Consistent Initial Condition Calculation for Differential-Algebraic Systems

Peter N. Brown

Alan C. Hindmarsh

Lawrence Livermore National Laboratory

Linda R. Petzold

University of Minnesota

Center for Applied Scientific Computing

UCRL-JC-122175, Rev. 1

June 1996

CONSISTENT INITIAL CONDITION CALCULATION FOR DIFFERENTIAL-ALGEBRAIC SYSTEMS*

PETER N. BROWN† , ALAN C. HINDMARSH† AND LINDA R. PETZOLD‡

June 10, 1996

Abstract. In this paper we describe a new algorithm for the calculation of consistent initial conditions for a class of systems of differential-algebraic equations which includes semi-explicit index-one systems. We consider initial condition problems of two types–one where the differential variables are specified, and one where the derivative vector is specified. The algorithm requires a minimum of additional information from the user. We outline the implementation in a general-purpose solver DASPK for differential-algebraic equations, and present some numerical experiments which illustrate its effectiveness.

1. Introduction. This paper is concerned with the calculation of initial conditions for systems of differential-algebraic equations (DAEs). We write the DAE system in the general form

$$G(t, y, y') = 0, \tag{1.1}$$

where G, y, and y' are N-dimensional vectors. The initial value problem for this system is the problem of finding a solution that satisfies a consistent set of initial conditions $y(t_0) = y_0$, $y'(t_0) = y'_0$. Two software packages have been written for solving initial value problems for the DAE system (1.1)—DASSL [4], and an extension of it called DASPK [9]. Both use variable-order, variable-stepsize backward differentiation formulas. DASSL solves the linear system at each time step by dense or banded direct linear system methods. In DASPK, the linear systems that arise at each time step are solved with either direct linear system methods, or with a preconditioned Krylov iterative method, namely GMRES [20]. For large-scale systems, the iterative method combined with a suitable preconditioner can be quite effective.

When using either of the solvers DASSL or DASPK, the integration must be started with a consistent set of initial conditions y_0 and y'_0. Consistency requires, in particular, that $G(t_0, y_0, y'_0) = 0$. Usually, not all of the components of y_0 and y'_0 are known directly from the original problem specification. The problem of finding consistent initial values can be a challenging task. The present DASSL and DASPK solvers offer an option for finding consistent y'_0 from a given initial y_0, by taking a small artificial step with the Backward Euler method. However, initialization problems do not always arise in this form, and even for the intended problem type, that technique is not always successful. In any case it is unsatisfactory in that it produces values at $t = t_0 + h$ (h = stepsize) rather than at $t = t_0$. In this paper, we propose an alternative procedure for a class of DAE problems. We will show that this method, in combination with the modified Newton methods of DASSL or the Newton-Krylov methods of [7] and [8], yields an algorithm which converges nearly as rapidly as the underlying Newton or Newton-Krylov method. The new method is very convenient for the user, because it makes use of the Jacobian or preconditioner matrices which are already required in DASSL or DASPK.

The class of problems that we consider is a generalization of semi-explicit index-one DAE systems. Semi-explicit index-one DAE systems are characterized as follows. The dependent variable vector y can be split into a vector u

* This research was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Dept. of Energy, by Lawrence Livermore National Laboratory under contract W-7405-ENG-48.

† Center for Applied Scientific Computing, L-561, Lawrence Livermore National Laboratory, Livermore, California 94551.

‡ Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. The work of this author was partially supported by ARO contract number DAAL03-92-G-0247, DOE contract number DE-FG02-92ER25130, NIST contract number 60NANB2D1272, and by the Minnesota Supercomputer Institute.

of size N_d, called the differential variables, and a vector v of size N_a, called the algebraic variables, such that the equations have the form

$$\begin{aligned} u' &= f(t,u,v) \\ g(t,u,v) &= 0, \end{aligned} \tag{1.2}$$

in which $g_v = \partial g/\partial v$ is a nonsingular square matrix. We will be concerned with the initialization problem of finding the initial value v_0 of v when the initial value u_0 for u is specified.

We can easily generalize the class of problems considered, to those where the ODE subsystem for u may be implicit. Thus we consider systems of the form

$$\begin{aligned} f(t,u,v,u') &= 0 \\ g(t,u,v) &= 0, \end{aligned} \tag{1.3}$$

where $u, f \in \mathbf{R}^{N_d}$ and $v, g \in \mathbf{R}^{N_a}$, with the matrix $f_{u'} = \partial f/\partial u'$ also being square and nonsingular. We will continue to refer to problems of the form (1.3), with $f_{u'}$ and g_v nonsingular, as semi-explicit index-one, even though they may be not be explicit in u'. In fact, our main initialization technique applies to an even more general class of problems, as we will explain later.

We also consider a second type of initialization problem, in which the initial derivatives are specified but all of the dependent variables are unknown. That is, we must solve for y_0 given y_0'. For example, beginning the DAE solution at a steady state corresponds to specifying $y_0' = 0$. This problem does not involve a split of the y vector into differential and algebraic parts or a semi-explicit form for the equations.

In later sections, we will refer to these two problems as *Initialization Problem 1* and *Initialization Problem 2*.

