GESDA: A Software Package for the Numerical Solution of General Switched Differential- Algebraic Equations *

Lena Wunderlich†

March 31, 2009

Abstract

We describe a new software package for the numerical solution of general linear or nonlinear switched differential-algebraic equations (DAEs). The package embeds the DAE solvers GELDA and GENDA into a hybrid mode controller that determines the switch points, organizes the mode switching, and provides consistent initial values to restart the integration method at the switch point. It can deal with systems of arbitrary index and with linear systems that do not have unique solutions or inconsistencies in the initial values or the inhomogeneity. Nonuniqueness and inconsistencies are treated in a least square sense. The package includes the possibility of sliding mode simulation that allows an efficient treatment of chattering behavior during the simulation of a hybrid system. We give explicit descriptions how to use the package and include a numerical example.

1 Purpose

DGESDA is a general purpose code for the solution of initial value problems for **GE**neral **S**witched **D**ifferential-**A**lgebraic equations. It is designed to solve switched systems that are composed of several different constrained dynamical systems described by differential-algebraic equations (DAEs) that switch between different operation modes on the basis of transition conditions. The modeling concept is based on the general theory of DAEs as described in [13] and considers hybrid systems composed of a collection of continuous and discrete subsystems and the possible interaction between these subsystems following an approach given in [1, 9]. We assume that the discrete and continuous subsystems only interact via instantaneous discrete transitions at distinct points in time called *events*. In the following, a *hybrid system of differential-algebraic equations* $\mathcal{H}$ is defined as a collection of N_F systems of differential-algebraic equations, given either in general nonlinear form

$$F^\ell(t, x^\ell, \dot{x}^\ell) = 0, \quad \ell \in \mathbb{M}, \tag{1}$$

with sufficiently smooth functions $F^\ell : D_\ell \times \mathbb{R}^{n_\ell} \times \mathbb{R}^{n_\ell} \to \mathbb{R}^{m_\ell}$, or as linear DAEs in the form

$$E^\ell(t)\dot{x}^\ell = A^\ell(t)x^\ell + f^\ell(t), \quad \ell \in \mathbb{M}, \tag{2}$$

with sufficiently smooth functions $f^\ell : D_\ell \to \mathbb{R}^{m_\ell}$ and $E^\ell, A^\ell : D_\ell \to \mathbb{R}^{m_\ell, n_\ell}$, where $\mathbb{M} = \{1, \ldots, N_F\}$, $N_F \in \mathbb{N}$ is the finite *set of modes*. In this setting, (1) or (2) are the DAEs that describe the dynamics of the hybrid system in mode $\ell \in \mathbb{M}$ defined on the domain D_ℓ that is given as the union of certain intervals $\mathbb{I}_i$. Hereby, the integration interval $\mathbb{I} = [t_0, t_f] \subset \mathbb{R}$ is decomposed into subintervals $\mathbb{I}_i = [\tau_i, \tau_i')$ for $i = 1, \ldots, N_\mathbb{I} - 1$ and $\mathbb{I}_{N_\mathbb{I}} = [\tau_{N_\mathbb{I}}, \tau_{N_\mathbb{I}}']$, $N_\mathbb{I} \in \mathbb{N}$ such that $\mathbb{I} = \bigcup_{i=1}^{N_\mathbb{I}} \mathbb{I}_i$ and $\bigcup_{\ell \in \mathbb{M}} D_\ell = \mathbb{I}$ as well as $D_\ell \cap D_k = \emptyset$ for $\ell \neq k$, in such a way that in each of these intervals

*Supported by DFG Research Center MATHEON, 'Mathematics for key technologies' in Berlin.

†Institut für Mathematik, Technische Universität Berlin, D-10623 Berlin. wunder@math.tu-berlin.de

the dynamics of the system is governed by only one DAE. Here, we have $\tau_1 = t_0$, $\tau'_{N_\mathbb{I}} = t_f$ and $\tau'_i = \tau_{i+1}$ for $i = 1, \ldots, N_\mathbb{I} - 1$ as well as $\tau_i < \tau'_i$ for $i = 1, \ldots, N_\mathbb{I}$. The hybrid system changes between different modes on the basis of transition conditions. Each mode $\ell \in \mathbb{M}$ has an index set of autonomous *transitions* $J^\ell = \{1, 2, \ldots, n_T^\ell\}$, where $n_T^\ell \in \mathbb{N}$ is the number of possible transitions of mode ℓ, and each transition $j \in J^\ell$ has a *transition condition*

$$L_j^\ell(t, x^\ell, \dot{x}^\ell), \quad \text{with} \quad L_j^\ell : D_\ell \times \mathbb{R}^{n_\ell} \times \mathbb{R}^{n_\ell} \to \{TRUE, FALSE\}, \tag{3}$$

which determines when the transition should be made. If $L_j^\ell(\hat{t}, x^\ell(\hat{t}), \dot{x}^\ell(\hat{t})) = FALSE$ for all $j \in J^\ell$ at a time $\hat{t} \in D_\ell$, then the system stays in the current mode. On the other hand, if there exists an integer $j \in J^\ell$ such that $L_j^\ell(\hat{t}, x^\ell(\hat{t}), \dot{x}^\ell(\hat{t})) = TRUE$ at time $\hat{t}$, then the system switches to another mode. When a transition condition is satisfied, i.e., a so-called *state event* has occurred, the event time is localized as root of a switching function. The relations that make up a transition condition $L_j^\ell(t, x^\ell, \dot{x}^\ell)$ are formulated as roots of so-called *switching functions*

$$g_{j,i}^\ell : D_\ell \times \mathbb{R}^{n_\ell} \times \mathbb{R}^{n_\ell} \to \mathbb{R}, \quad \text{for } i = 1, \ldots, n_j^\ell, \quad j \in J^\ell,$$

with $g_{j,i}^\ell(t, x^\ell, \dot{x}^\ell) > 0$ in mode ℓ. Each time a switching function crosses zero, i.e., if $g_{j,i}^\ell(t, x^\ell, \dot{x}^\ell) \leq 0$ for a $j \in J^\ell$ and some i, the associated transition condition may switch its logical value, such that the roots of the switching functions are the switch points (or event times) $\tau'_i = \tau_{i+1}$ of the system. Note that the switch points are not known in advance but determined during the integration depending on the state of the system. Here, a transition condition L_j^ℓ is described by n_j^ℓ separated switching functions $g_{j,i}^\ell$ whose logical combination determines if the transition condition is satisfied. In this way, the switching functions can be chosen as simple as possible, allowing an efficient and reliable computation of the switch points by a root finding procedure.
After the location of a state event the successor mode is determined by the so-called *mode allocation function*

$$S^\ell : J^\ell \to \mathbb{M}, \text{ with } S^\ell(j) = k \tag{4}$$

that assigns the successor mode k corresponding to the jth transition comming from mode ℓ. Finally, each transition $j \in J^\ell$ has a *transition function* $T_\ell^k : \mathbb{R}^{n_\ell} \times \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_k,2}$ of the form

$$T_\ell^k(x^\ell(\tau'_i), \dot{x}^\ell(\tau'_i)) = [x^k(\tau_{i+1}), \dot{x}^k(\tau_{i+1})] \tag{5}$$

that maps the final values of the variables x^ℓ in the predecessor mode ℓ to the corresponding initial values in the successor mode k at time $\tau'_i = \tau_{i+1} \in D_k$ according to the jth transition.
Hybrid systems defined in this way have no limitations with respect to the types of DAEs in the different modes, as long as these are solvable in the interval set D_ℓ, but there are restrictions concerning the transition processes. The number of mode changes must be finite, as otherwise no reasonable numerical integration is possible. Furthermore, we assume that the integration intervals have a nonzero measure, i.e., instantaneous multiple mode changes are not possible and that only one transition condition is becoming true at a time.
The numerical simulation of hybrid differential-algebraic systems requires the efficient treatment of certain aspects in the hybrid system behavior. Besides the robust numerical integration of the DAEs in each operation mode, the points in time at which a transition condition changes its logical value have to be detected as the roots of the switching functions. These switch points have to be located accurately and in strict temporal sequence as they are the initial points for further integration and influence the mode switching and the future behavior of the hybrid systems. Further, the system state at the switch point has to be determined to restart the integration method in the new mode. Since the switch points will in general not coincide with the points chosen by the stepsize control, an interpolation method is needed that interpolates the computed solution at the detected switch point. Finally, chattering behavior, i.e., repeated mode switching during the integration has to be treated in an appropriate way. The solver DGESDA provides a tool for the numerical integration of hybrid systems defined as described above. It combines the DAE

solver for the integration of the system in each mode with an outer mode controller that triggers root finding, mode switching and the restart of the integration procedure. Furthermore, the solver provides the possibility of sliding mode simulation to suppress chattering during the integration.

The report is organized as follows. Section 2 summarizes the mathematical background concerning the numerical treatment of switched DAE systems. In Section 2.1 we discuss index reduction for switched systems that enables the reformulation as reduced strangeness-free system that will be used for the discretization and in Section 2.2 we discuss the concept of sliding motion for switched DAE systems. In Section 3 we present the hybrid system solver DGESDA, describe the procedures used in the solver and discuss its features in detail. Section 4 gives a documentation of the solver DGESDA. In the Appendix A we present a numerical example to demonstrate the usage of the solver.

2 Mathematical Background

In this section we shortly summarize the theory for the numerical solution of switched DAE systems, for details see also [9, 16, 23]. For switched differential-algebraic systems a reduction to a strangeness-free system must be realized just as for standard DAEs to be able to apply numerical methods suited for DAEs in the integration process, see [13]. Depending on the form of the DAE in each mode a strangeness-free system can be determined as described in [12, 15] for nonlinear DAEs or in [14] for linear DAEs. In the following, we describe the more general nonlinear case. Furthermore, we introduce the concept of sliding motion that enables an appropriate treatment of chattering behavior in hybrid system simulation.

2.1 Index Reduction for Switched DAE Systems

We consider a hybrid system described by nonlinear DAEs of the form (1) in each mode $\ell \in \mathbb{M}$. The index reduction procedure for nonlinear DAEs described e.g. in [12, 13] can be applied independently for each mode as has been shown in [9]. We introduce the nonlinear derivative array $\mathcal{F}_i^\ell$ of level i in mode ℓ of the form

$$\mathcal{F}_i^\ell(t, x^\ell, \dot{x}^\ell, \ldots, \frac{d^{i+1}}{dt^{i+1}} x^\ell) = 0, \tag{6}$$

which stacks the original equations of the DAE in mode ℓ and all its derivatives up to level i into one large system

$$\mathcal{F}_i^\ell(t, x^\ell, \dot{x}^\ell, \ldots, x^{\ell(i+1)}) = \begin{bmatrix} F^\ell(t, x^\ell, \dot{x}^\ell) \\ \frac{d}{dt} F^\ell(t, x^\ell, \dot{x}^\ell) \\ \vdots \\ \frac{d^i}{dt^i} F^\ell(t, x^\ell, \dot{x}^\ell) \end{bmatrix}.$$

In the following, partial derivatives of $\mathcal{F}_i^\ell$ with respect to selected variables p from $(t, x^\ell, \ldots, x^{\ell(i+1)})$ are denoted by $\mathcal{F}_{i;p}^\ell$, e.g.,

$$\mathcal{F}_{i;x^\ell}^\ell = \frac{\partial}{\partial x^\ell} \mathcal{F}_i^\ell, \quad \mathcal{F}_{i;\dot{x}^\ell,\ldots,x^{\ell(i+1)}}^\ell = \left[\frac{\partial}{\partial \dot{x}^\ell} \mathcal{F}_i^\ell \ldots \frac{\partial^{(i+1)}}{\partial x^{\ell(i+1)}} \mathcal{F}_i^\ell \right].$$

To obtain a reduced system a hypothesis was introduced in [12]. Adapted to the case of hybrid DAE systems the hypothesis can be stated as follows.

Hypothesis 1. *Consider a nonlinear differential-algebraic equations* (1) *in mode* $\ell \in \mathbb{M}$. *There exist integers* $\mu^\ell, r^\ell, a^\ell, d^\ell$ *and* v^ℓ *such that the solution set of the derivative array* $\mathcal{F}_{\mu^\ell}^\ell$

$$\mathbb{L}_{\mu^\ell}^\ell = \{(t, x^\ell, \dot{x}^\ell, \ldots, x^{\ell(\mu^\ell+1)}) \in D_\ell \times \mathbb{R}^{(\mu^\ell+2)n_\ell} |\ \mathcal{F}_{\mu^\ell}^\ell(t, x^\ell, \dot{x}^\ell, \ldots, x^{\ell(\mu^\ell+1)}) = 0\}$$

is not empty, and the following properties hold:

1. *The set* $\mathbb{L}^\ell_{\mu^\ell} \subset \mathbb{R}^{(\mu^\ell+2)n_\ell+1}$ *forms a manifold of dimension* $(\mu^\ell+2)n_\ell+1-r^\ell$.

2. *We have* $\operatorname{rank} \mathcal{F}^\ell_{\mu^\ell;x^\ell,\dot{x}^\ell,\ldots,x^{\ell(\mu^\ell+1)}} = r^\ell$ *on* $\mathbb{L}^\ell_{\mu^\ell}$.

3. *We have* $\operatorname{corank} \mathcal{F}^\ell_{\mu^\ell;x^\ell,\dot{x}^\ell,\ldots,x^{\ell(\mu^\ell+1)}} - \operatorname{corank} \mathcal{F}^\ell_{\mu^\ell-1;x^\ell,\dot{x}^\ell,\ldots,x^{\ell(\mu^\ell)}} = v^\ell$ *on* $\mathbb{L}^\ell_{\mu^\ell}$. *(The corank is the codimension of the range and we use the convention that* $\operatorname{corank} \mathcal{F}^\ell_{-1;x^\ell} = 0$.*)*

4. *We have* $\operatorname{rank} \mathcal{F}^\ell_{\mu^\ell;\dot{x}^\ell,\ldots,x^{\ell(\mu^\ell+1)}} = r^\ell - a^\ell$ *on* $\mathbb{L}^\ell_{\mu^\ell}$ *such that there are smooth full rank matrix functions* $Z_{\ell,2}$ *and* $T_{\ell,2}$ *defined on* $\mathbb{L}^\ell_{\mu^\ell}$ *of size* $((\mu^\ell+1)m_\ell, a^\ell)$ *and* $(n_\ell, n_\ell - a^\ell)$ *satisfying*

$$Z^T_{\ell,2}\mathcal{F}^\ell_{\mu^\ell;\dot{x}^\ell,\ldots,\dot{x}^{\ell(\mu^\ell+1)}} = 0, \quad \operatorname{rank} Z^T_{\ell,2}\mathcal{F}^\ell_{\mu^\ell;x^\ell} = a^\ell, \quad Z^T_{\ell,2}\mathcal{F}^\ell_{\mu^\ell;x^\ell}T_{\ell,2} = 0$$

   on $\mathbb{L}^\ell_{\mu^\ell}$.

