METHODS AND SOFTWARE FOR DIFFERENTIAL-ALGEBRAIC SYSTEMS

Linda Petzold

This paper was prepared for submittal to the NATO Advanced Research Workshop in Real-Time Integration Methods for Mechanical System Simulation
Snowbird, Utah
August 7-11, 1989

June 1989

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.
Methods and Software for Differential-Algebraic Systems

Linda R. Petzold*
Computing and Mathematics Research Division, L-316
Lawrence Livermore National Laboratory
Livermore, California 94550

June 20, 1989

Abstract

In recent years there has been a great deal of progress in the development of numerical methods and software for the solution of initial value problems in differential-algebraic equations (DAE's). In this paper we review the current state of the development and analysis of numerical methods such as multistep and implicit Runge-Kutta applied to classes of DAE's occurring in applications. We describe some of the available software, giving in particular details on the implementation and use of DASSL and its extensions.

*This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.
1 Introduction

In this paper we give an introduction to the solution by numerical ODE methods and software of nonlinear differential-algebraic (DAE) systems of the form

\[ 0 = F(t, y, y'), \]  

(1.1)

where \( \frac{\partial F}{\partial y'} \) may be singular. Differential-algebraic systems include standard form ODE's as a special case, but they also include problems which are in many ways quite different from ODE's. Initial value problems in DAE's arise in a wide variety of applications, including circuit and control theory, chemical kinetics, modeling of constrained mechanical systems, fluid dynamics and robotics. We explore some classes of initial value problems which can be solved by backward differentiation formulas, and discuss some results on the order of convergence of implicit Runge-Kutta methods applied to DAE systems. We give a description of the algorithms and strategies used in the DAE solver DASSL [30], along with a brief description of how to get started using this code. Finally, we describe some of the extensions to DASSL which have recently been completed or are in progress. Further details on these subjects can be found in [3].

The basic idea of using a numerical method for solving DAE systems was introduced by Gear [16], and consists of replacing \( y \) and/or \( y' \) in (1.1) by a difference approximation, and then solving the resulting equation for an approximation to \( y \). The simplest example of a numerical ODE method for (1.1) is the backward Euler method. Using this approach, the derivative \( y'(t_n) \) at time \( t_n \) is approximated by a backward difference of \( y(t) \) and the resulting system of nonlinear equations is solved for \( y_n \),

\[ 0 = F(\ t_n, y_n, \frac{y_n - y_{n-1}}{h} \), \]  

(1.2)

where \( h = t_n - t_{n-1} \). In this way the solution is advanced from time \( t_{n-1} \) to \( t_n \).

This scheme of approximating the solution and/or the derivative in (1.1) and solving the resulting nonlinear equations is quite convenient for the solution of many different problems arising from practical applications. Systems in applications often are written naturally in the form (1.1). If we consider numerical methods for solving these very general systems, there will be no need to rewrite the system to fit into some special form. In addition, even though some DAE systems can be rewritten as standard form ODE's
or in other very special forms, often this is very inconvenient and it sometimes leads to a loss of sparsity in the matrices which are involved in the solution process.

The disadvantage of this quite general formulation is that there are problems which can be written in the form (1.1) which are not solvable by ODE methods. Also there are problems which are solvable by some numerical ODE methods, but we must take very special care in choosing an appropriate method and also in implementing it. To distinguish these classes of problems requires a means of investigating the underlying structure of the DAE’s.

2 Structure of DAE Systems

In some sense the simplest class of DAE’s is the class of linear constant-coefficient systems of the form

$$Ey' + Fy = g(t).$$  \hspace{1cm} (2.1)

When $E + \lambda F$ is singular for all values of $\lambda$, then no solutions exist, or infinitely many solutions exist. We are not interested in the solution of these systems. The remaining systems, where $E + \lambda F$ is nonsingular, can be completely understood by transforming the system to Kronecker canonical form [36], [15]. These systems are solvable in the sense that solutions exist for all sufficiently smooth input functions $g(t)$, and two solutions which have the same initial value are identical. For solvable systems, there exist nonsingular matrices $P$ and $Q$ which decouple the system,

$$y_1'(t) + Cy_1(t) = g_1(t)$$
$$Ny_2(t) + y_2(t) = g_2(t)$$  \hspace{1cm} (2.2)

where

$$Q^{-1}y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}, \quad Pg(t) = \begin{pmatrix} g_1(t) \\ g_2(t) \end{pmatrix}.$$

into a “differential” part and a “singular” part. The matrix $E$ is a block diagonal matrix whose blocks have the form

$$
\begin{pmatrix}
0 & 0 \\
1 & 0 \\
& & \ddots \\
& & & 0 \\
& & & & 1 & 0
\end{pmatrix}.$$