The consistent initialization problem has been studied in a number of contexts. The method used by Berzins et al. in SPRINT [3] is based on a small artificial step with the Backward Euler method. An improvement to the error estimate for this approach is suggested in [14]. Campbell [10, 11] suggests an approach based on Taylor's series expansions for determining consistent initial conditions. This method requires a knowledge of the derivatives of the Jacobian matrix of the problem, which can be obtained via automatic differentiation software. Leimkuhler et al. [16] propose an algorithm based on Taylor's series expansions where the derivatives are approximated numerically. Kröner et al. [14] show how to reduce the computational complexity of these methods by making use of structural information on the DAEs derived via symbolic preprocessing. Pantelides [18, 19] uses a graph-theoretic algorithm to determine the minimal set of equations to be differentiated in order to solve for consistent initial values. The differentiations are then to be carried out exactly, as in Campbell's algorithm. Barton and Pantelides [2] describe an implementation of these ideas in a general-purpose chemical process modeling environment. Bachman et al. [1] propose an algorithm for determining consistent initial values based on a thorough decomposition of the system structure. This algorithm isolates the algebraic constraints by index reduction and also requires exact differentiation. Simeon et al. [21] initialize the higher-index DAEs describing multibody mechanical systems via static equilibrium solution, analogous to solving Initialization Problem 2. The potential of continuation methods for robust solution of the nonlinear consistency conditions in initial condition calculations is investigated in [17, 21].

The main contribution of this paper is an algorithm to solve Initialization Problems 1 and 2 in the context of DAE solvers like DASSL and DASPK, with the help of mechanisms already in place for the solution of the DAE system itself. The algorithm is convenient for users because, unlike most of the approaches cited above, it requires almost no information from the user beyond what is already required for solving the time-dependent DAE system. It does not need symbolic or automatic differentiation software, and is immediately applicable to a wide range of problems, including very large-scale systems, for which the solution of a DAE system is sought.

2. The Basic Method. Consider first Initialization Problem 1 for the semi-explicit index-one system (1.3), where $v_0 = v(t_0)$ is to be determined, given $u_0 = u(t_0)$ at the initial point $t = t_0$. We expand this problem to include the calculation of $u'_0 = u'(t_0)$. Thus we can form a nonlinear system in the N-vector

$$x = \begin{pmatrix} u'_0 \\ v_0 \end{pmatrix} , \tag{2.1}$$

namely

$$F(x) \equiv \begin{pmatrix} f(t_0, u_0, v_0, u'_0) \\ g(t_0, u_0, v_0) \end{pmatrix} = 0 . \tag{2.2}$$

This general approach of solving the expanded problem has also been used in [15]. A Newton iteration for the solution of $F(x) = 0$ would require the Jacobian matrix

$$F'(x) = \begin{pmatrix} f_{u'} & f_v \\ 0 & g_v \end{pmatrix} . \tag{2.3}$$

By assumption, this matrix is nonsingular, at least in a neighborhood of the desired solution.

In the course of integrating a DAE system with DASSL or DASPK, the user must call upon one of several linear system algorithms to solve $N \times N$ linear systems at every time step. These arise from a Newton-like method for solving algebraic systems $G(t, y, c(y - a)) = 0$ for y, where a is a vector containing past values, and c is a constant, set by the solver, that is inversely proportional to the stepsize h. Thus the linear systems have the form

$$J\Delta y = R ,$$

in which R is a residual vector, Δy is a correction to y, and the matrix J is the DAE system iteration matrix

$$J = c\frac{\partial G}{\partial y'} + \frac{\partial G}{\partial y} . \tag{2.4}$$

The user is encouraged to supply an approximation to J, for use either as the Newton matrix itself in the case of direct methods, or as a preconditioner in the case of a Krylov method. In the direct case, J is generated by difference quotient approximations if not supplied by the user. In the case of the system (1.3), we have

$$J = c \begin{pmatrix} f_{u'} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} f_u & f_v \\ g_u & g_v \end{pmatrix} = \begin{pmatrix} cf_{u'} + f_u & f_v \\ g_u & g_v \end{pmatrix} . \tag{2.5}$$

In order to make use of J in solving $F(x) = 0$, we pick an artificial stepsize h, and set $c = 1/h$ in (2.5). Then, to recover the block $f_{u'}$, we rescale the first block-column of J by h, using the scaling matrix

$$S = \begin{pmatrix} hI_d & 0 \\ 0 & I_a \end{pmatrix} , \tag{2.6}$$

where I_d and I_a are the identity matrices of size N_d and N_a, respectively. Thus we consider the matrix

$$\bar{J}(x) \equiv JS = \begin{pmatrix} f_{u'} + hf_u & f_v \\ hg_u & g_v \end{pmatrix} , \tag{2.7}$$

evaluated at $t = t_0$, $(u, v) = (u_0, v_0)$, and $u' = u'_0$. Note that $\bar{J}(x) = F'(x) + hC(x)$ with

$$C(x) = \begin{pmatrix} f_u & 0 \\ g_u & 0 \end{pmatrix} .$$

If h is small in some appropriate sense, we can expect that $\bar{J}$ will be a good approximation to $F'(x)$.

