Technische Universität Berlin
Institut für Mathematik

# GENDA: A software package for the numerical solution of General Nonlinear Differential-Algebraic equations

P. Kunkel, V. Mehrmann and I.Seufer

Technical Report 730-02

Preprint-Reihe des Instituts für Mathematik
Technische Universität Berlin

Report 730-02 February 2002

# GENDA: A software package for the numerical solution of **GE**neral **N**onlinear **D**ifferential-**A**lgebraic equations *

Peter Kunkel † Volker Mehrmann ‡ Ingo Seufer §

February 13, 2002

## 1 Purpose

DGENDA solves general differentiell algebraic equations (DAEs) of the form

$$\begin{array}{rcl} F(t, x(t), \dot{x}(t)) & = & 0, \\ x(t_0) & = & x_0, \end{array} \tag{1}$$

for $x$ in a specified range $\mathbb{I}$ of the independent variable $t$. No restrictions on the (differentiation) index are needed.

## 2 The theory

We need information about several derivatives of the given DAE for the solution of general DAEs. For this we denote by

$$F_l(t, x, \dot{x}, ..., x^{(l+1)}) = \begin{bmatrix} F(t, x, \dot{x}) \\ \frac{dF}{dt}(t, x, \dot{x}, \ddot{x}) \\ \vdots \\ \frac{d^lF}{dt^l}(t, x, \dot{x}, ..., x^{(l+1)}) \end{bmatrix} = 0 \tag{2}$$

the *inflated* nonlinear DAE obtained by successive differentiation and we denote by

$$\begin{array}{rcl} M_l(t, x, \dot{x}, ..., x^{(l+1)}) & = & F_{l;\dot{x},...,x^{(l+1)}}(t, x, \dot{x}, ..., x^{(l+1)}), \\ N_l(t, x, \dot{x}, ..., x^{(l+1)}) & = & -(F_{l;x}(t, x, \dot{x}, ..., x^{(l+1)}), 0, ..., 0) \end{array}$$

its Jacobians. The DAE has to satisfy the following Hypothesis.

*The work has been supported by DFG research grant no. Ku964/4 and no. Me790/11.

†Mathematisches Institut, Universität Leipzig, Augustusplatz 10–11, D-04109 Leipzig, Fed. Rep. Germany.

‡Institut für Mathematik, MA 4-5, Technische Universität Berlin, Straße des 17. Juni 136, D-10623 Berlin, Fed. Rep. Germany.

§Institut für Mathematik, MA 4-5, Technische Universität Berlin, Straße des 17. Juni 136, D-10623 Berlin, Fed. Rep. Germany.

**Hypothesis 1** *There exist integers $\hat{\mu}$, $\hat{d}$ and $\hat{a}$, such that for all values $(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) \in \mathbb{L}$ with*

$$\mathbb{L} = \{(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) \in \mathbb{I} \times \mathbb{R}^n \times \mathbb{R}^n \times ... \times \mathbb{R}^n | F_{\hat{\mu}}(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) = 0\}$$

*associated with $F$ the following properties hold*

1. *We have* rank $M_{\hat{\mu}}(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) = (\hat{\mu}+1)n - \hat{a}$, *and there exists a matrix function $\hat{Z}_2$ being smooth on $\mathbb{L}$ with orthonormal columns and size $((\hat{\mu}+1)n, \hat{a})$ satisfying $\hat{Z}_2^T M_{\hat{\mu}} = 0$.*

2. *We have* rank $\hat{A}_2(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) = \hat{a}$, *where $\hat{A}_2 = \hat{Z}_2^T N_{\hat{\mu}}[I_n 0 \cdots 0]^T$, and there exists a matrix function $\hat{T}_2$ being smooth on $\mathbb{L}$ with orthonormal columns and size $(n, \hat{d})$, $\hat{d} = n - \hat{a}$, satisfying $\hat{A}_2 \hat{T}_2 = 0$.*

3. *We have* rank $F_{\dot{x}}(t, x, \dot{x}) \hat{T}_2(t, x, \dot{x}, ..., x^{(\hat{\mu}+1)}) = \hat{d}$, *and there exists a matrix function $\hat{Z}_1$ being smooth on $\mathbb{L}$ with orthonormal columns and size $(n, \hat{d})$ yielding that $\hat{E}_1 = \hat{Z}_1^T F_{\dot{x}}(t, x, \dot{x})$ has constant rank $\hat{d}$.*

When DAE (1) satisfies Hypothesis 1, the *(global) strangeness index* $\mu$ of (1) is defined as the least possible $\hat{\mu}$ for which the above properties hold. The corresponding numbers $d$ and $a$ are the numbers of differential and algebraic equations of the DAE.

If the *differentiation index* (see, e.g., [2]) is well-defined, Hypothesis 1 is always satisfied, and the strangeness index is zero if the differentiation index is zero and equal to the differentiation index lowered by 1 otherwise.

In [4] it has been shown that every sufficiently smooth solution $x^*$ of (1), where $F$ satisfies Hypothesis 1, is a locally unique solution of a problem of the form

$$\begin{aligned} \hat{Z}_1^T F(t, x_1, x_2, \dot{x}_1, \dot{x}_2) &= 0, & (3) \\ x_2 &= \hat{G}_2(t, x_1), & (4) \end{aligned}$$

which has a vanishing strangeness index, i.e. differentiation index at most one.

## 2.1 Initial values

Consistency of an initial value means that (4) is satisfied while arbitrary initial values can be chosen for the differential variables $x_1$. In [4] it has been shown that every $y_0 \in \mathbb{L}$ can be locally extended to a solution of (1). Thus every part $(t_0, x_0)$ of $y_0 \in \mathbb{L}$ is a consistent initial value. Therefore to determine a consistent initial value we must solve

$$F_\mu(t_0, x, \dot{x}, ..., x^{(\mu+1)}) = 0 \tag{5}$$

for $(x, \dot{x}, ..., x^{(\mu+1)})$. The solution of this underdetermined system of nonlinear equations is computed in a least squares sense with the subroutine NLSCON (see [6]) which is an implementation of the Gauss-Newton method [7]. The user must supply a guess of the initial values as a starting value for the Gauss-Newton iteration.

The initial values for the $d$ differential variables can be set to any value, similar to the case of an ODE. The user can choose these variables to be kept fixed during the Gauss-Newton iterations by setting up the IFIX-array (see (4.2)). In this case the corresponding columns of the Jacobian are set to zero. Note that this will lead to a rank drop of the Jacobian if any of the algebraic variables are fixed. In this case the code will return an error message.

## 2.2 Integration

DGENDA uses a BDF method with order and stepsize control. The BDF solver used here is an adaption of the code implemented in DASSL (see [8]) for solving DAEs of index at most one. Before discretizing the DAE at a timestep proceeding from $t_0$ to $t_0 + h$ we have to compute a locally equivalent system similar to (3),(4). In every step we then solve a system of nonlinear equations of the form

$$
\begin{aligned}
F_\mu(t_0 + h, x, \dot{x}, ..., x^{(\mu+1)}) &= 0, &\quad (6)\\
\tilde{Z}_1^T F(t_0 + h, x, \alpha x + \beta) &= 0, &\quad (7)
\end{aligned}
$$

where $\tilde{Z}_1$ denotes some approximation to $\hat{Z}_1$ at the desired solution. Equation (6) yields a solution for which (4) holds so that the algebraic constraints are satisfied and (7) is a discretization of (3).