3
The most important concept in classifying these systems is the index. The index is the dimension of the largest block of \( E \). The index is zero if there is no "singular" part to the system (that is, if the system is a standard form ODE or a system which can be rewritten as a standard form ODE by multiplying by \( A^{-1} \)). In general, the higher the index the more difficulties we are likely to encounter in trying to solve the system by a numerical ODE method.

To put these definitions into perspective, consider a canonical index two problem

\[
\begin{align*}
y'_1 & = g(t) \\
y_2 & = y'_1
\end{align*}
\]

and note that for this system, in contrast to an ODE, the initial values must be consistent with the input function \( g(t) \), and the solution is not as smooth as the input.

For nonlinear systems (1.1), we can define the local index as the index of the local constant-coefficient system at any given time, or the global index, which we will be primarily concerned with here, in terms of the number of differentiations of the system which are necessary to be able to solve for \( y' \) uniquely in terms of \( y \) and \( t \) [10].

**Definition 2.1** The global index of a solvable DAE (1.1) is the smallest nonnegative integer \( m \) such that \( F \) has \( m \) continuous derivatives and the nonlinear system

\[
\begin{align*}
F(t, y, y') & = 0 \\
\frac{dF}{dt}(t, y, y', y'') & = \frac{\partial F}{\partial y} y' + \frac{\partial F}{\partial y'} y'' + \frac{\partial F}{\partial t} = 0 \\
& \quad \vdots \\
\frac{d^m F}{dt^m}(t, y, y', y'', \ldots, y^{(m+1)}) & = 0
\end{align*}
\]  

(2.3)

can be solved for \( y' \) uniquely in terms of \( y \) and \( t \): \( y' = \phi(y, t) \).

There are two important classes of index one systems. The first is the general fully-implicit index one system (1.1). If we assume that the rank of \( \partial F/\partial y' \) is constant, and that the index is identically equal to one in a neighborhood of the solution, then we will refer to these systems as uniform
index one. The second class of index one systems is that of semi-explicit index one systems. These are systems which are written in the special form

\[ 0 = F_1(x, x', y, t) \]
\[ 0 = F_2(x, y, t), \] (2.4)

where \( \partial F_1 / \partial x' \) is nonsingular. The system (2.4) is index one iff \( \partial F_2 / \partial y \) is nonsingular. These semi-explicit index one systems arise frequently in applications. It is important to distinguish them from the general index one system (1.1) because there are numerical methods which perform well for (2.4) but poorly for (1.1).

For high index systems in the general form (1.1), numerical ODE methods such as the backward Euler method can be unstable for small step sizes [20]. It is often possible to reduce the index of a semi-explicit system by analytically differentiating the constraints [20]. Campbell [10] has devised an algorithm for solving higher index systems which repeatedly differentiates the entire system and solves the resulting overdetermined system for \( y' \) in terms of \( y \) and \( t \) in the spirit of Definition 2.1. However, in its present form this algorithm is not useful for the solution of most problems in applications because it requires knowledge of the analytic partial derivative matrices of \( F \) and time derivatives of these matrices and requires a large degree of smoothness which may not be present in all applications.

Despite the discouraging results on the instability of numerical ODE methods for general high index linear systems, we can identify important subsets of high index nonlinear problems for which we can obtain positive results. These systems arise in the modelling of electrical networks and constrained mechanical systems and in the solution of the equations of fluid flow. For index two, these are semi-explicit DAE systems of the form (2.4). The system is index two if \((\partial F_2 / \partial x)(\partial F_1 / \partial y)\) is nonsingular. For index three systems, these are semi-explicit systems which can be written in a triangular form which includes the class of index three constrained mechanical systems described in the next section.

Gear [17] has noted a simple relationship between the fully-implicit index one system and the semi-explicit index two system with which it is often possible to transfer methods and convergence results from one class of problems to the other.
3 Backward Differentiation Formulas

In this section we explore some results on the solution by BDF methods of nonlinear index one systems and some special index two and three nonlinear systems which occur frequently in practical applications. We find that BDF methods converge with the expected order of accuracy for several large classes of DAE's of practical interest.

