# MUSCOD-II Users' Manual

Moritz Diehl  Daniel B. Leineweber *
Andreas A. S. Schäfer

Interdisciplinary Center for Scientific Computing (IWR),
University of Heidelberg,
Germany

February 12, 2001

*Now at Bayer AG, D-51368 Leverkusen, Germany

# Contents

**1 Introduction** **4**

**2 Multistage Optimal Control Problems in DAE** **4**
2.1 Transition between model stages . . . . . 4
2.2 Interior Point and Path Constraints . . . . . 5
2.3 The Objective Function . . . . . 6
2.4 Least Squares Objective Contributions . . . . . 6

**3 The Direct Multiple Shooting Method** **6**
3.1 Piecewise Control Discretization . . . . . 7
3.2 Multiple Shooting State Parametrization . . . . . 7
3.3 Discretization of Bounds, Interior Point and Path Constraints . . . . . 9
3.4 Discretization of Least Squares Terms . . . . . 9
3.5 The resulting Nonlinear Programming problem . . . . . 10
3.6 The SQP Algorithm . . . . . 10

**4 Installation of MUSCOD-II** **12**
4.1 Installation steps . . . . . 12

**5 How to set up a problem** **13**
5.1 Locating the Problem Files . . . . . 14

**6 The Model Source File** **15**

**7 The Data File** **18**
7.1 Description of Keywords . . . . . 20

## Appendix **23**

**A Global Compilation Flags** **23**

**B MUSCOD-II Modules** **24**

**C Five examples of MUSCOD-II algorithms** **27**

**D Runtime Options** **29**

**E Templates for the Source File** **31**

**F Available ODE/DAE Solvers** **33**

**G Example: DAE Test Problem `dowbat`** **34**

**H Using Analytical Derivatives generated by ADIFOR** **39**
H.1 Preparing MUSCOD-II for sparse analytical derivatives . . . . . 39
H.2 Generating sparse model derivatives for MUSCOD-II with ADIFOR 2.0 . . . . . 39

# 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 the first section we present class of problems which can be solved, already introducing the problem syntax used in MUSCOD-II.

2. In the second section a very 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 step for step. Some compilation information, a brief description of the MUSCOD-II modules and some example configurations are given in appendices A, B, and C

4. How to set up and run a problem is described in the fourth section. The runtime options for the executable are explained in appendix D. Appendix G gives an example problem.

# 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:

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

with the matrix function $B_i$ in $\mathbb{R}^{n_i^x \times n_i^x}$ and the derivative $\dfrac{\partial g_i}{\partial z_i} \in \mathbb{R}^{n_i^z \times n_i^z}$ 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: $n^x_{i+1} = n^x$

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 ($n^x_{j+1}$ is not necessarily equal to $n^x_j$). 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 $r^d_k$ (with dimension $n^{rd}_k$) can be specified that require at single points $t = t_k$ in time or on complete model stages (i.e. $\forall_{t\in[t_k,t_{k+1}]}$):

$$r^d_k(t, x(t), z(t), u(t), p, p^r_k) \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} 0$$

Here, the first $n^{rde}_k$ components are equalities and the remaining ones (of altogether $n^{rd}_k$) 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 $r^c_k$ (at time points $t_k$) all of equal dimension $n^{rc}$. The vector sum of these functions is then required to satisfy:

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

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

In both, decoupled and coupled constraints, *local* parameters $p^r_k$ 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].

[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.

[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.

## 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[3]:

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

which again could in principle be covered by Lagrange terms as in (1), but allows a favourable numerical treatment if explicitly formulated in east-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]This feature is only implemented for the integrators DAESOL and adfDAESOL so far.

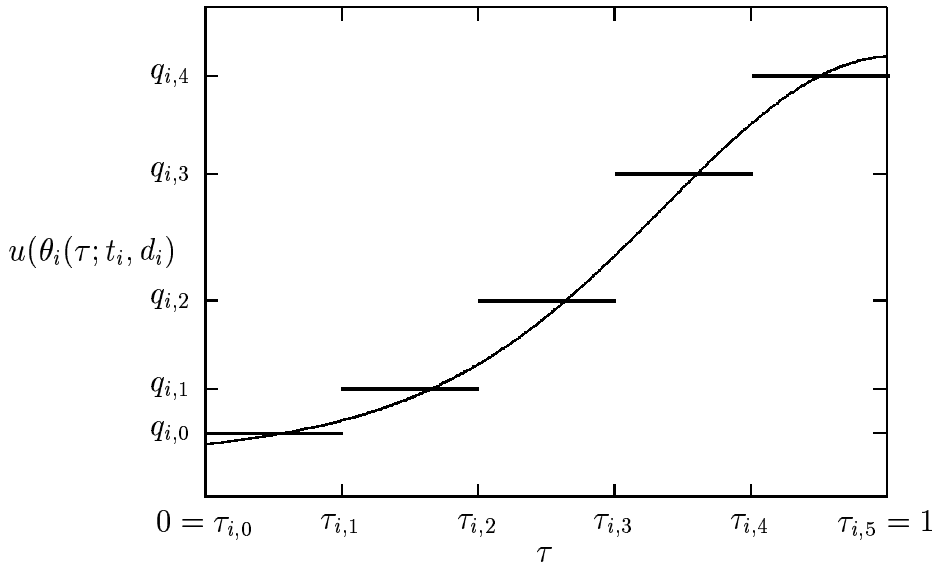



Figure 1: Piecewise constant representation of a control ($m_i = 5$).

## 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

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

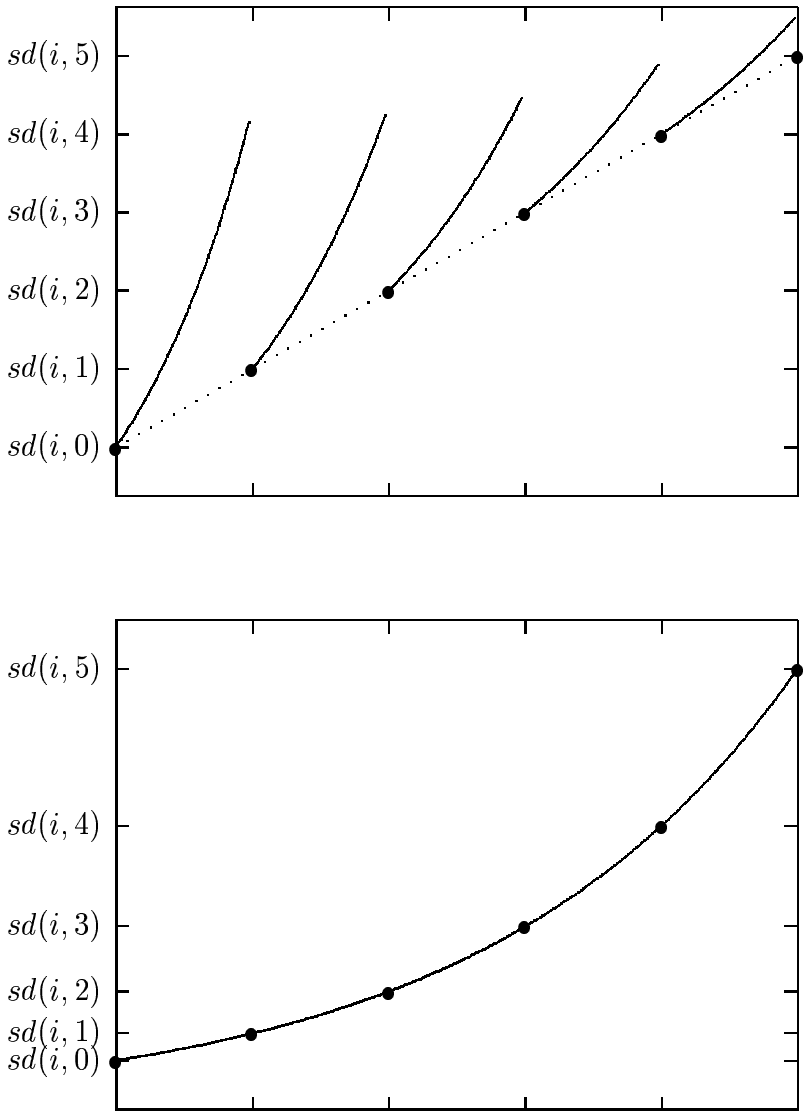
using "local" control parameters $q_{ij}$. The functions $\varphi_{ij}$ are typically vectors of polynomials[4]. If for example piecewise constant approximations are used for all control functions, we simply have $\varphi_{ij}(t, q_{i,j}) = q_{i,j}$ for $t \in I_{i,j}$, see the scalar example shown in Figure 1.