The solution is computed with a Gauss-Newton method, see e.g. [7]. An initial guess $\bar{x}_0 = (x_0, \dot{x}_0, ..., x_0^{(\mu+1)})$ is iteratively improved by a correction $\Delta\bar{x}_i$, i.e.

$$\bar{x}_{i+1} = \bar{x}_i + \Delta\bar{x}_i, \quad i = 0, 1, ... \quad , \tag{8}$$

where

$$\Delta\bar{x}_i = -\bar{J}_i^+ \bar{F}(t_0 + h, \bar{x}_i). \tag{9}$$

Here, $\bar{F}$ denotes the system given by (6), (7),

$$\bar{J}_i = \begin{bmatrix} -N_\mu[I_n 0 \cdots 0] & M_\mu \\ \tilde{Z}_1^T(F_x + \alpha F_{\dot{x}}) & 0 \end{bmatrix} \tag{10}$$

is its Jacobian at state $\bar{x}_i$ and $\bar{J}_i^+$ the Moore-Penrose pseudoinverse of $\bar{J}_i$ (see e.g. [1]). Note that the Jacobian of (6), (7) is generally nonsquare but has full row rank at every solution $(x, \dot{x}, ..., x^{(\mu+1)})$ of (1) if $\tilde{Z}_1$ is a sufficiently good approximation to $\hat{Z}_1$ 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 we can apply a simplified Gauss-Newton method [7] by fixing the Jacobian at any timestep.

By default, DGENDA uses the simplified Gauss-Newton method [7] with an initial $\bar{x}_0$ where $x_0$ and $\dot{x}_0$ are obtained by evaluating a predictor polynomial at time $t_0 + h$ and $(\ddot{x}_0, ..., x_0^{(\mu+1)})$ is obtained by classical homotopy [7], i.e. the results at the previous timestep $t_0$ are chosen as initial values at $t_0 + h$. Optionally, the user can force the code to use the classical Gauss-Newton method where the Jacobian is reevaluated at every iteration or to use the approach implemented in DASSL, where the Jacobian is reevaluated after several timesteps when a certain criterion is fulfilled. Furthermore, the user can tell the code to compute some of the components of the initial solution vector $\bar{x}_0$ by linear extrapolation.

The corrector iterations (8) are terminated if an estimate for the local relative and absolute error is small enough. This error test requires roughly that

$$|\text{LOCAL ERROR}| \le \text{RTOL} * |\text{X}| + \text{ATOL} \tag{11}$$

for each component of the solution vector $(x, \dot{x}, ..., x^{(\mu+1)})$ at every timestep. RTOL and ATOL can be scalars, such that (11) must hold for every component, or they can be vectors of size $(\mu + 2)n$, such that the user can set different tolerances for every component of the solution.

If it is known that $\hat{Z}_2$ of Hypothesis 1 only depends on $t$, then one can choose $\tilde{Z}_2 = \hat{Z}_2$ and equation (6) can be reduced to

$$\tilde{Z}_2^T F_\mu(t_0 + h, x, \dot{x}, ..., x^{(\mu+1)}) \quad = \quad 0. \tag{12}$$

Due to Hypothesis 1 the system (12), (7) does only depend on $x$ and is uniquely solvable. This approach is also implemented in the code but no check will be made if $\hat{Z}_2$ is independent of $(x, \dot{x}, ..., x^{(\mu+1)})$.

## 2.3 Computing the characteristic values

By default the user has to set the characteristic values $\mu$, $d$, $a$ and $u$ to their correct values ($u = m - a - d$ must be zero in this version of DGENDA). Note that the rank assumptions in Hypothesis 1 only hold for $y_0 \in \mathbb{L}$, such that they can be violated at $(t_0 + h, \bar{x}_0)$, where $\bar{x}_0$ is the initial solution vector for the Gauss-Newton iterations. Thus in general it is not possible to detect these values when computing the projectors $\tilde{Z}_1$, $\tilde{T}_2$ and $\tilde{Z}_2$. It can also be difficult to apply the approach used for the linear solver DGELDA [5] to a linearization of (1) after a successfull iteration of the BDF-solver because the computed approximation to the solution generally lies in a neighborhood of $\mathbb{L}$ prescribed by the tolerances, but not in $\mathbb{L}$ itself. Furthermore the rank decisions may suffer of badly conditioned problems, e.g. if the time scale is extremely small.

Nevertheless 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 characterisic values are detected. The code also returns a suggestion for the correct characteristic values.

This feature may be used to compute these values by setting the parameter MXINDX (see 6.1) sufficiently high (it must be at least equal to $\mu$) and by supplying sufficient derivatives of the DAE.

## 2.4 Scaling

Before any rank decisions are made during the computation, the matrix $[-N_\mu \;\; M_\mu]$ is equilibrated to lower its condition number. The code computes row and column scaling vectors $s_r$ and $s_c$ (stored in the arrays SCALC and SCALR), such that

$$[-N_\mu \;\; M_\mu] = \mathrm{diag}(s_r)^{-1}[-\bar{N}_\mu \;\; \bar{M}_\mu] \; \mathrm{diag}(s_c)^{-1} \;, \tag{13}$$

where the $i$-th element of $s_c$ is an estimate for the highest absolute value of the $i$-th component of the solution approximation computed so far, and $s_r$ is chosen such that the highest absolute value of every row of $[-\bar{N}_\mu \;\; \bar{M}_\mu]$ is equal to one. After computing the projectors $\tilde{Z}_1$, $\tilde{T}_2$ and $\tilde{Z}_2$ a backtransformation is made. Note that these transformed projectors are no longer orthogonal but still project onto the appropriate subspaces.

The same scaling method is applied to the Jacobian (10) before the correction $\Delta\bar{x}_i$ is computed to minimize the numerical errors caused by a badly conditioned problem.

Optionally the user can supply his own scaling subroutine USCAL if an appropriate scaling method adapted to a certain problem is known (see (4.2) for details). On the other hand it is also possible to deactivate any scaling.

