ca
                 + xd[0]*cos(xd[1])/R/(1.0+xd[2])
                 - G*cos(xd[1])/xd[0]/(1.0+xd[2])/(1.0+xd[2]);
        rhs[2] = xd[0]*sin(xd[1])/R;
}

static void rdfcn_s(double *ts, double *sd, double *sa, double *u,
  double *p, double *pr, double *res, long *dpnd, long *info)
{
        if (*dpnd) { *dpnd = RFCN_DPND(0, *sd, 0, 0, 0, 0); return; }

        res[0] = sd[0] - 0.36;
        res[1] = sd[1] + 8.1*PI/180.0;
        res[2] = sd[2] - 4.0/R;
}

static void rdfcn_e(double *ts, double *sd, double *sa, double *u,
  double *p, double *pr, double *res, long *dpnd, long *info)
{
        if (*dpnd) { *dpnd = RFCN_DPND(0, *sd, 0, 0, 0, 0); return; }

        res[0] = sd[0] - 0.27;
        res[1] = sd[1];
        res[2] = sd[2] - 2.5/R;
}

extern "C" void def_model(void)
{
        def_mdims(NMOS, NP, NRC, NRCE);
        def_mstage( 0, NXD, NXA, NU, NULL, lfcn, 0, 0, 0, NULL, ffcn, NULL, NULL, NULL );
        def_mpc(0, "Start_Point", NPR, NRD_S, NRDE_S, rdfcn_s, NULL);
        def_mpc(0, "End_Point", NPR, NRD_E, NRDE_E, rdfcn_e, NULL);
}
```

Listing 2: reentry dat file

```
*
*
*  MUSCOD-II/Apps/TEST/DAT/reentry.dat
*  (c) Daniel B. Leineweber, 1995
*
*  reentry of spacecraft (Bock/Plitt, 1984;
*                         Stoer/Bulirsch, 1992)
*
*  $Id: reentry.dat 651 2009-05-18 10:25:48Z ckirches $
*
*

*  # of multiple shooting intervals on each model stage
nshoot
0: 6

*  multiple shooting grids on model stages
grid(*)
0: 0.0
1: 0.25
2: 0.375
3: 0.5
4: 0.675
5: 0.75
6: 1.0

*  model stage duration start values, scales, bounds
h
0: 230.0

h_sca
0: 225.0

h_min
0: 220.0

h_max
0: 240.0

*  mode for differential state variable start values
s_spec
1

*  differential state start values, scales, bounds
sd(0,S)
0: 0.36
1: -0.1414
2: 0.01914

sd(0,E)
0: 0.27
1: 0.0
2: 0.01196

sd_sca(*,*)
0: 0.4
1: 0.1
2: 0.02

sd_min(*,*)
0: 0.2
1: -0.2
2: 0.006

sd_max(*,*)
0: 0.4
1: 0.1
2: 0.03

*  control parameterization types
u_type(*)
0: 1

*  control start values, scales, bounds
u(*,*)
0: 0.5

u_sca(*,*)
0: 1.0

u_min(*,*)
0: -3.0

u_max(*,*)
0: 2.0
*  control slope start values, scales, bounds
udot(*,*)
0: 0.0

udot_sca(*,*)
0: 1.0

udot_min(*,*)
0: -0.1

udot_max(*,*)
0: 0.1

*  decoupled i.p.c. scale factors
rd_sca(0,S)
0: 0.4
1: 0.2
2: 0.02

rd_sca(0,E)
0: 0.4
1: 0.2
2: 0.02

*  objective scale and expected range
of_sca
0.0275

of_min
0.0

of_max
0.05

* Number of values in history plot
nhist
30

**********************
* Choosing libraries *
**********************
libmodel
SRC/libreentry

libhessian
hess_finitediff

libsolve
solve_tbox

libcond
cond_std

libtchk
tchk

libmssqp
mssqp_standard

libeval
eval_ind

libind
0: ind_daesol

libqps
qps_qpopt

libplot
plot_pgplot

**********************************
* Setting algorithmic parameters *
**********************************
options_acc
1e-6
options_ftol
-1.0
options_itol
-1.0
options_rfac
0.0
options_levmar
0.0
options_qp_featol
1.0e-8
options_qp_relax
1.1
options_nhtopy
0
```

```
options_frstart
0
options_frmax
0
options_itmax
100
options_plevel_screen
0
options_plevel_file
1
options_plevel_matlab
0
options_bflag
-1
options_qp_itmax
10000
options_qp_expand
99999999
options_sflag
0
options_wflag
0
options_cflag
0
options_output_ps
0
options_output_gif
0
```

14 Example: MS MINTOC Problem `lotka`

Listing 3: lotka source file

```
/*
 *
 *  MUSCOD-II/Apps/MIP/SRC/lotka.cpp
 *  Sebastian Sager (2003)
 */

#include <math.h>
#include <stdio.h>

#include "def_usrmod.hpp"

#define  NMOS    1
#define  NP      2
#define  NRC     0
#define  NRCE    0

#define  NXD     3
#define  NXA     0
#define  NU      1
#define  NPR     0

static void ffcn(double *t, double *xd, double *xa, double *u,
  double *p, double *rhs, double *rwh, long *iwh, long *info)
{
  double ref0 = 1, ref1 = 1;               /* steady state with u == 0 */

  rhs[0] = xd[0] - xd[0]*xd[1] - p[0]*u[0]*xd[0];
  rhs[1] = - xd[1] + xd[0]*xd[1] - p[1]*u[0]*xd[1];
  rhs[2] = (xd[0]-ref0)*(xd[0]-ref0) + (xd[1]-ref1)*(xd[1]-ref1);
}

static void mfcn(double *ts, double *sd, double *sa, double *p,
  double *mval, long *dpnd, long *info)
{
  if (*dpnd) { *dpnd = MFCN_DPND(0, *sd, 0, 0); return; }

  *mval = sd[2];
}

extern "C" void def_model(void)
{
  def_mdims(NMOS, NP, NRC, NRCE);
  def_mstage( 0,  NXD, NXA, NU,  mfcn, NULL, 0, 0, 0, NULL, ffcn, NULL,  NULL, NULL );
  def_mio(NULL, NULL, NULL);
}



static void rdfcn_e(double *ts, double *sd, double *sa, double *u, double *p,
double *pr, double *res, long *dpnd, long *info) {
  if (*dpnd) { *dpnd = RFCN_DPND(*ts, 0, 0, 0, 0, 0); return; }
  res[0] = *ts - 12;
}

extern "C" void def_model_ct(void)
{
  long i;

  LVec newnmos = getNmos(NMOS);
  long nmos = V_EACC( newnmos, NMOS );

  def_mdims(nmos, NP, NRC, NRCE);
  for( i=0; i<nmos-1; i++) {
    def_mstage( i,  NXD, NXA, NU,  NULL, NULL,  0, 0, 0, NULL, ffcn, NULL,  NULL, NULL );
  }
  def_mstage( nmos-1,  NXD, NXA, NU,  mfcn, NULL,  0, 0, 0, NULL, ffcn, NULL,  NULL, NULL );

  def_mpc(nmos-1, "End_Point", NPR, 1, 1, rdfcn_e, NULL);