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 $i$th model stage ($j \in \{0, 1, \ldots, n_i^{ms}\}$) the initial value for the DAE solution is given by the so called *node*

[4]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.




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.

*values* $s^x_{i,j}$, $s^z_{i,j}$ for differential and algebraic states[5]. 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.

## 3.4 Discretization of Least Squares Terms

The pointwise defined least squares functions $l^p_k$ can be evaluated at all specified multiple shooting nodes, analogously to the constraint functions $r^c_k$, 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.

$$\frac{\partial^2 \|l^p_k\|^2_2}{\partial s^2_k} \approx 2 \frac{\partial l^p_k}{\partial s_k}^T \frac{\partial l^p_k}{\partial s_k},$$

which is good for small residuals $\|l^p_k\|^2_2$.

If a *continuous* least squares function $l^c_k$ 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:

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

[5]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]

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[6].

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. 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 dimensional Nonlinear Program:

$$\begin{aligned} \min_w \quad & F(w) \\ \text{subject to} \quad & \\ G(w) \quad &= \quad 0 \\ H(w) \quad &\geq \quad 0 \end{aligned} \tag{5}$$

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

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

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

with step directions $\Delta w_k$ (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:

$$\begin{aligned} \min_{\Delta w_k \in \Omega} \quad & \nabla F(w_k)^T \Delta w_k \;+\; \tfrac{1}{2} \Delta w_k^T H_k w_k \\ & \text{subject to} \\ G(w_k) + \nabla_w G(w_k)^T \Delta w_k \;&=\; 0 \\ H(w_k) + \nabla_w H(w_k)^T \Delta w_k \;&\geq\; 0 \end{aligned} \tag{6}$$

where $\Omega$ 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 $\Delta w_k$ 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 $\alpha_k$ ; they are described in appendix B.

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[7] which is then revised during the SQP iterations by appropriate update procedures (described in appendix B) 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 $\Omega$ in Equation 6) has to be specified to have a well defined QP. See e.g. appendix C, example 3, for a trust region algorithm with "exact" Hessian.

## 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 appendix D.

[7] 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.

# 4 Installation of MUSCOD-II

The MUSCOD-II package is delivered together with the linear algebra library LIBLAC (Leineweber and Jost, 1996 [LJ96]). However, for MUSCOD-II to be fully functionable, some extra software is required:

- MUSCOD-II requires BLAS routines (Lawson *et al.*, 1979 [LHKK79]; Dongarra *et al.*, 1988 [DDHH88] and 1990 [DDDH90]).
- The graphics package PGPLOT 5.2 (Pearson, 1997 [Pea97]) is used for all online graphics. It is not essential, but online graphics helps a lot to better understand possible difficulties.

Furthermore, MUSCOD-II contains some external software modules for which appropriate license must be obtained by the user (commercial products in italics):

- DAE solvers of BDF-type:
  - DAESOL (Bauer *et al.*, 1997 [BFD+97, Bau00])
  - *DDASAC* (Caracotsios and Stewart, 1985 [CS85])
- standard QP solvers, of which at least *one* has to be licensed, preferably QPSOL:
  - *NAG E04NAF* (or QPSOL) (Gill *et al.*, 1983 [GMSW83])
  - *Harwell VE17* (Goldfarb and Idnani, 1983 [GI83]; Powell, 1985 [Pow85])
  - *Harwell VE02* (Fletcher, 1971 [Fle71])
- direct linear solvers:
  - LAPACK DGETRF, DGETRS (Anderson *et al.*, 1995 [ABB+95])
  - *Harwell MA48* (Reid and Duff, 1993 [RD93] and 1996 [DR96])
- sparse numerical derivative subroutine:
  - *Harwell TD12* (Reid, 1972 [Rei72])

## 4.1 Installation steps

1. PGPLOT (Version 5.2) : To install the package PGPLOT copy the distribution file by anonymous ftp from Caltech. Use anonymous ftp (user: anonymous, password: your id username@machine) to node astro.caltech.edu (131.215.240.1).

   The distribution file is a UNIX tar file compressed with gzip. Issue the following ftp commands to retrieve the file:

   cd pub/pgplot binary get pgplot5.2.tar.gz

   The text files in this directory are also included in the tar file.

   Alternatively, the PGPLOT distribution file can be fetched from URL
   `ftp://astro.caltech.edu/pub/pgplot/pgplot5.2.tar.gz`

Follow the instructions given in the file install-unix.txt, and observe that in the file `drivers.list` the following devices have to be included (uncomment the corresponding rows by erasing the exclamation mark at the beginning):

```
PSDRIV 1 /PS        PostScript printers, monochrome, landscape        Std F77
PSDRIV 2 /VPS       Postscript printers, monochrome, portrait         Std F77
PSDRIV 3 /CPS       PostScript printers, color, landscape             Std F77
PSDRIV 4 /VCPS      PostScript printers, color, portrait              Std F77

XWDRIV 1 /XWINDOW   Workstations running X Window System             C
XWDRIV 2 /XSERVE    Persistent window on X Window System              C
```

Note that here only the installation on a Unix machine is described - for use on a Windows NT environment currently it is better to exclude graphics output of MUSCOD-II.

Provide the appropriate paths in your `.cshrc` file or equivalent, e.g.:

```
setenv PGPLOT_DIR ~/PGPLOT/PGPLIB
setenv PGPLOT_FONT ~/PGPLOT/PGPLIB/grfont.dat
setenv PGPLOT_RGB ~/PGPLOT/PGPLIB/rgb.txt
```

2. LIBLAC: choose the target platform include file `inc_*.mk` in the LIBLAC main makefile `~/MUSCOD-II/LIBLAC/makefile`, specified by the target `MACHINE` (you can create a new one in the directory `.../LIBLAC/MKINCL` if your machine is not represented). Type `make` in the `~/MUSCOD-II/LIBLAC` directory.

3. MUSCOD: choose the target platform include file in the MUSCOD main makefile `~/MUSCOD-II/MC2/makefile`, specified by the target `MACHINE` - it should be the same as in the LIBLAC main makefile. Type `make` in the directory `~/MUSCOD-II/MC2` .

   Note that the packages QPSOL, VE17 and VE02 are included in the MC2-subdirectory `.../MC2/QPS`, also the integrators DAESOL and DDASAC (Version 1996.2) are included in the subdirectory `.../MC2/IND` and are compiled automatically.

4. ADIFOR 2.0 and SparsLinC: For *optional* installation of ADIFOR 2.0 please consult `http://www.cs.rice.edu/~adifor/`. The *ADIntrinsics* and *SparsLinC* libraries should be located in a directory `~/ADIFOR` to be consistent with the MUSCOD-II makefiles.

In the MUSCOD main makefile `~/MUSCOD-II/MC2/makefile` some compilation flags `CFLAGS` can be changed in order to obtain optional algorithmic modifications. These are commented in appendix A. The same applies to some compiler flags in the source code of modules themselves, described in appendix B.

# 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 and linked with the object files and libraries of MUSCOD-II as described below.

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

The example model files `MC2_DOWBAT/SRC/dowbat.c` and `.../dowbat_f77.F` contain the Dow batch reactor problem formulated in C and alternatively in Fortran 77 and are listed in appendix G followed by the corresponding data file `.../DAT/dowbat.dat`. Additionally, it may be helpful to have a look into some of the small test problems contained in the directories `MC2_TEST/SRC` and `.../DAT`.

## 5.1 Locating the Problem Files

For the implementation of a new problem we recommend to create a directory `~/MUSCOD-II/NAME`, where all problem specific data are kept and the executable is generated. It should contain the subdirectories `NAME/SRC`, `NAME/DAT`, and `NAME/RES` for model source, data, and result files, respectively. The makefile in the directory `MUSCOD-II/MC2_DOWBAT` provides a template for the general problem makefile. It has to be edited in the following way:

1. First choose the target platform include file (same as in `LIBLAC` and `MC2`).
2. Define a new problem path by editing `DOWPATH`.
3. Choose the user-defined parts of `CFLAGS` (same as in `MC2`, see appendix A).
4. Take a look at the `LIBS` macro and edit if necessary (Cf. appendix C).
5. Define the problem object files by editing `DOWOBJS`, or `DOWOBJS_F77` respectively.
6. Edit the macro `MC2_OBJECTS` to define the actual combination of the algorithmic features. A brief description of these is given in appendix B, as well as four standard combinations.
7. Edit the targets `mc2d` or `mc2df77` that give the executable.

In addition, the makefile in the `SRC` subdirectory has to be edited to provide correct compilation of the model source.

After having set up the model, the executable can be generated by calling `make` in the directory `MUSCOD-II/NAME`. The executable can then be called by:

```
mc2d datafilename
```

**The Test Library:** By analogously adapting the makefile in the test library `MC2_TEST` you can quickly test if MUSCOD-II works well on your machine. Use one of the executables `mc2ts`, `mc2tw` or `mc2tb`, followed by a suitable problem name. The available problem names are listed if the executable is called without argument.