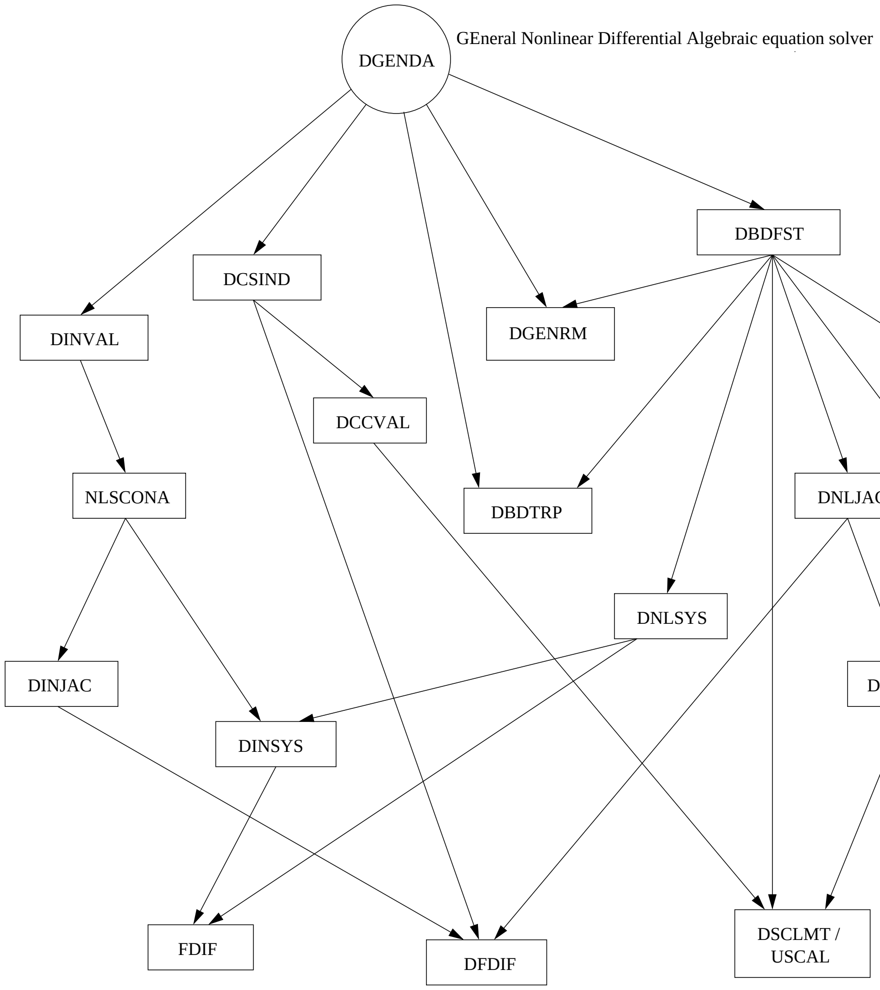


Fig.1: DGENDA and its subroutines

| | |
|---|---|
| **DINVAL** | computes consistent **IN**itial **VAL**ues |
| **NLSCONA** | **N**onlinear **L**east **S**quares solver for systems with **CON**straints [6], **A**dapted for DGENDA |
| **DINJAC/DINSYS** | sets up the **IN**flated **SYS**tem and its **JAC**obian |
| **DCSIND** | **C**omputes the **S**trangeness **IND**ex of the problem |
| **DCCVAL** | **C**omputes the **C**haracteristic **VAL**ues of the inflated system |
| **DBDFST** | performs a **BDF ST**ep |
| **DGENRM** | computes the **GE**neral **N**o**RM** of a solution |
| **DBDTRP** | computes a solution at a fixed time by **B**ackward **D**ifferentiation in**T**e**RP**olation |
| **DCMPRJ** | **C**o**M**putes the **PR**o**J**ectors $Z_1$ and $Z_2$ |
| **DNLJAC/DNLSYS** | compute the strangenessfree **N**on**L**inear **SYS**tem and its **JAC**obian |
| **DUPDXW** | **UPD**ates the weight vector XW |
| **DECCON/SOLCON** | perform a QR **DEC**omposition of the Jacobian of a nonlinear system with **CON**straints and compute least squares **SOL**utions [3] |
| **DSCLMT** | computes a **SC**a**L**ing of the **MAT**rices |
| **FDIF/DFDIF/USCAL** | user supplied subroutines, see (4.2) |

Tab.1: The subroutines and their purposes

## 3 Specification

```
 SUBROUTINE DGENDA (INFO, FDIF, DFDIF, USCAL, M, N, T, TOUT, X,
$                  XPRIME, CVAL, IPAR, RPAR, IFIX, SCALC, SCALR,
$                  RTOL, ATOL, IWORK, LIW, RWORK, LRW, IWARN,
$                  IERR)
 INTEGER           INFO(22), M, N, CVAL(4), IPAR(*), IFIX(*),
$                  IWORK(*), LIW, LRW, IWARN, IERR
 DOUBLE PRECISION  T, TOUT, X(*), XPRIME(*), RPAR(*), SCALC(*),
$                  SCALR(*), RTOL(*), ATOL(*), RWORK(*)
 EXTERNAL          FDIF, DFDIF, USCAL
```

## 4 Argument List

### 4.1 Mode Parameters

**INFO – INTEGER array of DIMENSION (22).**

The basic task of the code is to solve the 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 (See 6.1 for details).

### 4.2 User–supplied Subroutines

**FDIF – User–supplied function.**

This is a subroutine which the user provides to define the left hand side of the inflated DAE $F_\mu(x, \dot{x}, ..., x^{(\mu+1)})$ as described in (2). It has the form

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

The subroutine takes as input the time T, the vector X containing an approximation to the solution $\bar{x} = (x, \dot{x}, ..., x^{(\mathrm{MXINDX}+1)})$, where MXINDX$\geq \mu$ (see (4.3) for further explanations), and the integer parameter IDIF.

As output, the subroutine produces the IDIF-th derivative of the DAE at time T and state X in the first $m$ elements of the 1-dimensional array F.

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

**DFDIF – User–supplied function.**

This is a subroutine which the user provides to define the Jacobian the inflated DAE $F_\mu(t, x, \dot{x}, ..., x^{(\mu+1)})$. It is of the form

SUBROUTINE DFDIF (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 $\bar{x} = (x, \dot{x}, ..., x^{(\mathrm{MXINDX}+1)})$ where MXINDX$\geq \mu$ (see (4.3) for further explanations), and the integer parameter IDIF. As output, the subroutine produces all partial derivatives of the IDIF-th derivative of $F(t, x, \dot{x})$ with respect to all entrys of X in the first M rows and (IDIF+2)*N columns of the 2-dimensional array JAC. **Note:** In the calling program, DFDIF 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. 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 $[-N_\mu \;\; M_\mu]$ or the Jacobian (10), 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 $B = SCALR * A * SCALC$ similar to (13). 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)$\leq$1, USCAL must be a dummy-subroutine.

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

## 4.3 Arguments In

**M – INTEGER.**

The number of equations.
M $\geq$ 1.

**N – INTEGER.**

The number of components of $x$.
One has to set N = M in this version of DGENDA.

**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 (INFO(1)=0, see below) 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 ((MXINDX+2)N).**

If INFO(11)=0, this array may contain a guess for the initial value $(x, \dot{x}, ..., x^{(\mu+1)})$ 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 INFO(11)=1, this array must contain consistent initial values of the N solution components and the first (MXINDX+1) derivatives at the initial time T.
**Note:** This array is overwritten.

**CVAL – INTEGER array of DIMENSION (4).**

Contains the characteristic values of the DAE:

**CVAL(1)** contains the strangeness index $\mu$.

**CVAL(2)** contains the number $d_\mu$ of differential components.

**CVAL(3)** contains the number $a_\mu$ of algebraic components.

**CVAL(4)** contains the number $u_\mu$ of undetermined components.

Note that in this version $u_\mu$ has to be 0.