  def_mio(NULL, NULL, NULL);
}
```

Listing 4: lotka dat file

```
*
*
*  MUSCOD-II/Apps/MIP/DAT/lotka.dat
*  (c) Sebastian Sager, 2003
*
*

*  # of multiple shooting intervals on each model stage
nshoot
0: 60

* parameters
p
0: 0.4
1: 0.2

p_int
0: 0
1: 0
2: 0

p_sca
0: 1.0
1: 1.0
2: 1.0

p_min
0: 0.0
1: 0.0
2: 0.0

p_max
0: 1.0
1: 1.0
2: 1.0

p_fix
0: 1
1: 1
2: 1

*  model stage duration start values, scales, bounds
h
0: 12.0

h_sca
0: 1.0

h_min
0: 3.0

h_max
0: 12.0

*  model stage duration fixed value flags
h_fix
0: 1

s_spec
2

*  differential state start values, scales, bounds
sd(0,0)
0: 0.5
1: 0.7
2: 0.0

sd_fix(0,0)
0: 1
1: 1
2: 1

sd_sca(*,*)
0: 1.0
1: 1.0
2: 1.0

sd_min(*,*)
0: 0.0
1: 0.0
2: 0.0

sd_max(*,*)
0: 20.0
1: 20.0
2: 25.0

*  objective scale and expected range
of_sca
1.0

of_min
0.0

of_max
25.0

of_name
!Quadratic deviation

h_name
0: !Zeit

p_name
0: ! ParamA
1: ! ParamB
2: ! Force parameter eps

u_name
0: Control function u(t)

xd_name
0: !Biomass of Prey
1: !Biomass of Predator
2: !Integrated deviation

nhist
30

u_type(*)
0: 0

u_sca(0,*)
0: 1.0

u(0,*)
0: 1.0
u_min(0,*)
0: 0.0
u_max(0,*)
0: 1.0

********************
* MINTOC variables *
********************

* x=0  continuous
* x=1  binary
* x=-1 integer
* x>1  SOS1 variable
* x<-1 SOS1 variable with control n_w eliminated
u_int(0,*)
0: 1