# 6 The Model Source File

For generation of the model source we recommend to edit e.g. the dowbat example files `dowbat.c` or `dowbat_f77.F`. For better readability it is helpful to define the relevant dimensions as preprocessor macros.

All *model functions* (F77: subroutines) are defined in this file; their syntax is described in `template.c` and `template_f77.F`. Their names are arbitrary in principle but, for better readability, should not deviate much from the conventions used in the template.

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.h` or `def_usrmod_f77.h`). 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_msolver(N, name0, name1, ..., name(N-1))` is used to select and build a list of the model solver(s) by first indicating their total number and then listing their names. Index numbers are assigned to the solvers in the order of appearance. See `def_ind.h` or `def_ind_f77.h` for a list of available model solvers. It contains the Runge-Kutte-Fehlberg integrators `def_RKFXXX` of different order for ordinary differential equations, the DAE solvers `def_DDASAC` and `def_DAESOL`, the DAE solver for sparse analytical derivatives `def_adfDAESOL` (cf. appendix H), all equipped with *internal numerical differentiation (IND)* (cf. Bock 1981 [Boc81]), and the trivial solver `def_STRANS` for transition stages. Additionally, you can specify the nullsolver `def_NULLSLV`: it is equivalent to the transition stage solver with an identity operation.

- `def_mdims(NMOS, NP, NRC, NRCE)` must be used to specify the global model dimensions: number of model stages `NMOS`, number of global parameters `NP`, total number `NRC` of *coupled* interior point constraints (i.p.c.), and number `NRCE` of those that are equality constraints (`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, ISLV)` to define a model stage with index `I`, differential and algebraic state

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

| 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` | $n^x$ | 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` | $n^u$ | 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` | $\Phi$ | Mayer term of objective |
| `lfcn` | $L$ | Lagrange term of objective |
| `lsqfcn` | $l_k^p$,$l_k^c$ | least squares residual vector function |

and control dimensions NXD,NXA, NU. Pointers to a Mayer term function mfcn (to be evaluated at the end of the stage) and a Lagrange termlfcn can be specified (otherwise they must be replaced by the NULL pointer), as well as the differential and algebraic right hand side functions ffcn, gfcn. 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. The real and integer work arrays rwh and iwh can be used to pass a common workspace to the stage functions.

Finally, the solver index number ISLV has to be chosen to select a solver out of the list defined in def_msolver(...). If the solver used is def_STRANS the stage is a transition stage; instead of ffcn a transition function ftrans as explained in section 2.1 is then used, with input dimensions NXD, NXA, NU and output dimensions corresponding to the following stage's differential state dimension.

- def_mpc(I,SCOPE,NPR, NRD, NRDE, rdfcn, rcfcn), *optional*, to define interior point constraints on stage I, with SCOPE a string "s","i","e" or "*", indicating if the following constraint functions shall be evaluated at the start point of the stage only, at the interior multiple shooting nodes, at the end point, or at all multiple shooting nodes of the stage together (note that the end point can only appear in the final model stage, otherwise the first point of the following stage must be used). The number of *local* parameters pr used in rdfcn() and rcfcn() is NPR, and NRD defines the dimension of the *decoupled* residual vector res in rdfcn of which the first NRDE componenents 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, NLSQ, lsqfcn), *optional*, to define least squares terms contributing to the objective on stage I, at the nodes defined by SCOPE. The string SCOPE may contain "s","i","e" or "*" to define least squares terms $l^p()$ at the corresponding multiple shooting nodes, as 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[8].

  The number 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()[9].

  Attention: this command is only allowed if the global compilation flags LSQ resp. LSQ_CONT are set.

- def_mio(minp,mout,mplo) is 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) has been read. (Data passed through minp supersedes

[8]The continuous least squares terms can so far only be treated by the integrators DAESOL and adfDAESOL

[9]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.

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 $i$. |

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.

# 7 The Data File

MUSCOD-II uses a keyword based data file for problem data input. The order of data items in the problem data 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.

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.

Examples for data items (the keywords are explained in the next section):

```
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
```

Note that vectors are written one element a line with element indices starting from zero.

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.)

Examples for 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

rd_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.

## 7.1 Description of Keywords

`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

default: 0

[s_itol], [s_pert]
start integration tolerance, state perturbation factor (double) (relevant only if s_spec is 2)
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)

[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)

[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
defaults: Differential State Function, Algebraic State Function, Control Function, Model Stage Duration, Global Model Parameter

[of_name]
: objective name string (Str)
default: Objective

[xd_unit], [xa_unit], [u_unit], [h_unit], [p_unit]
: state, control, duration, and parameter unit strings (StrVec)
default: empty string

[of_unit]
: objective unit string (Str)
default: empty string

[t_unit]
: time unit string (Str)
default: empty string

# A Global 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
```

# B MUSCOD-II Modules

The directory `~/MUSCOD-II/MC2` contains a number of subdirectories called *modules*. Some modules offer alternative source files and compilation flags (`#defines`) described in the following. Please do not forget to perform an additional make in the `MC2`-directory whenever you have changed some flags in the files.

The final choice between algorithmic alternatives – if they exist in different object files – is made in the `makefile` of your problem, which performs the linking (cf. 5.1).

- `MSSQP`: overall control of SQP solution process: `mssqp.c`
  Additionally to the standard object `mssqp.o`, three different object files are automatically generated: `mssqp_ch.o`, `mssqp_fdh.o` and `mssqp_gn.o` to use a constant or finite difference or a Gauss-Newton Hessian approximation (via the flags `CONST_HESSIAN` or `FDIFF_HESSIAN` or `GAUSS_NEWTON`). If you always want to use a full multiplier step (even if the variable step is relaxed by the line search) then set the flag `FULL_MULTIPLIER_STEP`. And if you want to discard the Hessian after each homotopy step then set `DISCARDHESS`.

- `PRSQP`: *partially reduced SQP* decomposition: `prsqp.c`
  The makefile uses the flag `MA48` to get a sparse version of the module (`prsqp_spa.o`) and the flag `LAPACK` to get a dense version (`prsqp_den.o`).

- `EVAL`: function and gradient evaluation routines in `eval.c`.

- `IND`: integrator interfaces and integrators (all equipped with *internal numerical differentiation*). Three libraries are contained in this module, that have to be made accessible to the linker via the `LIBS` environment variable (cf. 5.1):

- The library `ind.a` contains the Runge-Kutta-Fehlberg integrators: `rkf12s.F`, `rkf23s.F`, `rkf45s.F`, `rkf78s.F`, their interface `ind_rkfXXs.c`; the DAE solver `ddasac.F` and its interface `ind_ddasac.c`; the DAESOL interfaces `ind_daesol.c` and `ind_adfdaesol.c`; the stage transition handler `ind_strans.c`; and and dummy model stage solver `ind_nullslv.c`.
- `daesol_sparse.a` contains a sparse (MA48) version of the DAE solver `DAESOL/daesol.F`. Compilation is performed by the makefile `DAESOL/makefile_sparse`.
- `daesol_dense.a` contains a dense version of the DAE solver `DAESOL/daesol.F`. Compilation is performed by the makefile `DAESOL/makefile_dense`.

As `daesol_sparse.a` and `daesol_dense.a` use the same function names they cannot be used simultaneously.

In the files `ind_rkfXXs.c`, `ind_daesol.c`, and `ind_adfdaesol.c` it can be chosen whether the initial trial for the stepsize should be adapted or stay fixed (via the flag `FIXED_INITIAL_STEPSIZE`). By default, this option is *not* set.

- `MODEL`: user model interface `model.c`.
- `HESS`: variable metric Hessian approximation, contains four alternatives:
  - In the standard module `hess_std.c` you can choose one of the following updates using internal flags: `BFGS_UPDATE`, `SR1_UPDATE`, `PSB_UPDATE`, and `DFP_UPDATE`. Note that with the SR1 and PSB update the hessian matrices do not need to be positive definite so that you have to use a trust region strategy. For the BFGS update there exist modifications to keep the Hessian matrices positive definite: `MOD_BFGS_POWELL`, `MOD_BFGS_SWANEY`, and `MOD_BFGS_NOCEDAL`.

    Another choice to be taken by the user is the initial scaling of the Hessian matrix: either according to Plitt by setting the flag `IEST_HESS_PLITT`, or as a unit matrix by not setting this flag.
  - In the limited memory module `hess_lim.c` only the BFGS update is realized. You can choose the update modification, and the initial scaling of the Hessian matrix (using the same flags as in `hess_std.c`). 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.
  - The file `fdhess.c` supports the finite-difference Hessian approximation, and has to be used together with the object file `mssqp_fdh.o`. Note that `fdhess.c` includes `hess_std.c`.
  - The file `gnhess.c` supports the Gauss-Newton Hessian approximation, and has to be used together with the object file `mssqp_gn.o`. Note that `gnhess.c` includes `hess_std.c`.
- `SOLVE`: calculation of SQP correction step. It contains

1. `solve_slse.c` for a standard line search,
2. `solve_vmcwd.c` for a line search with a watchdog technique, and
3. `solve_tbox.c` for a boxstep trust region technique.

In the first and third module you have the choice of using a second order correction step (via the flag `SOC_STEP`).

The option `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 `SOLVE` module calls the lower level `COND` module, that by itself calls the QP solvers of module `QPS`.

- `COND`: The condensing algorithm in `cond_std.c` 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 in module `QPS`.

  Here 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.

  The file `cond_min.c` is normally *not* used. It leaves out the condensing of the state variables.

- `QPS`: QP solver and interfaces. The library `qps.a` contains the interfaces `qps_qpsol.c`, `qps_ve02.c`, `qps_ve17.c` and the corresponding solvers.

  `qpsol.F` is by far the most important QP solver and it should be checked that the machine precision in functions `X02AJF` and `X02AMF` is correctly adjusted.

  In its interface `qps_qpsol.c` the active set is stabilized by setting the flag `ACTSTAB` (default: active) that introduces a relaxed tolerance for previously inactive constraints.

- `TCHK`: termination check in file `tchk.c`. The KKT-tolerance is calculated by neglecting all simple bounds that are assumed to be satisfied.

- `MSPLOT`: graphics routines contained in `msplot.c`.

- PDAUX, UTIL, SCALE, ADFAUX: auxiliary routines for problem data (pdaux.c), input/output (util.c), scaling (scale.c) and handling of ADIFOR generated analytical derivatives (adfaux.c).

# C Five examples of MUSCOD-II algorithms

With the choice of MUSCOD-II modules you can define the following algorithms. In all examples we use the standard condensing strategy. Note that the following lines are taken out of your model makefile where the object files for your MUSCOD-II algorithms are specified and linked.

The first three algorithms employ a dense version of the PRSQP strategy (they can be found in the test library's MC2_TEST/makefile) whereas the last example shows how sparse analytical derivatives can be used (to be found e.g. in MC2_BDIST/makefile).

In the following, we list the MC2-objects that are used to build the different algorithmic modifications.

1. **A line search algorithm with a standard BFGS update**