**IPAR – INTEGER array of DIMENSION (*).**

This integer array can be used for communication between the calling program and the FDIF and DFDIF subroutines.

**RPAR – DOUBLE PRECISION array of DIMENSION (*).**

This real array can be used for communication between the calling program and the FDIF and DFDIF subroutines.

IPAR and RPAR are not altered by DGENDA or its subprograms. If these arrays are used, they must be dimensioned in the calling program and in FDIF and DFDIF as arrays of appropriate length. Otherwise, ignore these arrays by treating them as dummy arrays of length one.

**IFIX – INTEGER array of DIMENSION ((MXINDX+2)N).**

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

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

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)N).**

User row scaling of the Jacobian. Only applicable as input if INFO(13)=1 (see also 6.1) and the user supplied scaling subroutine USCAL takes SCALR as an input argument.

### 4.4 Arguments Out

**T – DOUBLE PRECISION.**

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

**X – DOUBLE PRECISION array of DIMENSION ((MXINDX+2)N).**

Contains the computed approximation of the solution $x$ at T in its first N elements. The further elements contain approximations to the first (MXINDX+1) derivatives of x at the last timestep of the BDF-solver.

**XPRIME – DOUBLE PRECISION array of DIMENSION (N).**

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

**CVAL – INTEGER array of DIMENSION (4).**

Contains the characteristic values of the DAE as described in 4.3.

If INFO(16)=1 (see 6.1) 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.

**RWORK, IWORK – See 4.6.**

These real and integer arrays provide the workspace (see below). 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.

**IWORK(12)** contains the number of calls to EDIF, ADIF and FDIF so far.

**IWORK(13)** contains the number of factorizations of the Jacobian so far.

**IWORK(14)** contains the total number of error test failures so far.

**IWORK(15)** contains the total number of convergence test failures so far.

**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.

### 4.5 Tolerances

**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.
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).

The tolerances are used by the code in a local error test at each step which requires roughly that

$$|\text{LOCAL ERROR}| \leq \text{RTOL} * |\text{X}| + \text{ATOL}$$

for each vector component.

The true (global) error is the difference between the true solution of the initial value problem and the computed approximation. Practically all present day codes, including this one, control the local error at each step and do not even attempt to control the global error directly.

Usually, but not always, the true accuracy of the computed X is comparable to the error tolerances. This code will usually, but not always, deliver a more accurate solution if the user reduces the tolerances and integrate again. By comparing two such solutions the user can get a fairly reliable idea of the true error in the solution at the bigger tolerances.

Setting ATOL=0 results in a pure relative error test on that component. Setting RTOL=0 results in a pure absolute error test on that component. A mixed test with non-zero RTOL and ATOL corresponds roughly to a relative error test when the solution component is much bigger than ATOL and to an absolute error test when the solution component is smaller than the threshold ATOL.

The code will not attempt to compute a solution at an accuracy unreasonable for the machine being used. It will advise the user if he asks for too much accuracy and inform the user as to the maximum accuracy it believes possible.

## 4.6 Workspace

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

The length of IWORK.
LIW $\geq 127 + (\text{MXINDX} + 2)\text{N}$.

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

The length of RWORK.
LRW $\geq 3NQ$.

## 4.7 Warning Indicator

**IWARN – INTEGER.**

Is always zero in this version of DGENDA.

### 4.8 Error Indicator

**IERR – INTEGER.**

Unless the code detects an error (see next section), IERR contains a positive value on exit.

IERR = 1: 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.

## 5 Warnings and Errors detected by the Routine

*** Task interrupted ***

IERR = −1: A large amount of work has been expended (More then NMAX steps).

IERR = −2: The error tolerances are too stringent.

IERR = −3: The local error test cannot be satisfied because the user specified a zero component in ATOL and the corresponding computed solution component is zero. Thus, a pure relative error test is impossible for this component.

IERR = −6: DGENDA had repeated error test failures on the last attempted step.

IERR = −7: DGENDA had repeated convergence test failures on the last attempted step.

Note that in the case of repeated convergence test failures it may help to change the computation of the starting values for the Gauss-Newton iterations, see 6.1, INFO(15).

IERR = −8: The Jacobian is singular.

IERR = −9: DGENDA had repeated convergence test failures on the last attempted step and there were several error test failures.

IERR = −20: DGENDA failed to determine characteristic values because IERR in DFDIF had a negative value.

IERR = −21: DGENDA failed to continue because IERR in FDIF or DFDIF had a negative value.

IERR = $-22$: DGENDA failed to determine the characteristical values.

IERR = $-23$: DGENDA failed to compute an equivalent strangeness index 0 system.

IERR = $-24$: DGENDA failed to compute an initial X. An error was signalled by the LAPACK subroutine NLSCON.

IERR = $-25$: DGENDA failed to compute an initial X because IERR in FDIF or DFDIF had a negative value.

IERR = $-26$: DGENDA is unable to continue due to a change in the characteristic values.

IERR = $-27$: DGENDA failed to compute characteristic values. The strangeness index may be greater than MXINDX.

*** Task terminated ***

IERR = $-101$: Some element of the INFO vector is not zero or one.

IERR = $-102$: $N \leq 0$ or $M \neq N$.

IERR = $-103$: MAXORD not in range.

IERR = $-104$: MXINDX $< 0$.

IERR = $-105$: LRW is less than the required length for RWORK. See IWORK(18) for the required length.

IERR = $-106$: LIW is less than the required length for IWORK. See IWORK(18) for the required length.

IERR = $-107$: Some element of RTOL is $< 0$.

IERR = $-108$: Some element of ATOL is $< 0$.

IERR = $-109$: All elements of RTOL and ATOL are zero.

IERR = $-110$: INFO(4)=1 and TOUT is behind TSTOP.

IERR = $-111$: HMAX $\leq 0.0$

IERR = $-112$: TOUT is behind T.

IERR = $-113$: INFO(8)=1 and H0=0.0.

IERR = $-114$: Some element of $\mathrm{RTOL} * |\mathrm{X}| + \mathrm{ATOL}$ is $\leq 0.0$.

IERR = $-115$: TOUT is too close to T to start integration.

IERR = $-116$: INFO(4)=1 and T is behind TSTOP.

IERR = −118: INFO(13)=1 and IWORK(16) is out of range.

IERR = −119: INFO(1) = 1 and TOUT = T.

IERR = −120: NMAX $\leq 0$.

IERR = −121: Output unit not in range.

IERR = −122: INFO(20)=1 and IWORK(23)=NONLIN is out of range.

IERR = −123: INFO(18)=1 and RWORK(10) is too small.

IERR = −124: INFO(22)=1 and RWORK(11) is too small.

IERR = −125: INFO(21)=1 and IWORK(24) is out of range.

IERR = −126: INFO(16)=1 and IWORK(2)=0 is not allowed.

IERR = −127: Scaling is allowed (INFO(13)=1 and IWORK(16)$\geq$ 1) but SCALC has a negative element.

IERR = −128: INFO(19)=1 and IWORK(22)=KPRINT is out of range.