p_int
ALL: 0

******************
* MINTOC options *
******************

### penalty term parameters

# Initialization of penalty parameter
eps_init
0.0001

# Multiplier of penalty parameter
eps_step
2

# Penalty tolerance
penTolZero
0.001

# Penalty integer tolerance
penIntTol
1e-05

# Penalty convergence tolerance
penConTol
0.0001

# Number of QP iterations in Penalty strategy
penNumQPSteps
100
```

```
# Maximum number of iterations in Penalty strategy
penMaxSteps
20

# Maximum number of iterations
# before stuck in Penalty strategy
penMaxStuck
30


### adaptparam parameters

# Minimum length of stages
minstagelength
0.0001

# Number of successive adaptations
numadaptiters
3

# Adaptmode: 0 bisection 1 middle peak 2 adaptive
adaptmode
0

# Start of Penalty after adaptivity
adaptPenStart
3

# What was this again
adaptPenIter
1


### General parameters

# Zero tolerance
tolZero
1e-05

# Index of control to be used for simulation (internal)
simIndex
7

# Maximum of how many iterations
# for NO restart in solution
maxIterationsForConvergence
6

# What s_spec for restarts?
restart_s_spec
2


### Rounding parameters

# Offset that has been added to u_max to avoid cycling
roundOffset
0

# What after rounding? 0 Optimize 1 Simulate only
simulateOnly
0

# Rounding:
# 1 fix only bounds, 0 set also variable value
roundedEmbedding
1

# Open additional plot window?
plotAlgorithmicData
1

*********************
* Choosing libraries *
*********************

libmodel
SRC/liblotka

libhessian
hess_update

libsolve
solve_slse

libcond
cond_std

libtchk
tchk

libmssqp
mssqp_standard

libeval
eval_ind

libind
0: ind_rkf45

libqps
qps_qpopt

libplot
plot_pgplot

libmintoc
mintoc

*********************************
* Setting algorithmic parameters *
*********************************

options_acc
1e-6
options_ftol
-1.0
options_itol
-1.0
options_rfac
0.0
options_levmar
0.0
options_qp_featol
1.0e-8
options_qp_relax
1.1
options_nhtopy
0
options_frstart
0
options_frmax
0
options_itmax
100
options_plevel_screen
0
options_plevel_file
1
options_plevel_matlab
0
options_bflag
-1
options_qp_itmax
10000
options_qp_expand
99999999
options_sflag
0
options_wflag
0
options_cflag
0
options_output_ps
0
options_output_gif
0
```

15 Appendix A: MUSCOD-II test library

This section gives an overview over the testproblems in the software MUSCOD, that can be found in /Apps/Test/Src/. Note that some of the literature is no longer available and that description, solution characteristics or literature do not exist for all problems.

15.1 List of all Testproblems

academy	batchdist	batchdistrob	batchdistRobEx	batchdistUT
brac	brgr1	brgr2	ccbat	ccrane
chain1d	chain	container_bridge	cstr	cstr_est
dcbat1	dcbat2	dcbat3	eason	energy_f77
eocar1	eocar2	extrosen	fedbat1	fedbat1m
fedbat2	fedbat2m	fedbat3	fedbat3m	fedbat4
freudenstein	ftlos1	ftlos2	ftlos2_nlp	ftlos3
ftlos4	hang	helical	hydroscal	inventory_f77
kite	lbat	macro1	macro2	macro3
macro4	macro5	maratos	nlbat	nlp1
nlp2	nlp3	nlp4	nmpc1	ocean3a
ocean3b	optcar_lego	orbit	oven	pbreac
powerkite	qlin	reactdiff	reentry	rob2link
rob2link_flex	rob3link	rob3link_flex	rocket_f77	rosen
singular	skeleton	smb	soccer	stp1
stp1_gn	stp2a	stp2b	stp2c	stp2d
stp2e	stp3	stp3_f77	stp3_gn	stp4a
stp4a_gn	stp4b	swtball	tocar1	tocar2
tocar3	tolin1	tolin2	tolos1	tolos2
tolos3	tolos4	twobat1	twobat2	unload1
unload2	vdpol	watson	wood	

15.2 List of solution characteristics

The tables are taken from PhD thesis of D. Leineweber 1999. They use the following symbols and abbreviations:

- *name* gives a mnemonic description of the test problem (usually this is also the name of the corresponding model and data files),
- $\hat{M}$ denotes the total number of model stages (each set of stage transition conditions - if explicitly specified - counts as an additional “algebraic” model stage),
- n_i^x, n_i^z, n_i^u are the numbers of differential states, algebraic states and control functions (specified separately for each model stage i if there are changes in the model dimensions),
- n^p and n^v are the numbers of the *free* model parameters and model stage durations (n^p includes global as well as local model parameters),

- *ndis* denotes the total number of discretization points used (i.e. the number of multiple shooting intervals plus one),

- *nvar* is the total number of variables in the resulting structured NLP problem (including all discretized state variables, control parameters and free model parameters/model stage durations),

- *neq* and *nin* refer to the numbers of equality and inequality constraints in the NLP problem (*nin* includes lower and upper bounds on all variables),

- *mc2* specifies the variant of MUSCOD-II used for the solution of the problem (way of Hessian approximation, globalization strategy)

  - *ch*: constant diagonal Hessian,
  - *fd*: forward-difference Hessian approximation,
  - *lm*: limited-memory BFGS Hessian approximation,
  - *vm*: standard BFGS Hessian approximation,
  - *bt*: boxstep trust region strategy with SOC,
  - *sl*: standard line search strategy without SOC,
  - *wl*: watchdog line search strategy,

- *int* specifies the ODE or DAE integrator employed within MUSCOD-II,

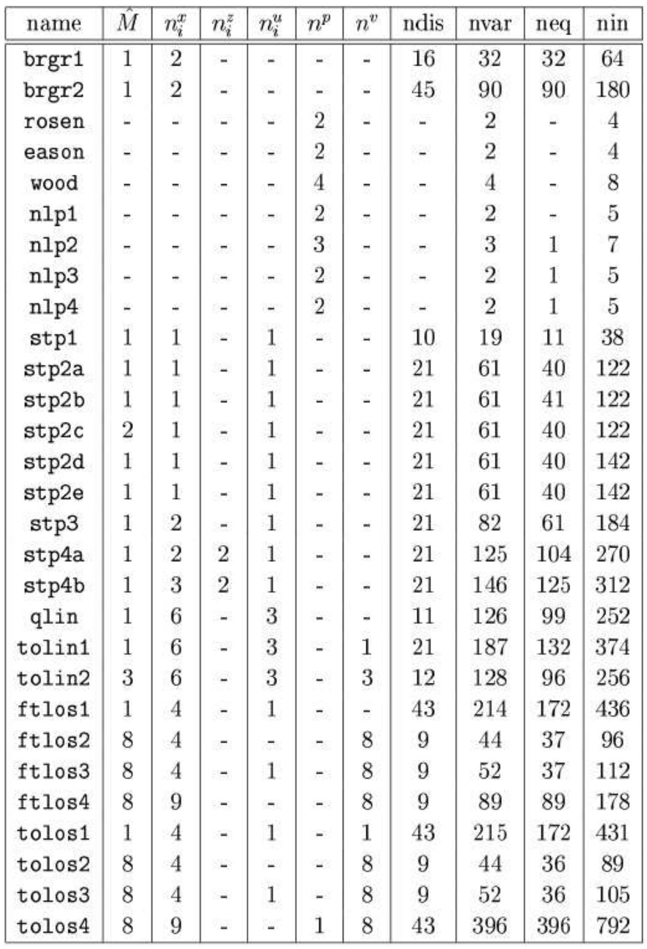
- *itr* is the number of SQP iterations,

- *cpu* is the CPU time in seconds - excluding graphics - on an SGI Indy workstation (MIPS R4000 CPU with 100Mhz speed, MIPS R4010 FPU) running IRIX Version 5.3,

- *obj* is the value of the objective function at the solution,

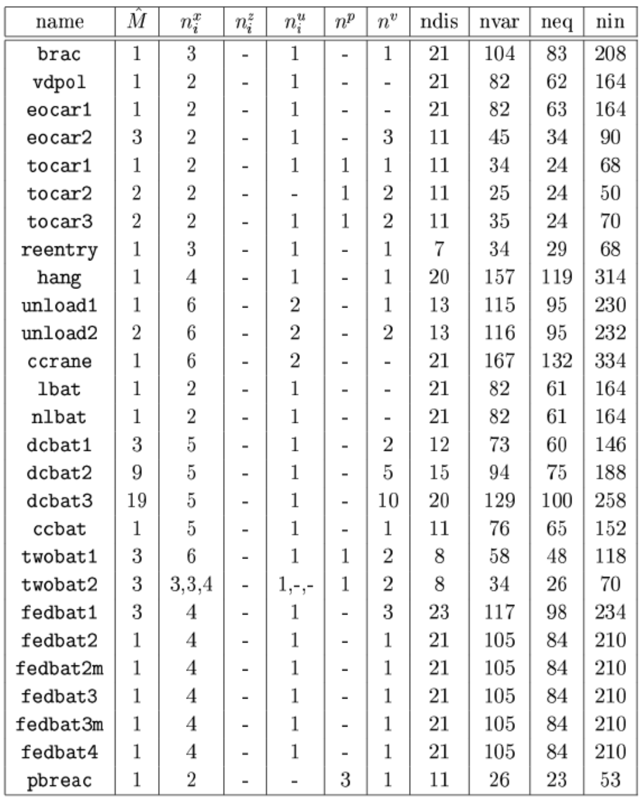
- *inf* is the (scaled) norm of the constraint infeasibilities at the solution,

- *lag* is the (scaled) norm of the Lagrangian gradient at the solution,



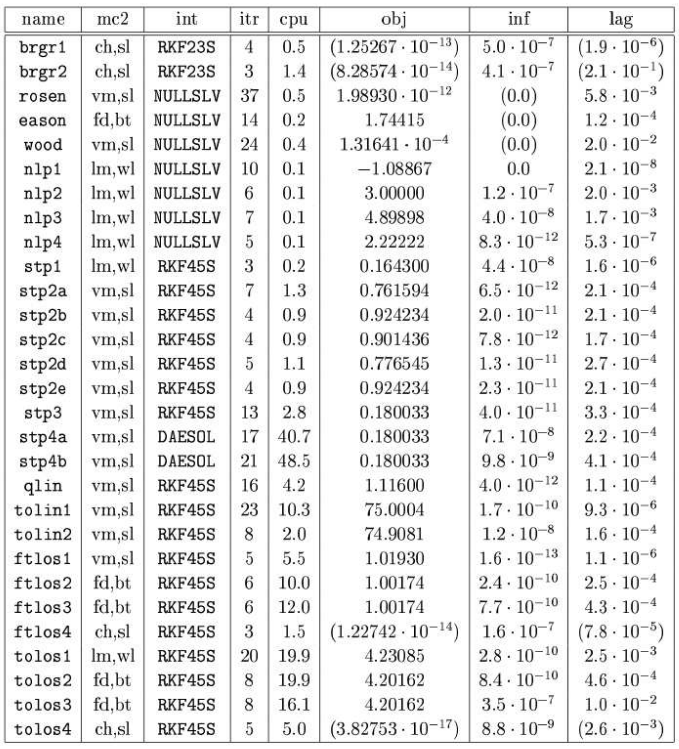
name	$\hat{M}$	n_i^x	n_i^z	n_i^u	n^p	n^v	ndis	nvar	neq	nin
brgr1	1	2	-	-	-	-	16	32	32	64
brgr2	1	2	-	-	-	-	45	90	90	180
rosen	-	-	-	-	2	-	-	2	-	4
eason	-	-	-	-	2	-	-	2	-	4