To solve (1.1) by a BDF method, we replace \( y'(t_n) \) by a \( k \)-step backward differentiation formula,

\[
\rho y_n = \sum_{i=0}^{k} \alpha_i y_{n-i}
\]

(3.1)

to obtain the system of nonlinear equations

\[
F\left(t_n, y_n, \frac{\rho y_n}{\hat{h}}\right) = 0.
\]

(3.2)

Then we have the following result for BDF applied to index one systems [20].

**Theorem 3.1** If \( F \) is uniform index one and is differentiable with respect to \( y \) and \( y' \), the solution of (1.1) by the \( k \)-step BDF method with fixed stepsize \( \hat{h} \) for \( k < 7 \) converges to order \( O(h^k) \) if all initial values are correct to order \( O(h^k) \).

An extension of this theorem to variable stepsizes is given in [19].

While the BDF methods converge as expected for index one problems, there are still practical difficulties in implementing these methods for this class of problems [30] [32]. It should be noted that systems whose index is less than or equal to one are the problems that general-purpose codes [30] are designed to handle.

For higher index systems, we have noted earlier that it is not possible to obtain convergence even of backward Euler, in general. However, we can obtain some encouraging results [27] if we restrict our attention to semi-explicit higher index systems (2.4). Systems of this form arise in the solution of constrained mechanical systems of the form

\[
M(q)q'' = f(q, q', t) + G(q)\lambda
\]

\[
\phi(q) = 0, \quad \frac{\partial \phi}{\partial q} = G^T,
\]

(3.3)
where \( q \) are the positions and \( \lambda \) the Lagrange multipliers. The index of the mechanical systems (3.3) is three.

A simple example of a system of form (3.3) is given by the equations describing a pendulum in Cartesian coordinates. Let \( L \) denote the length of the bar, \( \lambda \) the force in the bar, and \( x \) and \( y \) the coordinates of the infinitesimal ball of mass one located at the free end of the bar. Then \( x, y, \) and \( \lambda \) solve the DAE system

\[
\begin{align*}
x'' &= \lambda x \\
y'' &= \lambda y - g \\
0 &= \frac{1}{2}(x^2 + y^2 - L^2),
\end{align*}
\]

where \( g \) is the gravity constant. The index of this system is three.

For semi-explicit index two systems and for index three constrained mechanical systems of the form (3.3), the \( k \)-step constant-stepsizes BDF method converges to order of accuracy \( O(h^k) \) if the initial values are sufficiently accurate [27] [4]. Gear et al. [19] generalized this result to show that variable stepsizes BDF methods converge for semi-explicit index two systems. Gear and Keiper [18] have recently shown that variable stepsizes BDF methods converge for triangular index three systems if the order of the method is greater than one.

It is possible to use a general purpose code based on backward differentiation formulas to solve these special high index nonlinear systems [32]. However, there are some practical difficulties which must be dealt with. It can be shown that for an index \( m \) system, the iteration matrix which the code uses in the Newton iteration for solving the nonlinear equation (3.2) has a condition number which is \( O(1/h^m) \). This difficulty can be remedied by scaling the equations and the variables. The convergence test and error test must also be modified to allow a variable-stepsizes BDF code to solve these types of problems.

4 Implicit Runge-Kutta Methods

In this section we describe some results on the properties of implicit Runge-Kutta Methods applied to index one DAE systems. These methods are potentially advantageous over multistep methods for some systems (for example, for large systems with frequent discontinuities). However, some care
must be taken in choosing an implicit Runge-Kutta method which is appropriate for DAE's, as these methods do not in general attain the same order of accuracy for DAE's as they do for ODE's.

An $M$-stage implicit Runge-Kutta (IRK) method applied to the system of DAE's (1.1) is written as

\[
F \left( t_{n-1} + c_i h, y_{n-1} + h \sum_{j=1}^{M} a_{ij} Y'_j, Y'_i \right) = 0
\]

\[
y_n = y_{n-1} + h \sum_{i=1}^{M} b_i Y'_i. \tag{4.1}
\]

The method can be written in the shorthand notation which displays the matrix of coefficients,

\[
\begin{array}{cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1M} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2M} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_M & a_{M1} & a_{M2} & \cdots & a_{MM} \\
b_1 & b_2 & \cdots & \cdots & b_M
\end{array}
\]

We will assume that the matrix $A = (a_{ij})$ is nonsingular.