5. *We have* $\operatorname{rank} F^\ell_{;\dot{x}^\ell}T_{\ell,2} = d^\ell = m_\ell - a^\ell - v^\ell$ *on* $\mathbb{L}^\ell_{\mu^\ell}$ *such that there exists a smooth matrix function* $Z_{\ell,1}$ *defined on* $\mathbb{L}^\ell_{\mu^\ell}$ *of size* (m_ℓ, d^ℓ) *with* $Z^T_{\ell,1}F^\ell_{;\dot{x}^\ell}T_{\ell,2}$ *having full rank.*

The smallest possible μ^ℓ in Hypothesis 1 is called the *strangeness index* of the DAE (1) in mode ℓ and systems with vanishing strangeness-index are called *strangeness-free*. The *maximal strangeness index* μ_{max} of a hybrid system $\mathcal{H}$ is given by

$$\mu_{max} = \max_{\ell\in\mathbb{M}}\{\mu^\ell\},$$

and a hybrid system $\mathcal{H}$ is called *strangeness-free* if $\mu_{max} = 0$. Further, a DAE (1) that satisfies Hypothesis 1 with $n_\ell = m_\ell = d^\ell + a^\ell$ is called *regular*. The corresponding numbers d^ℓ and a^ℓ are the numbers of differential and algebraic equations of the DAE in mode ℓ. It has been shown in [12] that Hypothesis 1 implies locally the existence of a reduced system of the form

$$\hat{F}^\ell(t, x^\ell_1, x^\ell_2, x^\ell_3, \dot{x}^\ell_1, \dot{x}^\ell_2, \dot{x}^\ell_3) = 0, \tag{7a}$$

$$x^\ell_3 - \mathcal{R}^\ell(t, x^\ell_1, x^\ell_2) = 0, \tag{7b}$$

independently in each mode $\ell \in \mathbb{M}$, with $\hat{F}^\ell = Z^T_{\ell,1}F^\ell$. Note that an initial condition is consistent with the DAE in mode ℓ if it satisfies the algebraic equation $x^\ell_3 - \mathcal{R}^\ell(t, x^\ell_1, x^\ell_2) = 0$. Eliminating x^ℓ_3 and $\dot{x}^\ell_3$ in (7a) with the help of (7b) and its derivative leads to a system

$$\hat{F}^\ell(t, x^\ell_1, x^\ell_2, \mathcal{R}^\ell(t, x^\ell_1, x^\ell_2), \dot{x}^\ell_1, \dot{x}^\ell_2, \mathcal{R}^\ell_t(t, x^\ell_1, x^\ell_2) + \mathcal{R}^\ell_{x^\ell_1}(t, x^\ell_1, x^\ell_2)\dot{x}^\ell_1 + \mathcal{R}^\ell_{x^\ell_2}(t, x^\ell_1, x^\ell_2)\dot{x}^\ell_2) = 0.$$

By part 5 of Hypothesis 1 we may assume w.l.o.g. that this system can (locally via the implicit function theorem) be solved for $\dot{x}^\ell_1$, leading to a system of the form

$$\begin{aligned} \dot{x}^\ell_1 &= \mathcal{L}^\ell(t, x^\ell_1, x^\ell_2, \dot{x}^\ell_2), \\ x^\ell_3 &= \mathcal{R}^\ell(t, x^\ell_1, x^\ell_2), \end{aligned} \tag{8}$$

which has a vanishing strangeness index. Obviously, in this system $x^\ell_2 \in C^1(D_\ell, \mathbb{R}^{u^\ell})$, with $u^\ell = n_\ell - d^\ell - a^\ell$, can be chosen arbitrarily (at least when staying in the domain of definition of $\mathcal{R}^\ell$ and $\mathcal{L}^\ell$), i.e., it plays the role of a control. When x^ℓ_2 has been chosen, then the resulting system has (locally) a unique solution for x^ℓ_1 and x^ℓ_3, provided that a consistent initial condition is given.

Theorem 1. *[12] Let* F^ℓ *in* (1) *be sufficiently smooth and satisfy Hypothesis 1 with* μ^ℓ, r^ℓ, a^ℓ, d^ℓ, v^ℓ *and* $u^\ell = n_\ell - d^\ell - a^\ell$. *Then, every solution of* (1) *also solves the reduced problems* (7) *and* (8) *consisting of* d^ℓ *differential and* a^ℓ *algebraic equations.*

Remark 2. In the reduced systems (7) and (8) we have not used the quantity v^ℓ. This quantity measures the number of equations in the original system that give rise to trivial equations $0 = 0$, i.e., it counts the number of redundancies in the system. Together with a^ℓ and d^ℓ it gives a complete classification of the m_ℓ equations into d^ℓ differential equations, a^ℓ algebraic equations and v^ℓ trivial equations.

In order to be able to derive a reduced hybrid system that is strangeness-free we need to assume that Hypothesis 1 holds for each separate mode.

Hypothesis 2. *In each mode $\ell \in \mathbb{M}$ and each domain D_ℓ the strangeness index μ^ℓ is well-defined, i.e., the DAE in mode ℓ satisfies Hypothesis 1 with constant characteristic values $\mu^\ell, r^\ell, a^\ell, d^\ell, v^\ell$.*

If Hypothesis 2 is violated at a finite number of points, then we can introduce new modes and further switching points to satisfy Hypothesis 2. Then, a reduced system as in (7) or (8) with the same solution as the original DAE (1) in mode ℓ can be extracted independently in each mode and, therefore, also for the complete hybrid system. Connecting all the reduced systems together we locally obtain an equivalent reduced hybrid system denoted by $\hat{\mathcal{H}}$, which is strangeness-free. By solving the corresponding transformed systems, the solution of the complete hybrid system can be computed for every time t. For hybrid systems satisfying Hypothesis 2 it has been shown in [9, 23] that every sufficiently smooth solution of the original hybrid system is also a solution of this equivalent strangeness-free hybrid system.

2.2 Sliding Motion

A special phenomena that can occur during the simulation of hybrid systems is a fast changing between different modes of continuous operation, called *chattering*. The numerical solution of a hybrid system exhibiting chattering behavior requires high computational costs and, in the worst case, the numerical integration breaks down as it does not proceed in time, but chatters between modes. To prevent chattering and to reduce the computational costs, the system dynamics along the switching surface can be regularized in the regions where chattering occurs by adding an additional mode, the so-called *sliding mode*, that represents the dynamics during sliding. Sliding motion is well understood for ordinary differential equations, see e.g. [6, 21]. These ideas have been extended to the case of DAEs in [16, 23].
We consider a hybrid DAE system $\mathcal{H}$ that switches from mode ℓ to mode k, assuming that each transition condition L_j^ℓ is described by exactly one switching function g_j^ℓ, i.e., $n_j^\ell = 1$. Further, we restrict to switching functions independent of the state derivative, such that mode switching occurs at points on the *switching surfaces*

$$\Gamma_j^\ell = \left\{(t, x^\ell) \in D_\ell \times \mathbb{R}^{n_\ell} \mid g_j^\ell(t, x^\ell) = 0\right\}, \quad j \in J^\ell,\ \ell \in \mathbb{M}. \tag{9}$$

Further, we assume that $g_{j;x^\ell}^\ell(t, x^\ell) \neq 0$ in a neighborhood of the switching surface Γ_j^ℓ and that the two modes are separated by the j-th switching surface Γ_j^ℓ. Assuming regularity and well-definedness of the strangeness index in each mode, the DAEs in the modes ℓ and k can be transformed to the corresponding reduced systems of the form (8) given by

$$\begin{aligned} \dot{x}_1^\ell &= \mathcal{L}^\ell(t, x_1^\ell), & \dot{x}_1^k &= \mathcal{L}^k(t, x_1^k), \\ x_2^\ell &= \mathcal{R}^\ell(t, x_1^\ell), & x_2^k &= \mathcal{R}^k(t, x_1^k). \end{aligned} \tag{10}$$

The hybrid system exhibits chattering behavior if all solutions near the switching surface Γ_j^ℓ approach it from both sides, i.e., if the projections of the velocity vectors describing the dynamical parts of the corresponding reduced systems onto the surface gradient are of opposite signs and directed towards the surface from both sides in a neighborhood of the switching surface. The *sliding condition* can be formulated as

$$\mathcal{L}_N^\ell = g_{j;x_1^\ell}^\ell(t, x_1^\ell, x_2^\ell)\mathcal{L}^\ell(t, x_1^\ell) < 0 \quad \text{and} \quad \mathcal{L}_N^k = g_{\tilde{j};x_1^k}^k(t, x_1^k, x_2^k)\mathcal{L}^k(t, x_1^k) > 0, \tag{11}$$

for $(t, x) \in \Gamma_j^\ell = \Gamma_{\tilde{j}}^k$. These directional derivatives can be approximated numerically by

$$\mathcal{L}_N^\ell \approx \frac{1}{\delta} g_j^\ell(t, x_1^\ell + \delta\mathcal{L}^\ell(t, x_1^\ell), x_2^\ell), \quad \mathcal{L}_N^k \approx \frac{1}{\delta} g_{\tilde{j}}^k(t, x_1^k + \delta\mathcal{L}^k(t, x_1^k), x_2^k), \tag{12}$$

for small enough δ. During the numerical simulation an immediate switch back to mode ℓ after one or a few integration steps in mode k would result if the sliding condition is satisfied. To avoid this, we add an additional mode defining the dynamics of the DAE during sliding and switch to the sliding mode instead. To define the behavior during sliding the system dynamics are approximated in such a way that the state trajectory moves along the switching surface. We generalize an approach called *Filippov regularization*, see e.g. [6, 21], where the velocity vector of sliding motion is approximated as a convex combination of the velocity vectors on both side of the switching surface in such a way that it lies on a plane tangential to the switching surface. The system evolves in the sliding mode as long as the sliding condition is satisfied, and resumes in mode ℓ or k afterwards, depending on the sign of the directional derivatives.
Let d^ℓ, d^k and a^ℓ, a^k denote the number of differential and algebraic equations in mode ℓ and mode k, i.e., the dimension of x_1^ℓ, x_1^k and x_2^ℓ, x_2^k in the corresponding reduced systems (10). Further, let $d^\ell + a^\ell = d^k + a^k = n$ and without loss of generality assume that $d^\ell \geq d^k$ and $a^\ell \leq a^k$. Then, x_1^ℓ and x_2^k can be further partitioned into

$$x_1^\ell = \begin{bmatrix} x_{1,1}^\ell \\ x_{1,2}^\ell \end{bmatrix}, \quad x_2^k = \begin{bmatrix} x_{2,1}^k \\ x_{2,2}^k \end{bmatrix},$$

with $x_{1,1}^\ell \in \mathbb{R}^{d^k}$, $x_{1,2}^\ell \in \mathbb{R}^{d^\ell - d^k}$ and $x_{2,1}^k \in \mathbb{R}^{a^\ell}$, $x_{2,2}^k \in \mathbb{R}^{a^k - a^\ell}$. Then, the corresponding reduced systems can be partitioned accordingly into

$$\begin{bmatrix} \dot{x}_{1,1}^\ell \\ \dot{x}_{1,2}^\ell \end{bmatrix} = \begin{bmatrix} \mathcal{L}_1^\ell(t, x_{1,1}^\ell, x_{1,2}^\ell) \\ \mathcal{L}_2^\ell(t, x_{1,1}^\ell, x_{1,2}^\ell) \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} \dot{x}_1^k = \mathcal{L}^k(t, x_1^k), \\ x_2^\ell = \mathcal{R}^\ell(t, x_{1,1}^\ell, x_{1,2}^\ell), \end{bmatrix} \quad \begin{bmatrix} x_{2,1}^k \\ x_{2,2}^k \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^k(t, x_1^k) \\ \mathcal{R}_2^k(t, x_1^k) \end{bmatrix}. \tag{13}$$

Using the ideas of the Filippov regularization we define the *differential-algebraic system in sliding motion* by

$$\dot{x}_1 = \alpha \begin{bmatrix} \mathcal{L}_1^\ell(t, x_1) \\ \mathcal{L}_2^\ell(t, x_1) \end{bmatrix} + (1 - \alpha) \begin{bmatrix} \mathcal{L}^k(t, x_1) \\ 0 \end{bmatrix}, \tag{14a}$$

$$x_2 = \alpha \begin{bmatrix} \mathcal{R}^\ell(t, x_1) \\ 0 \end{bmatrix} + (1 - \alpha) \begin{bmatrix} \mathcal{R}_1^k(t, x_1) \\ \mathcal{R}_2^k(t, x_1) \end{bmatrix}, \tag{14b}$$

$$0 = g_j^\ell(t, x_1, x_2). \tag{14c}$$

It has been shown in [16, 23] that the DAE in sliding motion (14) is regular and of strangeness index $\mu = 1$ if

$$g_{j;x_2}^\ell(t, x_1, x_2) \left(\mathcal{R}^k(t, x_1) - \mathcal{R}^\ell(t, x_1) \right)$$

is nonsingular for all $(t, x_1, x_2) \in D_\ell \times \mathbb{R}^n$. Here, the algebraic variable α ensures that the solution stays on the manifold Γ_j^ℓ described by the algebraic constraint (14c). The differential equation (14a) describes the equivalent dynamics of the system during sliding motion and the algebraic equations (14b) is defined as a transformation of the constraint manifolds in the two modes such that $\mathcal{R}^\ell(t, x_1)$ is turned into $\mathcal{R}^k(t, x_1)$ or vice versa depending on the direction of the discontinuity crossing.

3 The Hybrid System Solver

The solver DGESDA is designed for the numerical solution of initial value problems for hybrid differential-algebraic systems that consist of a number of either nonlinear DAEs of the form (1) or linear DAEs of the form (2) in each mode $\ell \in \mathbb{M}$. For the numerical integration of the DAEs in each mode existing integration methods for continuous differential-algebraic systems have been embedded in the mode controller. The mode controller that determines the switch points, organizes the mode switching, and provides consistent initial values to restart the integration method at the switch point is designed independently of the integration method, the index reduction procedure,

or the switch point location method. Ideally, the index reduction technique and the integration method should be chosen according to the type of the DAE in the current mode. Currently, the DAE solvers DGELDA [14] and DGENDA [15], see Section 3.1, have been embedded for the solution of linear and nonlinear DAEs, respectively. Note, that the DAE in each mode has to be mathematically well-behaved in a small interval following an event to ensure that the solution is a smooth function over the whole integration step and, in particular, that a solution of the DAE in each mode exists until the end of an integration step. Using this assumption we can lock the function evaluation for the DAE solver during an integration step such that the equations evaluated cannot be changed while a small time step is taken even if the transition condition is satisfied.

The user has to provide the descriptions of the DAEs in each mode together with a sufficient number of derivatives of the system depending on the index of the DAE in each mode. For linear DAEs of the form (2) the matrices E^ℓ, A^ℓ and the right-hand sides f^ℓ together with the corresponding derivatives up to a certain order have to be provided in the subroutine MATSUB. For nonlinear DAE systems of the form (1) the functions F^ℓ together with the corresponding derivatives have to be provided in the subroutine FUN, and the corresponding Jacobians in the subroutine DFUN. Further, the user has to specify the transition conditions for each mode in the subroutine UINTER and the corresponding switching functions in the subroutine GFUN. The roots of the switching functions defined in GFUN have to correspond to the points in time where a transition condition defined in UINTER changes its logical value. The successor modes are determined by the mode allocation functions implemented in a user-provided subroutine MCHNG. Finally, the transition functions have to be provided in the subroutine TRANS that transfer the state of the system at a switch point to the initial state in the successor mode. For a detailed description of the user-supplied subroutines see Section 4.2.1.