wood	-	-	-	-	4	-	-	4	-	8
nlp1	-	-	-	-	2	-	-	2	-	5
nlp2	-	-	-	-	3	-	-	3	1	7
nlp3	-	-	-	-	2	-	-	2	1	5
nlp4	-	-	-	-	2	-	-	2	1	5
stp1	1	1	-	1	-	-	10	19	11	38
stp2a	1	1	-	1	-	-	21	61	40	122
stp2b	1	1	-	1	-	-	21	61	41	122
stp2c	2	1	-	1	-	-	21	61	40	122
stp2d	1	1	-	1	-	-	21	61	40	142
stp2e	1	1	-	1	-	-	21	61	40	142
stp3	1	2	-	1	-	-	21	82	61	184
stp4a	1	2	2	1	-	-	21	125	104	270
stp4b	1	3	2	1	-	-	21	146	125	312
qlin	1	6	-	3	-	-	11	126	99	252
tolin1	1	6	-	3	-	1	21	187	132	374
tolin2	3	6	-	3	-	3	12	128	96	256
ftlos1	1	4	-	1	-	-	43	214	172	436
ftlos2	8	4	-	-	-	8	9	44	37	96
ftlos3	8	4	-	1	-	8	9	52	37	112
ftlos4	8	9	-	-	-	8	9	89	89	178
tolos1	1	4	-	1	-	1	43	215	172	431
tolos2	8	4	-	-	-	8	9	44	36	89
tolos3	8	4	-	1	-	8	9	52	36	105
tolos4	8	9	-	-	1	8	43	396	396	792

Figure 2: Table 1: List of Test Problems



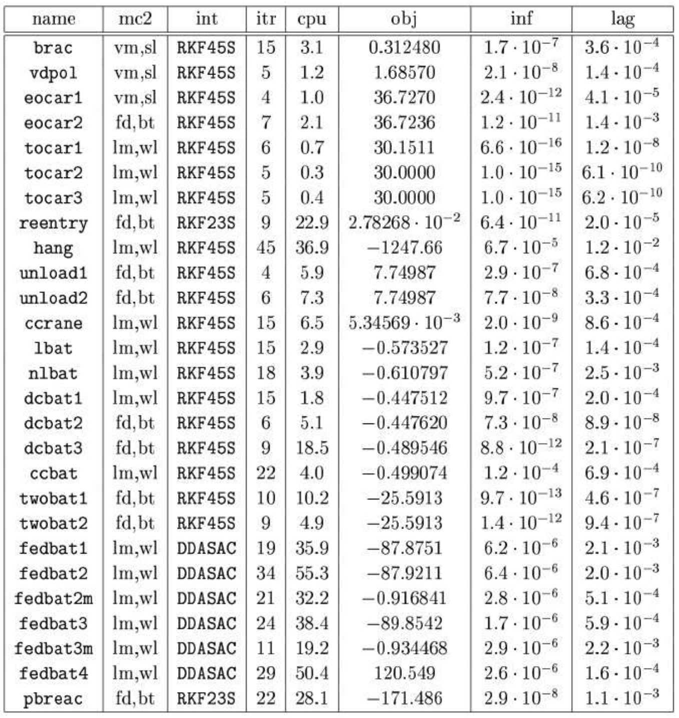
name	$\hat{M}$	n_i^x	n_i^z	n_i^u	n^p	n^v	ndis	nvar	neq	nin
brac	1	3	-	1	-	1	21	104	83	208
vdpol	1	2	-	1	-	-	21	82	62	164
eocar1	1	2	-	1	-	-	21	82	63	164
eocar2	3	2	-	1	-	3	11	45	34	90
tocar1	1	2	-	1	1	1	11	34	24	68
tocar2	2	2	-	-	1	2	11	25	24	50
tocar3	2	2	-	1	1	2	11	35	24	70
reentry	1	3	-	1	-	1	7	34	29	68
hang	1	4	-	1	-	1	20	157	119	314
unload1	1	6	-	2	-	1	13	115	95	230
unload2	2	6	-	2	-	2	13	116	95	232
ccrane	1	6	-	2	-	-	21	167	132	334
lbat	1	2	-	1	-	-	21	82	61	164
nlbat	1	2	-	1	-	-	21	82	61	164
dcbat1	3	5	-	1	-	2	12	73	60	146
dcbat2	9	5	-	1	-	5	15	94	75	188
dcbat3	19	5	-	1	-	10	20	129	100	258
ccbat	1	5	-	1	-	1	11	76	65	152
twobat1	3	6	-	1	1	2	8	58	48	118
twobat2	3	3,3,4	-	1,-,-	1	2	8	34	26	70
fedbat1	3	4	-	1	-	3	23	117	98	234
fedbat2	1	4	-	1	-	1	21	105	84	210
fedbat2m	1	4	-	1	-	1	21	105	84	210
fedbat3	1	4	-	1	-	1	21	105	84	210
fedbat3m	1	4	-	1	-	1	21	105	84	210
fedbat4	1	4	-	1	-	1	21	105	84	210
pbreac	1	2	-	-	3	1	11	26	23	53

Figure 3: Table 2: List of Test Problems



name	mc2	int	itr	cpu	obj	inf	lag
brgr1	ch,sl	RKF23S	4	0.5	$(1.25267 \cdot 10^{-13})$	$5.0 \cdot 10^{-7}$	$(1.9 \cdot 10^{-6})$
brgr2	ch,sl	RKF23S	3	1.4	$(8.28574 \cdot 10^{-14})$	$4.1 \cdot 10^{-7}$	$(2.1 \cdot 10^{-1})$
rosen	vm,sl	NULLSLV	37	0.5	$1.98930 \cdot 10^{-12}$	(0.0)	$5.8 \cdot 10^{-3}$
eason	fd,bt	NULLSLV	14	0.2	1.74415	(0.0)	$1.2 \cdot 10^{-4}$
wood	vm,sl	NULLSLV	24	0.4	$1.31641 \cdot 10^{-4}$	(0.0)	$2.0 \cdot 10^{-2}$
nlp1	lm,wl	NULLSLV	10	0.1	-1.08867	0.0	$2.1 \cdot 10^{-8}$
nlp2	lm,wl	NULLSLV	6	0.1	3.00000	$1.2 \cdot 10^{-7}$	$2.0 \cdot 10^{-3}$
nlp3	lm,wl	NULLSLV	7	0.1	4.89898	$4.0 \cdot 10^{-8}$	$1.7 \cdot 10^{-3}$
nlp4	lm,wl	NULLSLV	5	0.1	2.22222	$8.3 \cdot 10^{-12}$	$5.3 \cdot 10^{-7}$
stp1	lm,wl	RKF45S	3	0.2	0.164300	$4.4 \cdot 10^{-8}$	$1.6 \cdot 10^{-6}$
stp2a	vm,sl	RKF45S	7	1.3	0.761594	$6.5 \cdot 10^{-12}$	$2.1 \cdot 10^{-4}$
stp2b	vm,sl	RKF45S	4	0.9	0.924234	$2.0 \cdot 10^{-11}$	$2.1 \cdot 10^{-4}$
stp2c	vm,sl	RKF45S	4	0.9	0.901436	$7.8 \cdot 10^{-12}$	$1.7 \cdot 10^{-4}$
stp2d	vm,sl	RKF45S	5	1.1	0.776545	$1.3 \cdot 10^{-11}$	$2.7 \cdot 10^{-4}$
stp2e	vm,sl	RKF45S	4	0.9	0.924234	$2.3 \cdot 10^{-11}$	$2.1 \cdot 10^{-4}$
stp3	vm,sl	RKF45S	13	2.8	0.180033	$4.0 \cdot 10^{-11}$	$3.3 \cdot 10^{-4}$
stp4a	vm,sl	DAESOL	17	40.7	0.180033	$7.1 \cdot 10^{-8}$	$2.2 \cdot 10^{-4}$
stp4b	vm,sl	DAESOL	21	48.5	0.180033	$9.8 \cdot 10^{-9}$	$4.1 \cdot 10^{-4}$
qlin	vm,sl	RKF45S	16	4.2	1.11600	$4.0 \cdot 10^{-12}$	$1.1 \cdot 10^{-4}$
tolin1	vm,sl	RKF45S	23	10.3	75.0004	$1.7 \cdot 10^{-10}$	$9.3 \cdot 10^{-6}$
tolin2	vm,sl	RKF45S	8	2.0	74.9081	$1.2 \cdot 10^{-8}$	$1.6 \cdot 10^{-4}$
ftlos1	vm,sl	RKF45S	5	5.5	1.01930	$1.6 \cdot 10^{-13}$	$1.1 \cdot 10^{-6}$
ftlos2	fd,bt	RKF45S	6	10.0	1.00174	$2.4 \cdot 10^{-10}$	$2.5 \cdot 10^{-4}$
ftlos3	fd,bt	RKF45S	6	12.0	1.00174	$7.7 \cdot 10^{-10}$	$4.3 \cdot 10^{-4}$
ftlos4	ch,sl	RKF45S	3	1.5	$(1.22742 \cdot 10^{-14})$	$1.6 \cdot 10^{-7}$	$(7.8 \cdot 10^{-5})$
tolos1	lm,wl	RKF45S	20	19.9	4.23085	$2.8 \cdot 10^{-10}$	$2.5 \cdot 10^{-3}$
tolos2	fd,bt	RKF45S	8	19.9	4.20162	$8.4 \cdot 10^{-10}$	$4.6 \cdot 10^{-4}$
tolos3	fd,bt	RKF45S	8	16.1	4.20162	$3.5 \cdot 10^{-7}$	$1.0 \cdot 10^{-2}$
tolos4	ch,sl	RKF45S	5	5.0	$(3.82753 \cdot 10^{-17})$	$8.8 \cdot 10^{-9}$	$(2.6 \cdot 10^{-3})$

Figure 4: Table 3: List of Test Problems



name	mc2	int	itr	cpu	obj	inf	lag
brac	vm,sl	RKF45S	15	3.1	0.312480	$1.7 \cdot 10^{-7}$	$3.6 \cdot 10^{-4}$
vdpol	vm,sl	RKF45S	5	1.2	1.68570	$2.1 \cdot 10^{-8}$	$1.4 \cdot 10^{-4}$
eocar1	vm,sl	RKF45S	4	1.0	36.7270	$2.4 \cdot 10^{-12}$	$4.1 \cdot 10^{-5}$
eocar2	fd,bt	RKF45S	7	2.1	36.7236	$1.2 \cdot 10^{-11}$	$1.4 \cdot 10^{-3}$
tocar1	lm,wl	RKF45S	6	0.7	30.1511	$6.6 \cdot 10^{-16}$	$1.2 \cdot 10^{-8}$
tocar2	lm,wl	RKF45S	5	0.3	30.0000	$1.0 \cdot 10^{-15}$	$6.1 \cdot 10^{-10}$
tocar3	lm,wl	RKF45S	5	0.4	30.0000	$1.0 \cdot 10^{-15}$	$6.2 \cdot 10^{-10}$
reentry	fd,bt	RKF23S	9	22.9	$2.78268 \cdot 10^{-2}$	$6.4 \cdot 10^{-11}$	$2.0 \cdot 10^{-5}$
hang	lm,wl	RKF45S	45	36.9	-1247.66	$6.7 \cdot 10^{-5}$	$1.2 \cdot 10^{-2}$
unload1	fd,bt	RKF45S	4	5.9	7.74987	$2.9 \cdot 10^{-7}$	$6.8 \cdot 10^{-4}$
unload2	fd,bt	RKF45S	6	7.3	7.74987	$7.7 \cdot 10^{-8}$	$3.3 \cdot 10^{-4}$
ccrane	lm,wl	RKF45S	15	6.5	$5.34569 \cdot 10^{-3}$	$2.0 \cdot 10^{-9}$	$8.6 \cdot 10^{-4}$
lbat	lm,wl	RKF45S	15	2.9	-0.573527	$1.2 \cdot 10^{-7}$	$1.4 \cdot 10^{-4}$
nlbat	lm,wl	RKF45S	18	3.9	-0.610797	$5.2 \cdot 10^{-7}$	$2.5 \cdot 10^{-3}$
dcbat1	lm,wl	RKF45S	15	1.8	-0.447512	$9.7 \cdot 10^{-7}$	$2.0 \cdot 10^{-4}$
dcbat2	fd,bt	RKF45S	6	5.1	-0.447620	$7.3 \cdot 10^{-8}$	$8.9 \cdot 10^{-8}$
dcbat3	fd,bt	RKF45S	9	18.5	-0.489546	$8.8 \cdot 10^{-12}$	$2.1 \cdot 10^{-7}$
ccbat	lm,wl	RKF45S	22	4.0	-0.499074	$1.2 \cdot 10^{-4}$	$6.9 \cdot 10^{-4}$
twobat1	fd,bt	RKF45S	10	10.2	-25.5913	$9.7 \cdot 10^{-13}$	$4.6 \cdot 10^{-7}$
twobat2	fd,bt	RKF45S	9	4.9	-25.5913	$1.4 \cdot 10^{-12}$	$9.4 \cdot 10^{-7}$
fedbat1	lm,wl	DDASAC	19	35.9	-87.8751	$6.2 \cdot 10^{-6}$	$2.1 \cdot 10^{-3}$
fedbat2	lm,wl	DDASAC	34	55.3	-87.9211	$6.4 \cdot 10^{-6}$	$2.0 \cdot 10^{-3}$