IERR = −129: One of the characteristic values in CVAL or the MXINDX has an illegal value.

IERR = −130: INFO(15)=1 and IWORK(17) is out of range.

IERR = −131: INFO(14)=1 and IWORK(2) is out of range.

IERR = −998: The last step was terminated with a negative value of IERR larger than −100, and no appropriate action was taken.

IERR = −999: The previous call was terminated because of illegal input (IERR $< -100$) and there is illegal input in the present call as well. (Suspect infinite loop.)

## References

[1] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications.* John Wiley, New York, 1973.

[2] 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.

[3] P. Deuflhard and V. Apostolescu. An underrelaxed Gauss-Newton method for equality constrained nonlinear least squares problems. *Lect. Notes Control Inf. Sci.*, 7:22–32, 1978.

[4] P. Kunkel and V. Mehrmann. Regular solutions of nonlinear differential-algebraic equations and their numerical determination. *Numer. Math.*, 79:581–600, 1998.

[5] 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.

[6] U. Nowak and L. Weimann. A family of newton codes for systems of highly nonlinear equations - algorithm, implementation, application. Report, ZIB, 1990.

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

[8] L. R. Petzold. A description of DASSL: A differential/algebraic system solver. In R. S. Stepleman et al., editors, *IMACS Trans. Scientific Computing Vol. 1*, pages 65–68. North-Holland, Amsterdam, 1983.

# 6 Further Comments

## 6.1 Setting up the INFO array before the first call

This array is used to give the code more details about how the user wants the problem to be solved. The user must respond to all of the following 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)** 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.
See below for continuation calls.

**INFO(2)** The error tolerances RTOL and ATOL are used to specify how much accuracy the user wants. The simplest use is to take them both to be scalars. To obtain more flexibility, they can both be vectors, see 2.2. Are both error tolerances RTOL, ATOL scalars ?

Yes – Set INFO(2) = 0
and input scalars for both RTOL and ATOL.

No – Set INFO(2) = 1
and input arrays for both RTOL and ATOL.

**INFO(3)** The code integrates from T in the direction of TOUT by steps. If the user wishes, it will return the computed solution at the next intermediate step (the intermediate-output mode) or TOUT, whichever comes first. If the user must have solutions at many specific TOUT points, this code will compute them efficiently. Do you want the solution only at TOUT (and not at the next intermediate step) ?

Yes – Set INFO(3) = 0

No – Set INFO(3) = 1

**INFO(4)** The code may integrate past TOUT and interpolate to obtain the result at TOUT, to handle solutions at many specific values TOUT efficiently. Sometimes it is not possible to integrate beyond some point TSTOP because the equation changes there or it is not defined past TSTOP. Then the user must tell the code not to go past. Can the integration be carried out without any restrictions on the independent variable T ?

Yes – Set INFO(4)=0

No – Set INFO(4)=1
and define the stopping point TSTOP by setting RWORK(1)=TSTOP.

**INFO(6)** A maximum number of steps NMAX must be specified in order to prevent the code from computing infinitely further in the case of repeated step rejection. The default value for NMAX is 10 000. Do you want the maximum number of steps to default to 10 000 ?

Yes – Set INFO(6)=0

No – Set INFO(6)=1
and define NMAX by setting IWORK(20)=NMAX.

**INFO(7)** The user can specify a maximum (absolute value of) stepsize, so that the code will avoid passing over very large regions. 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(8)** Differential/algebraic problems may occasionally suffer from severe scaling difficulties on the first step. If the user knows a great deal about the scaling of his problem, he can help to alleviate this problem by specifying an initial stepsize H0. Do you want the code to define its own initial stepsize ?

Yes – Set INFO(8)=0

No – Set INFO(8)=1
and define H0 by setting RWORK(3)=H0.

**INFO(9)** If storage is a severe problem you can save some memory by restricting the maximum order MAXORD of the BDF-method. The default value is 5. For each order decrease below 5, the code requires fewer locations, however it is likely to be slower. In any case, you must have $1 \leq$ MAXORD $\leq 5$. Do you want the maximum order to default to 5 ?

Yes – Set INFO(9)=0

No – Set INFO(9)=1
and define MAXORD by setting IWORK(3)=MAXORD

**INFO(10)** If the user wishes, the code tries to check the strangeness index $\mu$ and the other characteristic values of the problem (see also INFO(16) and INFO(17)). The default value for the maximum index MXINDX is $\mu$=CVAL(1). The user can increase it if the index of the problem may be higher. Note, that FDIF and DFDIF must provide $F(t, x(t), \dot{x}(t))$, the first MXINDX derivatives of $F$ and the partial derivatives with respect to $\dot{x}, ..., x^{(\mathrm{MXINDX}+1)}$. In any case, the user must have MXINDX $\geq 0$. Do you want the maximum index to default to 3 ?

Yes – Set INFO(10)=0

No – Set INFO(10)=1
and define MXINDX by setting IWORK(4)=MXINDX.

**INFO(11)** In this code it is not neccesary to provide consistent initial conditions. Using the special structure of the strangeness free DAE, the code can compute consistent initial values to start the integration (see 2.1). However, often consistent initial values are known and the code should use these values. Do you want the code to compute consistent initial values ?

Yes – Set INFO(11)=0

No – Set INFO(11)=1
and input consistent initial values in X.

**INFO(12)** If INFO(11) = 0, the code computes consistent initial values in the least squares sense. The default method is to compute consistent initial values which are close (in the least squares sense) to the given X. Sometimes the user knows which are the differential variables and he wants to prescribe these variables. In this case, the user can use a different method, which keeps some of the variables fixed (see 2.1). Note that this may cause a rank-deficient Jacobian. Do you want the code to use the default method for computing consistent initial values ?

Yes – Set INFO(12)=0

No – Set INFO(12)=1
and set IFIX(I)=1 if you want to keep X(I) fixed.

**INFO(13)** Nonlinear DAE's sometimes suffer from severe scaling problems. DGENDA uses row- and column scaling of the iteration matrices, where the scaling vectors SCALR and SCALC are automatically updated. If the problem does not suffer from scaling problems, the user can disable this automatic scaling. If, on the other hand, the user knows a lot about the correct scaling of the problem, its possible to supply a user-defined scaling routine USCAL (see also 2.4, 4.2). Do you want the code to use the default scaling method ?

Yes – Set INFO(13)=0

No – Set INFO(13)=1
and set IWORK(16)=0 if you want to disable any scaling or set IWORK(16)=2 if you want to use your own scaling subroutine.

**INFO(14)** In every step the BDF-solver uses the simplified Gauss-Newton method [7] to solve the nonlinear system. If $\mu$=0, you can let the code decide when a new iteration-matrix is needed. The method used for this decision is the same as in DASSL [8]. You can also let the code use the classical Gauss-Newton method, which is much slower than the default method. Do you want the code to use the simplified Gauss-Newton method ?

Yes – Set INFO(14)=0

No – Set INFO(14)=1
and set IWORK(2)=0 if you prefer the method used in DASSL or set IWORK(2)=2 if you want the code to use the classical Gauss-Newton method in every step.