Starting from given or computed consistent initial conditions in the initial mode, the state of the system evolves according to the current DAE that is numerically integrated according to its structure with an appropriate DAE solver until an event occurs. After each successful time step of the numerical integration the DAE solver checks whether a transition condition is satisfied by calls to the user-provided subroutine UINTER. Further, the DAE solver checks for changes in the index or in the characteristic values of the current DAE. If a transition condition is satisfied or changes in the characteristic values occur, then the integration is stopped. Changes in the characteristic values can only be handled by introducing further mode changes and additional modes, such that each DAE is well-defined in a small interval following a switch point. Otherwise, if a transition condition is satisfied, the switch point is localized as the root of a switching function. The root finding procedure implemented in the subroutine DRTFND uses a modified secant method described in Section 3.2. The successor mode is then determined by the mode allocation function (4) implemented in the user-provided subroutine MCHNG. The solver determines the state at the switch point of the current solution by interpolation. The subroutines DCONTS and DBDTRP provide interpolation routines for Runge-Kutta and BDF methods, respectively, as described in Section 3.3. Next, with the help of the transition function (5) implemented in the user-provided subroutine TRANS, the interpolated state is transferred to the initial state of the successor mode. If the initial state is consistent with the DAE in the new mode, the integrator is restarted, otherwise new consistent initial values are determined in a least square sense as described in Section 3.4. Note that we may have different characteristic values in the new mode.

One-step methods like Runge-Kutta methods can be restarted at any time point, since no past values are used. This is different for multi-step method like BDF methods that use past values to approximate the solution. In this case, changing to a lower order method is necessary if discontinuities in the solution occur due to an event. For safety reasons we always restart the integration with the first order BDF method and the initial stepsize is set to the stepsize used in the last successful step. It may happen that after a mode switching event an immediate mode change is detected in the new mode. If only one transition is possible from the new mode, then this transition is carried out and the integration is resumed. If several transitions are possible, then it is not clear which transition should be carried out, such that the code exits with an error message. If immediate mode changes occur repeatedly, i.e., if more than a maximal number

Subroutines within DGESDA	
DGELDA	general linear differential-algebraic equation solver (see Section 3.1)
DGENDA	general nonlinear differential-algebraic equation solver (see Section 3.1)
DRTFND	root finding procedure (see Section 3.2)
DBDTRP	backward differentiation interpolation routine (see Section 3.3)
DCONTS	computes a solution at a fixed time by interpolation (see Section 3.3)
DCKCON	checks consistency and if necessary computes consistent initial values (see Section 3.4)

User-supplied subroutines (see Section 4.2.1)	
USCAL	user-supplied scaling routine
UINTER	user-supplied subroutine defining transition conditions
TRANS	user-supplied subroutine defining transition functions
MCHNG	user-supplied subroutine defining mode allocation functions
GFUN	user-supplied subroutine defining switching functions
MATSUB	user-supplied subroutine for linear problems
FUN/DFUN	user-supplied subroutines for nonlinear problems

Table 1: The subroutines of DGESDA and their purposes

MAXCGN of successive mode changes occur, the integration is stuck and the code exits with an error message. In this case, the transition conditions should be reformulated, e.g., if applicable, hysteresis can be used. The default value for the maximal number of immediate mode changes is set to MAXCGN=100, it can be adapted by the user, see Section 4.3. In addition, the maximal stepsize is restricted to HMAX in order to avoid stepping over regions. By default HMAX is set to HMAX=1.0. Also the maximal stepsize can be adapted by the user, see Section 4.3. If no smooth transition at a mode change is possible, due to inconsistency of initial values or due to the user-defined transition functions, the code displays a warning. In Table 1 the subroutines of DGESDA and their purposes as well as the user-supplied subroutines are summarized. The hybrid mode controller is depicted in Figure 1. In the following sections, we will discuss each part of the hybrid system solver in more detail.

Remark 3. We assume that it is possible to extend the solution of the differential-algebraic equation in the current mode in a small interval beyond the event time. This is sometimes not possible or not in a unique way, e.g., at impasse points [19], or when characteristic values change. Sometimes events are employed to switch the DAE system at these critical points. However, if the solution cannot be extended past the event, the proposed event location algorithm does not apply. In principle, the event can be moved slightly to the left of the critical point, but even this can effect the error and stepsize control if the integrator attempts to locate a point beyond the event. Further, we assume that only one transition condition can be satisfied at the same point in time and the successor mode is uniquely determined by the mode allocation function. In practical applications this might be difficult to realize since the successor mode can depend on the system state and multiple transitions at a switch point occur. It also may happen that an immediate transition occurs if a transition condition is satisfied directly after the transfer of the state or the computation of consistent initial values. Eventually, the transition condition that causes the mode switching may not be fulfilled anymore after the consistent reinitialization. This problem of so-called *discontinuity sticking* as well as consistent event location has been treated e.g. in [2].

3.1 Integration of the current DAE - The DAE solvers

The DAE in the current mode is solved according to its structure with an appropriate DAE solver. In general, any integration method for strangeness-free differential-algebraic systems that provides a continuous representation of the solution, as e.g. BDF methods [4] or Runge-Kutta

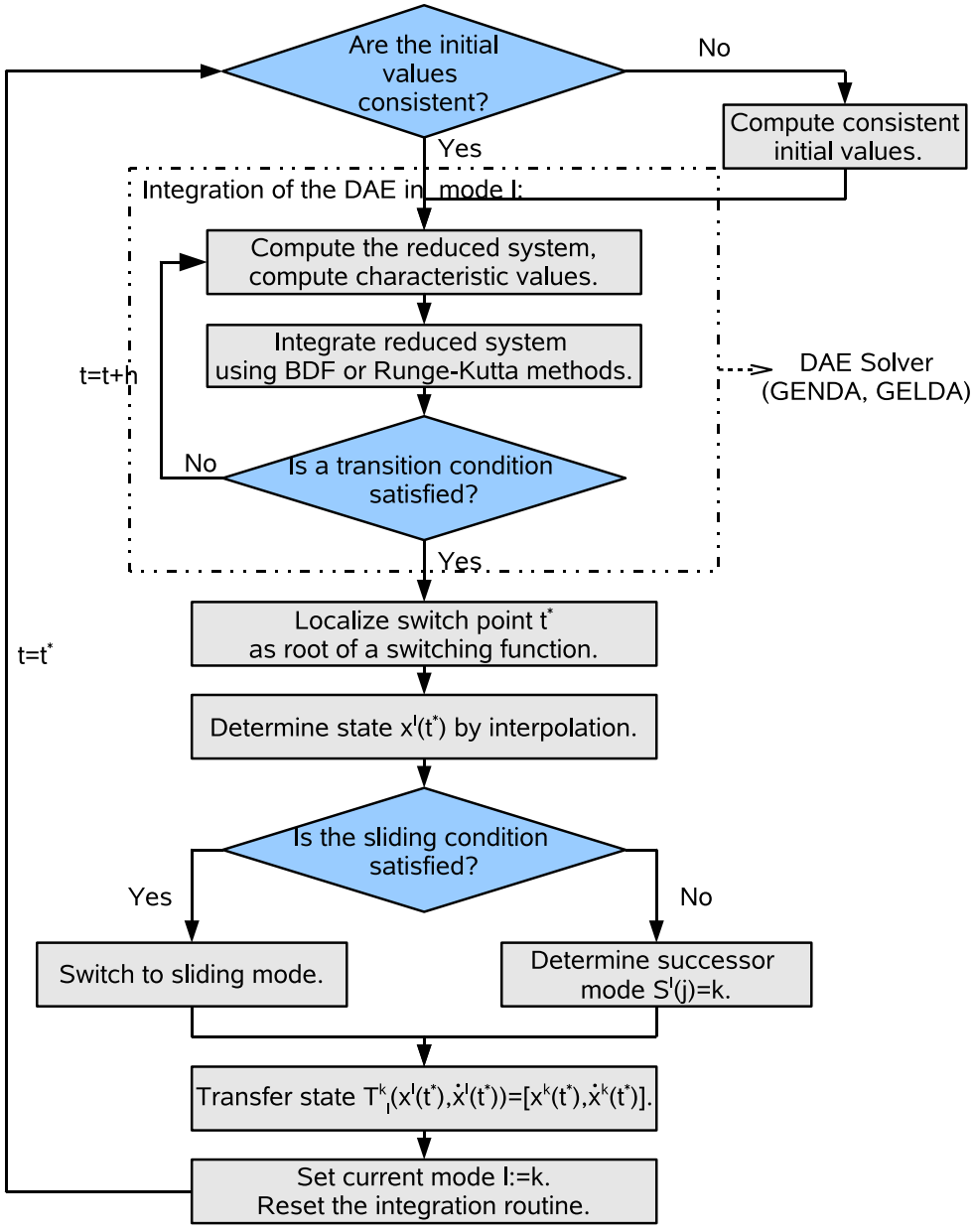

Figure 1: The hybrid mode controller

methods [8], can be used for the numerical integration of the DAEs in each mode. Currently, the differential-algebraic system solvers DGELDA [14] and DGENDA [15] are embedded in the hybrid system solver DGESDA. Note, that no restrictions on the index of the DAEs (1) or (2) are needed, since both solvers are constructed for the solution of DAE systems of arbitrary high index. There are several further routines for the numerical solution of differential-algebraic systems including solvers adapted to special structures arising e.g. in the equations of motion of multibody systems [20] that are planned to be appended to the hybrid system solver such that ideally an appropriate DAE solver can be chosen depending on the structure of the DAE in each mode.

DGELDA For linear DAEs of the form (2) with initial values that are not necessarily consistent, we use the solver DGELDA. While most of the standard integration methods are suitable only for regular strangeness-free DAEs, DGELDA is suitable for the numerical integration of linear DAEs of arbitrary index and also allows the solution of over- and underdetermined systems. The implementation of DGELDA is based on the combination of an index reduction procedure introduced in [11], which first determines all the local invariants and then transforms the linear DAE (2) into an equivalent strangeness-free DAE with the same solution set, followed by a discretization of the strangeness-free DAE either using a Runge-Kutta scheme adapted from the code RADAU5 [8] or a BDF method adapted from DASSL [4]. The user has to provide the necessary number of derivatives of all system matrices. The code DGELDA allows the numerical determination of the strangeness index and the computation of the structural invariants by recursive formulas. The index is computed at each time step to detect changes in the index or in the characteristic values. Consistent initial conditions are computed via correction of the given initial values by solving a minimization problem. For a detailed description of the code see [14].

DGENDA For nonlinear square problems of the form (1) with arbitrary index and initial values not necessarily consistent we use the solver DGENDA which uses a BDF method with order and stepsize control adapted from DASSL. The code DGENDA combines the index reduction technique described in Section 2.1 with the discretization of an equivalent strangeness-free formulation of the DAE. The user has to provide the necessary number of derivatives of the whole DAE, in particular, the whole derivative array $\mathcal{F}_k^\ell$ of level k given in (6), as well as the corresponding Jacobians. In addition, the characteristic quantities have to be provided by the user. Before discretizing the DAE at a time step proceeding from t_i to $t_i + h$ we have to compute a locally equivalent strangeness-free system similar to (8) by generating projectors $Z_{\ell,1}, Z_{\ell,2}$. Then, in every integration step a nonlinear system of equations of the form

$$\mathcal{F}_{\mu^\ell}^\ell(t_i + h, x^\ell, \dot{x}^\ell, ..., x^{\ell(\mu^\ell+1)}) = 0, \tag{15a}$$

$$\tilde{Z}_{\ell,1}^T F^\ell(t_i + h, x^\ell, D_h x^\ell) = 0, \tag{15b}$$

is solved, where $\tilde{Z}_{\ell,1}$ denotes some approximation to $Z_{\ell,1}$ at the desired solution and D_h denotes the discretization scheme. Equation (15a) ensures that the algebraic constraints are satisfied and (15b) is a discretization of the differential part of the reduced system. The solution of the underdetermined system of nonlinear equations (15a) is computed in a least squares sense using the subroutine NLSCON [17] which is an implementation of the Gauss-Newton method. Note, that the Jacobian of (15) is generally nonsquare but has full row rank at every solution $(x^\ell, \dot{x}^\ell, ..., x^{\ell(\mu^\ell+1)})$ of the DAE (1) if $\tilde{Z}_{\ell,1}$ is a sufficiently good approximation to $Z_{\ell,1}$ and the stepsize is sufficiently small. This property extends to a neighborhood of the solution set, thus we get quadratic convergence to a solution and a simplified Gauss-Newton method [18] can be applied by fixing the Jacobian at any time step. Rank decisions may suffer of badly conditioned problems, e.g., if the time scale is extremely small. Thus, before any rank decisions are made during the computation, the Jacobi matrix of (15) is equilibrated to lower its condition number by computing appropriate row and column scaling vectors. Optionally, the user can supply a scaling subroutine USCAL if an appropriate scaling method is known, see Section 4.2.1 for details. On the other hand, it is also possible to completely deactivate scaling. Further, the user can require the code to verify the

given characteristic values after consistent initial values have been computed or after the BDF solver successfully completed an iteration. In this case, DGENDA returns an error message if any changes in the characteristic values are detected. For details of the integration routine see [15].