fedbat2m	lm,wl	DDASAC	21	32.2	-0.916841	$2.8 \cdot 10^{-6}$	$5.1 \cdot 10^{-4}$
fedbat3	lm,wl	DDASAC	24	38.4	-89.8542	$1.7 \cdot 10^{-6}$	$5.9 \cdot 10^{-4}$
fedbat3m	lm,wl	DDASAC	11	19.2	-0.934468	$2.9 \cdot 10^{-6}$	$2.2 \cdot 10^{-3}$
fedbat4	lm,wl	DDASAC	29	50.4	120.549	$2.6 \cdot 10^{-6}$	$1.6 \cdot 10^{-4}$
pbreac	fd,bt	RKF23S	22	28.1	-171.486	$2.9 \cdot 10^{-8}$	$1.1 \cdot 10^{-3}$

Figure 5: Table 4: List of Test Problems

15.3 Further description of some of the problems

- academy
  This is a model of G. Feichtinger and coworkers. The idea is to optimize the recruitment strategy of an "Academy of Sciences", that aims at

  - keeping the average age of the academy down,
  - maximizing the number of recruitments.

  The size of the academy is fixed to 70 persons. Only persons between 40 and 70 are counted as members. The ratio between the two goals is given by the parameters *p[2]* and *p[3]*. The optimal control is given by the control constraint *rdfcn*, otherwise full recruitment of young resp. very old researchers is optimal.

- batchdistUT
  This model uses the *Unscented Transform* to obtain mean and variance of the inequality constraint functions and optimizes then with a security back-off. (It can also be used for simple dat-file controlled simulation runs. This version is active at the moment.)

- brac
  Classical brachistochrone problem (Betts, J.T.; Eldersveld, S.K.; Huffman, W.P.; Sparse nonlinear programming test problems (Release 1.0); Technical Report BCSTECH-93-047. Boeing Computer Services (1993)).

- brgr1
  Burgers equation for EPS_B = 1.0E-1 (Betts, J.T.; Eldersveld, S.K.; Huffman, W.P.; Sparse nonlinear programming test problems (Release 1.0); Technical Report BCSTECH-93-047. Boeing Computer Services (1993)).

- brgr2
  Burgers equation for EPS_B = 5.0E-2 (Betts, J.T.; Eldersveld, S.K.; Huffman, W.P.; Sparse nonlinear programming test problems (Release 1.0); Technical Report BCSTECH-93-047. Boeing Computer Services (1993)).

- ccbat
  Continuous charge batch reactor (Vassiliadis, V.S.; Pantelides, C.C.; Sargent, W.H.; Optimization of discrete charge batch reactors; Comput. Chem. Engng **18**, Suppl., p.415 - 419 (1994)).

- ccrane
  Container crane (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18 (1988)).

- cstr - constrained stirred tank reactor
  Model Equations according to: Chen, H., Kremling, A. and Allgwer, F.: Nonlinear predictive control of a benchmark CSTR, Proc. 3rd European Control Conference ECC'95, pp. 3247-3252, 1995.

(see also: Diehl, M.; Real-Time optimization for large scale nonlinear processes; Ph.D. thesis, Heidelberg University; 2001; p.12 ff).

- eason
  This is an example for unconstrained minimization, namely Eason's function. (Reklaitis, G.V.; Ravindran, A.; Ragsdell, K.M.; Engineering Optimization. Methods and Applications; 1983).

- dcbat1
  Discrete charge batch reactor I (2 charges) (Vassiliadis, V.S.; Pantelides, C.C.; Sargent, W.H.; Optimization of discrete charge batch reactors; Comput. Chem. Engng **18**, Suppl., p.415 - 419 (1994)).

- dcbat2
  discrete charge batch reactor II (5 charges) (Vassiliadis, V.S.; Pantelides, C.C.; Sargent, W.H.; Optimization of discrete charge batch reactors; Comput. Chem. Engng **18**, Suppl., p.415 - 419 (1994)).

- dcbat3
  discrete charge batch reactor III (10 charges) (Vassiliadis, V.S.; Pantelides, C.C.; Sargent, W.H.; Optimization of discrete charge batch reactors; Comput. Chem. Engng **18**, Suppl., p.415 - 419 (1994)).

- energy_f77
  Energy Problem (given in Bryson, A.E., and Ho, Y.-C., Applied Optimal Control, Hemisphere, Washington, D.C. 1976)

$$\min 0.5 \int\limits_0^1 a^2(t)dt$$

$$\begin{aligned} s.t.\ddot{x} &= a(t) \\ x(t) &\leq l(constant) \\ x(0) = 0, \quad \dot{x}(0) = 1, x(1) = 0, & \quad \dot{x}(1) = -1 \end{aligned}$$

Formulation0 (by setting $x_0 = x, x_1 = \dot{x}, x_2 = 0.5 \int_0^t a^2(t)dt$)

$$\begin{aligned} &\min x_2(1) \\ s.t.\ddot{x}_0 &= x2 \\ \ddot{x}_1 &= a \\ \ddot{x}_2 &= 0.5 \cdot a^2 \\ x_0(t) &\leq l(constant) \\ x_0(0) = 0, \quad x_1(0) = 1, &\quad x_2(0) = 0, \\ x_0(1) = 0, \quad x_1(1) = -1 & \end{aligned}$$

Formulation1 (by setting $x0 = x, x1 = \dot{x}$)

$$\min 0.5 \int\limits_0^1 a^2(t)dt$$

$$\begin{aligned} s.t.\ddot{x}0 &= x2 \\ \ddot{x}1 &= a \\ x0(t) &\leq l \ (constant) \\ x0(0) = 0, &\quad x1(0) = 1, \\ x0(1) = 0, &\quad x1(1) = -1 \end{aligned}$$

- eocar1
  Energy-optimal car I.

- eocar2
  Energy-optimal car II.

- extrosen
  This is another example for unconstrained minimization, the Rosenbrock function (in its extended version, using more variables).
  see Rosenbrock: 'An automatic method for finding the greatest or least value of a function', Comput. J. 3 (1960).

- fedbat1
  Fed-batch fermentor, case I (Cuthrell, J.E.; Biegler, L.T.; Simultaneous optimization and solution methods for batch reactor control profiles; Comp. Chem. Engng 13; p.49-62, 1989).

- fedbat1m
  Fed-batch fermentor, modified case I (max. productivity) (Cuthrell, J.E.; Biegler, L.T.; Simultaneous optimization and solution methods for batch reactor control profiles; Comp. Chem. Engng 13; p.49-62, 1989).

- fedbat2
  Fed-batch fermentor, case II (Cuthrell, J.E.; Biegler, L.T.; Simultaneous optimization and solution methods for batch reactor control profiles; Comp. Chem. Engng 13; p.49-62, 1989).

- fedbat2m
  Fed-batch fermentor, modified case II (max. productivity) (Cuthrell, J.E.; Biegler, L.T.; Simultaneous optimization and solution methods for batch reactor control profiles; Comp. Chem. Engng 13; p.49-62, 1989).

- fedbat3
  fed-batch fermentor (high initial substrate) (Lim, H.C.; Tayeb, Y.J.; Modak, J.M.; Bonte, P.; Computational algorithms for optimal feed rates for a class of fed-batch fermentation:

numerical results for penicillin and cell mass production; Biotechn. Bioengng. 28; p. 1408-1420, 1986).

- fedbat3m