The proposed initialization procedure is to carry out a Newton-like iteration with corrections

$$\Delta x = -\bar{J}^{-1} F(x) \ . \tag{2.8}$$

Each iteration will call on the linear system solution procedure that is to be used later in solving the DAE system itself. It will also require information about which components of y are differential and which are algebraic, in order to apply the correction Δx to the vectors y and y'. But otherwise, the procedure requires no additional information or methodology. Upon convergence, we have all components of $y(t_0)$, and we have the components of $y'(t_0)$ corresponding to u'_0, the derivatives of the differential variables. The remaining components of $y'(t_0)$, corresponding to v'_0, will simply be set to zero, as the integration procedure is insensitive to these (since v' does not appear in (1.3))[1], and the first time step will produce accurate values for them. The next two sections will present this procedure in a more formal manner and prove convergence for it.

For Initialization Problem 2, we are given the initial value of y' and must compute the initial y. In this case, we are simply interested in solving for $x = y_0$ in the system

$$F(x) \equiv G(t_0, x, y'_0) = 0 \ , \tag{2.9}$$

with y'_0 given. We assume that this problem is well-posed as provided by the user, so that $F'(x) = G_y$ is nonsingular in a neighborhood of the solution, including the initial guess supplied. As in the first problem, we will call for the user to supply the DAE iteration matrix J, but this time we set $c = 0$, so that the matrix involved is simply $J = \partial G/\partial y$; there is no stepsize h. We then proceed with Newton iterations using J, with corrections

$$\Delta y_0 = -J^{-1} G(t_0, y_0, y'_0) \ . \tag{2.10}$$

Finally, we remark that an extension of these ideas to Hessenberg index-two DAE systems is under way by the authors.

3. More General Problems. In the case of Initialization Problem 1, the full scope of problems for which the above idea can be applied is more general than systems of the form (1.3).

3.1. Index-0 systems. While the main focus of this work is the initialization of index-one systems, we note here that the easier case of index-0 systems is taken care of as a special case. Here the problem is to compute all of y'_0 given y_0, with the assumption that $G_{y'}$ is nonsingular. This problem is in fact treated by the general procedure of (2.1)–(2.8), by taking $N_a = 0$ (no algebraic components). The unknown vector is $x = y'_0$ and satisfies $F(x) = G(t_0, y_0, x) = 0$. The true Jacobian is $F'(x) = G_{y'}(t_0, y_0, x)$ and is approximated by $\bar{J} = JS = hJ = G_{y'} + hG_y$ for h suitably small.

3.2. Permuted variables. We first wish to generalize the system (1.3) by dropping the requirement that the differential and algebraic components are separated into blocks in y. Thus we assume there is a permutation matrix P of size N such that

$$Py = z = \begin{pmatrix} u \\ v \end{pmatrix} \ , \quad u \in \mathbf{R}^{N_d} \ , \quad v \in \mathbf{R}^{N_a} \ , \tag{3.1}$$

[1] Although the BDF formulas do not depend on these values, the error test at the end of the first step depends on them, unless the v variables are excluded from the error test (which is an option in DASPK).

and that, in terms of z, the system function G has the blocked form $H(t,z,z') = \begin{pmatrix} f(t,u,v,u') \\ g(t,u,v) \end{pmatrix}$ as in (1.3). The vector of unknowns in terms of z is $w = \begin{pmatrix} u_0' \\ v_0 \end{pmatrix}$, and in terms of y it is $x = P^{-1}w$. The system to be solved is

$$0 = F(x) \equiv G(t_0, y_0, y_0') \ , \tag{3.2}$$

in which only N_a components of y_0 and N_d components of y_0' are in the vector x of unknowns. The Jacobian $F'(x)$ and the iteration matrix J are given by the expressions in (2.3) and (2.5), multiplied on the right by P. The scaling by h of the differential variables in y is given by $\bar{S} = P^{-1}SP$, with S as in (2.6). The scaled system matrix (again with $c = 1/h$) is

$$\bar{J}(x) = J\bar{S} = (h^{-1}H_{z'} + H_z)SP = \begin{pmatrix} f_{u'} + hf_u & f_v \\ hg_u & g_v \end{pmatrix} P \ . \tag{3.3}$$

So it remains clear that $\bar{J}$ approximates $F'(x)$ for small h.

In the same way, we need not require that the components of G are blocked as in (1.3). Thus we allow a permutation Q in the components of G, such that QG has that blocked form. Then of course so does QF. However, in our initialization procedure, we can work with G (hence F) in its original ordering.

3.3. Implicit constraints. To generalize further the form of the problem we can solve, suppose that, after permuting the y vector to $z = Py$, the DAE system function G has the form

$$G(t,y,y') = H(t,u,v,u'), \quad u \in \mathbf{R}^{N_d} \ , \ v \in \mathbf{R}^{N_a} \ , \quad \text{with } H_{u'} \text{ having full rank } N_d \ . \tag{3.4}$$

This class of problems generalizes (1.3) in that the algebraic constraints, $g(t,u,v) = 0$ in (1.3), need not be identified explicitly. As above, we define the vectors $w = (u_0', v_0)^T$ and $x = P^{-1}w$ of unknowns, and the system to be solved is

$$0 = F(x) \equiv G(t_0, y_0, y_0') = H(t_0, u_0, v_0, u_0') \ . \tag{3.5}$$

Here the Jacobian $F'(x)$ is given by

$$F'(x) = [\partial H(t_0, u_0, v_0, u_0')/\partial w]P = (H_{u'}, H_v)P \ . \tag{3.6}$$