```
MC2OBJS = \
  ${MC2PATH}/MODEL/model.o \
  ${MC2PATH}/PDAUX/pdaux.o \
  ${MC2PATH}/MSSQP/mssqp.o \
  ${MC2PATH}/EVAL/eval.o \
  ${MC2PATH}/SCALE/scale.o \
  ${MC2PATH}/PRSQP/prsqp_den.o \
  ${MC2PATH}/MSPLOT/msplot.o \
  ${MC2PATH}/HESS/hess_std.o \
  ${MC2PATH}/SOLVE/solve_slse.o \
  ${MC2PATH}/COND/cond_std.o \
  ${MC2PATH}/TCHK/tchk.o \
  ${MC2PATH}/UTIL/util.o
```

   This is a standard SQP algorithm close to the original implementation of Powell [Pow78].

2. **A line search algorithm with the watchdog technique and a limited memory BFGS update**

   Only the HESS and the SOLVE module have to be changed:

```
... \
${MC2PATH}/HESS/hess_lim.o \
${MC2PATH}/SOLVE/solve_vmcwd.o \
...
```

The watchdog technique is more efficient for curved constraints (as it allows some full SQP iterations without immediate reduction of the merit function), and the limited memory updates help to preserve good conditioning of the hessian approximation.

3. **A box step trust region algorithm with an exact (finite difference) Hessian approximation**

   Here, compared to the first example, `MSSQP`, `HESS` and `SOLVE` are changed:

   ```
   ... \
   ${MC2PATH}/MSSQP/mssqp_fdh.o \
   ... \
   ${MC2PATH}/HESS/fd_hess.o \
   ${MC2PATH}/SOLVE/solve_tbox.o \
   ...
   ```

   This strategy has proven especially useful in multistage problems with free stage durations. However, for DAE or stiff systems it is not yet implemented.

4. **A line search algorithm with the watchdog technique and a limited memory BFGS update using *sparse* derivatives**

   Compared to the second example, only `PRSQP` is changed:

   ```
   ... \
   ${MC2PATH}/PRSQP/prsqp_spa.o \
   ... \
   ${MC2PATH}/HESS/hess_lim.o \
   ${MC2PATH}/SOLVE/solve_vmcwd.o \
   ...
   ```

5. **A Gauss-Newton method**

   Compared to the other examples, only `MSSQP` and `HESS` are changed:

   ```
   ... \
   ${MC2PATH}/MSSQP/mssqp_gn.o \
   ... \
   ${MC2PATH}/HESS/gnhess.o \
   ...
   ```

   This method is only applicable if the objective function is composed of least squares terms. It is mainly used for parameter estimation or tracking problems, and is especially successful if the residuals are expected to be small in the solution ("good fit"). See e.g. Bock [Boc87] for details on the Gauss-Newton Method.

**Libraries** The first three algorithms all use dense PRSQP and are linked with the following libraries:

```
LIBS    = -L${MC2PATH}/IND \
          -L${MC2PATH}/QPS \
          -L${LACPATH} \
          -L${HOME}/PGPLOT/PGPLIB \
          ${SYSPATHS} \
           -lind -ldaesol_dense -lqps -llapack -llac -lblas -lpgplot -lX11
          ${SYSLIBS}
```

The last example is somewhat different in that it employs a *sparse* version of the PRSQP strategy and should therefore also use a sparse version of the DAE integrator DAESOL. The previous last line should be replaced by:

```
          ... \
          ${SYSPATHS} \
          -lind -ldaesol_sparse -lqps  -llac -lblas -lpgplot -lX11
          ${SYSLIBS}
```

The sparse version of PRSQP and DAESOL can of course be combined with any of the other algorithmic features.

**Remark on analytical derivatives:** The sparse versions of PRSQP and DAESOL can also be used with analytical derivatives in ADIFOR 2.0 format. Please note that analytical derivatives have to be provided by the user, e.g. by following the instructions in appendix H.

# D Runtime Options

After compilation and linking, the executable (e.g. target `mc2x`) can be called with a number of options. The syntax is as follows:

```
mc2x -option1 -option2 ... -optionN  name
```

Here, only the last argument `name` is mandatory. It determines the data file to be used: `./DAT/name.dat` and the name of the output files `./RES/name.txt` (for detailed solution information), `./RES/name.log` (chronological history of solution process), and `./RES/name.bin` (binary solution information for warm starts).

The possible options depend on the global compilation flags (cf. appendix A); by calling the executable `mc2x` without any argument a list of the active ones is shown, together with a quick reference and the default values.

- Option `a`: Termination criterion: KKT-tolerance *kkttol* of the solution has to be lower than the given value *acc*. Example: `-a1e-6` (which gives the default $acc = 10^{-6}$).

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

- Option `h`: 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. Option `h` requires two arguments: `-h`*itol*,*N*. Example: `-h1e-3,1`.

- Option `f`: 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`.

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

- Option `p`: 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 printing even the internal QP solution procedure. Example: `-p0` (default). This option appears only when the global compilation flag `NDEBUG` is not activated in the `MC2/makefile`.

- Option `r`: 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`.

- Option `e`: "evaluate twice" – only possible if `PRSQP` is employed. 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]. Use `-e`.

- Option `s`: stop after each SQP iteration and wait for keystroke. Example: `-s`.

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

# E Templates for the Source File

## Source File, ANSI C Version: (template.c)

```
/*
 *
 *  MUSCOD-II/MC2_TEST/SRC/template.c
 *  (c) Daniel B. Leineweber, 1995--1997
 *
 *  user model definition file template (C version)
 *
 *
 */


#include "def_usrmod.h"
#include "def_ind.h"


static void mfcn(
  double *ts,    /* physical time (I) */
  double *sd,    /* differential state vector (I) */
  double *sa,    /* algebraic state vector (I) */
  double *p,     /* global model parameter vector (I) */
  double *mval,  /* Mayer term value (0) */
  long   *dpnd,  /* argument dependency code (I/0) */
  long   *info   /* error code (0) */
)
/*
 *  Mayer term of objective function (``end point term'')
 */
{
  if (*dpnd) {
    *dpnd = MFCN_DPND(*ts, *sd, *sa, *p);
    return;
  }