We will first give some results on the properties of implicit Runge-Kutta methods applied to different classes of index one systems, and then we will discuss the properties of these methods applied to nonlinear semi-explicit index two systems. First we need to define some terminology.

The Runge-Kutta method will be called **strictly stable** if the difference between a perturbed Runge-Kutta step

\[
F \left( t_{n-1} + c_i h, z_{n-1} + h \sum_{j=1}^{M} a_{ij} Z'_j + \delta^{(i)}_n, Z'_i \right) = 0 \quad i = 1, 2, \ldots, M
\]

\[
z_n = z_{n-1} + h \sum_{i=1}^{M} b_i Z'_i + \delta^{(M+1)}_n \tag{4.2}
\]

where $z_0 = y_0 + \delta^{(M+1)}_0$, $\|\delta^{(i)}_n\| \leq \Delta$, and an unperturbed Runge-Kutta step (4.1) satisfies $\|y_n - z_n\| \leq K_0 \Delta$, where $0 < h \leq h_0$ and $K_0, h_0$ are constants depending only on the method and the DAE.

Defining the **stability constant** $r = 1 - b^T A^{-1} \epsilon_M$, where $\epsilon_M = (1, 1, \ldots, 1)^T$, it is easy to show for index one DAE's that the IRK method (4.1) is stable
iff the method coefficients satisfy the strict stability condition $|r| < 1$. This
stability condition for DAE's is related to the stability criterion $|R(z)| \leq 1,$
$R(z) = 1 + z b^T (I - z A)^{-1} \epsilon_M$, where $z = h \lambda$, for the stiff model
problem $y' = \lambda y$ because $\lim_{|z| \to \infty} R(z) = 1 - b^T A^{-1} \epsilon_M$.

Applying the IRK method (4.1) to a canonical constant-coefficient index
one system $u = g(t)$, we find that the method is locally accurate to order
$O(h^{k_u+1})$ iff the method coefficients satisfy $A_1(k_u)$, where $A_1(q)$ is defined by

$$A_1(q) : \quad b^T A^{-1} c^j = 1 \quad j = 1, 2, \ldots, q \quad (4.3)$$

where $c^j = (c_1^j, c_2^j, \ldots, c_M^j)^T$. If the method satisfies the strict stability
condition and the order of the IRK method for purely differential systems
is $k_d$, then it is easy to show that the method is globally accurate to order
$O(h^{\min(k_d,k_u+1)})$ for index one constant-coefficient DAE systems.

One class of IRK methods which appears to be particularly promising
for the solution of DAE's is that of stiffly accurate methods. These are
methods whose coefficient matrices satisfy $c_M = 1$, $a_{Mj} = b_j$, $j = 1, \ldots, M$
and $A$ is nonsingular. For semi-explicit index one systems, these methods
have the property which is analogous to BDF that the constraint equations
are satisfied exactly at the end of each time step. Griespentog [21] and
Deufhard et al. [13] have shown that there is no order reduction for stiffly
accurate methods applied to semi-explicit index one systems.

For more general implicit Runge-Kutta methods, Roche [33] has derived
a set of order conditions which include (4.3) using the theory of Butcher
series and rooted trees, for semi-explicit index one systems.

The IRK methods are, in general, even less accurate for fully-implicit
nonlinear index one systems than for semi-explicit systems. The additional
loss of accuracy comes about because of mixing which can occur between
the errors in the differential and singular parts of the system. To state the
results, we must first define some terminology.

Defining

$$C(q) : \quad \sum_{j=1}^{M} a_{ij} c_j^{k-1} = c_i^{k}/k, \quad i = 1, 2, \ldots, M \quad k = 1, 2, \ldots, q$$

$$B(q) : \quad \sum_{j=1}^{M} b_j c_j^{k-1} = 1/k, \quad k = 1, 2, \ldots, q \quad (4.4)$$

Then we have the following result, which is shown in [8].
Theorem 4.1 Suppose that (1.1) is uniform index one and linear in $y'$, the Runge-Kutta method satisfies the stability condition $|r| \leq 1$, the errors in the initial conditions are $O(h^G)$ and the errors in terminating the Newton iterations are $O(h^{G+\delta})$, where $\delta = 1$ if $|r| = 1$ and $\delta = 0$ otherwise, and $G \geq 2$. Then the global errors satisfy $\|e_n\| = O(h^G)$ where

$$G = \begin{cases} q, & \text{if } C(q) \text{ and } B(q) \\ q + 1, & \text{if } C(q), B(q + 1) \text{ and } -1 \leq r < 1 \\ q + 1, & \text{if } C(q), B(q + 1), A_1(q + 1) \text{ and } r = 1 \end{cases}$$

It should be noted that implicit Runge-Kutta methods with $|r| = 1$ can be unstable for some classes of fully-implicit index one systems [1]. These results for index one DAE's are in agreement with the order reduction results for stiff ODE's given in [9] in the limit as the stiffness becomes infinite [8].