The nonsingularity of F' can be shown from the assumptions of full rank for $H_{u'}$ and index one for the system. The system iteration matrix J being supplied (or approximated) by the user is now

$$J = cG_{y'} + G_y = cH_{z'}P + H_zP = (cH_{u'} + H_u, H_v)P \ . \tag{3.7}$$

Taking $c = 1/h$, and using the scaling matrix $\bar{S} = P^{-1}SP$ as above, the scaled system matrix is

$$\bar{J}(x) \equiv J\bar{S} = (h^{-1}H_{u'} + H_u, H_v)SP = (H_{u'} + hH_u, H_v)P \ . \tag{3.8}$$

Comparing (3.6) and (3.8), we again expect $\bar{J}$ to work well as an approximation to $F'(x)$ in a Newton iteration, for sufficiently small h. The corrections to x now take the form

$$\Delta x = -\bar{J}(x)^{-1}F(x) = -\bar{S}^{-1}J(x)^{-1}F(x) \ . \tag{3.9}$$

Here $\bar{S}^{-1}$ is a scaling the differential variables in y by h^{-1}.

It is the class of problems given by (3.4) that we take as the scope of Initialization Problem 1, for which we have implemented the algorithm described above.

3.4. General index-one systems. Note that (3.4) does not include all fully-implicit index-one DAEs, because the rank and dependency conditions combined exclude certain index-one systems. A simple example is the system

$$\begin{aligned} y_1' + y_2' &= g_1(t, y_1) \\ y_2 &= g_2(t) \ . \end{aligned} \tag{3.10}$$

This has index one, and it is well-posed for any given value of $y_1(t_0)$. But it does not fit into the scheme of (3.4), because it contains the derivatives of both variables, but the rank of the 2×2 matrix $G_{y'}$ is only 1.

In principle, our scheme can be applied to more general index-one DAE systems by allowing P to be a more general matrix, not just a permutation. If a constant nonsingular matrix P can be found that transforms y into $z = Py = (u, v)^T$ such that $G(t, y, y') = H(t, u, v, u')$ with $H_{u'}$ having full rank, as in (3.4), then our procedure can be applied, as follows: Defining w, $x = P^{-1}w$, and $F(x)$ as before, we again have $F'(x) = H_w P = (H_{u'}, H_v)P$, and $\bar{J}(x) \equiv J\bar{S} = (H_{u'} + hH_u, H_v)P$. Thus, as long as the problem is well-posed (hence F' is nonsingular), the Newton iteration using $\bar{J}$ should work well. However, in contrast to the case where P is a permutation, once a solution vector x is found, the vector $y_0 = P^{-1}z_0 = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$, where v_0 is defined by $\begin{pmatrix} u_0' \\ v_0 \end{pmatrix} = Px$, may differ from the input value of y_0 in *all* of its components.

For the example system (3.10) above, an appropriate matrix P is

$$P = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \ ,$$

making $u = y_1 + y_2$ and $v = y_2$. Our procedure would determine $v_0 = y_{2,0}$ correctly. But if the input initial value of y_2 differs from $g_2(t_0)$, both components of y_0 would be altered by the procedure, while $u_0 = y_1 + y_2$ is unaltered. If preserving $y_1(t_0) = y_{1,0}$ is required, another approach would have to be used.

The procedure with more general constant P determines a consistent set of initial conditions only in the sense that the initial values of the transformed variables are consistent. Initial values for the derivatives of the original variables may or may not be consistent. For example, in the system (3.10) above, $y_1' + y_2'$ is initialized correctly, whereas y_1' or y_2' individually can only be determined if information about the constraint derivative $g_2'(t)$ is used [16]. Our procedure would leave $v_0' = y_{2,0}'$ undefined.

For the most general index-one systems, the matrix P which achieves the form (3.4) would generally not be constant: $P = P(t, y, y')$. Due to the complications and expense of finding such a smooth P and continuing with this change of variables in later time steps, we have chosen not to implement this extension of our algorithm, instead restricting P to the class of permutation matrices. If necessary, the user may be able to bring the problem to the form (3.4) by a change of variables as described above.

4. Convergence Theory. In Initialization Problem 1, given by (3.4)–(3.5), the use of an approximate Jacobian $\bar{J}(x, h)$ raises the question of convergence of the Newton iteration, for which we give a convergence theorem here. In order to cover all situations, the theorem below includes both full and modified Newton iteration for the same problem. Here "full Newton" refers to the fact that the Jacobian, though approximate, is evaluated at every iteration x_k, while "modified Newton" means that it is evaluated at a fixed argument $\bar{x}$. We give the main convergence theorem without proof, but with citations of similar results in [13]. Then we discuss the application of the theorem to the DAE initialization problem, and give a similar convergence theorem that applies to the use of Newton-Krylov iteration.