}

static void lfcn(
  double *t,     /* physical time (I) */
  double *xd,    /* differential state vector (I) */
  double *xa,    /* algebraic state vector (I) */
  double *u,     /* control vector (I) */
  double *p,     /* global model parameter vector (I) */
  double *lval,  /* Lagrange term value (integrand) (0) */
  double *rwh,   /* real work array (I/0) */
  long *iwh,     /* integer work array (I/0) */
  long *info     /* error code (0) */
)
/*
 *  Lagrange term of objective function (``integral term'')
 */
{

}

static void afcn(
  double *t,     /* physical time (I) */
  double *xd,    /* differential state vector (I) */
  double *xa,    /* algebraic state vector (I) */
  double *u,     /* control vector (I) */
  double *p,     /* global model parameter vector (I) */
  double *amat,  /* left-hand side matrix (0) */
  long   *lda,   /* leading dimension of amat (I) */
  double *rwh,   /* real work array (I/0) */
  long   *iwh,   /* integer work array (I/0) */
  long   *info   /* error code (0) */
)
/*
 *  left-hand side matrix of DAE system
 */
{

}

static void ffcn(
  double *t,    /* physical time (I) */
  double *xd,   /* differential state vector (I) */
  double *xa,   /* algebraic state vector (I) */
  double *u,    /* control vector (I) */
  double *p,    /* global model parameter vector (I) */
  double *rhs,  /* right-hand side vector (0) */
  double *rwh,  /* real work array (I/0) */
  long *iwh,    /* integer work array (I/0) */
  long *info    /* error code (0) */
)
/*
 *  differential right-hand side of ODE or DAE system
 */
{

}

static void gfcn(
  double *t,    /* physical time (I) */
  double *xd,   /* differential state vector (I) */
  double *xa,   /* algebraic state vector (I) */
  double *u,    /* control vector (I) */
  double *p,    /* global model parameter vector (I) */
  double *rhs,  /* right-hand side vector (0) */
  double *rwh,  /* real work array (I/0) */
  long *iwh,    /* integer work array (I/0) */
  long *info    /* error code (0) */
)
/*
 *  algebraic right-hand side of DAE system
 */
{

}

static void rfcn(
  double *ts,   /* physical time (I) */
  double *sd,   /* differential state vector (I) */
  double *sa,   /* algebraic state vector (I) */
  double *u,    /* control vector (I) */
  double *p,    /* global model parameter vector (I) */
  double *pr,   /* local i.p.c. parameter vector (I) */
  double *res,  /* i.p.c. residual (0) */
  long *dpnd,   /* argument dependency code (I/0) */
  long *info    /* error code (0) */
)
/*
 *  decoupled or coupled interior point constraint (i.p.c.) residual
 */
{
  if (*dpnd) {
    *dpnd = RFCN_DPND(*ts, *sd, *sa, *u, *p, *pr);
    return;
  }

}

static void mout(
  long   *imos,      /* index of model stage (I) */
  long   *imsn,      /* index of m.s. node on current model stage (I) */
  double *ts,        /* physical time at m.s. node (I) */
  double *te,        /* physical time at end of m.s. interval (I) */
  double *sd,        /* differential states at m.s. node (I) */
  double *sa,        /* algebraic states at m.s. node (I) */
  double *u,         /* controls at m.s. node (I) */
  double *udot,      /* control slopes at m.s. node (I) */
  double *ue,        /* controls at end of m.s. interval (I) */
  double *uedot,     /* control slopes at end of m.s. interval (I) */
  double *p,         /* global model parameters (I) */
  double *pr,        /* local i.p.c. parameters (I) */
  double *mul_ccxd,  /* multipliers of continuity conditions (I) */
  double *mul_rd,    /* multipliers of decoupled i.p.c. (I) */
  double *mul_rc,    /* multipliers of coupled i.p.c. (I) */
  double *rwh,       /* real work array (I) */
  long   *iwh        /* integer work array (I) */
)
/*
 *  model output function (optional)
 */
{

}

void def_model(void)
/*
 *  formal definition of multistage optimal control problem using
 *
 *    def_msolver()  ...  define model solver(s)
 *    def_mdims()    ...  define global model dimensions
 *    def_mstage()   ...  define model stage
```

```
 *    def_mpc()       ...  define model point constraints
 *    def_lsq()       ...  define model point least squares terms
 *    def_mio()       ...  define model input/output functions (optional)
 *
 *  see "def_usrmod.h" for argument descriptions and "def_ind.h" for list
 *  of available model solvers
 */
{

}
```

## Source File, Fortran 77 Version: (template_f77.F)

```
*
*
*  MUSCOD-II/MC2_TEST/SRC/template_f77.F
*  (c) Daniel B. Leineweber, 1995--1997
*
*  user model definition file template (Fortran 77 version)
*  see "template.c" for argument descriptions of subroutines
*  mfcn, lfcn, ffcn, gfcn, rfcn
*
*
*


#include "def_usrmod_f77.h"
#include "def_ind_f77.h"


      subroutine mfcn(ts, sd, sa, p, mval, dpnd, info)
      implicit none
      real*8  ts, sd(0:*), sa(0:*), p(0:*)
      real*8  mval
      integer dpnd, info

      if (dpnd .ne. 0) then
        dpnd = MFCN_DPND(ts, sd(0), sa(0), p(0))
        return
      endif

      return
      end


      subroutine lfcn(t, xd, xa, u, p, lval, rwh, iwh, info)
      implicit none
      real*8  t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      real*8  lval
      integer iwh(0:*), info

      return
      end


      subroutine afcn(t, xd, xa, u, p, amat, lda, rwh, iwh, info)
      implicit none
      real*8  t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      real*8  amat(0:lda-1,*)
      integer lda, iwh(0:*), info

      return
      end


      subroutine ffcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
      implicit none
      real*8  t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      real*8  rhs(0:*)
      integer iwh(0:*), info

      return
      end


      subroutine gfcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
      implicit none
      real*8  t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      real*8  rhs(0:*)
      integer iwh(0:*), info

      return
      end


      subroutine rfcn(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
      real*8  ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      real*8  res(0:*)
      integer dpnd, info

      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(ts, sd(0), sa(0), u(0), p(0), pr(0))
        return
      endif

      return
      end


      subroutine mout(imos, imsn, ts, te, sd, sa, u, udot, ue,
     &  uedot, p, pr, mul_ccxd, mul_rd, mul_rc, rwh, iwh)
      implicit none
      real*8  ts, te, sd(0:*), sa(0:*), u(0:*), udot(0:*)
      real*8  ue(0:*), uedot(0:*), p(0:*), pr(0:*), mul_ccxd(0:*)
      real*8  mul_rd(0:*), mul_rc(0:*), rwh(0:*)
      integer imos, imsn, iwh(0:*)

      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(ts, sd(0), sa(0), u(0), p(0), pr(0))
        return
      endif

      return
      end


      subroutine def_model()
      implicit none
*
*     formal definition of multistage optimal control problem using
*
*       def_msolver()   ...  define model solver(s)
*       def_mdims()     ...  define global model dimensions
*       def_mstage()    ...  define model stage
*       def_mpc()       ...  define model point constraints
*       def_lsq()       ...  define model point least squares terms
*       def_mio()       ...  define model input/output functions (optional)
*
*     see "def_usrmod_f77.h" for argument descriptions and "def_ind_f77.h"
*     for list of available model solvers
*

      return
      end
```

# F Available ODE/DAE Solvers

The available ODE or DAE solver definition routines are listed in the `INCLUDE` file `def_ind.h`(`def_ind_f77.h`, resp.) as follows:

`def_ind.h`

```
/*
 *  MUSCOD-II/MC2/INCLUDE/def_ind.h
 */

void def_RKF12S();
void def_RKF23S();
void def_RKF45S();
void def_RKF78S();

void def_RKF7B();

void def_METANB();
void def_DAESOL();
void def_adfDAESOL();
void def_DDASAC();
void def_adfDDASAC();

void def_STRANS();

void def_NULLSLV();
```

The first five solvers are based on the Runge-Kutta-Fehlberg method and are suitable for ODE only. The following five solvers are also suitable for DAE, where the prefix `adf` denotes a sparse version.

`def_STRANS` is the solver used for transition stages, where the right hand side `ffcn` is interpreted as a transition function.

`def_NULLSLV` is a dummy solver that just defines a constant transition. It can be used to define NLP-Problems without any underlying dynamic system.

Please note that the use of continuous least squares terms (`def_lsq(...,"")CLSQ` global compilation flag) requires

# G Example: DAE Test Problem dowbat

## Source File, ANSI C Version: (dowbat.c)