  fed-batch fermentor (high initial substrate, max. productivity) (Lim, H.C.; Tayeb, Y.J.; Modak, J.M.; Bonte, P.; Computational algorithms for optimal feed rates for a class of fed-batch fermentation: numerical results for penicillin and cell mass production; Biotechn. Bioengng. 28; p. 1408-1420, 1986).

- fedbat4
  Fed-batch fermentor, case IV (Cuthrell, J.E.; Biegler, L.T.; Simultaneous optimization and solution methods for batch reactor control profiles; Comp. Chem. Engng 13; p.49-62, 1989).

- freudenstein
  This is a modell of the Freudenstein-Roth function (see Fletcher, Roger; Practical methods of optimization; Chichester [et.al.] 1987, p. 120).

- ftlos1
  Fixed-time linear oscillating system I (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- ftlos2
  Fixed-time linear oscillating system II (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- ftlos2_nlp
  The problem ftlos2 in NLP formulation (eliminated states).

- ftlos3
  Fixed-time linear oscillating system III (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992)

- ftlos4
  An indirect approach (maximum principle) to the fixed-time linear oscillating system, (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- hang
  Hang glider problem
  Betts, J.T.; Eldersveld, S.K.; Huffman, W.P.; Sparse nonlinear programming test problems (Release 1.0); Technical Report BCSTECH-93-047; Boeing Computer Services, 1993.
  Bulirsch, R.; Nerz, E.; Pesch, H.J.; Stryk, O. von; Combining Direct and Indirect Methods in Optimal Control: Range Maximization of a Hang Glider; **in**: International Series of Numerical Mathematics, Vol. 111, 1991; p. 273-288.

- helical
  A helical valley function. See Fletcher; Powell; A rapidly convergent descent method for minimization; Comput. J., 6, (1963).

- hydroscal
  Distillation model, 82 diff + 122 alg var.
  For a description see e.g. Chapter 7 in the PhD thesis "Real-Time Optimization for Large Scale Nonlinear Processes" (2001) by Moritz Diehl (Download at: http://www.ub.uni-heidelberg.de/archiv/1659/)
  Optimization problem is to steer the column from a disturbed state back into the nominal operating point, minimizing an integrated least-squares deviation of two temperatures and the controls.

- inventory_f77

$$
\begin{array}{rcl}
\min & \int_0^T & (exp(-\rho_2 \cdot ts) \cdot a \cdot xd(0) + exp(-\rho_0 \cdot ts) \cdot b0 \cdot u(0) - exp(-\rho_1 \cdot ts) \cdot b1 \cdot u(\\
s.t.\dot{x} & = & u0 - u1 - d, \\
\alpha_1 & \leq x & \leq \alpha_2, \\
0 \leq u_1 & \leq \delta_1, & 0 \leq u_2 \delta_2, \\
\gamma_1 \cdot u_0 + \gamma_2 \cdot u_1 & \leq & \delta_3, \\
x(0) & = & x_0, \\
x(T) & = & x_T.
\end{array}
$$

discount rates $\rho_I \in [0,1]$, $\gamma 1 + \gamma 2 = 1$, a, b_0, b_1 are unit cost of storage (w.r.p. ρ_2), replenishment (w.r.t. ro0) and selling price (with respect to ro1), respectively, d = own demand, x is the stock function, u_0 is replenishment, u_1 is selling amount, 4te constraint financial restriction, where

$$\begin{array}{cccccc}
b_0 & = & 5 & \mathrm{t} & \in & [0,2] \\
 & = & 5 + 2^*(\mathrm{t}\text{-}2) & \mathrm{t} & \in & [2,2.4] \\
 & = & 5.8 & \mathrm{t} & \in & [2.4,4] \\
 & = & 5.8 + 2^*(\mathrm{t}\text{-}4), & \mathrm{t} & \in & [4,7] \\
b_1 & = & b_0 + 1 & & & \\
\rho_0 & = & 0.0 & \mathrm{t} & \in & [0,2] \\
 & = & 0.1, & \mathrm{t} & \in &]2,7] \\
\rho_1 & = & 0.9 & \mathrm{t} & \in & [0,2] \\
 & = & 0.2, & \mathrm{t} & \in &]2,7] \\
\rho_2 & = & 0.1 & \mathrm{t} & \in & [0,2] \\
 & = & 0.7, & \mathrm{t} & \in &]2,7] \\
\mathrm{d} & = & 0.5 & \mathrm{t} & \in & [0,2] \\
 & = & 1.5, & \mathrm{t} & \in &]2,4] \\
 & = & 1.0\text{+}0.5^*(\mathrm{t}\text{-}3.0) & \mathrm{t} & \in &]4,7] \\
\mathrm{a} & = & 3.0\text{-}2^*(\mathrm{t}\text{-}1.0) & \mathrm{t} & \in & [0,2] \\
 & = & 1.0, & \mathrm{t} & \in &]2,7] \\
\gamma_1 & = & \gamma_2 & = & 0.5 & \\
\alpha_1 & = 0, & \alpha_2 & = 2,\ x_0 & = x_T & = 1, \\
\delta_1 & = \delta_2 & = 2,\ \delta_3 & = & 3. &
\end{array}$$

- kite
  Kite Tracking Problem for Stability Optimization. Model as in Diehl, Magni, Scattolini: "Online NMPC of a Looping Kite using Approximate Infinite Horizon Closed Loop Costing", Bratislava, 2003.

- lbat
  Batch reactor, linear in states (Ray, W.H.; Advanced Process Control; New York, 1981). (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- maratos
  Maratos example (avoiding fullstep with nondifferentiable meritfunction).

- macro1
  Simple consumption savings problem: Maximize utility function, dependent on consumption. Capital increases by a deterministic life-cycle income profile.

- macro2
  Neoclassical growth model with endogeneous labor supply.

- macro3
  Neoclassical growth model.

- macro4
  Optimal growth model, extended as to macro3 with additional agent government and taxing:

  - spends variable amount gov=u[3] (thrown away),
  - takes $\tau = p[6]$ percent tax of everything (capital and labour),
  - bonds influence government capital.

  Make sure you modify NSHOOT in both DAT and C file.

- macro5
  Optimal growth model, extended as to macro4 with split up of the worker: now two generations (young and old). Make sure you modify NSHOOT in both DAT and C file.

- nlbat
  Batch reactor, nonlinear in states
  (Ray, W.H.; Advanced Process Control; New York 1981).
  (Renfro, J.G.; Morshedi, A.M.; Asbjornsen, O.A.; Simultaneous optimization and solution of systems described by differential/algebraic equations; Comput. Chem. Engng. 11; p. 503-517 1987).
  (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- nlp1
  Standard NLP test example I. (Gill, P.E.; Murray, W.; Wright, M.H.; Practical Optimization; London [et.al] 2006).

- nlp2
  Standard NLP test example II. See Harwell Subroutine Library Specification, subroutine VF13.

- nlp3
  Standard NLP test example III. (Reklaitis, G.V.; Ravindran, A.; Ragsdell, K.M.; Engineering Optimization. Methods and Applications; 1983).

- nlp4