4.1. Newton iteration convergence. For a general system $F(x) = 0$, consider the iteration

$$x_{k+1} = x_k - \bar{J}(\bar{x}_k, h)^{-1} F(x_k) \ , \tag{4.1}$$

where both types of Newton iteration are included by taking

$$\bar{x}_k = \begin{cases} x_k & \text{for full Newton iteration, or} \\ \bar{x} & \text{for modified Newton iteration} , \end{cases} \tag{4.2}$$

and $\bar{x}$ is fixed. The iteration uses an approximate Jacobian $\bar{J}(x,h)$ assumed to have the form $\bar{J}(x,h) = F'(x) + hC(x)$ for all x and h of interest. The following theorem says that this iteration converges, at least linearly, under mild smoothness assumptions on the functions F and C. The norm $\|\cdot\|$ used here is arbitrary.

THEOREM 4.1. *Let $F : R^N \to R^N$ be continuously differentiable in an open convex set $D \subset R^N$. Assume that*

(a) there exists $x_* \in D$ such that $F(x_*) = 0$;

(b) $F'(x_*)$ is nonsingular, and $\|F'(x_*)^{-1}\| \le \beta$ for a constant $\beta > 0$;

(c) there exists $\rho > 0$ such that the neighborhood $N(x_*, \rho) = \{x : \|x - x_*\| < \rho\} \subset D$, and $F'(x)$ satisfies a Lipschitz condition in $N(x_*, \rho)$ with constant γ; and

(d) there exist matrix-valued functions $\bar{J}(x,h)$ and $C(x)$, for all $h \ge 0$ and $x \in D$, related by

$$\bar{J}(x,h) = F'(x) + hC(x) , \tag{4.3}$$

with $C(x)$ satisfying a Lipschitz condition in $N(x_*, \rho)$ with constant γ_c.

Then there exist constants $\epsilon > 0$ and $\bar{h} > 0$ such that for $x_0 \in N(x_*, \epsilon)$, any $0 \le h \le \bar{h}$, and any $\bar{x} \in N(x_*, \epsilon)$ in the modified Newton case, the sequence generated by (4.1) is well-defined and converges to x_*. Under these conditions, the iterates obey

$$\|x_{k+1} - x_*\| \le 2\beta\gamma\|x_k - x_*\|^2 + 4\beta(\gamma\eta_k + \bar{h}C_0)\|x_k - x_*\| , \tag{4.4}$$

where $\eta_k = \|x_k - \bar{x}_k\|$ and $C_0 \equiv \epsilon\gamma_c + \|C(x_*)\|$. In the full Newton case, $\eta_k = 0$, and in the modified Newton case, $\eta_k < 2\epsilon$.

The proof is an extension of the proof in Dennis and Schnabel [13], p. 90, which treats the full Newton case with exact Jacobian. The main complication is due to the inaccuracy in the iteration matrix. For that, the perturbation relation (3.1.20) in [13], p. 45, is useful. We also make use of Lemma 4.1.12 of [13], p. 75.

4.2. Application to DAE systems. The general time step within DASSL and DASPK involves the solution of linear systems $J\Delta y = R$ in which the matrix is the DAE system iteration matrix of (2.4), $J = cG_{y'}(t,y,y') + G_y(t,y,y')$. In order to use values of J inside a Newton or modified Newton iteration for solving the system $F(x) = 0$ from Initialization Problem 1, we exploit the relationship between the Jacobian matrix $F'(x)$ and J. From (3.6)–(3.8), we have

$$\bar{J} \equiv J\bar{S} = F'(x) + hC(x) , \tag{4.5}$$

where $\bar{S} = P^{-1}SP$ in terms of the permutation P from y to u and v, and $C(x) = (H_u, 0)P$ in terms of the system function $H(t,u,v,u')$. Starting from input initial guesses y_0 and y'_0 for $y(t_0)$ and $y'(t_0)$, we will set h and $c = 1/h$ in J. The Newton corrections are given by (3.9), $\Delta x = -\bar{S}^{-1}J^{-1}F(x)$.

In DASPK, when direct methods are selected, then $J(t,y,y')$ is either supplied by the user (possibly in approximate form), or generated by difference quotients, and J^{-1} is realized by the LU method. In this case, $J = J(t_0, y_0, y'_0)$ is fixed, and therefore (3.9) represents a modified Newton method ($\bar{x}_k = x_0$). Theorem 4.1 can be applied to this iteration. If one assumes differentiability of G with respect to y and y', and that the partial derivatives of G with respect to y and y' are locally Lipschitz continuous, then for well-posed initialization problems of the type discussed in Sections 2 and 3, it is clear that the assumptions of the theorem hold. Thus, the iteration on x will converge, given h small enough, and given sufficiently good initial guesses.

When Newton-Krylov iteration is selected, the linear systems $Jp = -F$ are solved with preconditioned GMRES [20] as the linear iteration, with finite-difference approximations involving $G(t, y, c(y-a))$ to approximate the action of J on an arbitrary vector, and with a user-defined preconditioner that approximates J. Theorem 4.1 applies here only if the linear iteration error is neglected. In fact, GMRES computes an approximate solution p_k such that $\|F(x_k) + Jp_k\|$ is small, and for the step given by $\Delta x_k = \bar{S}^{-1} p_k$, $\|F(x_k) + \bar{J}\Delta x_k\|$ is also small.

The reason finite differences of G are used above (instead of F) is that the GMRES solver in DASPK uses finite differences of G to approximate J times an arbitrary vector for the DAE time step. Thus, we are able to apply machinery that is already available, and the user only need be concerned with J, not F'. Because of the relationship between $\bar{J}$ and F', it is also clear that a good preconditioner for J will suffice in the iteration.