```
/*
 *
 *  MUSCOD-II/MC2_TEST/SRC/dowbat.c
 *  (c) Daniel B. Leineweber, 1996--1997
 *
 *  batch reactor example of Dow Chemical Company, modified
 *  (Caracotsios and Stewart, 1985; Biegler, Damiano, and Blau, 1986)
 *
 *  $Id: dowbat.c,v 1.1.1.1 1998/04/27 08:40:31 daniel Exp $
 *
 */

#include <math.h>
#include "def_usrmod.h"
#include "def_ind.h"

#define  NMOS   2   /* # of model stages */
#define  NP     10  /* # of global model parameters */
#define  NRC    1   /* total # of coupled i.p.c. */
#define  NRCE   1   /* # of coupled equality i.p.c. */

#define  NXD    6   /* # of differential states */
#define  NXA    4   /* # of algebraic states */
#define  NU     1   /* # of control functions */
#define  NPR    0   /* # of local i.p.c. parameters */

#define  NRD_S  6   /* total # of decoupled i.p.c. in S */
#define  NRDE_S 6   /* # of decoupled equality i.p.c. in S */
#define  NRD_I  1   /* total # of decoupled i.p.c. in I */
#define  NRDE_I 0   /* # of decoupled equality i.p.c. in I */
#define  NRD_E  1   /* total # of decoupled i.p.c. in E */
#define  NRDE_E 0   /* # of decoupled equality i.p.c. in E */

/*
 *  xd[0]  ...  [HA] + [A-]
 *  xd[1]  ...  [BM]
 *  xd[2]  ...  [HABM] + [ABM-]
 *  xd[3]  ...  [AB]
 *  xd[4]  ...  [MBMH] + [MBM-]
 *  xd[5]  ...  [M-]
 *
 *  xa[0]  ...  -log([H+])
 *  xa[1]  ...  [A-]
 *  xa[2]  ...  [ABM-]
 *  xa[3]  ...  [MBM-]
 *
 *   u[0]  ...  T
 *
 *   p[0]  ...  alph1
 *   p[1]  ...  beta1 = E1/R
 *   p[2]  ...  alph_1
 *   p[3]  ...  beta_1 = E_1/R
 *   p[4]  ...  alph2
 *   p[5]  ...  beta2 = E2/R
 *   p[6]  ...  K1
 *   p[7]  ...  K2
 *   p[8]  ...  K3
 *   p[9]  ...  [Q+] (= initial catalyst concentration)
 *
 */

static void mfcn(double *ts, double *sd, double *sa,
  double *p, double *mval, long *dpnd, long *info)
{
  if (*dpnd) {
    *dpnd = MFCN_DPND(*ts, 0, 0, *p);
    return;
  }
  *mval = *ts + 100.0*p[9];
}

static void ffcn(double *t, double *xd, double *xa, double *u,
  double *p, double *rhs, double *rwh, long *iwh, long *info)
{
  double k1, k_1, k2, k3, k_3;
  double rr1, rr2, rr3;
  k1  = p[0]*exp(-p[1]/u[0]);
  k_1 = p[2]*exp(-p[3]/u[0]);
  k2  = p[4]*exp(-p[5]/u[0]);
  k3  = k1;
  k_3 = 0.5*k_1;
  rr1 = k1*xd[1]*xd[5] - k_1*xa[3];
  rr2 = k2*xd[1]*xa[1];
  rr3 = k3*xd[3]*xd[5] - k_3*xa[2];
  rhs[0] = -rr2;
  rhs[1] = -rr1 - rr2;
  rhs[2] =  rr2 + rr3;
  rhs[3] = -rr3;
  rhs[4] =  rr1;
  rhs[5] = -rr1 - rr3;
}

static void gfcn(double *t, double *xd, double *xa, double *u,
  double *p, double *rhs, double *rwh, long *iwh, long *info)
{
  double powxa;
  powxa = pow(10.0, -xa[0]);
  rhs[0] = p[9] - xd[5] + powxa - xa[1] - xa[2] - xa[3];
  rhs[1] = xa[1] - p[7]*xd[0]/(p[7] + powxa);
  rhs[2] = xa[2] - p[8]*xd[2]/(p[8] + powxa);
  rhs[3] = xa[3] - p[6]*xd[4]/(p[6] + powxa);
}

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, *p, 0);
    return;
  }
  res[0] = sd[0] - 1.5776;
  res[1] = sd[1] - 8.32;
  res[2] = sd[2];
  res[3] = sd[3];
  res[4] = sd[4];
  res[5] = sd[5] - p[9];
}

static void rdfcn_i(double *ts, double *sd, double *sa, double *u,
  double *p, double *pr, double *res, long *dpnd, long *info)
{
  if (*dpnd) {
    *dpnd = RFCN_DPND(*ts, *sd, 0, 0, 0, 0);
    return;
  }
  res[0] = 2.0*(*ts)*(*ts) - sd[3];
}

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[3] - 1.0;
}

static void rcfcn_i(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] = -4.0*(*ts);
}

static void rcfcn_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;
}

void dowbat(void)
{
  def_msolver(1, def_DDASAC);
  def_mdims(NMOS, NP, NRC, NRCE);
  def_mstage(
```

```
    0,
    NXD, NXA, NU,
    NULL, NULL,
    0, 0, 0, NULL, ffcn, gfcn,
    NULL, NULL,
    0
  );
  def_mstage(
    1,
    NXD, NXA, NU,
    mfcn, NULL,
    0, 0, 0, NULL, ffcn, gfcn,
    NULL, NULL,
    0
  );
  def_mpc(
    0,
    "Start Point",
    NPR,
    NRD_S, NRDE_S,
    rdfcn_s, NULL
  );
  def_mpc(
    0,
    "Interior Points",
    NPR,
    NRD_I, NRDE_I,
    rdfcn_i, NULL
  );
  def_mpc(
    1,
    "Start Point",
    NPR,
    NRD_I, NRDE_I,
    rdfcn_i, rcfcn_i
  );
  def_mpc(
    1,
    "End Point",
    NPR,
    NRD_E, NRDE_E,
    rdfcn_e, rcfcn_e
  );
}
```

## Source File, Fortran 77 Version: (dowbat_f77.F)

```
*
*
*  MUSCOD-II/MC2_TEST/SRC/dowbat_f77.F
*  (c) Daniel B. Leineweber, 1997
*
*  batch reactor example of Dow Chemical Company, modified
*  (Caracotsios and Stewart, 1985 Biegler, Damiano, and Blau, 1986)
*
*  $Id: dowbat_f77.F,v 1.1.1.1 1998/04/27 08:40:31 daniel Exp $
*
*

#include "def_usrmod_f77.h"
#include "def_ind_f77.h"

#define  NMOS   2
#define  NP     10
#define  NRC    1
#define  NRCE   1

#define  NXD    6
#define  NXA    4
#define  NU     1
#define  NPR    0

#define  NRD_S  6
#define  NRDE_S 6
#define  NRD_I  1
#define  NRDE_I 0
#define  NRD_E  1
#define  NRDE_E 0

*
*  xd(0)  ...  [HA] + [A-]
*  xd(1)  ...  [BM]
*  xd(2)  ...  [HABM] + [ABM-]
*  xd(3)  ...  [AB]
*  xd(4)  ...  [MBMH] + [MBM-]
*  xd(5)  ...  [M-]
*
*  xa(0)  ...  -log([H+])
*  xa(1)  ...  [A-]
*  xa(2)  ...  [ABM-]
*  xa(3)  ...  [MBM-]
*
*   u(0)  ...  T
*
*   p(0)  ...  alph1
*   p(1)  ...  beta1 = E1/R
*   p(2)  ...  alph_1
*   p(3)  ...  beta_1 = E_1/R
*   p(4)  ...  alph2
*   p(5)  ...  beta2 = E2/R
*   p(6)  ...  K1
*   p(7)  ...  K2
*   p(8)  ...  K3
*   p(9)  ...  [Q+] (= initial catalyst concentration)
*

      subroutine dow_mfcn(ts, sd, sa, p, mval, dpnd, info)
      implicit none
      double precision ts, sd(0:*), sa(0:*), p(0:*), mval
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = MFCN_DPND(ts, 0, 0, p(0))
        return
      endif
      mval = ts + 100.0D0*p(9)
      return
      end

      subroutine dow_ffcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
      implicit none
      double precision t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      double precision rhs(0:*)
      integer iwh(0:*), info
      double precision k1, k_1, k2, k3, k_3
      double precision rr1, rr2, rr3
      k1  = p(0)*exp(-p(1)/u(0))
      k_1 = p(2)*exp(-p(3)/u(0))
      k2  = p(4)*exp(-p(5)/u(0))
      k3  = k1
      k_3 = 0.5*k_1
      rr1 = k1*xd(1)*xd(5) - k_1*xa(3)
      rr2 = k2*xd(1)*xa(1)
      rr3 = k3*xd(3)*xd(5) - k_3*xa(2)
      rhs(0) = -rr2
      rhs(1) = -rr1 - rr2
      rhs(2) =  rr2 + rr3
      rhs(3) = -rr3
      rhs(4) =  rr1
      rhs(5) = -rr1 - rr3
      return
      end