3.2 The Root Finding Procedure

In order to localize an event, i.e., the earliest time in an integration interval $[t_i, t_{i+1}]$ at which a transition condition is satisfied, the roots of the corresponding switching functions have to be found with sufficiently high accuracy. In particular, the root finding procedure must ensure that all events are located precisely and if multiple roots exist in an interval the earliest event time must be located. As the switching functions are evaluated over the whole integration interval, the system state has to be interpolated between meshpoints. Thus, the root finding procedure depends on interpolation formulas which are provided by the BDF formulas or by the Runge-Kutta method, see Section 3.3. In the following, we describe the procedure used to find the roots of a set of switching functions

$$g_j^\ell(t, x^\ell, \dot{x}^\ell), \quad \text{with } j = 1, \ldots, n_T^\ell,\ \ell \in \mathbb{M}$$

in an interval $[t_i, t_{i+1}]$. The chosen method is an adapted version of the root finding procedure of the SUNDIALS code IDA [10] that checks for sign changes of any $g_j^\ell(t, x^\ell, \dot{x}^\ell)$ in $[t_i, t_{i+1}]$ and then computes the roots with a modified secant method. At first, the algorithm checks if g_j^ℓ has an exact root at t_i. If an exact root of any g_j^ℓ is found at time t_i, then $g_j^\ell(t_i + \delta, x^\ell, \dot{x}^\ell)$ is computed for a small increment $\delta > 0$. If $g_j^\ell(t_i + \delta, x^\ell, \dot{x}^\ell) = 0$ also has an exact root, then the procedure stops with an error message. In this way, it is guaranteed that the values of all g_j^ℓ are nonzero at some past value of t_i, beyond which a search for roots is done. In the next step, if no roots at t_i were found, the algorithm checks g_j^ℓ at t_{i+1} for exact zeros and detect sign changes in (t_i, t_{i+1}). If no sign changes are found, then either a root is reported if some $g_j^\ell(t_{i+1}, x^\ell, \dot{x}^\ell) = 0$, or no root is found in (t_i, t_{i+1}). If one or more sign changes were found, then a loop is entered to locate the roots within a tolerance $TTOL$ given by

$$TTOL = 100\, U(|t_{i+1}| + |h|), \tag{16}$$

where U is the rounding unit of the machine. This is the default value for the tolerance $TTOL$ used in the root finding procedure, it can be adapted to a specific problem by the user, see Section 4.3. When sign changes are found in two or more switching functions g_j^ℓ, $j = 1, \ldots, n_T^\ell$, then the one with the largest value of

$$\frac{|g_j^\ell(t_{i+1}, x^\ell, \dot{x}^\ell)|}{|g_j^\ell(t_{i+1}, x^\ell, \dot{x}^\ell) - g_j^\ell(t_i, x^\ell, \dot{x}^\ell)|},$$

corresponding to the secant method value closest to t_i, is the one where most likely the sign change occurs first. At each pass through the loop, a new value t_{mid} within the search interval is set and the values of $g_j^\ell(t_{mid}, x^\ell, \dot{x}^\ell)$ are checked. The point t_{mid} is computed via

$$t_{mid} = t_{i+1} - g_j^\ell(t_{i+1}, x^\ell, \dot{x}^\ell) \frac{t_{i+1} - t_i}{g_j^\ell(t_{i+1}, x^\ell, \dot{x}^\ell) - \omega g_j^\ell(t_i, x^\ell, \dot{x}^\ell)}, \tag{17}$$

where ω is a weight parameter. On the first two passes through the loop, ω is set to 1, such that t_{mid} is the classical secant method value. Afterwards, ω is reset according to the side of the subinterval in which the sign change was found in the previous two steps. The value of t_{mid} is closer to t_i when $\omega < 1$ and closer to t_{i+1} when $\omega > 1$. If the value of t_{mid} in (17) is within $\frac{TTOL}{2}$ of t_i or t_{i+1}, it is adjusted inward, such that its distance from the endpoint relatively to the interval size is between 0.1 and 0.5, with 0.5 being the midpoint, and the actual distance from the endpoint is at least $\frac{TTOL}{2}$. Then, either t_i or t_{i+1} is reset to t_{mid} according to which subinterval

is found to have the sign change. If there is no sign change in (t_i, t_{mid}), then that root is reported. The loop continues until $|t_{i+1} - t_i| < TTOL$, and then the reported root location is t_{i+1}.
In general, the root finding procedure is only able to find roots of odd multiplicity that correspond to a sign change in one of the switching functions $g_j^\ell(t, x^\ell, \dot{x}^\ell)$, or exact zeros at t_i or t_{i+1}. If more than one switching function g_j^ℓ has a root in the given interval or if multiple roots are found for one switching function, then the one closest to t_i is returned. If a switching function has a root of even multiplicity, it may be missed. If such a root is desired, the switching function should be reformulated such that it changes sign at the desired root. In general, the switching functions should be chosen as simple as possible, e.g. linear, and if possible, different switching functions should be used for different mode transitions.

3.3 Interpolation

During the numerical solution of a hybrid system the state of the system at a switch point is required. In addition, in order to evaluate the switching functions in the root-finding process, the numerical solution is required between gridpoints. Therefore, we need a continuous representation of the solution that enables an efficient interpolation of the approximate solution between gridpoints. For discretization methods based on a polynomial representation of the solution or its derivatives, like BDF methods or Runge-Kutta methods based on collocation, the construction of a continuous solution representation is straightforward as it is given by construction of the method, see [8]. Since the interpolated values are used for continuing the integration the order q of the interpolation should be the same as the order p of the discretization method. Further, the interpolation error is propagated as it influences the determination of the switch points and the computation of the initial values used to restart the integration method such that an error controlled continuous representation of the solution is required. Unfortunately, for many collocation methods the order of the continuous representation is lower that the convergence order of the method, e.g., the continuous representation of the 5th order Radau IIa method has order $q = 3$, see [7]. Alternatively, other interpolation schemes, as e.g. Hermite interpolation, can be applied if collocation Runge-Kutta methods are used as integration methods. For BDF methods the interpolant is of the same order as the method that was used to advance the solution from t_i to t_{i+1}. Note, that for integration methods based on a polynomial representation as BDF or Runge-Kutta methods the interpolation error is automatically error controlled by the error control of the numerical solution. The interpolant is continuous, but it has a discontinuous derivative at the meshpoints. It is also possible to define a continuously differentiable interpolant, see [3, 22], which leads to a more robust code for root finding. Note, that BDF and Runge-Kutta methods guarantee consistency of differential and algebraic variables at meshpoints, but not necessarily at interpolated points such that the interpolated values at the switch points are in general not consistent with the algebraic equations and consistent values have to be computed to restart the integration.

3.4 Consistent Reinitialization After Mode Switching

One of the difficulties in the numerical integration of hybrid differential-algebraic equations is the computation of consistent initial values at the switch points to restart the numerical integration. The code checks the initial values specified by the user-defined transition functions (5) for consistency, and if necessary computes new consistent initial value on the basis of the given but inconsistent final state of the predecessor mode. To determine a consistent initial value the underdetermined nonlinear system

$$\mathcal{F}_{\mu^\ell}^\ell(\tau_i, x^\ell, \dot{x}^\ell, ..., x^{\ell(\mu^\ell+1)}) = 0 \tag{18}$$

has to be solved for $(x^\ell, \dot{x}^\ell, ..., x^{\ell(\mu^\ell+1)})$ considered as algebraic variables. The solution of this nonlinear system is computed in a least squares sense with the subroutine NLSCON [17] which is an implementation of the Gauss-Newton method [18]. In the beginning of the integration the

user must supply a guess of the initial values as a starting value for the Gauss-Newton iteration. Since the Jacobian of (18) has full row rank at a solution by Hypothesis 1, we have local quadratic convergence of the Gauss-Newton method, provided that the initial guess is sufficiently close to the solution.
In the computation of consistent initial values the solver enables the user to choose the differential variables to be kept fixed during the Gauss-Newton iterations and only algebraic components are chosen consistently with the DAE in the current mode. Setting the columns of the Jacobian of (18) that corresponds to differential components to zero, then these components of the initial guess are kept fixed during the Gauss-Newton iteration and only consistent values for the algebraic variables are computed. We must only guarantee that the remaining columns of the Jacobian still has full row rank which corresponds to the classification of a component of x^ℓ to be a differential variable. Let $\tilde{T}_{\ell,2}$ be a fixed approximation to $T_{\ell,2}$ at the desired solution, then we solve

$$\mathcal{F}^\ell_{\mu^\ell}(\tau_i, \tilde{T}_{\ell,2}\tilde{T}^T_{\ell,2}\hat{x}^\ell + (I - \tilde{T}_{\ell,2}\tilde{T}^T_{\ell,2})x^\ell, \dot{x}^\ell, \ldots, x^{\ell(\mu^\ell+1)}) = 0$$

for $(x^\ell, \dot{x}^\ell, \ldots x^{\ell(\mu+1)})$, where $\tilde{T}_{\ell,2}\tilde{T}^T_{\ell,2}$ is an orthogonal projection of rank d^ℓ and $\hat{x}^\ell$ is the initial guess. In this way, differential variables can be progressed continuously after a mode switch. The user can choose variables to be kept fixed during the Gauss-Newton iterations by setting up the IFIX-array, see Section 4.2.1. In this case, the corresponding columns of the Jacobian are set to zero. Note that this will lead to a rank drop in the Jacobian if any of the algebraic variables are are chosen to be kept fixed. In this case the code returns an error message.

3.5 Sliding Mode Simulation

The code DGESDA enables sliding mode simulation. During the simulation the code detects chattering, i.e., repeated switching between modes, either by checking the sliding condition (11) using the approximation (12), or by checking if a maximal allowed number of successive mode switchings in a short time period is exceeded. If the distance between the last three detected switch points τ_{i+1}, τ_i, τ_{i-1} is lower than a chattering tolerance $TOLC$, i.e., if $\tau_{i+1} - \tau_i < TOLC$ and $\tau_i - \tau_{i-1} < TOLC$ with τ_{i+1} the last detected switch point, then the code assumes that chattering occurs. The default value for the chattering tolerance $TOLC$ is given by $TOLC = 10^{-3}$ and the default value for the parameter δ that is used in the approximation of the directional derivatives (12) is given by $\delta = 10^{-5}$. These default values can be adapted to the problem by the user, see Section 4.3. Sliding mode simulation is initiated by the user using reverse communication. If chattering occurs, then the code stops with a warning message. The user can decide to switch to sliding mode, otherwise, the integration is resumed until a user-defined maximal number of switchings has occurred and the code stops with an error message. In the case of sliding mode simulation, the user has to provide the DAE describing the system during sliding motion that can be defined as in (14). Further, the user can completely disable the detection of chattering and the checking of the sliding condition, see Section 4.3.

References

[1] P.I. Barton and C.K. Lee. Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Trans. on Modeling and Computer Simul.*, 12(4):256–289, 2002.

[2] P.I. Barton and T. Park. State event location in differential-algebraic models. *ACM Trans. on Modeling and Computer Simul.*, 6(2):137–165, 1996.

[3] M. Berzins. A C^1 interpolant for codes based on backward difference formulae. *Applied Numerical Mathematics*, 2:109–118, 1986.

[4] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*, volume 14 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 1996.

[5] E. Eich-Soellner and C. Führer. *Numerical Methods in Multibody Dynamics*. B.G. Teubner, Stuttgar, 1998.

[6] A.F. Filippov. *Differential Equations with Discontinuous Right-hand Sides*. Kluwer, 1998.

[7] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin, 1993.

[8] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, second edition, 1996.

[9] P. Hamann and V. Mehrmann. Numerical solution of hybrid differential-algebraic equations. *Comp. Meth. App. Mech. Eng*, 197(6-8):693–705, 2008.

[10] A. C. Hindmarsh, R. Serban, and A. Collier. *User Documentation for ida v2.5.0*. Center for Applied Scientific Computing Lawrence Livermore National Laboratory, November 6 2006.

[11] P. Kunkel and V. Mehrmann. Canonical forms for linear differential-algebraic equations with variable coefficients. *J. Comput. Appl. Math.*, 56:225–259, 1994.

[12] P. Kunkel and V. Mehrmann. Analysis of over- and underdetermined nonlinear differential-algebraic systems with application to nonlinear control problems. *Math. Control, Signals, Sys.*, 14:233–256, 2001.

[13] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations — Analysis and Numerical Solution*. EMS Publishing House, Zürich, Switzerland, 2006.

[14] P. Kunkel, V. Mehrmann, W. Rath, and J. Weickert. GELDA: A software package for the solution of general linear differential algebraic equations. *SIAM J. Sci. Comput.*, 18:115 – 138, 1997.

[15] P. Kunkel, V. Mehrmann, and I. Seufer. GENDA: A software package for the numerical solution of general nonlinear differential-algebraic equations. Report 730-02, Technische Universität Berlin, 2002.

[16] V. Mehrmann and L. Wunderlich. Hybrid systems of differential-algebraic equations – analysis and numerical solution. Technical Report 531-2008, DFG Research Center MATHEON in Berlin, 2008.

[17] U. Nowak and L. Weimann. A family of Newton codes for systems of highly nonlinear equations - Algorithm, Implementation, Application. Technical Report 91-10, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1991.

[18] J. Ortega and W. Rheinboldt. *Iterative Solutions of Nonlinear Equations in Several Variables*, volume 30 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 2000.

[19] G. Reißig. Differential-algebraic equations and impasse points. *IEEE Transactions on Circuits and Systems*, 43(2):122–133, 1996.

[20] A. Steinbrecher. *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems*. Dissertation, Technische Universität Berlin, 2006.

[21] V.I. Utkin. *Sliding Modes in Control Optimization*. Springer, 1992.

[22] H. A. Watts. A smooth output interpolation process for BDF codes. *Journal of Applied Mathematics and Computing*, 31:397–418, 1989.

[23] L. Wunderlich. *Analysis and Numerical Solution of Structured and Switched Differential-Algebraic Systems*. PhD thesis, TU Berlin, 2008.

4 Documentation

4.1 Specification

```
      SUBROUTINE DGESDA(INFO, MATSUB, FUN, DFUN, USCAL, UINTER, TRANSF,
     $                  MDCHNG, GFUN, MXTRAN, TRAN, SOLVER, METHOD, M, N,
     $                  NMAX, T, TOUT, TS, LTS, NSP, X, XPRIME, CVAL, IPAR,
     $                  LIPAR, RPAR, IFIX, IDIFCO, SCALC, SCALR, RTOL, ATOL,
     $                  MDHIST, IWORK, LIW, RWORK, LRW, STATS, DWORK, ISMTH,
     $                  IWARN, IERR)

      EXTERNAL          MATSUB, FUN, DFUN, USCAL, UINTER, TRANSF, MDCHNG, GFUN
      INTEGER           LIW, LRW, IWARN, IERR, MXTRAN, M(*), N(*), NMAX,
                        CVAL(5,*), IPAR(*), LIPAR, METHOD(*), IWORK(*),
     $                  INFO(27), IFIX(*), SOLVER(*), LTS, NSP, ISMTH,
     $                  MDHIST(*), STATS(5), IDIFCO(*), TRAN(*)
      DOUBLE PRECISION  T, TOUT, TS(*), X(*), XPRIME(*), RPAR(*), SCALC(*),
     $                  SCALR(*), RTOL(*), ATOL(*), RWORK(*), DWORK(*)
```

4.2 Argument List

4.2.1 User–supplied Subroutines

MATSUB – User–supplied function.

This is a subroutine which the user provides to define the matrices $E^\ell(t)$, $A^\ell(t)$ and the vector $f^\ell(t)$ and their derivatives for all modes $\ell \in \mathbb{M}$ for linear problems of the form (2). It is of the form

SUBROUTINE MATSUB(IMAT, M, N, T, IDIF, W, LDW, IPAR, RPAR, IERR).

The subroutine takes as input the number of equations M in the current mode, the number of unknowns N in the current mode, the time T and the integer parameters IMAT and IDIF. Further, the integer and double precision arrays IPAR and RPAR that can be used for communication between the calling program and the MATSUB subroutine. MATSUB must not alter these input parameters or the leading dimension LDW of the array W. As output, the subroutine produces the IDIF-th derivative of $E^\ell(t)$ if IMAT= 1, of $A^\ell(t)$ if IMAT= 2 and of $f^\ell(t)$ if IMAT= 3, each at time T in the leading M by N part of the array W. The integer flag IERR is always zero on input and MATSUB should alter IERR only if IDIF is larger than the highest derivative the subroutine provides (set IERR = -2) or if another problem occurs (set IERR = -1). IPAR(1) contains the number of the current mode $\ell \in \mathbb{M}$ and IPAR(2) contains the total number of modes N_F. MATSUB should alter IERR if IPAR(1) is larger than the number of modes the subroutine provides (set IERR=-3). If there are no linear DAEs in the system, MATSUB must be a dummy-subroutine.

Note: In the calling program MATSUB must be declared as external.

FUN – User–supplied function.

This is a subroutine which the user provides to define the left hand side of the inflated DAE $\mathcal{F}^\ell_{\mu^\ell}(x^\ell, \dot{x}^\ell, ..., x^{\ell(\mu^\ell+1)})$ as in (6) for nonlinear DAEs in modes $\ell \in \mathbb{M}$. It has the form

SUBROUTINE FUN (T, IDIF, X, F, IPAR, RPAR, IERR).