The presence of Krylov iteration error leads us to consider the convergence of the following inexact Newton iteration for a general function $F(x)$. Given x_0, consider the following iteration, where for $k = 0, 1, 2, \ldots$ we have

$$\begin{aligned} &\bar{J}(x_k, h)s_k = -F(x_k) + r_k \ , \quad \text{such that} \quad \|r_k\| \le \eta \|F(x_k)\| \ , \\ &x_{k+1} = x_k + s_k \ , \end{aligned} \tag{4.6}$$

with $0 \le \eta < 1$. In the general inexact Newton setting, it is not important how the step s_k is computed, only that such an s_k can be found. With the same assumptions as before, the following theorem says that this iteration converges. Its proof is also a fairly easy modification of that of Theorem 2.3 in Dembo et al. [12].

THEOREM 4.2. *Let $F : R^N \to R^N$ be continuously differentiable in an open convex set $D \subset R^N$. Assume that conditions (a)–(d) of Theorem 4.1 hold. Then there exist constants $\epsilon > 0$ and $\bar{h} > 0$ such that for $x_0 \in N(x_*, \epsilon)$, any $0 \le h \le \bar{h}$, and any $0 \le \eta < 1$, the sequence generated by (4.6) is well-defined and converges linearly to x_*.*

5. The Linesearch Algorithm. In order to improve the robustness of the Newton algorithm discussed above, we also employ a linesearch backtracking algorithm. Consider a general function $F(x) \in \mathbf{R}^N$ for $x \in \mathbf{R}^N$, and define

$$f(x) \equiv \frac{1}{2} F(x)^T F(x). \tag{5.1}$$

Let x and δ in $\mathbf{R}^N$ be such that the residual norm condition

$$\|F(x) + F'(x)\delta\|_2 \le \eta \|F(x)\|_2, \tag{5.2}$$

holds, where $0 \le \eta < 1$, and $\|\cdot\|_2$ is the Euclidean norm. The vector δ can be thought of as an approximate solution of the Newton equations $F'(x)\Delta x = -F(x)$. Given (5.2), it is shown in Brown and Saad [8] that δ is a *descent direction* for f at x, i.e., $f(x + \lambda\delta) < f(x)$ for small positive λ. Given a descent direction δ for f at x, we employ the following backtracking algorithm:

Algorithm 5.1: Given $0 < \theta_{\min} \le \theta_{\max} < 1$ and $0 < \alpha < 1$,

1. Set $\lambda = 1$.
2. If $f(x + \lambda\delta) \le f(x) + \alpha\lambda\nabla f(x)^T\delta$, then set $\lambda_{\text{final}} = \lambda$ and exit; otherwise, continue.
3. Choose $\hat{\lambda} \in [\theta_{\min}\lambda, \theta_{\max}\lambda]$, set $\lambda = \hat{\lambda}$, and go to step 2.

The global convergence of this algorithm used in connection with an inexact Newton iteration is discussed at length in [8]. The simplest choice for the θ's is to take $\theta_{\min} = \theta_{\max} = 1/2$, and this makes $\hat{\lambda} = \lambda/2$. We use this choice in the implementation here, and also use $\alpha = 10^{-4}$, so that only a small decrease in f is required.

From (5.1), we have $\nabla f(x)^T\delta = F(x)^T F'(x)\delta$. Now, if δ is the exact Newton step at x, $\delta = -F'(x)^{-1}F(x)$, then $\nabla f(x)^T\delta = -F(x)^T F(x) = -2f(x)$. Hence in this case, the Armijo condition in Step 2 of Algorithm 5.1 is simply

$$f(x + \lambda\delta) \le (1 - 2\alpha\lambda) f(x). \tag{5.3}$$

On the other hand, if δ is the GMRES solution at the m-th step when applied to $F'(x)\delta = -F(x)$, then it is shown in [7] that $F(x)^T F'(x)\delta = -F(x)^T F(x) + \rho_m^2$, where $\rho_m \equiv \|F(x) + F'(x)\delta\|_2$. Thus, in this case the condition in Step 2 of Algorithm 5.1 is

$$f(x + \lambda\delta) \le (1 - 2\alpha\lambda) f(x) + \alpha\lambda\rho_m^2. \tag{5.4}$$

In the present context, we want to solve the nonlinear system $F(x) = 0$ given by (2.9) or (3.5). However, because F is the DAE system function, we have no suitable measure of the size of $F(x)$ that is directly available. The weighted root-mean-square (WRMS) norm used in DASPK for norms of y suggests that we solve instead the problem

$$0 = \tilde{F}(x) \equiv DA^{-1}F(x),$$

where the matrix A is the current approximate system Jacobian matrix J of (2.4) in the direct case, or the preconditioner P in the Krylov case[2], and D is a diagonal matrix containing the weights used in the WRMS norm. The vector $A^{-1}F$ is dimensionally consistent with the vector y, and $\tilde{F}^T\tilde{F}$ is just the square of the WRMS norm of $A^{-1}F$. Thus it is likely that $\tilde{F}(x)$ is well-scaled in the WRMS norm, and we apply Algorithm 5.1 to $\tilde{f} = \tilde{F}^T\tilde{F}/2$ instead of to f.