For nonlinear semi-explicit index two systems, because of the close relationship between fully-implicit index one systems and semi-explicit index two systems, we have that the global error in the “differential” variable $x$ is given by Theorem 4.1. For methods satisfying the strict stability condition, the global errors in the $y$ variable are $O(h^{G_y})$, where $G_y$ is given by [5]

$$G_y = \begin{cases} q, & \text{if } C(q), B(q), A_1(q) \\ q + 1, & \text{if } C(q), B(q + 1), A_1(q + 1), A_2(q + 1) \end{cases},$$

where $A_2$ are the order conditions for Runge-Kutta methods applied to constant-coefficient index two systems and are given by

$$A_2(q) : \quad b^T A^{-1} \epsilon_M = b^T A^{-2} c$$
$$b^T A^{-2} c^i = i, \quad i = 2, 3, \ldots, q.$$

In a recent paper by Hairer, Lubich and Roche [23], a complete set of order conditions are given for index one and semi-explicit index two systems. Some preliminary results for index three systems are also discussed.

5 The DAE Solver DASSL

In this section we outline the basic algorithms and strategies used in the DAE solver DASSL [30], along with a brief description of the use of this code. DASSL is a code for solving index one DAE systems, although with some modification it can sometimes be used for the solution of higher index systems [32].
Methods and Strategies

The underlying methods in DASSL for solving DAE systems replace the derivative in (1.1) by a difference approximation, and then solve the resulting equation for the solution at the current time \( t_n \) using Newton’s method. For example, replacing the derivative by the backward difference in (1.1), we obtain the first order formula

\[
F \left( t_n, y_n, \frac{y_n - y_{n-1}}{h} \right) = 0. \tag{5.1}
\]

This equation is then solved using Newton’s method,

\[
y^{m+1}_n = y^m_n - \left( \frac{1}{h} \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y} \right)^{-1} F \left( t_n, y^m_n, \frac{y^m_n - y_{n-1}}{h} \right), \tag{5.2}
\]

where \( m \) is the iteration index. The algorithms used in DASSL are an extension of this approach. Instead of using the first order formula (5.1), DASSL approximates the derivative using the \( k \)th order backward differentiation formula (BDF), where \( k \) ranges from one to five. On every step it chooses the order \( k \) and stepsize \( h \), based on the behavior of the solution.

Newton’s method (5.2) converges most rapidly when the initial guess \( y^{(0)}_n \) is accurate. DASSL obtains an initial guess for \( y_n \) by evaluating the polynomial which interpolates the computed solution at the last \( k + 1 \) times \( t_{n-1}, t_{n-2}, \ldots, t_{n-(k+1)} \), at the current time \( t_n \). An initial guess for \( y'_n \) is obtained by evaluating the derivative of this polynomial at \( t_n \). Once \( y^{(0)}_n \) is found, Newton’s method is used to solve for \( y_n \) as in (5.2), except that in general the derivative is approximated by the \( k \)th order BDF formula, instead of by the backward difference of \( y_n \). When the stepsize is not constant, there is a choice as to which form of the BDF formula to use. DASSL uses the fixed leading coefficient form of the BDF formula [25].