**INFO(15)** In every step of the BDF-solver the code obtains the starting values for X(1 : N) and XPRIME for the Gauss-Newton iterations by interpolation. For the remaining starting values X(N+1 : ($\mu$ +2)N) it takes by default X(N+1 : 2N)=XPRIME and classical homotopy for the remaining values, i.e. it takes the last solutions as starting values for the next step. Do you want to use this choice for the starting values ?

Yes – Set INFO(15)=0

No – Set INFO(15)=1
and set IWORK(17)=0 if you want to use classical homotopy also for X(N+1:2N), or set INFO(17)=2 if you want the code to compute the starting values for X(N+1:2N) by linear extrapolation, or set IWORK(17)=3 if you want the code to obtain all values X(N+1:($\mu$ +2)N) by linear extrapolation.

**INFO(16)** The user must initialize the code with correct characteristic values of the DAE. However, DGENDA can try to calculate these values after a successful BDF-step or after consistent initial values have been computed. If it detects any changes in the characteristic values the code will return with an error flag. This is not necessary if the user is sure about the given characteristic values. Are you sure about the characteristic values of the DAE ?

Yes – Set INFO(16)=0

No – Set INFO(16)=1

**INFO(17)** If INFO(16)=1, you can let the code check the characteristic values once after every call of DGENDA or after every BDF-step. Do you think its enough to have the characteristic values checked once after every call to DGENDA ?

Yes – Set INFO(17)=0

No – Set INFO(17)=1

**INFO(18)** If INFO(16)=1, the rank decisions made to compute the characteristic values may suffer from scaling problems and of the error made in every BDF-step. A singular value of a matrix is taken to be zero, if it is smaller than the largest singular value multiplied by $COND^{-1}$. By default, we set $COND = 10^{12}$. Do you think this is the right choice ?

Yes – Set INFO(18)=0

No – Set INFO(18)=1
and define COND by setting IWORK(10)=COND.

**INFO(19)** DGENDA can print a lot of information like the intermediate data, convergence and rank decision monitors to a user-defined output unit. If the user needs this information, he has to set the logical output unit LUN and the KPRINT-parameter defining the level of output data DGENDA will print. Setting KPRINT=0 will produce no output, any value for KPRINT between 1 and 6 will lead to an increasing output level. Do you want no output printed by DGENDA ?

Yes – Set INFO(19)=0

No – Set INFO(19)=1
and define LUN and KPRINT by setting IWORK(21)=LUN and IWORK(22)=KPRINT.

**INFO(20)** The code assumes that the user wants to solve a general nonlinear problem. If it is known that the projector $Z_2$ for the algebraic equations does only depend on $t$, the code can compute a reduced system. This will result in a faster run of the program. If the problem is linear the user can also help the code to be faster. Is the problem nonlinear and does $Z_2$ also depend of the values of X ?

Yes – Set INFO(20)=0

No – Set INFO(20)=1
and set IWORK(23)=1 if $Z_2$ only depends of $t$ or set IWORK(23)=0 if the problem is linear.

**INFO(21)** The code uses the subroutine NLSCON for the calculation of consistent initial values if INFO(11)=0. By default the NONLIN parameter of NLSCON is set to 2 if the DAE is nonlinear and to 1 if the problem is linear. The user can change this value if DGENDA fails to compute consistent initial values (IERR=-24). Do you prefer the default settings for the NONLIN-parameter ?

Yes – Set INFO(21)=0

No – Set INFO(21)=1
and define NONLIN by setting IWORK(24)=NONLIN.

**INFO(22)** If INFO(11)=0 the user can prescribe the required relative precision of the consistent initial values by setting the RTOL parameter of the subroutine NLSCON. By default, RTOL is set to $10^{-10}$. Do you want the relative precision to be set to the default value ?

Yes – Set INFO(22)=0

No – Set INFO(22)=1
and define RTOL by setting RWORK(11)=RTOL.

## 6.2 Continuing the integration

The user must monitor the IERR parameter in order to determine what to do next.

Do not alter any quantity not specifically permitted below, in particular do not alter N, T, X(*), RWORK(*), IWORK(*) or the differential equation in subroutines FDIF and DFDIF. Any such alteration constitutes a new problem and must be treated as such, i.e., the user must start anew. The user cannot change from vector to scalar error control or vice versa (INFO(2)), but he can change the size of the entries of RTOL and ATOL. Increasing a tolerance makes the equation easier to integrate. Decreasing a tolerance will make the equation harder to integrate and should generally be avoided.

The user can switch from the intermediate-output mode to the interval mode (INFO(3)) or vice versa at any time.

If it has been necessary to prevent the integration from going past a point TSTOP (INFO(4), RWORK(1)), keep in mind that the code will not integrate to any TOUT beyond the currently specified TSTOP. Once TSTOP has been reached the user must change the value of TSTOP or set INFO(4)=0. The user may change INFO(4) or TSTOP at any time but he must supply the value of TSTOP in RWORK(1) whenever he sets INFO(4)=1.

The user should not change INFO(5), IWORK(1), or IWORK(2) unless he is going to restart the code.

*** Following a completed task ***

If

IERR = 1: call the code again to continue the integration another step in the direction of TOUT.

IERR = 2 or 3: define a new TOUT and call the code again. TOUT must be different from T. The user cannot change the direction of integration without restarting.

IERR = 4: define a new TOUT and call the code again. TOUT must be different from T.

*** Following an interrupted task ***

To show the code that the user realizes that the task was interrupted and that he wants to continue, he must take appropriate action and set INFO(1) = 1. If

IERR = −1: The code has taken 10000 steps. If the user wants to continue, set INFO(1) = 1 and call the code again. Additional 10000 steps will be allowed.

IERR = −2: The error tolerances RTOL and ATOL have been increased to values the code estimates appropriate for continuing. The user may want to change them. If the user is sure he wants to continue with relaxed error tolerances, set INFO(1)=1 and call the code again.

IERR = −3: A solution component is zero and the user sets the corresponding component of ATOL to zero. If the user is sure to continue, he must first alter the error criterion to use positive values for those components of ATOL corresponding to zero solution components, then set INFO(1)=1 and call the code again.

IERR = −6: Repeated error test failures occurred on the last attempted step in DGENDA. If you are absolutely certain you want to continue, you should restart the integration.

IERR = $-7$: Repeated convergence test failures occurred on the last attempted step in DGENDA. If you are absolutely certain you want to continue, you should restart the integration.

Note that in the case of repeated convergence test failures it may help to change the computation of the starting values for the Gauss-Newton iterations, see 6.1, INFO(15).

IERR = $-8$: A rank deficient Jacobian occurred several times on the last attempted step in DGENDA. If you are absolutely certain you want to continue, you should restart the integration.

IERR = $-9$: Repeated convergence test failures occurred on the last attempted step in DGENDA and additionally there were several error test failures. If the you are absolutely certain to continue, you should restart the integration.

IERR = $-20$: DGENDA could not determine the characteristic values of the problem because the error flag IERR in the subroutine DFDIF had a negative value.

IERR = $-21$: DGENDA could not continue the integration because the error flag IERR in the subroutine FDIF or DFDIF had a negative value.