      subroutine dow_gfcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
      implicit none
      double precision t, xd(0:*), xa(0:*), u(0:*), p(0:*), rwh(0:*)
      double precision rhs(0:*)
      integer iwh(0:*), info
      double precision powxa
      powxa = 10.0D0**(-xa(0))
      rhs(0) = p(9) - xd(5) + powxa - xa(1) - xa(2) - xa(3)
      rhs(1) = xa(1) - p(7)*xd(0)/(p(7) + powxa)
      rhs(2) = xa(2) - p(8)*xd(2)/(p(8) + powxa)
      rhs(3) = xa(3) - p(6)*xd(4)/(p(6) + powxa)
      return
      end

      subroutine dow_rdfcns(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
      double precision ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      double precision res(0:*)
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(0, sd(0), 0, 0, p(0), 0)
        return
      endif
      res(0) = sd(0) - 1.5776D0
      res(1) = sd(1) - 8.32D0
      res(2) = sd(2)
      res(3) = sd(3)
      res(4) = sd(4)
      res(5) = sd(5) - p(9)
      return
      end

      subroutine dow_rdfcni(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
```

```
      double precision ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      double precision res(0:*)
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(ts, sd(0), 0, 0, 0, 0)
        return
      endif
      res(0) = 2.0D0*ts*ts - sd(3)
      return
      end

      subroutine dow_rdfcne(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
      double precision ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      double precision res(0:*)
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(0, sd(0), 0, 0, 0, 0)
        return
      endif
      res(0) = sd(3) - 1.0D0
      return
      end

      subroutine dow_rcfcni(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
      double precision ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      double precision res(0:*)
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(ts, 0, 0, 0, 0, 0)
        return
      endif
      res(0) = -4.0D0*ts
      return
      end

      subroutine dow_rcfcne(ts, sd, sa, u, p, pr, res, dpnd, info)
      implicit none
      double precision ts, sd(0:*), sa(0:*), u(0:*), p(0:*), pr(0:*)
      double precision res(0:*)
      integer dpnd, info
      if (dpnd .ne. 0) then
        dpnd = RFCN_DPND(ts, 0, 0, 0, 0, 0)
        return
      endif
      res(0) = ts
      return
      end

      subroutine dowbatf77()
      implicit none
      external def_DDASAC, NULL
      external dow_mfcn, dow_ffcn, dow_gfcn
      external dow_rdfcns, dow_rdfcni, dow_rdfcne
      external dow_rcfcni, dow_rcfcne
      call def_msolver(1, def_DDASAC)
      call def_mdims(NMOS, NP, NRC, NRCE)
      call def_mstage(
     &  0,
     &  NXD, NXA, NU,
     &  NULL, NULL,
     &  0, 0, 0, NULL, dow_ffcn, dow_gfcn,
     &  NULL, NULL,
     &  0)
      call def_mstage(
     &  1,
     &  NXD, NXA, NU,
     &  dow_mfcn, NULL,
     &  0, 0, 0, NULL, dow_ffcn, dow_gfcn,
     &  NULL, NULL,
     &  0)
      call def_mpc(
     &  0,
     &  'Start Point',
     &  NPR,
     &  NRD_S, NRDE_S,
     &  dow_rdfcns, NULL)
      call def_mpc(
     &  0,
     &  'Interior Points',
     &  NPR,
     &  NRD_I, NRDE_I,
     &  dow_rdfcni, NULL)
      call def_mpc(
     &  1,
     &  'Start Point',
     &  NPR,
     &  NRD_I, NRDE_I,
     &  dow_rdfcni, dow_rcfcni)
      call def_mpc(
     &  1,
     &  'End Point',
     &  NPR,
     &  NRD_E, NRDE_E,
     &  dow_rdfcne, dow_rcfcne)
      return
      end
```

## Data File: (dowbat.dat)

```
*
*
*  MUSCOD-II/MDAT/dowbat.dat
*  (c) Daniel B. Leineweber, 1996--1997
*
*  batch reactor example of Dow Chemical Company
*  (Caracotsios and Stewart, 1985; Biegler, Damiano, and Blau, 1986)
*
*  $Id: dowbat.dat,v 1.1.1.1 1998/04/27 08:40:31 daniel Exp $
*
*

*
*  xd[0]  ...  [HA] + [A-]
*  xd[1]  ...  [BM]
*  xd[2]  ...  [HABM] + [ABM-]
*  xd[3]  ...  [AB]
*  xd[4]  ...  [MBMH] + [MBM-]
*  xd[5]  ...  [M-]
*
*  xa[0]  ...  -log([H+])
*  xa[1]  ...  [A-]
*  xa[2]  ...  [ABM-]
*  xa[3]  ...  [MBM-]
*
*   u[0]  ...  T
*
*   p[0]  ...  alph1
*   p[1]  ...  beta1 = E1/R
*   p[2]  ...  alph_1
*   p[3]  ...  beta_1 = E_1/R
*   p[4]  ...  alph2
*   p[5]  ...  beta2 = E2/R
*   p[6]  ...  K1
*   p[7]  ...  K2
*   p[8]  ...  K3
*   p[9]  ...  [Q+] (= initial catalyst concentration)
*

*  # of multiple shooting intervals on each model stage
nshoot
0: 6
1: 6

*  global model parameter start values, scale factors, and bounds
*  investigator (2) estimates (R = 1.987 cal/gmol/K)
p
0: 1.3708E12
1: 9.2984E03
2: 1.6215E20
3: 1.3108E04
4: 5.2282E12
5: 9.5999E03
6: 2.575E-16
7: 4.876E-14
8: 1.7884E-16
9: 0.0131

p_sca
0: 1.3708E12
1: 9.2984E03
2: 1.6215E20
3: 1.3108E04
4: 5.2282E12
5: 9.5999E03
6: 2.575E-16
7: 4.876E-14
8: 1.7884E-16
9: 0.0131

p_min
0: 1.3708E12
1: 9.2984E03
2: 1.6215E20
3: 1.3108E04
```

```
4: 5.2282E12
5: 9.5999E03
6: 2.575E-16
7: 4.876E-14
8: 1.7884E-16
9: 0.0

p_max
0: 1.3708E12
1: 9.2984E03
2: 1.6215E20
3: 1.3108E04
4: 5.2282E12
5: 9.5999E03
6: 2.575E-16
7: 4.876E-14
8: 1.7884E-16
9: 0.0262

p_fix
0: 1
1: 1
2: 1
3: 1
4: 1
5: 1
6: 1
7: 1
8: 1
9: 0

*  model stage duration start values, scale factors, and bounds
h
0: 2.175
1: 7.825

h_sca
0: 1.0
1: 1.0

h_min
0: 0.025
1: 0.100

h_max
0: 2.5
1: 10.0

*  differential state specification mode
s_spec
2

*  differential state start values, scale factors, and bounds
sd(0,S)
0: 1.5776
1: 8.32
2: 0.0
3: 0.0
4: 0.0
5: 0.0131

sd_sca(*,*)
0: 1.0
1: 4.0
2: 1.0
3: 1.0
4: 1.0
5: 0.015

sd_min(*,*)
0: 0.0
1: 0.0
2: 0.0
3: 0.0
4: 0.0
5: 0.0

sd_max(*,*)
0: 2.0
1: 10.0
2: 2.0
3: 2.0
4: 2.0
5: 0.03

*  algebraic state start values, scale factors, and bounds
sa(*,*)
0: 7.0
1: 1.0E-8
2: 0.0
3: 0.0

sa_sca(*,*)
0: 10.0
1: 0.010
2: 1.0E-5
3: 1.0E-5

sa_min(*,*)
0: 4.0
1: 0.0
2: 0.0
3: 0.0

sa_max(*,*)
0: 14.0
1: 0.03
2: 1.0E-4
3: 1.0E-4

*  algebraic RHS scale factors
g_sca(*,*)
0: 0.010
1: 0.010
2: 1.0E-5
3: 1.0E-5

*  control parameterization types
u_type(*)
0: 1

*  control start values, scale factors, and bounds
u(*,*)
0: 373.15

u_sca(*,*)
0: 343.15

u_min(*,*)
0: 293.15

u_max(*,*)
0: 393.15

*  decoupled i.p.c. scale factors
rd_sca(0,S)
0: 1.0
1: 4.0
2: 1.0
3: 1.0
4: 1.0
5: 0.015

rd_sca(0,I)
0: 1.0

rd_sca(1,S)
0: 1.0

rd_sca(1,E)
0: 1.0

*  coupled i.p.c. scale factors
rc_sca
0: 1.0

*  objective scale and expected range; # of values in history plot
of_sca
1.0

of_min
0.0

of_max
15.0

nhist
30

*  names
of_name
Production Cost

h_name
0: Duration of 1st Stage
1: Duration of 2nd Stage

p_name
0: !alph1
1: !beta1
```

```
2: !alph_1
3: !beta_1
4: !alph2
5: !beta2
6: !K1
7: !K2
8: !K3
9: [Q+]

u_name
0: Temperature

xd_name
0: [HA] + [A\u-\d]
1: [BM]
2: [HABM] + [ABM\u-\d]
3: [AB]
4: [MBMH] + [MBM\u-\d]
5: [M\u-\d]

xa_name
0: -log([H\u+\d])
1: [A\u-\d]
2: [ABM\u-\d]
3: [MBM\u-\d]