The subroutine takes as input the time T, the vector X containing an approximation to the solution $\bar{x}^\ell = (x^\ell, \dot{x}^\ell, ..., x^{\ell(\mathrm{MXINDX}+1)})$, where MXINDX$\geq \mu^\ell$ and the integer parameter IDIF. Further, the integer and double precision arrays IPAR and RPAR that can be used for

communication between the calling program and the FUN subroutine. IPAR(1) contains the number of the current mode $\ell \in \mathbb{M}$ and IPAR(2) contains the total number of modes N_F. As output, the subroutine produces the IDIF-th derivative of the DAE in the current mode at time T and state X in the first M elements of the 1-dimensional array F. FUN should alter IERR if IPAR(1) is larger than the number of modes the subroutine provides (set IERR=-3). If there are no nonlinear DAEs in the system FUN must be a dummy-subroutine.

Note: In the calling program, FUN must be declared as external.

DFUN – User–supplied function.

This is a subroutine which the user provides to define the Jacobian of the inflated DAE $\mathcal{F}^{\ell}_{\mu^{\ell}}(t, x^{\ell}, \dot{x}^{\ell}, ..., x^{\ell(\mu^{\ell}+1)})$ for all modes $\ell \in \mathbb{M}$. It is of the form

SUBROUTINE DFUN (T, IDIF, X, JAC, LDJAC, IPAR, RPAR, IERR).

The subroutine takes as input the time T, the vector X containing an approximation to the solution $\bar{x}^{\ell} = (x^{\ell}, \dot{x}^{\ell}, ..., x^{\ell(\mathrm{MXINDX}+1)})$ where MXINDX$\geq \mu^{\ell}$ and the integer parameter IDIF. IPAR(1) contains the number of the current mode $\ell \in \mathbb{M}$ and IPAR(2) contains the total number of modes N_F. As output, the subroutine produces all partial derivatives of the IDIF-th derivative of $F^{\ell}(t, x^{\ell}, \dot{x}^{\ell})$ of the DAE in the current mode with respect to all entries of X in the first M rows and (IDIF+2)∗N columns of the 2-dimensional array JAC. If there are no nonlinear DAEs in the system DFUN must be a dummy-subroutine.

Note: In the calling program, DFUN must be declared as external.

USCAL – User–supplied function.

This is a subroutine which the user can provide for scaling purposes if INFO(13)=1 and IWORK(16)=2 using the solver DGENDA. It is of the form

SUBROUTINE USCAL(MQ, NQ, A, LDA, SCALC, SCALR, IERR)

and takes as input the number of rows MQ and the number of columns NQ of the extended derivative array, and the array A of dimension (LDA, MQ). The leading NQ-by-MQ part of A must contain the matrix to be scaled. Furthermore, USCAL takes as input the arrays SCALC and SCALR of dimensions NQ and MQ respectively. As output, the subroutine produces column scale factors SCALC and row scale factors SCALR and overwrites the array A with the matrix B such that B=SCALR∗A∗SCALC. The integer error flag IERR should be set to a negative value if there was any illegal input. If INFO(13)=0 or IWORK(16)≤1, USCAL must be a dummy-subroutine.

Note: In the calling program, USCAL must be declared as external.

UINTER – User–supplied function.

This is a subroutine which the user provides to check transition conditions and to interrupt the integration. It is of the form

SUBROUTINE UINTER (T, X, XPRIME, RPAR, IPAR, ATOL, IRTRN)

The subroutine takes as input the time T, the vector X containing an approximation to the solution and the vector XPRIME containing the computed first derivative of the solution approximated at T and the absolute error tolerance ATOL. IPAR and RPAR are integer and real arrays which can be used for the communication between the user's calling program and the subroutine UINTER. IPAR(1) contains the number of the current mode $\ell \in \mathbb{M}$ and IPAR(2) contains the total number of modes N_F. If INFO(23)=0, then the subroutine UINTER is called after each intermediate step in the integration by the solver DGELDA or

DGENDA to check if a transition conditions is satisfied. As output, the subroutine produces the integer IRTRN, which indicates if the integration is to be interrupted.

If IRTRN = 0 the integration is continued,
If IRTRN =-1 the integration is interrupted.

If INFO(23)=1, i.e., no switching is desired, UINTER must be a dummy-subroutine.

Note: In the calling program, UINTER must be declared as external.

TRANSF – User–supplied function.

This is a subroutine which the user provides to define the transition function of the switched DAE. It is of the form

SUBROUTINE TRANSF (XNEW, XOLD, N, MODNEW, MODOLD, IERR)

The subroutine takes as input the vector XOLD containing an approximation to the solution in the predecessor mode, the integer N containing the size of X, and the integers MODNEW and MODOLD defining the successor and predecessor modes. As output, the subroutine provides the vector XNEW containing the solution after transfer to the successor mode. If an error occurs IERR should be < 0. The integer flag IERR is always zero on input and TRANSF should alter IERR if a problem occurs (set IERR $= -1$). If INFO(23)=1, i.e., no switching is desired TRANSF must be a dummy-subroutine.

Note: In the calling program, TRANSF must be declared as external.

MDCHNG – User–supplied function.

This is a subroutine which the user provides to define the mode changes of the switched DAE. It is of the form

SUBROUTINE MDCHNG (MODE, J, NMODE, MXTRAN, NEWMOD, IERR)

The subroutine takes as input the integer MODE defining the current mode, the integer J providing the active transition, the integer NMODE providing the number of modes and the integer MXTRAN providing the maximal number of transitions. As output, the subroutine produces the successor mode NEWMOD according to the Jth transition coming from mode MODE. The integer flag IERR is always zero on input and MDCHNG should alter IERR if a problem occurs (set IERR $= -1$). If INFO(23)=1, i.e., no switching is desired MDCHNG must be a dummy-subroutine.

Note: In the calling program, MDCHNG must be declared as external.

GFUN – User–supplied function.

This is a subroutine which the user provides to define the switching functions $g_j^\ell(t, x^\ell, \dot{x}^\ell)$ of the switched DAE for each mode $\ell \in \mathbb{M}$. It is of the form

SUBROUTINE GFUN(T, IDIF, X, XPRIME, GT, NRTFN, IPAR, RPAR, IERR).

The subroutine takes as input the time T, and integer IDIF, the vector X containing an approximation to the solution, the vector XPRIME containing an approximation to the derivative of the solution and NRTFN, the number of root functions in the current mode $\ell \in \mathbb{M}$. IPAR is an integer array which can be used for the communication between the user's calling program and the subroutine. IPAR(1) contains the current mode and IPAR(2) contains the total number of modes. As output, the subroutine produces the array GT, of length NRTFN, with components $g_j^\ell(t, x^\ell, \dot{x}^\ell)$. The integer flag IERR is always zero on input. The subroutine should return a non-zero value if an error occurred. If INFO(23)=1, i.e., no switching is desired GFUN must be a dummy-subroutine.

Note: In the calling program, GFUN must be declared as external.

4.2.2 Arguments In

INFO – INTEGER array of DIMENSION (27).

The basic task of the code is to solve the switched system from T to TOUT and return an answer at TOUT. INFO is an integer array which is used to communicate exactly how the user wants this task to be carried out. The simplest use of the code corresponds to setting all entries of INFO to 0. To enable mode switching INFO(23) must be set to 0. For details see Section 4.3.

METHOD – INTEGER array of DIMENSION (NMODE).

Indicates which integration method should be used in mode I:

METHOD(I)=1 the code uses the BDF solver,

METHOD(I)=2 the code uses the Runge-Kutta solver.

Note: For nonlinear problems only a BDF solver is implemented. Setting METHOD(I)=2 here, yields an error message.

SOLVER – INTEGER array of DIMENSION (NMODE).

Indicates which solver should be used in each mode I as follows:

SOLVER(I)=1 the code uses the solver DGENDA.

SOLVER(I)=2 the code uses the solver DGELDA. If the problem is nonlinear an error is reported.

M – INTEGER array of DIMENSION (NMODE).

The number of equations in mode I. $M(I) \geq 1$.

N – INTEGER array of DIMENSION (NMODE).

The number of components of x^I in mode I. $N(I) = M(I)$ in this version of DGENDA.

NMAX – INTEGER.

The maximal size of the vector X: NMAX= $MAX[(MXINDX + 2) * N(I)]$.

T – DOUBLE PRECISION.

The initial point of the integration.
Note: This scalar is overwritten.

TOUT – DOUBLE PRECISION.

The point at which a solution is desired. Integration either forward in T (TOUT > T) or backward in T (TOUT < T) is allowed. At the beginning of the integration the user can set T = TOUT. If INFO(11)=0 consistent initial values will be computed and stored in X.

X – DOUBLE PRECISION array of DIMENSION (NMAX).

If INFO(11)=0, this array may contain a guess for the initial value and for DGENDA in addition for the first NMAX derivatives at the initial time T. A consistent initial value and consistent initial derivatives close (in the least square sense) to this guess are then computed. If no guess is available, all elements of X are set to zero.

If INFO(11)=1, this array must contain consistent initial values of the NMAX solution components and for DGENDA in addition for the first NMAX derivatives at the initial time.
Note: This array is overwritten.

CVAL – INTEGER array of DIMENSION (5, NMODE).

Contains the characteristic values of the DAE in mode I:

CVAL(1, I) contains the strangeness index μ^I in mode I,

CVAL(2, I) contains the number d^I of differential components in mode I,

CVAL(3, I) contains the number a^I of algebraic components in mode I,

CVAL(4, I) contains the number u^I of undetermined components in mode I,

CVAL(5, I) contains the number v^I of vanishing equations in mode I.

IPAR – INTEGER array of DIMENSION (LIPAR).

This integer array can be used for communication between the calling program and the user supplied subroutines. The use of the first three entries is defined as follows.

IPAR(1) contains the initial mode (this is overwritten with the current mode),

IPAR(2) contains the number of modes,

IPAR(3) contains the sliding mode (if dot defined set =0).

If INFO(23)=1, i.e., no switching is desired, IPAR is not altered by DGESDA or its subprograms. If this array is used, it must be dimensioned in the calling program and in the user supplied subroutines as array of appropriate length. Otherwise, this array is ignored by treating it as dummy array of length one.

LIPAR – INTEGER.

The length of IPAR.

RPAR – DOUBLE PRECISION array of DIMENSION (*).

This real array can be used for communication between the calling program and the user supplied subroutines. RPAR is not altered by DGESDA or its subprograms. If this array is used, it must be dimensioned in the calling program and in the user supplied subroutines as array of appropriate length. Otherwise, this array is ignored by treating it as dummy array of length one.

IFIX – INTEGER array of DIMENSION (NMAX).

Only used in the solver DGENDA. If INFO(12)=1, the user can set IFIX(I)=1 if he wants to keep the value of X(I) fixed when DGENDA tries to compute consistent initial values. All other entries of IFIX should be set to zero. If INFO(23)=0, i.e., mode switching is desired the array IDIFCO has to be used to specify which components should be fixed.

Note: This may lead to a rank-deficient Jacobian.

IDIFCO – INTEGER array of DIMENSION (NMODE*NMAX).

Contains information of differential components. By setting IDIFCO(I)=1 the corresponding variables is kept fixed during the computations of consistent initial values.

SCALC – DOUBLE PRECISION array of DIMENSION ((MXINDX+2)NMAX).

Only used in DGENDA. User scaling of the iteration vector X. If set to zero DGENDA will try to calculate an appropriate SCALC.

SCALR – DOUBLE PRECISION array of DIMENSION ((MXINDX+1)NMAX).

Only used in DGENDA. User row scaling of the Jacobian. Only applicable as input if INFO(13)=1 and the user supplied scaling subroutine USCAL takes SCALR as an input argument.

MXTRAN – INTEGER.

The maximal number of transitions.

TRAN – INTEGER array of DIMENSION (NMODE).

The number of possible transitions for each mode.

LTS – INTEGER.

The length of the array TS containing the switch points. The length of TS must be set in the calling program. If more than LTS switching points occur during the integration the codes stops with an error message. In this case probably chattering has occurred.

RTOL – DOUBLE PRECISION array of DIMENSION (*).

The relative error tolerances which the user provides to indicate how accurately he wishes the solution to be computed. The user may choose RTOL and ATOL to be both scalars or else both vectors for all modes. If RTOL and ATOL are scalars (INFO(2) = 0) the user has to declare this array to be RTOL(1).

ATOL – DOUBLE PRECISION array of DIMENSION (*).

The absolute error tolerances which the user provides. If ATOL and RTOL are scalars (INFO(2) = 0) the user has to declare this array to be ATOL(1). For details on the use of the error tolerances see the documentation of DGELDA [14] and DGENDA [15].

4.2.3 Arguments Out

T – DOUBLE PRECISION.

The solution was successfully advanced to the output value of T.

X – DOUBLE PRECISION array of DIMENSION (NMAX).

Contains the computed approximation of the solution at time T in its first NMAX elements. For DGENDA the further elements contain approximations to the first (MXINDX+1) derivatives of the solution at the last time step of the BDF solver.

XPRIME – DOUBLE PRECISION array of DIMENSION (NMAX).

Contains the computed first derivative of the solution approximated at T.

CVAL – INTEGER array of DIMENSION (5, NMODE).

Contains the characteristic values of the DAE as described in 4.2.2. If INFO(16)=1 and DGENDA detects any changes in the characteristic values, the code will return with an error message and CVAL will be overwritten with the new characteristic values.

TS – DOUBLE PRECISION array of DIMENSION (LTS).

Contains the switching points found during the integration.

NSP – INTEGER.

Contains the number of switching points found during the integration.

IWORK – INTEGER array of DIMENSION at least (LIW).

This integer arrays provide the workspace. The entries are slightly different for the use in DGELDA and DGENDA. Usually the information they contain are of no interest, but sometimes the following may be useful:

IWORK(7) contains the order of the method to be attempted on the next step.

IWORK(8) contains the order of the method used on the last step.

IWORK(11) contains the number of steps taken so far in the current mode.

IWORK(12) contains the number of calls to MATSUB or FUN in the current mode.

IWORK(13) contains the number of factorizations of the Jacobian in the current mode.

IWORK(14) contains the total number of error test failures so far in the current mode.

LIW – INTEGER.

The length of IWORK.
For DGENDA we need: LIW $\geq 127 + (\text{MXINDX} + 2)\text{N}$.

For DGELDA we need: LIW $\geq 20+2\text{N}$ if METHOD $= 1$, LIW $\geq 20+6\text{N}$ if METHOD $= 2$.

RWORK – DOUBLE PRECISION array of DIMENSION at least (LRW).

This real arrays provide the workspace. The entries are slightly different for the use in DGELDA and DGENDA. Usually the information they contain are of no interest, but sometimes the following may be useful:

RWORK(3) contains the stepsize H to be attempted on the next step.

RWORK(4) contains the current value of the independent variable, i.e., the farthest point integration has reached. This will be different from T only when interpolation has been performed (IERR $= 3$).

RWORK(7) contains the stepsize used on the last successful step.

LRW – INTEGER.

The length of RWORK.
For DGENDA we need: LRW $\geq 3N$.