It is important to solve the nonlinear equation (5.1) efficiently. To simplify notation, we can rewrite this equation as

\[
F(t, y, \hat{\alpha} y + \beta) = 0, \tag{5.3}
\]

where \( \hat{\alpha} \) is a constant which changes whenever the stepsize or order changes, \( \beta \) is a vector which depends on the solution at past times, and \( t, y, \hat{\alpha}, \beta \) are evaluated at \( t_n \). This equation is solved in DASSL by a modified version of Newton’s method,

\[
y^{(m+1)} = y^m - c \left( \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y} \right)^{-1} F \left( t, y^m, \hat{\alpha} y^m + \beta \right). \tag{5.4}
\]
The iteration matrix $G = \alpha \partial F/\partial y' + \partial F/\partial y$ is computed and factored, and is then used for as many time steps as possible. In general, the value of $\alpha$ when $G$ was last computed is different from the current value of $\hat{\alpha}$. If $\alpha$ is too different from $\hat{\alpha}$, then (5.4) may not converge. The constant $c$ in (5.4) is chosen to speed up the convergence when $\alpha \neq \hat{\alpha}$, and is given by

$$c = \frac{2}{(1 + \hat{\alpha}/\alpha)}.$$  \hspace{1cm} (5.5)

The rate of convergence $\rho$ of (5.4) is estimated whenever two or more iterations have been taken by

$$\rho = \left( \frac{||y^{m+1} - y^m||}{||y^1 - y^0||} \right)^{1/m}.$$  \hspace{1cm} (5.6)

The norms are scaled norms which depend on the error tolerances specified by the user. The iteration has converged when

$$\frac{\rho}{1 - \rho} ||y^{m+1} - y^m|| < 0.3.$$  \hspace{1cm} (5.7)

If $\rho > 0.9$, or $m > 4$, and the iteration has not yet converged, then the step-size is reduced, and/or an iteration matrix based on current approximations to $y$, $y'$ and $\alpha$ is formed, and the step is attempted again.

The linear systems are solved using routines from the LINPACK [14] subroutine package. The matrix can either be dense or have a banded structure. For most problems, the iteration matrix is computed by finite differences. The $j^{th}$ column of $G$ is approximated by incrementing the $j^{th}$ component of $y$ in (5.3), and then forming the finite difference quotient. The choice of the increment is a delicate but important issue. When $G$ is banded, it is computed using the algorithm of Curtis et. al [12] so as to minimize the number of function evaluations required. There is an option available for the user to write a routine to compute $G$, given $t$, $y$, $y'$ and $\alpha$. For some problems, this can be more efficient than using finite differences to compute the matrix.

After the corrector iteration has converged, an error test is made to determine whether the solution satisfies a local error tolerance specified by the user. The test is satisfied whenever $C||y_n - y_n^0|| \leq 1$, where $C$ is a constant which depends on the order and recent stepsize history of the method. The constant $C$ is chosen to control both the variable stepsize local truncation error, and the error in interpolated values of $y$ between mesh points. If the
error test is satisfied, the code takes another step. Otherwise, the stepsize and/or order are reduced and the step is attempted again.

The stepsize and order for the next step are determined using basically the same strategies as in Shampine and Gordon [34]. The code estimates what the error would have been if the last few steps had been taken at constant stepsize, at the current order $k$, and at $k - 2$, $k - 1$, and $k + 1$. If these estimates increase as $k$ increases, the order is lowered; if they decrease, it is raised. The new stepsize $h_{n+1}$ is chosen so that the error estimate based on taking constant stepsizes $h_{n+1}$ at order $k_{n+1}$ satisfies the error test.

**Using DASSL**

DASSL is designed to be as easy to use as possible, while providing enough flexibility and control for solving a wide variety of problems. It is extensively documented in the source code. In this subsection we outline what needs to be done to solve a problem using this code.

The most important information the code needs is how to define and function $F$ in (1.1), which describes the equation to be solved. To define $F$, the user writes a subroutine RES which takes as input the time $T$ and the vectors $Y$ and $YPRIME$, and produces as output the vector DELTA, where $DELTA = F(T,Y,YPRIME)$ is the amount by which the function $F$ fails to be zero for the input values of $T$, $Y$, and $YPRIME$.

To get started, DASSL needs a consistent set of initial values $T$, $Y$, and $YPRIME$. This means that we must have $F(T,Y,YPRIME) = 0$ at the initial time. There is an option in DASSL to compute the initial $YPRIME$, in the case where all of the initial values for $Y$ are known. In other cases, where some of the components of $Y$ may not be known and need to be determined using the DAE, there is some recent research [26] which in principle solves this problem, but to our knowledge there is no currently available production code. Sometimes these values can be obtained for specific classes of physical problems.

Additional information which must be supplied to the solver is virtually identical to that needed by standard ODE solvers [35], so we will not discuss it further here.