  Standard NLP test example IV. (Reklaitis, G.V.; Ravindran, A.; Ragsdell, K.M.; Engineering Optimization. Methods and Applications; 1983).

- nmpc1
  Simple NMPC problem.

- optcar_lego
  Optimal control for little car, built with LEGO. Results can be obtained by calling first "mc2ts optcar" and then "mc2ts -c optcar". Plot of the resulting paths can be obtained by calling "gnuplot" in the RES directory and typing: plot "optcar_lego.gnuplot" using 2:3 w lines.

- orbit
  2d orbit transfer.

- qlin
  Quadratic-linear problem (Betts, J.T.; Eldersveld, S.K.; Huffman, W.P.; Sparse nonlinear programming test problems (Release 1.0); Technical Report BCSTECH-93-047; Boeing Computer Services, 1993).

- pbreac
  Packed-bed reactor optimization problem, case Ia, (Cuthrell, J.E.; Biegler, L.T.; On the optimization of differential-algebraic process systems; AIChE J. 33, p.1257-1270; 1987).

- reactdiff
  Constrained PDE minimization example. Finite Difference Discretization of elliptic PDE in 2-D on unit square
  $$0 = d^2u/dx^2 - \alpha u^2 + \beta q$$
  with bound restrictions $0 < u < 1, 0 < q < 1$. u corresponds to a the concentration of a species that

  1. diffuses,
  2. reacts with second order and
  3. is added with distributed controls q.

- reentry
  Reentry of Apollo type vehicle
  (Bock, H.G.; Plitt, K.J.; A multiple shooting algorithm for direct solution of optimal control problems; Proceedings of the 9th IFAC World Congress; Budapest 1984).
  (Stoer, J.; Bulirsch, R.; Introduction to Numerical Analysis; New York, 1992).

- rob2link
  Two link robot model without joint flexibility. Time optimal movement.

- rob2link_flex
  Two link robot model with joint flexibility. Time optimal movement with least squares penalty on accelerations.

- rob3link
  Three link robot model without joint flexibility. Time optimal movement, 6 state variables r, θ_1, θ_2, $\dot{r}$, $\dot{\theta}_1$, $\dot{\theta}_2$, 3 controls U_r, U_{θ_1}, U_{θ_2} (u[0], u[1], u[2]).

- rob3link_flex
  Three link robot model with joint flexibility. Time optimal movement, 12 state variables: $r, \theta_1, \theta_2, \phi_1, \phi_2, \phi_3, \dot{r}, \dot{\theta}_1, \dot{\theta}_2, \dot{\phi}_1, \dot{\phi}_2, \dot{\phi}_3$, 3 controls: $U_r, U_{\theta_1}, U_{\theta_2}$ (u[0], u[1], u[2]).

- rocket_f77

$$
\begin{array}{rcl}
\min T, & & \\
\dot{s} & = & v, \\
\dot{v} & = & \dfrac{c_1}{m} \cdot a - c_2 v^2, \\
\dot{m} & = & -c_3 a^2, \\
\text{with } |a| & \leq & 1, \\
s(0) = 0, & v(0) = 0, & m(0) = 1, \\
s(T) = 10, & v(T) = 0. &
\end{array}
$$

- rosen
  Unconstrained minimization example, Rosenbrock's function. (Gill, P.E.; Murray, W.; Wright, M.H.; Practical Optimization; London [et.al] 2006).

- singular
  Unconstrained minimization example, Powell's singular function. (Powell; An iterative method for finding stationary values of a function of several variables; Comput. J., 5; 1962).

- smb
  Model of a Simplified SMB process.

- stp1
  Simple test problem 1 (Steinbach, M.C.; Fast recursive SQP methods for large-scale optimal control problems; PhD thesis, University of Heidelberg, 1995).

- stp2a
  Simple test problem IIa (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp2b
  Simple test problem IIb (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp2c
  Simple test problem IIc (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp2d
  Simple test problem IId (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp2e
  Simple test problem IIe (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp3
  Simple test problem III (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp3_f77
  Simple test problem III, Fortran 77 version (Goh, C.J.; Teo, K.L.; Control parametrization: a unified approach to optimal control problems with general constraints; Automatica 24, p.3-18; 1988).

- stp4a
  Simple test problem IVa (DAE extension of simple test problem III).

- stp4b
  Simple test problem IVb (DAE extension of simple test problem III with Mayer objective instead of Lagrange objective).

- swtball
  This is a test for the sensitivity updates on switches implemented in RKF45SWT. The bouncing ball's energy loss upon contact with the floor is to be adjusted so that the ball hits the ground exactly at the end of the stage. In between, it bounces several times. Without sensitivity updates, there would be no advance in the objective, since the derivatives with respect to the responsible parameter are zero.

- tocar1
  Time-optimal car I
  (Cuthrell, J.E.; Biegler, L.T.; On the optimization of differential-algebraic process systems; AIChE J. 33, p.1257-1270; 1987).
  (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- tocar2
  Time-optimal car II
  (Cuthrell, J.E.; Biegler, L.T.; On the optimization of differential-algebraic process systems; AIChE J. 33, p.1257-1270; 1987).
  (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- tocar3
  Time-optimal car III
  (Cuthrell, J.E.; Biegler, L.T.; On the optimization of differential-algebraic process systems; AIChE J. 33, p.1257-1270; 1987).

(Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- tolin1
  Time-optimal linear problem I (fixed grid).

- tolin2
  Time-optimal linear problem II (model stage formulation).

- tolos1
  Time-optimal linear oscillating system I (Plant, J.B.; Athans, M.; An iterative technique for the computation of time-optimal controls; Proceedings of the 3rd International IFAC Conference; London, 1966).

- tolos2
  Time-optimal linear oscillating system II (Plant, J.B.; Athans, M.; An iterative technique for the computation of time-optimal controls; Proceedings of the 3rd International IFAC Conference; London, 1966).

- tolos3
  Time-optimal linear oscillating system III (Plant, J.B.; Athans, M.; An iterative technique for the computation of time-optimal controls; Proceedings of the 3rd International IFAC Conference; London, 1966).

- tolos4