*  units
h_unit
0: hr
1: hr

p_unit
0: !alph1
1: !beta1
2: !alph_1
3: !beta_1
4: !alph2
5: !beta2
6: !K1
7: !K2
8: !K3
9: gmol/kg

u_unit
0: K

xd_unit
0: gmol/kg
1: gmol/kg
2: gmol/kg
3: gmol/kg
4: gmol/kg
5: gmol/kg

xa_unit
0: -
1: gmol/kg
2: gmol/kg
3: gmol/kg

t_unit
hr
```

# H Using Analytical Derivatives generated by ADIFOR

MUSCOD-II supports the use of analytical derivatives of the model functions. So far, only *sparse* derivatives are implemented for the DAE solver DAESOL. Please note that the SparsLinC and the ADIntrinsics libraries have to be accessible to the linker (e.g. in the directory `~/ADIFOR`, cf. section 4).

If ADIFOR 2.0 is installed on your machine you can generate the sparse derivative code of the dynamic model FORTRAN subroutines, say, `name_ffcn()`, `name_gfcn()`, and `name_lfcn()`. The derivatives are assigned the ADIFOR default names `g_name_ffcn()`, `g_name_gfcn()`, and `g_name_lfcn()`.

How to set up MUSCOD-II for analytical derivatives, and how to generate the sparse derivatives in a suitable format, is described in the following two subsections.

## H.1 Preparing MUSCOD-II for sparse analytical derivatives

For dealing with sparse ADIFOR derivatives the modules `ADFAUX` and `ADIFOR` supporting ADIFOR have to be added in the object list in the problem makefile (see C):

```
MC2OBJS = \
  ... \
  ${MC2PATH}/ADFAUX/adfaux.o \
  ${HOME}/ADIFOR/ReqADIntrinsics-iris4d.o
  ...
```

and the library list should contain the `ADIntrinsics` and `SparsLinC` library, e.g.

```
LIBS    = ... \
          -lADIntrinsics-iris4d -lSparsLinC-iris4d \
          ...
```

in an IRIX-environment.

To prepare the DAE integrator DAESOL to support analytical derivatives, the flag `NUMJAC` has to be *undefined* in the file `../MC2/IND/ind_adfdaesol.c`; the same has to be done in the file `../MC2/PRSQP/prsqp.c`, to activate analytical derivatives there.

## H.2 Generating sparse model derivatives for MUSCOD-II with ADIFOR 2.0

In the following we assume that the MUSCOD-II Fortran model is contained in the file `name_f77.F`. It could e.g. be copied into the directory `~/MUSCOD-II/MC2_DOWBAT/ADIFOR/`, which contains all relevant files applied to the `dowbat` example.

1. Create files `ffcn.f`, and, if needed, `gfcn.f` and `lfcn.f` for the model functions from `name_f77.F` for which derivatives are desired. Each of them should contain only one of these functions (say, e.g. `name_ffcn()`) in standard Fortran 77, and nothing else.

If preprocessor macros are used in the function definitions, it may be useful to preprocess the file `name_f77.F` (in the working directory `MC2_DOWBAT/ADIFOR`) before extracting the functions: e.g. use the commands

```
cc -P name_f77.F
```

to generate a standard Fortran 77 file `name.i` from which the functions can now be extracted using an editor. (the C preprocessor needs access to the MUSCOD-II header files `def_ind_f77.h` and `def_usrmod_f77.h` that are already copied into `MC2_DOWBAT/ADIFOR`)

2. Edit the dummy main program `adfmain.f` for ADIFOR 2.0 to provide a call to the first function for which derivatives are desired, say `name_ffcn()`. It does no harm that the declared dimensions of the array arguments do not correspond to the real ones. Ensure that only the desired function is called by disactivating the other two function calls:

```
      call name_ffcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
*      call name_gfcn(t, xd, xa, u, p, rhs, rwh, iwh, info)
*      call lfcn(t, xd, xa, u, p, lval, rwh, iwh, info)
```

3. Invoke ADIFOR 2.0:

```
Adifor2.0 AD_PROG=ffcn.cmp AD_TOP=name_ffcn AD_SCRIPT=ffcn.adf \
 AD_FLAVOR=sparse AD_OUTPUT_DIR=.
```

(the predefined script files `ffcn.adf`, `gfcn.adf`, and `lfcn.adf` contain the corresponding independent and dependent variables if standard argument names are used, and the predefined composition files `ffcn.cmp`, `gfcn.cmp`, and `lfcn.cmp` contain the relevant file names.)

The derivated function is now written into the file `g_ffcn.f`.

Execute steps 2. and 3. for the other two functions as well, by first editing adfmain.f to provide a call to the concerned function and then invoking ADIFOR 2.0 by

```
Adifor2.0 AD_PROG=gfcn.cmp AD_TOP=name_gfcn AD_SCRIPT=gfcn.adf \
 AD_FLAVOR=sparse AD_OUTPUT_DIR=.
```

or

```
Adifor2.0 AD_PROG=lfcn.cmp AD_TOP=name_lfcn AD_SCRIPT=lfcn.adf \
 AD_FLAVOR=sparse AD_OUTPUT_DIR=.
```

respectively.

4. Free the temporarily needed disk space by

```
rm -rf AD_cache
```

5. Provide the derivative subroutines generated by ADIFOR 2.0 to the linker by directly including them in the model source file `name.F`, or by combining them into a new include file `name_der.F` that can then be included into the original file `name.F`.

6. List the driver definition routine `def_adfDAESOL` in the call of `def_msolver()` in the model source (see 6). It must be used on all model stages, by appropriately choosing `ISLV` in the calls of `def_mstage()`.

7. `def_model()` must also contain a call to the additional function `def_msders()` for each model stage, to make the sparse derivatives accessible to the DAE solver and `PRSQP` module in MUSCOD-II. It has the syntax:

   ```
   def_msders(I, g_lfcn, g_lhs, g_ffcn, g_gfcn)
   ```

   where `I` is the model stage index and `g_...` are the pointers to the sparse derivative functions generated by ADIFOR .

# References


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

[Bau00] I. Bauer. *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik*. PhD thesis, University of Heidelberg, 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.

[Boc81] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K. H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, volume 18 of *Springer Series in Chemical Physics*. Springer, Heidelberg, 1981.

[Boc87] H.G. Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, volume 183 of *Bonner Mathematische Schriften*. University of Bonn, Bonn, 1987.

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

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

[DDDH90] J. J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling. A set of level and basic linear algebra subprograms. *ACM Trans. on Math. Soft.*, 16,1:1–28, 1990.

[DDHH88] J. J. Dongarra, J. DuCroz, 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. Math. Softw.*, 22:187–226, 1996.

[Fle71] R. Fletcher. A general quadratic programming algorithm. *J. Inst. Math. Appl.*, 7:76–91, 1971.

[GI83] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Math. Progr.*, 27:1–33, 1983.

[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 *Fortschr.-Ber. 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.*, volume 96-56 of *IWR-Preprint*. University of Heidelberg, Heidelberg, 1996.

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

[Pow78] M. J. D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G. A. Watson, editor, *Numerical Analysis, Dundee 1977*, volume 630 of *Lecture Notes in Mathematics*, Berlin, 1978. Springer.

[Pow85] M. J. D. Powell. On the quadratic programming algorithm of goldfarb and idnani. *Math. Progr. Study*, 25:46–61, 1985.

[RD93] J. K. Reid and I.S. Duff. Technical report ma48, a fortran code for direct solution of sparse unsymmetric linear systems of equations. Technical Report 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, 1972.

[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*, volume 96-12 of *IWR-Preprint*. University of Heidelberg, 1996. Ph.D. thesis.

[Sch98] V.H. Schulz. Solving discretized optimization problems by partially reduced sqp methods. *Comput. Visual. Sci.*, 1:83–96, 1998.