For DGELDA we need:

If METHOD $= 1$ then

$$\begin{aligned}\text{LRW} \quad \geq \quad & 50 + (21 + \text{MAXORD} + 14\text{MXINDX})\text{N} \\ & +(6 + \text{MXINDX} + 6(\text{MXINDX} + 1)^2)\text{N}^2.\end{aligned}$$

If METHOD $= 2$ then

$$\begin{aligned}\text{LRW} \quad \geq \quad & 20 + (29 + 14\text{MXINDX})\text{N} \\ & +(15 + \text{MXINDX} + 6(\text{MXINDX} + 1)^2)\text{N}^2.\end{aligned}$$

For good performance, LRW should generally be larger.

DWORK – DOUBLE PRECISION array of DIMENSION (4+NMAX).

providing the workspace used in DGESDA. (See documentation of the INFO array).

DWORK(1) contains the tolerance TTOL used in the root finding procedure DRTFND.

DWORK(2) contains the chattering tolerance TOLC.

DWORK(3) contains the tolerance δ in the approximation of the sliding condition.

DWORK(4) contains the maximal allowed number MAXCGN of immediate transitions.

DWORK(5) contains the interpolated and transferred solution at the last switch point.

ISMTH – INTEGER.

Integer indicating if nonsmooth transitions occur.

ISMTH = 0 only smooth transitions occur.

ISMTH = -1 non-smooth transition due to transition function occur.

ISMTH = -2 non-smooth transition due to consistent initialization occur.

MDHIST – INTEGER array of DIMENSION (LTS+1).

Contains the mode history.

STATS – INTEGER array of DIMENSION(5).

The array STATS contains some statistic of the integration:

STATS(1) contains the number of integration steps so far.

STATS(2) contains the number of calls to MATSUB or FUN so far.

STATS(3) contains the number of factorizations so far.

STATS(4) contains the number of error test failures.

STATS(5) contains the number of convergence test failures.

4.2.4 Error and Warning Indicator

IERR – INTEGER.

Unless the code detects an error, IERR contains a positive value on exit.

IERR = 1: in the BDF solver a step was successfully taken in the intermediate output mode. The code has not yet reached TOUT.

IERR = 2: The integration to TOUT was successfully completed (T=TOUT) by stepping exactly to TOUT.

IERR = 3: The integration to TOUT was successfully completed (T=TOUT) by stepping past TOUT. X is obtained by interpolation.

IERR = 4: At the first step the user set T=TOUT and the code computed successfully the strangeness index and the other characteristic quantaties. Furthermore, if INFO(11)=0 consistent initial values are stored in X.

For a description of the errors occurring in the routines DGELDA or DGENDA see the documentations of these codes [14, 15].

IERR = $-1 \cdots -9$: An error occurred in the integration routine DGELDA or DGENDA.

IERR = -11: The integration was stopped due to a call to UINTER (IRTRN=-1).

IERR = $-20 \cdots -27$: An error occurred in the integration routine DGELDA or DGENDA.

IERR = $-101.. -131$: An error occurred in the integration routine DGELDA or DGENDA.

IERR = -201: Some element of the INFO vector is not zero or one.

IERR = -202: The number of modes is <= 0.

IERR = -203: The current mode exceeds the number of modes.

IERR = -204: $N^\ell \leq 0$ or $M^\ell \leq 0$.

IERR = −205: Mode chattering occurs

IERR = −206: METHOD(I)> 1 not possible for the nonlinear case.

IERR = −207: Wrong maximal transition number.

IERR = −208: TRAN(I) has an invalid value.

IERR = −209: TTOL has an invalid value.

IERR = −211: An error was signalled by the subroutine DRTFND.

IERR = −212: IERR in MDCHNG has a negative value.

IERR = −213: IERR in TRANSF has a negative value.

IERR = −215: Chosen solver not implemented yet.

IERR = −216: Wrong transition function detected: J≤ 0 or J> IPAR(3).

IERR = −217: Number of switch points exceeds LTS. Probably chattering has occurred!

IERR = −218: Wrong arguments in DWORK array.

IERR = −219: No root found by DRTFND, but an event was detected by UINTER.

IERR = −220: Too many immediate mode changes occur.

IERR = −221: An error was signalled by the subroutine DCKCON.

IERR = −222: Invalid value in IDIFCO.

IERR = −223: An immediate mode change occur and TRAN(I)>1.

IWARN – INTEGER.

Integer containing warnings:

IWARN > 0 the strangeness index has changed and IWARN contains its new value.

IWARN $= -1$ a high number of switch points occur, possibly chattering occurs!

IWARN $= -2$ an immediate mode change occurs.

4.3 Setting up the INFO array

The INFO array is used to give the code more details about how the user wants the problem to be solved. The entries are slightly different in the use of the codes DGELDA and DGENDA, for details see documentation of these codes [14, 15]. The user must respond to the items, which are arranged as questions. The simplest use of the code corresponds to answering all questions as yes, i.e., setting all entries of INFO to 0. INFO(1)-INFO(22) are used in the solvers DGELDA and DGENDA. The following items are important in the use of DGESDA.

INFO(1) This parameter enables the code to initialize itself. The user must set it to indicate the start of every new problem. Is this the first call for this problem ?

Yes – Set INFO(1) = 0

No – Not applicable here. For continuing the integration, see [14, 15].

INFO(7) The user can specify a maximum stepsize HMAX, so that the code will avoid passing over very large regions. The default value for HMAX is 1.0. Do you want the code to decide on its own maximum stepsize ?

Yes – Set INFO(7)=0

No – Set INFO(7)=1 and define HMAX by setting RWORK(2)=HMAX.

INFO(23) If INFO(23)=0 the subroutine UINTER is called after each successful step in the integration, i.e., the integration can be interrupted if a transition condition is defined in UINTER. If INFO(23)=1 then no calls to UINTER are performed, i.e., no switching is possible. Do you want the code to call UINTER after each successful step ?

Yes – Set INFO(23)=0

No – Set INFO(23)=1

INFO(24) The root finding procedure DRTFND determines the switch point within a tolerance of TTOL. By default the TTOL is computed by TTOL =100U $(|t_{i+1}|+|h|)$ where U is the unit roundoff. Do you want to use the default tolerance ?

Yes – Set INFO(24)=0

No – Set INFO(24)=1 and define TTOL by setting DWORK(1)=TTOL.

INFO(25) The codes detect chattering behavior. By default the chattering tolerance TOLC is set to TOLC=10^{-5} and the parameter δ used for the approximation of the sliding condition is $\delta = 10^{-5}$. Do you want to use the default tolerances ?

Yes – Set INFO(25)=0

No – Set INFO(25)=1 and set DWORK(2)=TOLC and DWORK(3)= δ.

INFO(26) The codes enables sliding mode simulation. The detection of chattering can be disabled. Do you want the code to detect chattering ?

Yes – Set INFO(26)=0

No – Set INFO(26)=1

INFO(27) The codes allows a maximal number of immediate mode changes. The default value for the maximal allowed number of immediate transitions MAXCGN is 100. Do you want to use the default value ?

Yes – Set INFO(27)=0

No – Set INFO(27)=1 and define MAXCGN by setting DWORK(4)=MAXCGN.

A Numerical Example

We consider the example of stick-slip friction between two rigid bodies as depicted in Figure 2, see also [5]. The equations of motion of the two-body system with dry friction are given by

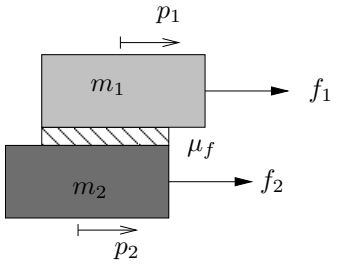

Figure 2: Stick-slip friction between rigid bodies

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & m_1 & 0 \\ 0 & 0 & 0 & m_2 \end{bmatrix} \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \\ \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ f_1 - \mu_f \|F_N\| \operatorname{sign}(v_1 - v_2) \\ f_2 + \mu_f \|F_N\| \operatorname{sign}(v_1 - v_2) \end{bmatrix} \tag{19}$$

where p_1, p_2 describe the positions and v_1, v_2 are the velocities of the bodies, f_1, f_2 are the applied forces, μ_f is the coefficient of friction, and F_N is the normal force on the contact surface. System (19) represents a hybrid system with two modes depending on the direction of the relative velocity, i.e., the system is in mode 1 if $\operatorname{sign}(v_1 - v_2) = 1$ and the system is in mode 2 if $\operatorname{sign}(v_1 - v_2) = -1$. Note that the model equations (19) cannot be applied if the relative velocity is $v_{rel} = v_1 - v_2 = 0$, since sign(0) is not defined. We apply a small hysteresis band of width 2ε around zero relative velocity to define the transition conditions

$$L_1^1(v_1, v_2) = v_1 - v_2 < -\varepsilon, \qquad L_1^2(v_1, v_2) = v_1 - v_2 > \varepsilon.$$

The corresponding switching functions are given by

$$g_1^1(v_1, v_2) = v_1 - v_2 + \varepsilon, \qquad g_1^2(v_1, v_2) = v_1 - v_2 - \varepsilon,$$

the mode allocation functions are simply given by

$$S^1(1) = 2, \qquad S^2(1) = 1,$$

and the transition functions are the identity mappings. The hybrid system (19) is solved with the solver DGESDA using the BDF method of the DAE solver DGELDA. Since (19) is an ordinary differential equation, the characteristic values in both modes are given by $\mu = 0$, $d_\mu = 4$, $a_\mu = v_\mu = u_\mu = 0$. The solution is computed in the interval $[0, 10]$ using $m_1 = m_2 = 1$, $f_1 = \sin(t)$, $f_2 = 0$, $F_N = 1$, $\mu_f = 0.4$, and $\varepsilon = 0.003$ with relative and absolute error tolerance $ATOL = RTOL = 10^{-8}$ and initial values $p_1(0) = p_2(0) = 1$, $v_1(0) = v_2(0) = 0$ in initial mode 1. The computed solution of the system and the relative velocity v_{rel} between the two bodies are given in Figure 3. In the beginning, both bodies move together since the applied force is less than the friction force. If the applied force exceeds the friction force the bodies slid over each other until the applied force is again lower than the friction force and the two bodies stick together once more. The relative velocity oscillates around zero during stiction resulting in a large number of integration steps. Altogether, the code requires 8209 integration steps, and 396 mode switches are detected. Physically, the oscillations around zero relative velocity do not occur, since during stiction both bodies move together. Using sliding mode simulation we can define the system behavior during

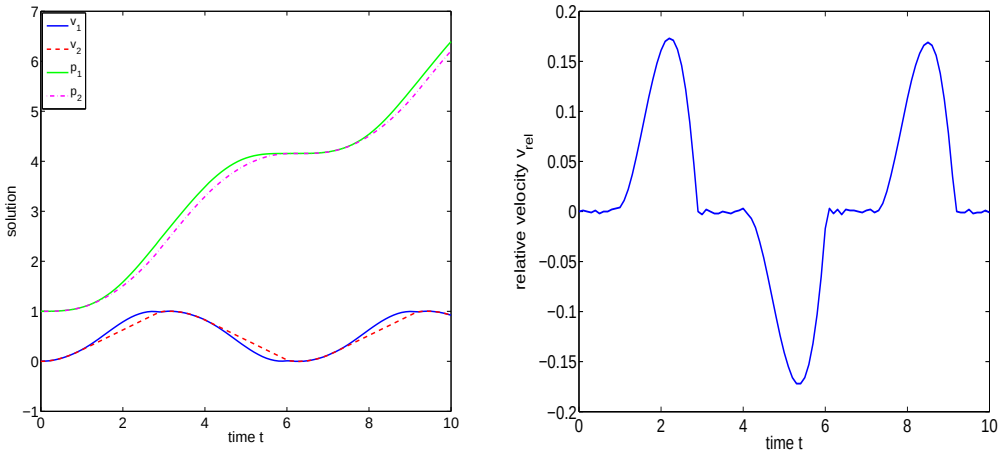

Figure 3: Solution of (19) and relative velocities between the two bodies

stiction for $v_{rel} = 0$ by

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 & 0 \\ 0 & 0 & 0 & m_2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \\ \dot{v}_1 \\ \dot{v}_2 \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_f\|F_N\| \\ 0 & 0 & 0 & 0 & -2\mu_f\|F_N\| \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ v_1 \\ v_2 \\ \alpha \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ f_1 - \mu_f\|F_N\| \\ f_2 + \mu_f\|F_N\| \\ 0 \end{bmatrix} \tag{20}$$

defining the DAE in sliding mode (mode 3). Now, system (20) represents a DAE with characteristic quantities $\mu = 1, d_\mu = 3$, and $a_\mu = 2$. The system stays in sliding mode as long as the applied force is less than the friction force. If the applied force exceeds the friction force, then the system leaves the sliding mode, i.e., if $f_1 + f_2 - 2\mu_f\|F_N\| > 0$, then the system switches back to mode 1, and if $f_1 + f_2 + 2\mu_f\|F_N\| < 0$, then the system switches back to mode 2. Again, the system is solved with DGESDA now using sliding mode simulation with chattering tolerance TOLC=0.01 and $\delta = 10^{-5}$. The solution of the system together with the relative velocity v_{rel} using sliding mode simulation is given in Figure 4. We can see that the oscillations in the relative velocity during

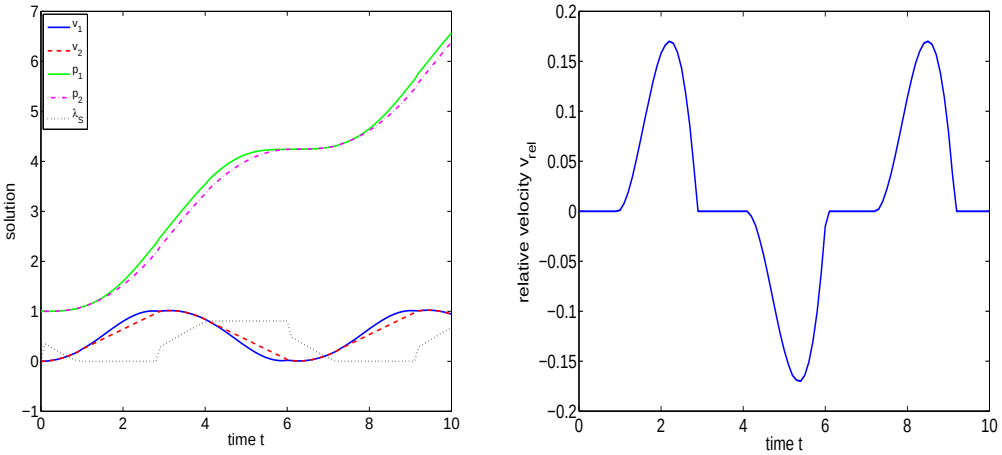

Figure 4: Solution and relative velocity using sliding mode simulation

stiction disappear. Altogether, 12 switch points are detected during the numerical integration

using sliding mode simulation and the number of integration step is reduced to 4737. The switch points together with the corresponding mode changes are given in Table 2.