The direction vector δ that is available to us is the Newton correction given by (2.10) or (3.9). Thus $\delta = -\bar{J}^{-1}F(x)$, where $\bar{J} = J\bar{S}$ from (3.8) in the case of Initialization Problem 1, or $\bar{J} = J$ in the case of Initialization Problem 2. In both cases, we expect $\bar{J} \approx F'(x)$, but the question arises then as to whether or not δ will be a descent direction for $\tilde{f}$ at the current approximate solution x. An easy calculation gives

$$\delta = -\bar{J}^{-1}(AD^{-1})\tilde{F}(x) = -(\bar{J}^{-1}F'(x))\tilde{F}'(x)^{-1}\tilde{F}(x) \ .$$

Thus if we can assure that $\bar{J}^{-1}F'(x) - I$ is small, δ will be a descent direction for $\tilde{f}$ at x.

6. Implementation. We implemented the algorithms described above for Initialization Problems 1 and 2 as new options in the general-purpose DAE solver DASPK [9]. Initialization Problem 1 is solved for the more general class of index-one systems (3.4) described in Section 3.3. Here we give a few details of the implementation, and describe briefly how to use the new options. Within the dependent variable vector Y in DASPK, we denote by Y_d and Y_a the subvectors of differential and algebraic variables, respectively.

DASPK has an integer array argument INFO which is used to specify a variety of options. In this latest version of DASPK, the user is to set INFO(11) to have DASPK solve one of the two initialization problems:

- INFO(11) = 0 means initial values are already consistent (the default).
- INFO(11) = 1 means solve Initialization Problem 1 (given Y_d, calculate Y_a and Y_d'). In this case, the user must identify the differential and algebraic components of Y, by setting an array ID (of length N) as part of the integer work array IWORK:
  - ID(I) = +1 if Y(I) is a differential variable, and
  - ID(I) = −1 if Y(I) is an algebraic variable.
- INFO(11) = 2 means solve Initialization Problem 2 (given Y', calculate Y).

In any case, the input arrays Y and Y' must contain the initial values for the given components and initial guesses for the unknown components.

The algorithm for Problem 1 requires a stepsize h, to determine $c = h^{-1}$. Initially, we try the initial stepsize h_0 which is used by DASSL and DASPK ([4], p. 128). If convergence is not achieved in the Newton iteration, we divide h by 10. If the algorithm fails for MXNH (nominally = 5) different values of h, DASPK returns an error flag to the

[2] The preconditioner P is not to be confused with the permutation matrix of Section 3.

user. In our experience, if the initialization succeeds, it usually succeeds with the initial choice h_0. For Initialization Problem 2, we always set $c = 0$. Once the initialization has been completed, we reset the initial stepsize h_0 for the first time step of DASPK, based on the newly computed initial values.

For a given value of c, the initialization problem is solved with either a modified Newton method or an inexact Newton method similar to that used in the general time step, augmented by the linesearch algorithm. For both problems, we must solve a system $F(x) = 0$, where $F(x)$ is the residual of the DAE system at t_0, y, y', and x is the vector of unknown components in y or y'. A Jacobian or preconditioner matrix is obtained, either internally or by calling a user routine, depending on the option specified. The user routines to specify the system function G and the Jacobian or preconditioner are exactly the same ones which are needed for the time integration. The Newton iteration is given in terms of an approximation to the system Jacobian J by

$$\Delta x = -\lambda (J\bar{S})^{-1} F(x) = -\lambda \bar{S}^{-1} J^{-1} F(x) \ ,$$

where λ is the relaxation steplength ($0 < \lambda \leq 1$) from the linesearch algorithm. For Initialization Problem 2, the scaling matrix $\bar{S}$ is absent. The vector $p = J^{-1}F(x)$ is computed either by direct LU/backsolve operations, or by the preconditioned GMRES algorithm. The code is organized so that the Newton solver is independent of which initialization problem is being solved. After p is computed, a separate routine is called to apply the increment $\Delta x = -\lambda \bar{S}^{-1} p$ as follows: For Problem 1, we increment Y_a by the algebraic components of $-\lambda p$ and increment Y_d' by the differential components of $-\lambda c p$. For Problem 2, we increment Y by $-\lambda p$.

The complete algorithm involves three loop levels for Problem 1, and two levels for Problem 2. At the innermost level, up to MXNIT Newton iterations are performed with a given value of h and J or P. We consider the iteration to have converged if the scaled residual is small in norm:

$$\rho \equiv \|A^{-1} F(x)\| \leq \text{EPCONI},$$

where A is the current approximate system Jacobian J in the direct case, and the preconditioner matrix P in the Krylov case. The test constant is EPCONI = EPINIT*EPCON, where EPCON = 0.33 is the tolerance for the Newton iteration in the subsequent time steps, and EPINIT is a heuristic factor, nominally 0.01. The WRMS norm is used throughout.