IERR = $-22$: DGENDA could not determine the strangeness index. It is possible that the problem is ill-posed, and cannot be solved using this code.

IERR = $-23$: DGENDA could not compute an equivalent strangeness index zero system. It is possible that your problem is ill-posed, and cannot be solved using this code.

IERR = $-24$: DGENDA could not compute an initial X, an error was signalled by the subroutine NLSCON. It is possible that the problem is ill-posed, and cannot be solved using this code.

IERR = $-25$: DGENDA could not compute an initial X because the error flag IERR in the subroutine FDIF or DFDIF had a negative value.

IERR = $-26$: The strangeness index or the characteristic values changed on the last step. It is possible that a solution to the problem does not exist.

IERR = $-27$: The strangeness index or the characteristic values changed on the last step. It is possible that a solution to the problem either does not exist or that the strangeness index is higher than expected.

*** Following a terminated task ***

IF IERR $< -100$ the user cannot continue the solution of this problem. An attempt to do so will result in the users run being terminated.

# 7 Example

Solve the DAE given by

$$\begin{bmatrix} \dot{x}_2 - x_1 - e^{t-1} \\ \dot{x}_1(\dot{x}_2 - x_1 - e^{t-1}) + x_2 - t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \dot{x}_1 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_2 - x_1 - e^{t-1} \\ x_2 - t \end{bmatrix} = 0, \quad t \in [0,1],$$

which has the characteristic values

$$\mu = 1, d_\mu = 0, a_\mu = 2, u_\mu = 0.$$

Since the DAE is equivalent to a purely algebraic equation, there is no freedom in the choice of consistent initial values and the unique solution is

$$x(t) = \begin{bmatrix} 1 - e^{t-1} \\ t \end{bmatrix}.$$

## 7.1 Program Text

```
C     DEMO: Demonstration program for DGENDA.
C
C     The problem is a 2x2 strangeness index 1 DAE wich is equivalent
C     to a purely algebraic equation.
C
C     REVISIONS
C
C     2001, March 9 (I. Seufer).
C
C     ******************************************************************
C
C     .. Parameters ..
      INTEGER          M, N, NQ, LRW, LIW
      PARAMETER        (M = 2, N = 2, NQ = 6, LRW = 1000, LIW = 200)
      INTEGER          NOUT
      PARAMETER        (NOUT = 10)
      DOUBLE PRECISION TSTART
      PARAMETER        (TSTART = 0.0D0)
C     .. Local Scalars ..
      DOUBLE PRECISION DSEC, ERO, HU, T, TOUT
      INTEGER          I, IERR, IOUT, IWARN, NFE, NJE, NQU, NST
C     .. Local Arrays ..
      DOUBLE PRECISION ATOL(1), DTOUT(NOUT), ERROR(N), RPAR(1),
     $                 RTOL(1), RWORK(LRW), X(NQ), XPRIME(N),
     $                 SCALC(NQ), SCALR(NQ)
      INTEGER          INFO(22), CVAL(4), IPAR(1), IWORK(LIW), IFIX(NQ)
C     .. External Functions ..
      DOUBLE PRECISION DNRM2, DSECND
      EXTERNAL         DNRM2, DSECND
      EXTERNAL         FDIF, DFDIF, USCAL
```

```
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 /
C     .. Executable Statements ..
C
C     Set the INFO array to tell the code how to solve the problem.
C     We use the standard setup by setting all entries to zero.
C
      DO 10 I=1,22
         INFO(I) = 0
   10 CONTINUE
C
C     Set the starting time and the initial values.
C     Note, that we choose inconsistent initial values.
C
      T = TSTART
      DO 20 I=1,NQ
         X(I)=0.0D0
         SCALC(I)=0.0D0
   20 CONTINUE
C
C     Set the characteristic values
C
      CVAL(1) = 1
      CVAL(2) = 0
      CVAL(3) = 2
      CVAL(4) = 0
C
C     Now we have to set the tolerances for DGENDA 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
      ATOL(1) = 1.0D-5
      RTOL(1) = 1.0D-5
C
C     Write some information about the problem to be solved.
C
      WRITE (6,100) N,RTOL,ATOL,(CVAL(I),I=1,4)
C
  100 FORMAT(/1X,' DEMONSTRATION PROGRAM FOR DGENDA',///
     $     1X,' EXAMPLE 5.1 FROM P. KUNKEL AND V. MEHRMANN,',/
     $     1X,' REGULAR SOLUTIONS OF NONLINEAR DIFFERENTIAL-ALGEBRAIC',/
     $     1X,' EQUATIONS AND THEIR NUMERICAL DETERMINATION:',//
     $     1X,' N =',I3,   /1X,' RTOL =',E10.1,'   ATOL =',E10.1,//
     $     ' PROBLEM DESCRIPTION'/
     $     '   STRANGENESS INDEX   ',I2,4X,
```

```
     $     '   DIFFERENTIAL COMPONENTS   ',I2/29X,
     $     '   ALGEBRAIC    COMPONENTS   ',I2/29X,
     $     '   UNDETERMINED COMPONENTS   ',I2/)
C
C     Before we solve the problem, we compute the characteristic values
C     of the DAE and consistent initial values by setting TOUT = T and
C     calling DGENDA.
C
      WRITE (6,115) (X(I),I=1,NQ)
 115  FORMAT(' GIVEN INITIAL VALUES'/(' ',6E12.4/))
      TOUT = T
      CALL DGENDA (INFO, FDIF, DFDIF, USCAL, M, N, T, TOUT, X,
     $            XPRIME, CVAL, IPAR, RPAR, IFIX, SCALC, SCALR,
     $            RTOL, ATOL, IWORK, LIW, RWORK, LRW, IWARN,
     $            IERR)
      IF (IERR.LT.0) THEN
         CALL DNLERM (INFO, FDIF, DFDIF, USCAL, M, N, T, TOUT , X,
     $               XPRIME, CVAL, IPAR, RPAR, IFIX, SCALC, SCALR,
     $               RTOL, ATOL, IWORK, LIW, RWORK, LRW, IWARN,
     $               IERR)
         STOP
      END IF
      WRITE (6,116) (X(I),I=1,NQ)
 116  FORMAT(' CORRECTED INITIAL VALUES'/(' ',6E12.4/))
C
C     The next step is to solve the problem.
C     We will use DGENDA to compute NOUT intermediate solutions from
C     DTOUT(1) to DTOUT(10) (see above).
C
C     For each intermediate solution the weighted error is computed and
C     some statistics are displayed, where
C     T    is the actual time,
C     X(1) is the computed first solution component at time T,
C     ERO  is the actual weighted error,
C     ORD  is the order of the BDF method used in the last step (if METHOD=1),
C     H    is the stepsize used in the last step.
C
      WRITE (6,110)
C
 110  FORMAT(///
     $     10X,'T',14X,'X(1)',12X,'ERO',8X,'ORD',8X,'H'/)
      DSEC = DSECND()
      DO 200 IOUT = 1,NOUT
         TOUT = DTOUT(IOUT)
         CALL DGENDA (INFO, FDIF, DFDIF, USCAL, M, N, T, TOUT, X,
     $            XPRIME, CVAL, IPAR, RPAR, IFIX, SCALC, SCALR,
     $            RTOL, ATOL, IWORK, LIW, RWORK, LRW, IWARN,
```

```
     $           IERR)
         IF (IERR.LT.0) THEN
            CALL DNLERM (INFO, FDIF, DFDIF, USCAL, M, N, T, TOUT , X,
     $                XPRIME, CVAL, IPAR, RPAR, IFIX, SCALC, SCALR,
     $                RTOL, ATOL, IWORK, LIW, RWORK, LRW, IWARN,
     $                IERR)
            STOP
         END IF
         ERROR(1) = (1.0D0 - EXP(T-1.0D0) - X(1))
     $              /(1.0D0 - EXP(T-1.0D0) + 1.0D0)
         ERROR(2) = (T - X(2))
     $              /(T + 1)
         ERO = DMAX1(ERO,DNRM2(N,ERROR,1)/DSQRT(DFLOAT(N)))
         HU = RWORK(7)
         NQU = IWORK(8)
         WRITE (6,130)
     $           T,X(1),DNRM2(N,ERROR,1)/DSQRT(DFLOAT(N)),NQU,HU
C
 130     FORMAT(1X,E15.5,E16.5,E16.5,I6,E14.3)
C
 200  CONTINUE
C
C     Finally, we display some final statistics
C
      DSEC = DSECND()-DSEC
      NST = IWORK(11)
      NFE = IWORK(12)
      NJE = IWORK(13)
      WRITE (6,210) NST,NFE,NJE,IWORK(14),IWORK(15),ERO,DSEC
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,' MAX ERROR                          =',E10.2/
     $     1X,' RUNTIME                            =',E10.2)
C
      STOP
      END

      SUBROUTINE FDIF(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 of E(t) the subroutine provides
C
C     REVISIONS
C
C     2001, March 9 (I. Seufer).
C
C     ******************************************************************
C
C     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IDIF, IERR
C     .. Array Arguments ..
      INTEGER          IPAR(*)
      DOUBLE PRECISION RPAR(*), X(*), F(*)
C     .. Local Scalars ..
      DOUBLE PRECISION EXPO
C     .. Executable Statements ..
      IERR = 0
      EXPO = DEXP(T-1.D0)
      IF (IDIF .GE. 2) THEN
         IERR = -1
         RETURN
      ELSE IF (IDIF .EQ. 0) THEN
         F(1) = X(4) - X(1) - EXPO
         F(2) = X(3)*F(1) + X(2) - T
      ELSE IF (IDIF .EQ. 1) THEN
         F(1) = X(6) - X(3) - EXPO
```

```
         F(2) = X(5)*(X(4) - X(1) - EXPO) + X(3)*F(1) + X(4) - 1.0D0
      END IF
      RETURN
C *** Last line of FDIF ***
      END

      SUBROUTINE DFDIF(T, IDIF, X, JAC, LJAC, IPAR, RPAR, IERR)
C
C     ARGUMENT LIST
C
C        ARGUMENTS IN
C
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
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
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 of E(t) the subroutine provides
C
C     REVISIONS
C
C     2001, March 9 (I. Seufer).
C
C     ****************************************************************
C
C     .. Scalar Arguments ..
C     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          IDIF, IERR, LJAC
```

```
C     .. Array Arguments ..
      DOUBLE PRECISION X(*), JAC(LJAC,*), RPAR(*)
      INTEGER          IPAR(*)
C     .. Local Scalars ..
      DOUBLE PRECISION EXPO
C     .. Executable Statements ..
      IERR = 0
      CALL DLASET('N', 2, 6, 0.0D0, 0.0D0, JAC, LJAC)
      EXPO = DEXP(T-1.D0)
      IF (IDIF .GE. 2) THEN
         IERR = -1
         RETURN
      ELSE IF (IDIF .EQ. 0) THEN
         JAC(1,1) = -1.0D0
         JAC(1,4) = 1.0D0
         JAC(2,1) = -X(3)
         JAC(2,2) = 1.0D0
         JAC(2,3) = X(4) - X(1) - EXPO
         JAC(2,4) = X(3)
      ELSE IF (IDIF .EQ. 1) THEN
         JAC(1,3) = -1.0D0
         JAC(1,6) = 1.0D0
         JAC(2,1) = -X(5)
         JAC(2,3) = X(6) - 2.0D0*X(3) - EXPO
         JAC(2,4) = X(5) + 1.0D0
         JAC(2,5) = X(4) - X(1) - EXPO
         JAC(2,6) = X(3)
      ENDIF
      RETURN
C *** Last line of DFDIF ***
      END

      SUBROUTINE USCAL(M, N, A, LDA, SCALC, SCALR, IERR)
C
C     This is a dummy routine
C
C     .. Subroutine Arguments ..
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
```