Switch points $t^\star$	Mode changes	Switch points $t^\star$	Mode changes
0.0038	$1 \to 2$	2.8886	$1 \to 3$
0.0112	$2 \to 1$	4.0689	$3 \to 2$
0.0188	$1 \to 3$	6.0340	$2 \to 3$
0.9273	$3 \to 1$	7.2105	$3 \to 1$
2.8719	$1 \to 2$	9.1465	$1 \to 2$
2.8776	$2 \to 1$	9.1521	$2 \to 3$

Table 2: Detected switch points in the solution of (19) using sliding mode simulation

A.1 Program Text

```
C     .. Subroutine Arguments ..
C     .. Scalar Arguments ...
      INTEGER          NMODE, M1, N1, M2, N2, M3, N3, NQMAX, LIW,LRW,
     $                 NIPAR, NRPAR, MXTRAN, LTS
      PARAMETER       (NMODE=3, N1=4, M1=4,N2=4, M2=4, M3=5, N3=5,
     $                 NQMAX=20, LIW=200, LRW=50000, NIPAR=5, NRPAR=2,
     $                 MXTRAN=2, LTS=2000)
C     .. Array Arguments ...
      INTEGER          IWORK(LIW), CVAL(5,NMODE), M(NMODE), N(NMODE),
     $                 INFO(27), IFIX(NQMAX), IPAR(NIPAR), STATS(5),
     $                 TRAN(NMODE), IDIFCO(3*N3),
     $                 SOLVER(NMODE), METHOD(NMODE), MDHIST(LTS+1)
      DOUBLE PRECISION X(NQMAX), XPRIME(NQMAX), RWORK(LRW),RPAR(NRPAR),
     $                 SCALC(NQMAX), SCALR(NQMAX), RTOL, ATOL, TS(LTS),
     $                 DWORK(9)
      DOUBLE PRECISION T, TOUT, DSEC
      INTEGER          KPRINT, IERR, LUN, I, IWARN, NOUT, IOUT, NFE,
     $                 NJE, NST, NSP, NETF, NCTF, ISMTH
      PARAMETER        (KPRINT = 3, LUN = 6, NOUT = 100)
      DOUBLE PRECISION DTOUT(NOUT)
C     .. External Functions ..
      DOUBLE PRECISION DSECND
      EXTERNAL         DSECND
      EXTERNAL         FUN, DFUN, USCAL, UINTER, TRANSF, MDCHNG, GFUN,
     $                 MATSUB
C     .. Data Statements ..
      DATA             DTOUT / 0.1D0, 0.2D0, 0.3D0, 0.4D0, 0.5D0,
     $                         0.6D0, 0.7D0, 0.8D0, 0.9D0, 1.0D0,
     $                         1.1D0, 1.2D0, 1.3D0, 1.4D0, 1.5D0,
     $                         1.6D0, 1.7D0, 1.8D0, 1.9D0, 2.0D0,
     $                         2.1D0, 2.2D0, 2.3D0, 2.4D0, 2.5D0,
     $                         2.6D0, 2.7D0, 2.8D0, 2.9D0, 3.0D0,
     $                         3.1D0, 3.2D0, 3.3D0, 3.4D0, 3.5D0,
     $                         3.6D0, 3.7D0, 3.8D0, 3.9D0, 4.0D0,
     $                         4.1D0, 4.2D0, 4.3D0, 4.4D0, 4.5D0,
     $                         4.6D0, 4.7D0, 4.8D0, 4.9D0, 5.0D0,
     $                         5.1D0, 5.2D0, 5.3D0, 5.4D0, 5.5D0,
```

```
     $                          5.6D0, 5.7D0, 5.8D0, 5.9D0, 6.0D0,
     $                          6.1D0, 6.2D0, 6.3D0, 6.4D0, 6.5D0,
     $                          6.6D0, 6.7D0, 6.8D0, 6.9D0, 7.0D0,
     $                          7.1D0, 7.2D0, 7.3D0, 7.4D0, 7.5D0,
     $                          7.6D0, 7.7D0, 7.8D0, 7.9D0, 8.0D0,
     $                          8.1D0, 8.2D0, 8.3D0, 8.4D0, 8.5D0,
     $                          8.6D0, 8.7D0, 8.8D0, 8.9D0, 9.0D0,
     $                          9.1D0, 9.2D0, 9.3D0, 9.4D0, 9.5D0,
     $                          9.6D0, 9.7D0, 9.8D0, 9.9D0, 10.0D0/
C     .. Executable Statements ..
C
C     Set the INFO array to tell the code how to solve the problem.
      DO 10 I=1,27
         INFO(I)=0
  10  CONTINUE
C
C     Set TTOL
C      INFO(24)=0
C      DWORK(1)=1D-10
C
C     chattering tolerance
      INFO(25) = 1
      DWORK(2) = 0.01D0
      DWORK(3) = 1.0D-5
C
C     Set the characteristic values
C
      CVAL(1,1) = 0
      CVAL(2,1) = 4
      CVAL(3,1) = 0
      CVAL(4,1) = 0
      CVAL(5,1) = 0
      CVAL(1,2) = 0
      CVAL(2,2) = 4
      CVAL(3,2) = 0
      CVAL(4,2) = 0
      CVAL(5,2) = 0
      CVAL(1,3) = 1
      CVAL(2,3) = 3
      CVAL(3,3) = 2
      CVAL(4,3) = 0
      CVAL(5,3) = 0
C
C     Initial mode
      IPAR(1) = 1
      IPAR(2) = NMODE
c     SLIDING MODE
      IPAR(3) = 3
      IPAR(4) = 1
C
      RPAR(2) = 0.003D0
C
      TRAN(1) = 1
      TRAN(2) = 1
```

```
      TRAN(3) = 2
C
C     SOLVER to be used in each mode
C     SOLVER(I) = 1 -> use DGENDA
      SOLVER(1) = 2
      SOLVER(2) = 2
      SOLVER(3) = 2
C
      METHOD(1)=1
      METHOD(2)=1
      METHOD(3)=1
C
C     use SVD =1
      INFO(18)=1
C
      IERR=0
      IWARN=0
      ISMTH=0
C
C     Set the starting time and the initial values.
      T     =  0.0D0
      DO 20 I=1,NQMAX
         X(I)=0.0D0
         SCALC(I)=0.0D0
         IFIX(I)=0
         XPRIME(I)=0.0D0
 20   CONTINUE
      X(3)=1.0D0
      X(4)=1.0D0
C
C     differential components in mode 1:
      IDIFCO(1)= 1
      IDIFCO(2)= 1
      IDIFCO(3)= 1
      IDIFCO(4)= 1
      IDIFCO(5)= 1
C     differential components in mode 2:
      IDIFCO(6)= 1
      IDIFCO(7)= 1
      IDIFCO(8)= 1
      IDIFCO(9)= 1
      IDIFCO(10)=1
C     differential components in mode 3:
      IDIFCO(11)=1
      IDIFCO(12)=1
      IDIFCO(13)=1
      IDIFCO(14)=1
      IDIFCO(15)=0
C
C     Display results in MATLAB
      CALL WRMLAB(1)
C
C     Set sizes for each mode
      M(1)=M1
```

```
      M(2)=M2
      N(1)=N1
      N(2)=N2
      M(3)=M3
      N(3)=N3
C
C     Now we have to set the tolerances for the solver to indicate how
C     accurate we want the solution to be computed.
C     In this case we use a combined error test with the same absolute
C     and relative tolerance.
C
      RTOL=1.0D-8
      ATOL=1.0D-8
C
      IF (KPRINT .GT. 2) THEN
         WRITE (LUN,115) (X(I),I=1,NQMAX)
      ENDIF
 115  FORMAT(' GIVEN INITIAL VALUES'/(' ',6E12.4/))
C
C        Display initial values in MATLAB
         WRITE(10,307) 1,T
 307     FORMAT( 'T(',I5,')=',F25.10,';')
         WRITE(10,308) 1,X(1),X(2),X(3),X(4),X(5)
 308     FORMAT( 'X(:,',I5,')=[',F25.10,';',F25.10,';',F25.10,';',
     $                           F25.10,';',F25.10,'];')
         WRITE(10,309) 1,XPRIME(1),XPRIME(2)
 309     FORMAT( 'XPRIME(:,',I5,')=[',F25.10,';',F25.10,'];')
C
C     The next step is to solve the problem.
C     We will use DGESDA to compute NOUT intermediate solutions from
C     DTOUT(1) to DTOUT(100) (see above).
C
C     For each intermediate solution some statistics are displayed, where
C     T       is the actual time,
C     X(1)    is the computed first solution component at time T,
C     X(2)    is the computed second solution component at time T,
C     IPAR(1) is the current mode.
C
      IF (KPRINT .GT. 2) THEN
         WRITE (LUN,110)
      ENDIF
C
 110  FORMAT(///
     $     10X,'T',14X,'X(1)',10X,'X(2)',8X,'MODE'/)
      DSEC = DSECND()
      DO 200 IOUT = 1,NOUT
         TOUT = DTOUT(IOUT)
         CALL DGESDA(INFO, MATSUB, FUN, DFUN, USCAL, UINTER,
     $          TRANSF, MDCHNG, GFUN, MXTRAN, TRAN, SOLVER, METHOD,
     $          M, N, NQMAX, T, TOUT, TS, LTS, NSP, X, XPRIME, CVAL,
     $          IPAR, NIPAR,  RPAR, IFIX, IDIFCO,SCALC,SCALR, RTOL, ATOL,
     $          MDHIST, IWORK, LIW, RWORK, LRW, STATS,DWORK, ISMTH,
     $          IWARN, IERR)
         IF (IERR.LT.0) THEN
```

```
            CALL DGSERM (INFO, MATSUB, FUN, DFUN, USCAL,
     $               SOLVER, METHOD, M, N, T, TOUT , X,
     $               XPRIME, CVAL, IPAR, RPAR, IFIX, IDIFCO,SCALC,SCALR,
     $               RTOL, ATOL, IWORK, LIW, RWORK, LRW, ISMTH, IWARN,
     $                IERR)
            STOP
         END IF
         IF (KPRINT .GT. 2) THEN
            WRITE (LUN,130)  T,X(1),X(2),IPAR(1)
         ENDIF
C
C        Display results in MATLAB
 130     FORMAT(1X,E15.5,E16.5,E16.5,I6)
         WRITE(10,407) IPAR(4)+1,T
 407     FORMAT( 'T(',I5,')=',F25.10,';')
         WRITE(10,408) IPAR(4)+1,X(1),X(2),X(3),X(4),X(5)
 408     FORMAT( 'X(:,',I5,')=[',F25.10,';',F25.10,';',F25.10,';',
     $                           F25.10,';',F25.10,'];')
         WRITE(10,409) IPAR(4)+1,XPRIME(1),XPRIME(2)
 409     FORMAT( 'XPRIME(:,',I5,')=[',F25.10,';',F25.10,'];')
         IPAR(4)=IPAR(4)+1
C
 200  CONTINUE
C
C     Finally, we display some final statistics
C
      DSEC = DSECND()-DSEC
      NST = STATS(1)
      NFE = STATS(2)
      NJE = STATS(3)
      NETF= STATS(4)
      NCTF= STATS(5)
      IF (KPRINT .GT. 2) THEN
         WRITE (LUN,210) NST,NFE,NJE,NETF,NCTF, DSEC
      ENDIF
C
 210  FORMAT(//1X,' FINAL STATISTICS FOR THIS RUN..',/
     $     1X,' NUMBER OF STEPS                      =',I5/
     $     1X,' NUMBER OF EVALUATIONS                =',I5/
     $     1X,' NUMBER OF FACTORIZATIONS             =',I5/
     $     1X,' NUMBER OF ERROR TEST FAILURES        =',I5/
     $     1X,' NUMBER OF CONVERGENCE TEST FAILURES =',I5/
     $     1X,' RUNTIME                              =',E10.2)
C
      IF (KPRINT .GT. 2) THEN
         WRITE (LUN,220) NSP
         IF(NSP.GT.0) THEN
            DO 221 I=1,NSP
               WRITE(LUN,230) TS(I)
 221        CONTINUE
         WRITE (LUN,240)
            DO 222 I=1,NSP+1
               WRITE(LUN,250) MDHIST(I)
 222        CONTINUE
```

```
         ENDIF
      ENDIF
C
 220  FORMAT(//1X,'NUMBER OF SWITCH POINTS =',I5)
 230  FORMAT(1X,E15.5)
 240  FORMAT(//1X,'MODE HISTORY:')
 250  FORMAT(1X,I5)
      IF (KPRINT .GT. 2) THEN
         WRITE(LUN,260) ISMTH
      ENDIF
 260  FORMAT(//1X,'SMOOTH TRANSITION POSSIBLE? ISMTH=',I5)
C
C     Display results in MATLAB
      CALL WRMLAB(2)
C
      STOP
C *** Last line  ***
      END


      SUBROUTINE USCAL(M, N, A, LDA, SCALC, SCALR, IERR)
C     .. Scalar Arguments ...
      INTEGER          M, N, LDA, IERR
C     .. Array Arguments ...
      DOUBLE PRECISION A(LDA,*), SCALC(*), SCALR(*)
C     .. Executable Statements ..
      RETURN
C *** Last line of USCAL ***
      END


      SUBROUTINE UINTER(T,X,XPRIME,RPAR,IPAR,ATOL,IRTRN)
C
C     ARGUMENT LIST
C
C       ARGUMENTS IN
C                 T - current mesh point
C            X,XPRIME - current state at time T
C           RPAR,IPAR - user supplied parameter arrays
C
C       ARGUMENTS OUT
C             IRTRN - Serves to interrupt the integration
C                     If IRTRN
C                    * is set  0, the integration is continued
C                    * is set -1, DGENDA returns to the calling program
C     ----------------------------------------------------------------
C     ..Scalar Arguments..
      DOUBLE PRECISION MU
      PARAMETER        (MU=0.4D0)
      DOUBLE PRECISION T
      INTEGER          IRTRN
C     ..Array Arguments..
      INTEGER          IPAR(*)
      DOUBLE PRECISION  X(*), XPRIME(*), RPAR(*), ATOL(*), VC
```

```
C     .. Executable Statements ..
C
      VC= RPAR(2)
      IF(IPAR(1) .EQ. 1) THEN
         IF( (X(1)-X(2).LT.-VC)) THEN
               IRTRN = -1
         ENDIF
      ENDIF
      IF (IPAR(1) .EQ. 2) THEN
         IF( X(1)-X(2).GT.VC ) THEN
            IRTRN=-1
         END IF
      END IF
      IF (IPAR(1) .EQ. 3) THEN
        IF( DSIN(T)-2*MU.GT. 0.0D0  ) THEN
            IRTRN=-1
        END IF
        IF( DSIN(T)+2*MU.LT. 0.0D0  ) THEN
            IRTRN=-1
        END IF
      END IF
      RETURN
      END
C *** Last line of UINTER ***


      SUBROUTINE GFUN(T,IDIF,  X,  XPRIME, GT, NRTFN, IPAR, RPAR, IERR)
C
C     ARGUMENT LIST
C
C       ARGUMENTS IN
C                 T - DOUBLE PRECISION. The current mesh point
C         X, XPRIME - DOUBLE PRECISION arrays of DIMENSION (*).
C                     The current state at time T.
C               IPAR - INTEGER array of DIMENSION (*).
C                      User supplied parameter arrays
C              NRTFN - INTEGER. The number of root functions for
C                      the current mode.
C
C       ARGUMENTS OUT
C                GT - DOUBLE PRECISION array of DIMENSION (NRTFN).
C                     The root functions for the current mode.
C
C        ERROR INDICATOR
C
C               IERR - INTEGER.
C                      Unless the routine detects an error,
C                      IERR contains 0 on exit.
C
C        WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = -1 :  On entry, the number of the current mode stored
C                     in IPAR(1) is greater than the total number
C                     of modes in IPAR(2).
```

```
C     ------------------------------------------------
C     ..Scalar Arguments..
      DOUBLE PRECISION MU
      PARAMETER        (MU=0.4D0)
      DOUBLE PRECISION T