When DASSL is finished (either successfully or unsuccessfully), it returns to the user's calling program with a flag IDID which indicates what happened. If the flag is positive, the problem was solved successfully. Otherwise, something went wrong. DAE systems are in general quite a bit more complex than ODE systems, and the number of complications which can occur is correspondingly greater. The code documentation gives information about the most likely cause of the problem in the event that a negative
IDID is encountered, and diagnostic information is available through the work arrays which DASSL uses for storage.

**Extensions to DASSL**

Although DASSL is a powerful code which can handle a wide variety of problems, some problems require capabilities which are not implemented in the standard version of DASSL or which can be performed more efficiently with a code based on DASSL but oriented towards a more specific task. These problems include root-finding, sensitivity analysis, and the solution of very large systems of DAE's whose iteration matrices do not fit naturally into a banded structure.

The standard version of DASSL solves a DAE system from time $T$ to time $TOUT$, where $TOUT$ is specified by the user. In some problems it is more natural to stop the code at the root of some function $g(t, y)$. For example, in computing the trajectory of an object, it does not make sense to continue the solution past the time when the object hits the ground. In other problems, it is necessary to stop the code at the root of $g$ because the function $F$ changes at the roots of $g$. The code DASSLRT has this capability. A vector of functions $g_i(t, y), i = 1, 2, \ldots, NG$, may be supplied to DASSLRT such that the root of any of the $NG$ functions $g_i$ is desired. If there are several roots in a given output interval, DASSLRT returns them one at a time, in the order in which they occur along the solution. An integer array tells the user which $g_i$, if any, were found to have a root on any given return. At the present time, a version of DASSLRT exists and is routinely used to solve application problems at several national laboratories. A version for outside release is planned.

A second extension of DASSL has been developed to handle sensitivity analysis for DAE's. Suppose we are given a DAE

$$F(t, y, y', p) = 0,$$  \hfill (5.8a)

$$y(t_0, p) = y_0(p)$$  \hfill (5.8b)

where $y, y' \in \mathcal{R}^N$, whose solution depends on a vector of $M$ time independent parameters $p$. We would like to compute the $(N, M)$ matrix $W(t)$ of sensitivity functions

$$W(t) = \frac{\partial y(t)}{\partial p}$$  \hfill (5.9)

which describes how the solution components change as a result of changes in the parameters.
The sensitivity matrix satisfies a system of DAE’s which can be derived by partial differentiation of equations (5.8) with respect to the parameter vector $p$

$$\frac{\partial F}{\partial y} W'(t) + \frac{\partial F}{\partial y} W(t) + \frac{\partial F}{\partial t} + \frac{\partial F}{\partial p} = 0$$

$$W(t_0) = \frac{\partial y_0}{\partial p}. \quad (5.10)$$

Computationally, the important observation with respect to the sensitivity equations (5.10) is that they are linear and that they have the same iteration matrix as the original system (5.8) if they are solved using the same sequence of methods and stepsizes. Caracotsios and Stewart [11] have written a code DASSAC for solving the sensitivity system which makes use of these observations and is based on DASSL. The code works as follows. The sensitivity equations (5.10) are appended to the original system. The system is solved using a version of DASSL which has been modified to evaluate the iteration matrix on every step, require only one Newton iteration for the linear system (5.10), and make use of the repetitive block diagonal structure of the iteration matrix to save storage and operations in forming the matrix and solving the linear system. The error estimates and stepsize adjustment are based on the entire system (5.10) appended to (5.8). It is to be noted that the sensitivities may not be accurate if the iteration matrix is computed via finite differencing.

One of the obvious deficiencies of the standard DASSL is the lack of options for dealing with Jacobian matrices which do not fit naturally into a dense or banded category. Marquardt [29] has recently implemented a version of DASSL which makes use of the HARWELL sparse direct linear system solver. A version of DASSL which uses the preconditioned GMRES method to solve the linear system at each Newton iteration is currently under development by Brown, Hindmarsh, Petzold and Ulrich, and is expected to be released soon. This code implements many of the strategies described for ODE’s in [6]. An alternative to the GMRES method for solving nonsymmetric linear systems is the adaptive Chebyshev method of Manteuffel [28] which has been implemented by Ashby [2]. A version of DASSL which uses the preconditioned Chebyshev method is currently under development by Ashby, Lee, Petzold and Saylor.

Finally, we note that Skjellum et al.[37] have recently implemented a version of DASSL for use on massively parallel computers, called Concurrent DASSL.
References