## 7.2 Program Data

None.

## 7.3 Program Results

```
 DEMONSTRATION PROGRAM FOR DGENDA


 EXAMPLE 5.1 FROM P. KUNKEL AND V. MEHRMANN,
 REGULAR SOLUTIONS OF NONLINEAR DIFFERENTIAL-ALGEBRAIC
 EQUATIONS AND THEIR NUMERICAL DETERMINATION:

 N =  2
 RTOL =   0.1E-04   ATOL =   0.1E-04

PROBLEM DESCRIPTION
  STRANGENESS INDEX    1       DIFFERENTIAL COMPONENTS    0
                               ALGEBRAIC    COMPONENTS    2
                               UNDETERMINED COMPONENTS    0

GIVEN INITIAL VALUES
  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00

CORRECTED INITIAL VALUES
  0.6321E+00  0.2737E-47 -0.3679E+00  0.1000E+01  0.0000E+00  0.0000E+00




        T              X(1)            ERO        ORD        H

    0.10000E+00     0.59343E+00     0.61536E-07     3     0.565E-01
    0.20000E+00     0.55067E+00     0.16244E-06     3     0.565E-01
    0.30000E+00     0.50341E+00     0.19188E-06     3     0.565E-01
    0.40000E+00     0.45119E+00     0.17661E-06     4     0.113E+00
    0.50000E+00     0.39347E+00     0.46043E-09     4     0.113E+00
    0.60000E+00     0.32968E+00     0.15423E-06     4     0.113E+00
    0.70000E+00     0.25918E+00     0.13972E-06     4     0.113E+00
    0.80000E+00     0.18127E+00     0.16535E-06     4     0.113E+00
    0.90000E+00     0.95162E-01     0.31738E-06     4     0.113E+00
    0.10000E+01    -0.10578E-05     0.74797E-06     4     0.113E+00


 FINAL STATISTICS FOR THIS RUN..
 NUMBER OF STEPS                   =   24
```

```
NUMBER OF EVALUATIONS               =   53
NUMBER OF FACTORIZATIONS            =   24
NUMBER OF ERROR TEST FAILURES       =    0
NUMBER OF CONVERGENCE TEST FAILURES =    0
MAX ERROR                           =  0.75E-06
RUNTIME                             =  0.71E-02
```