  Fixed-time linear oscillating system IV, indirect approach (maximum principle)
  (Logsdon, J.S.; Biegler, L.T.; Decomposition strategies for large-scale dynamic optimization problems; Chem. Engng Sci., 47, p.851-864; 1992).

- twobat1
  Two-stage batch reactor system I (Vassiliadis, V.S.; Sargent, R.W.H.; Pantelides, C.C.; Solution of a class of multistage dynamic optimization problems. 1. Problems without path constraints; Ind. Eng. Chem. Res. 33; p.2111-2122; 1994).

- twobat2
  Two-stage batch reactor system II (Vassiliadis, V.S.; Sargent, R.W.H.; Pantelides, C.C.; Solution of a class of multistage dynamic optimization problems. 2. Problems with path constraints; Ind. Eng. Chem. Res. 33; p.2123-2133; 1994).

- unload1
  Ore unloading system
  (Plitt, K.J.; Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschrnkter optimaler Steuerungen; Diploma thesis; University of Bonn, 1981).

- unload2
  Ore unloading system
  (Plitt, K.J.; Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschrnkter optimaler Steuerungen; Diploma thesis; University of Bonn, 1981).

- vdpol
  Van der Pol problem (Bock, H.G.; Plitt, K.J.; A multiple shooting algorithm for direct solution of optimal control problems; Proceedings of the 9th IFAC World Congress; Budapest 1984).

- watson
  Nonlinear Complementary Problem (restricted version). (Watson; Solving the nonlinear complementary problem by a homotopy method; SIAM J. Cont. Appl., 17, 1979).

- wood
  Unconstrained minimization example, Wood's function. (Reklaitis, G.V.; Ravindran, A.; Ragsdell, K.M.; Engineering Optimization. Methods and Applications; 1983).

- oven
  Model Equations according to: Bertsekas, Dimitri; Dynamic Programming and Optimal Control (Vol. I); Belmont, MA; 1995, pp. 21ff.
  Discrete dynamical system of material passed through two ovens

$$\min r \cdot (x_2 - T)^2 + u_0^2 + u_1^2,$$
$$s.t. x_{k+1} = (1-a) \cdot x_k + a \cdot_k \, (k = 0, 1).$$

analytical solution for feedback laws:

$$\mu_0(x_0) = \frac{r \cdot (1-a) \cdot a \cdot (T - (1-a)^2 \cdot x_0)}{1 + r \cdot a^2 \cdot (1 + (1-a)^2)},$$

$$\mu_1(x_1) = \frac{r \cdot a \cdot (T - (1-a) \cdot x_1)}{1 + r \cdot a^2}.$$

References

[ABB⁺95] E. Anderson, Z. Bai, C. Bischof, J.W. Demmel, J.J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *Lapack Users' Guide.* SIAM, Philadelphia, 2nd edition edition, 1995.

[BBKS00] I. Bauer, H.G. Bock, S. Körkel, and J.P. Schlöder. Numerical methods for optimum experimental design in DAE systems. *J. Comput. Appl. Math.*, 120(1-2):1–15, 2000.

[BBS99] I. Bauer, H.G. Bock, and J.P. Schlöder. DAESOL – a BDF-code for the numerical solution of differential algebraic equations. Internal report, IWR, SFB 359, Universität Heidelberg, 1999.

[BFD$^+$97] I. Bauer, F. Finocchi, W.J. Duschl, H.-P. Gail, and J.P. Schlöder. Simulation of chemical reactions and dust destruction in protoplanetary accretion discs. *Astron. Astrophys.*, 317:273–289, 1997.

[BP84] H.G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.

[CS85] M. Caracotsios and W.E. Stewart. Sensitivity analysis of initial value problems with mixed odes and algebraic equations. *Computers and Chemical Engineering.*, 9:359–365, 1985.

[DBK06] M. Diehl, H.G. Bock, and E. Kostina. An approximation technique for robust nonlinear optimization. *Mathematical Programming*, 107:213–230, 2006.

[DCDH90] J.J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. on Math. Soft.*, 16,1:1–28, 1990.

[DCHH88] J.J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Trans. on Math. Soft.*, 14,1:1–32, 1988.

[DR96] I.S. Duff and J.K. Reid. The design of MA48: A code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. on Math. Soft.*, 22:187–226, 1996.

[GMSW83] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. *User's guide for SOL/QPSOL: a Fortran package for quadratic programming*, volume SOL 83-7 of *Technical Report*. Stanford University, Systems Optimization Laboratory, Department of Operations Research, 1983.

[Lei99] D.B. Leineweber. *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, volume 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. VDI Verlag, Düsseldorf, 1999.

[LHKK79] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subrograms for fortran usage. *ACM Trans. on Math. Soft.*, 5:308–325, 1979.

[LJ96] D.B. Leineweber and J.A. Jost. Liblac – structured data types and basic operations for numerical linear algebra in an ansi c/fortran 77 environment. IWR-Preprint 96-56, Universität Heidelberg, Heidelberg, 1996.

[Pea97] T.J. Pearson. *PGPLOT Graphics Subroutine Library*. California Institute of Technology, Pasadena, 1997.

[RD93] J.K. Reid and I.S. Duff. MA48, A fortran code for direct solution of sparse unsymmetric linear systems of equations. Technical Report RAL 93 072, Rutherford Appleton Laboratory, 1993.

[Rei72] J.K. Reid. *Fortran subroutines for the solution of sparse systems of nonlinear equations*, volume R. 7293 of *Harwell Report*. Harwell Laboratory, Harwell, 1972.

[Sag09] S. Sager. Reformulations and algorithms for the optimization of switching decisions in nonlinear optimal control. *Journal of Process Control*, (accepted), 2009.

[Sch96] V.H. Schulz. *Reduced SQP methods for large-scale optimal control problems in DAE with application to path planning problems for satellite mounted robots*. PhD thesis, Universität Heidelberg, 1996.

[Sch98] V.H. Schulz. Solving discretized optimization problems by partially reduced SQP methods. *Computing and Visualization in Science*, 1:83–96, 1998.

[Sch09] A. Schäfer. On scaling techniques and termination criteria for sqp methods. diploma thesis, Universität Heidelbeg, 2009.

[SRB09] S. Sager, G. Reinelt, and H.G. Bock. Direct methods with maximal lower bound for mixed-integer optimal control problems. *Mathematical Programming*, 118(1):109–149, 2009. published online at http://dx.doi.org/10.1007/s10107-007-0185-6 on 14 August 2007.