      DOUBLE PRECISION X(*), XPRIME(*), GT(NRTFN), RPAR(*), VC
      INTEGER          IERR, IPAR(*), NRTFN, MODE, NMODE, IDIF
C     .. Executable Statements ..
      IERR =0
      MODE = IPAR(1)
      NMODE = IPAR(2)
      VC = RPAR(2)
C
      IF( MODE .GT. NMODE) THEN
         IERR = -1
         RETURN
C     MODE 1
      ELSE IF( MODE .EQ. 1) THEN
         IF(IDIF.EQ.0)THEN
            GT(1)=X(1)-X(2)+VC
         ELSE
            IERR=-1
         ENDIF
C     MODE 2
      ELSE IF( MODE.EQ.2) THEN
         IF(IDIF.EQ.0) THEN
            GT(1)=X(1)-X(2)-VC
         ELSE
            IERR=-1
         ENDIF
C     SLIDING MODE
      ELSE IF(MODE .EQ. 3) THEN
         IF(IDIF.EQ.0) THEN
            GT(1)=DSIN(T)-2*MU
            GT(2)=DSIN(T)+2*MU
         ELSE
            IERR=-1
         ENDIF
      ENDIF
      RETURN
      END
C *** Last line of GFUN ***



      SUBROUTINE TRANSF(XNEW,XOLD,N,MODNEW,MODOLD,IERR)
C
C     ARGUMENT LIST
C
C       ARGUMENTS IN
C            XOLD - The state of the system in the old mode.
C          MODNEW - the new mode of the system.
C          MODOLD - The old mode of the system.
C
```

```
C       ARGUMENTS OUT
C            XNEW - The state of the system in the new mode.
C
C       ERROR INDICATOR
C
C            IERR - INTEGER.
C                     Unless the routine detects an error,
C                     IERR contains 0 on exit.
C
C        WARNING AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = -1 : Wrong mode change is detected.
C     ------------------------------------------------------------------
C     ..Scalar Arguments..
      INTEGER MODNEW,MODOLD, IERR, N(*)
C     ..Array Arguments..
      DOUBLE PRECISION XNEW(*), XOLD(*)
C     ..Local Scalars..
      IERR=0
      XNEW(1)=XOLD(1)
      XNEW(2)=XOLD(2)
      XNEW(3)=XOLD(3)
      XNEW(4)=XOLD(4)
      XNEW(5)=XOLD(5)
      XNEW(6)=XOLD(6)
      XNEW(7)=XOLD(7)
      XNEW(8)=XOLD(8)
      XNEW(9)=XOLD(9)
      XNEW(10)=XOLD(10)
      RETURN
      END
C *** Last line of TRANSF ***


      SUBROUTINE MDCHNG(MODE, J, NMODE, MXTRAN, NEWMOD, IERR)
C
C     Argument List
C
C       Arguments In
C           MODE - old mode
C          NMODE - number of modes
C         MXTRAN - maximal number of transitions
C              J - transition number
C
C       Arguments Out
C           NEWMOD - new mode
C
C       ERROR INDICATOR
C           IERR - INTEGER
C                   = 0 , ok
C                   = -1 wrong transition number J
C                   = -2 wrong MODE
C     ------------------------------------------------------------------
C     ..Scalar Arguments..
```

```
      INTEGER          MODE, NMODE, MXTRAN, IERR, NEWMOD, J
C     ..Array Arguments..
C     ..Local Scalars..
      INTEGER M(NMODE,MXTRAN)
C     ..Executable Statements..
      IERR=0
      DO 10 I=1,NMODE
         DO 20 K=1,MXTRAN
            M(I,K)=0
 20      CONTINUE
 10   CONTINUE
C
      M(1,1)=2
      M(2,1)=1
      M(1,2)=3
      M(2,2)=3
      M(3,1)=1
      M(3,2)=2
      IF(J .LE. 0 .OR. J .GT. MXTRAN) THEN
         IERR = -1
         RETURN
      ELSE IF (MODE .LE. 0 .OR. MODE .GT. NMODE) THEN
         IERR = -2
         RETURN
      ELSE
         NEWMOD=M(MODE,J)
      END IF
      RETURN
      END
C     *** Last line of MDCHNG ***


      SUBROUTINE FUN(T, IDIF, X, F, IPAR, RPAR, IERR)
C
C     ARGUMENT LIST
C
C        ARGUMENTS IN
C
C             T - DOUBLE PRECISION.
C           IDIF - INTEGER.
C             X - DOUBLE PRECISION array of DIMENSION (*)
C
C        ARGUMENTS OUT
C
C             F - DOUBLE PRECISION array of DIMENSION (*).
C                 The leading N part of this array contains the
C                 IDIF-th derivative of F(t,x(t),dx/dt(t)) at time T.
C
C        ERROR INDICATOR
C
C           IERR - INTEGER.
C                  Unless the routine detects an error (see next section),
C                  IERR contains 0 on exit.
C
```

```
C        WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = -1 : On entry, IDIF is larger than the highest
C                    derivative the subroutine provides
C
C     ------------------------------------------------------------------
C     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IDIF, IERR
C     .. Array Arguments ..
      INTEGER          IPAR(*)
      DOUBLE PRECISION RPAR(*), X(*), F(*)
      RETURN
      END
C     *** Last line of FUN ***


      SUBROUTINE DFUN(T, IDIF, X, JAC, LJAC, IPAR, RPAR, IERR)
C
C     ARGUMENT LIST
C
C        ARGUMENTS IN
C             T - DOUBLE PRECISION.
C          IDIF - INTEGER.
C             X - DOUBLE PRECISION arrai of DIMENSION (*)
C          LJAC - INTEGER.
C                 The leading dimension of array JAC as declared in the
C                 calling program.
C
C        ARGUMENTS OUT
C           JAC - DOUBLE PRECISION array of DIMENSION (LJAC,*).
C                 The leading N by (IDIF+1)*N part of this array
C                 contains the partial derivatives of the IDIF-th
C                 derivative of F(t,x(t),dx/dt(t)) at time T.
C
C        ERROR INDICATOR
C          IERR - INTEGER.
C                 Unless the routine detects an error (see next section),
C                 IERR contains 0 on exit.
C
C        WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C        IERR = -1 : On entry, IDIF is larger than the highest
C                    derivative the subroutine provides
C     ------------------------------------------------------------------
C     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IDIF, IERR, LJAC
C     .. Array Arguments ..
      DOUBLE PRECISION X(*), JAC(LJAC,*), RPAR(*)
      INTEGER          IPAR(*)
      RETURN
      END
C *** Last line of DFUN ***
```

```
      SUBROUTINE WRMLAB(N)
C
C     Writes an m-Files for displaying the results in Matlab
C     (WRiteMatlab)
C     ACHTUNG: Die erzeugte Datei enthaelt ein "clear all".
C
C     INPUT PARAMETERS
C     ----------------
C     N /1 Initialisierung der Datei
C       /2 Plot-anweisungen und Ende der Datei
C
      IMPLICIT NONE
      INTEGER N
      LOGICAL FILEEXIST
C     ------------------------------------------------------------------
      IF (N .EQ. 1) THEN
C
      INQUIRE(FILE='res.m',EXIST=FILEEXIST)
      IF (FILEEXIST) THEN
         OPEN(UNIT=10,FILE='res.m',STATUS='OLD')
         CLOSE(10, STATUS='DELETE')
      END IF
      OPEN(UNIT=10,FILE='res.m',STATUS='NEW')
C
      WRITE(10,99)
  99  FORMAT( 'clear all;close all;')
C
      ELSE IF (N .EQ. 2) THEN
         WRITE(10,110)
 110     FORMAT( 'plot(T,X(1,:),''b'');hold on;')
         WRITE(10,111)
 111     FORMAT( 'plot(T,X(2,:),''r'');')
         WRITE(10,112)
 112     FORMAT( 'plot(T,X(3,:),''g'');')
         WRITE(10,113)
 113     FORMAT( 'plot(T,X(4,:),''m'');')
         WRITE(10,114)
 114     FORMAT( 'plot(T,X(5,:),''y'');')
         WRITE(10,115)
 115     FORMAT('figure(2);plot(T,X(1,:)-X(2,:),''b'');')
         CLOSE(10)
C
      END IF
      RETURN
      END
C     *** Last line of WRMLAB ***


      SUBROUTINE MATSUB(IMAT, M, N, T, IDIF, W, LDW, IPAR,RPAR,IERR)
C
C     Provides the matrices E, A, B, C, D (or the vector F, resp.)
C     and their derivatives up to order IDIF
C     of the given descriptor system (or the DAE)
```

```
C     ------------------------------------------------------------
C     .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER        (ZERO = 0.0D0)
C     .. Scalars ..
      INTEGER          IDIF, IERR, IMAT, LDW, I, J, M, N
      DOUBLE PRECISION T
C     .. Arrays ..
      INTEGER          IPAR(*)
      DOUBLE PRECISION RPAR(*), W(LDW,*)
C     .. External Subroutines ..
      EXTERNAL         EDIF, ADIF, FDIF
C     .. Executable statements ..
C     Matrix to be provided
      GOTO (100, 200, 300), IMAT
C
C     Provide the derivative coefficient E
  100 CONTINUE
      DO 112 I=1,M
         DO 111 J=1,N
            W(I,J) = ZERO
  111    CONTINUE
  112 CONTINUE
C
      CALL EDIF(M, N, T, IDIF, W(1,1), LDW, IPAR, RPAR, IERR)
      RETURN
C
C     Provide the solution coefficient A
  200 CONTINUE
      DO 212 I=1,M
         DO 211 J=1,N
            W(I,J) = ZERO
  211    CONTINUE
  212 CONTINUE
      CALL ADIF(M, N, T, IDIF, W(1,1), LDW, IPAR, RPAR, IERR)
      RETURN
C
C     Provide the inhomogeneity F
  300 CONTINUE
      DO 310 I=1,M
         W(I,1) = ZERO
  310 CONTINUE
      CALL FDIF(M, T, IDIF, W(1,1), IPAR, RPAR, IERR)
      RETURN
      END
C *** Last line of MATSUB ***



      SUBROUTINE EDIF(M,N,T,IDIF,E,LDE,IPAR,RPAR,IERR)
C
C     .. Scalar Arguments ..
      DOUBLE PRECISION T, RPAR(*)
      INTEGER          IDIF, IERR, IPAR(*), LDE, N, M
```

```
C     .. Array Arguments ..
      DOUBLE PRECISION E(LDE,*)
      INTEGER          I,J, MODE, NMODE
C     .. Executable Statements ..
      IERR = 0
      MODE=IPAR(1)
      NMODE=IPAR(2)
      IF(MODE.GT.NMODE)THEN
         IERR=-2
         RETURN
      ENDIF
      DO 10 I=1,M
         DO 20 J=1,N
            E(I,J)=0.0D0
 20      CONTINUE
 10   CONTINUE
      IF(MODE.EQ.1 .OR. MODE.EQ.2) THEN
         IF(IDIF.GT.0)THEN
            IERR=-2
            RETURN
         ENDIF
         DO 30 J=1,N
            E(J,J)=1.0D0
 30      CONTINUE
      ENDIF
      IF(MODE.EQ.3)THEN
         IF(IDIF.GT.2)THEN
            IERR=-2
            RETURN
         ENDIF
         IF(IDIF.EQ.0)THEN
            DO 40 J=1,N-1
               E(J,J)=1.0D0
 40         CONTINUE
         ENDIF
      ENDIF
      RETURN
C     *** Last line of EDIF ***
      END


      SUBROUTINE ADIF(M,N,T,IDIF,A,LDA,IPAR,RPAR,IERR)
C
C     .. Scalar Arguments ..
      DOUBLE PRECISION RPAR(*), T
      INTEGER          IDIF, IERR, IPAR(*), LDA, N, M
C     .. Array Arguments ..
      DOUBLE PRECISION A(LDA,*)
      INTEGER          I,J
C     .. Executable Statements ..
      IERR = 0
      MODE=IPAR(1)
      NMODE=IPAR(2)
      IF(MODE.GT.NMODE) THEN
```

```
         IERR=-1
         RETURN
      ENDIF
      DO 10 I=1,M
         DO 20 J=1,N
            A(I,J)=0.0D0
 20      CONTINUE
 10   CONTINUE
C
      IF(MODE.EQ.1 .OR. MODE .EQ.2) THEN
         IF(IDIF.GT.0) THEN
            IERR=-2
            RETURN
         ENDIF
         A(3,1)=1.0D0
         A(4,2)=1.0D0
      ENDIF
      IF(MODE.EQ.3) THEN
         IF(IDIF.GT.2) THEN
            IERR=-2
            RETURN
         ENDIF
         IF(IDIF.EQ.0) THEN
            A(1,5)= 1.0D0
            A(2,5)=-1.0D0
            A(3,1)= 1.0D0
            A(4,2)= 1.0D0
            A(5,1)= 1.0D0
            A(5,2)=-1.0D0
         ENDIF
      ENDIF
      RETURN
C *** Last line of ADIF ***
      END


      SUBROUTINE FDIF(M,T,IDIF,F,IPAR,RPAR,IERR)
C
C     .. Constants
      DOUBLE PRECISION MU
      PARAMETER        (MU=0.4D0)
C     .. Scalar Arguments ..
      DOUBLE PRECISION T, RPAR(*)
      INTEGER          IDIF, IERR, IPAR(*), M
C     .. Local Arrays ..
      DOUBLE PRECISION F(*)
      INTEGER          MODE, NMODE
C     .. Executable Statements ..
      IERR =0
      MODE = IPAR(1)
      NMODE = IPAR(2)
C
      IF( MODE .GT. NMODE) THEN
         IERR = -2
```

```
         RETURN
C     MODE 1
      ELSEIF( MODE .EQ. 1) THEN
         IF(IDIF.EQ.0) THEN
            F(1)=DSIN(T)-MU
            F(2)=MU
            F(3)=0.0D0
            F(4)=0.0D0
         ELSE
             IERR = -2
             RETURN
          ENDIF
      ELSEIF( MODE .EQ. 2) THEN
         IF(IDIF.EQ.0) THEN
            F(1)=DSIN(T)+MU
            F(2)=-MU
            F(3)=0.0D0
            F(4)=0.0D0
         ELSE
             IERR = -2
             RETURN
          ENDIF
      ELSEIF( MODE .EQ. 3) THEN
         IF(IDIF.EQ.0) THEN
            F(1)=DSIN(T)-MU
            F(2)=MU
            F(3)=0.0D0
            F(4)=0.0D0
            F(5)=0.0D0
         ELSEIF(IDIF.EQ.1)THEN
            F(1)=DCOS(T)
            F(2)=0.0D0
            F(3)=0.0D0
            F(4)=0.0D0
            F(5)=0.0D0
         ELSEIF(IDIF.EQ.2)THEN
            F(1)=-DSIN(T)
            F(2)=0.0D0
            F(3)=0.0D0
            F(4)=0.0D0
            F(5)=0.0D0
         ELSE
             IERR = -2
             RETURN
          ENDIF
      ENDIF
      RETURN
C *** Last line of FDIF ***
      END
```