The values $\rho = \rho_m$ from the mth iteration are used to infer a convergence rate, RATE $= \rho_m / \rho_{m-1}$. If convergence is not achieved in MXNIT iterations, the strategy for repeated attempts depends on RATE. If convergence failed, but RATE ≤ 0.8 (the iterations are converging, but slowly), then we retry the Newton iteration with the current values of y and y' and a new value for A (i.e. for J or P), up to a limit of MXNJ such attempts. In addition, in the case of the Krylov method, if the GMRES solver failed to converge after at least two Newton iterations, but RATE < 1, the Newton iteration is retried with a new value for P. If the limit of MXNJ retries is reached, we reduce h and retry the iteration (again with a new A and the current y and y') in the case of Problem 1, or give up and return an error flag in the case of Problem 2. If convergence of the inner Newton iteration failed but RATE > 0.8 (or some other recoverable failure occurred), we retry the iteration with a reduced value of h and the initial y and y' (Problem 1) or give up (Problem 2). The total number of iterations performed can therefore be as large as MXNH*MXNJ*MXNIT in Problem 1, and MXNJ*MXNIT in Problem 2.

Currently we have set MXNH = 5 and EPINIT = 0.01. We have set MXNIT = 5, MXNJ = 6 in the case of direct methods, and MXNIT = 15, MXNJ = 2 in the case of Krylov methods. However, all four of these controls are optional inputs to DASPK, so that a user may specify different values. In addition, an option is provided to turn off the linesearch algorithm.

An additional outer layer of logic is required, because the error weights involved in all convergence tests depend on the current solution vector: $w_i = RTOL_i |y^i| + ATOL_i$. With weights set using the input y vector, the initialization

algorithm is called, and if it succeeds, we update the weights and call it a second time. If it again succeeds, we update the weights again, and proceed to the first time step. If either initialization fails, an error flag is returned to the user. In the case of the Krylov method, on the second initialization call, the preconditioner is not updated unless and until there is a convergence failure.

We have added another useful feature to our implementation of the initialization algorithm—the ability to enforce certain constraints on the components of Y. In addition to the option (previously included in DASSL and DASPK) of forcing all Y(I) ≥ 0, the revised version allows the user to specify that each Y(I) is to be either positive, nonnegative, nonpositive, or negative, or else is to be unconstrained, during the initialization. This option is activated by way of INFO(10), and the user must then load a second integer array (also part of IWORK) with the constraint specifications on each Y(I).

7. Numerical Experiments. We tested the initialization algorithm on several problems and found that it performed much as expected. In the course of development and debugging, we used a simple index-one system of size 2, having a known analytic solution. For both the first and second initialization problem types, and for a wide range of initial guesses, the initialization algorithm converged within the limits imposed, for both the direct and Krylov method options. All attempts to integrate the system without the initialization option failed except when the initial values were consistent.

For a more realistic test, we used a model of a multi-species food web [5], in which mutual competition and/or predator-prey relationships in a spatial domain are simulated. Here we consider a 2-species model, species 1 being the prey and species 2 being the predator, and with the predator assumed to have an infinitely fast reaction rate. Specifically, the model equations for the concentration vector $c = (c^1, c^2)^T$ are:

$$\left\{ \begin{array}{l} \dfrac{\partial c^1}{\partial t} = f_1(x,y,t,c) + d_1(c^1_{xx} + c^1_{yy}) \\ \qquad 0 = f_2(x,y,t,c) + d_2(c^2_{xx} + c^2_{yy}) \end{array} \right. \tag{7.1}$$

with

$$f_i(x,y,t,c) = c^i(b_i + \sum_{j=1}^{2} a_{ij}\, c^j) \ . \tag{7.2}$$

The interaction and diffusion coefficients (a_{ij}, b_i, d_i) could be functions of (x,y,t) in general. The choices made for this test problem are as follows:

$$A = (a_{ij}) = \begin{pmatrix} -1 & -0.5 \cdot 10^{-6} \\ 10^4 & -1 \end{pmatrix} \ , \tag{7.3}$$

$$b_1 = 1 + \alpha xy + \beta \sin(4\pi x) \sin(4\pi y) = -b_2 \ , \tag{7.4}$$

and

$$d_1 = 1 \ , \quad d_2 = .05 \ . \tag{7.5}$$

The domain is the unit square $0 \leq x, y \leq 1$, and $0 \leq t \leq 10$. The boundary conditions are of homogeneous Neumann type (zero normal derivatives) everywhere. The coefficients are such that a unique stable equilibrium is guaranteed to exist when $\alpha = \beta = 0$ and time derivatives appear in the equations for species 2 [5]. Empirically, a stable equilibrium appears to exist for (7.1) when α and β are positive, although it may not be unique. In our tests on this problem we take $\alpha = 50$ and $\beta = 100$, for which there is considerable spatial variation in the solution.

The PDE system (7.1), together with the boundary conditions, was discretized with central differencing on an $L \times L$ mesh, as described in [9]. We have taken $L = 20$, which is quite sufficient for accurate spatial resolution. The resulting DAE system $G(t, Y, Y') = 0$ has size $N = 2L^2 = 800$. The tolerances used were RTOL = ATOL = 10^{-5}. All tests were run on a Sun Sparc-10 workstation.

7.1. Initialization Problem 1. In the tests on this problem reported in [9], the initial conditions were taken to be mildly peaked functions that nearly satisfy the constraint equations:

$$\left\{ \begin{array}{l} c^1 = 10 + [16x(1-x)y(1-y)]^2 \\ c^2 = -(b_2 + a_{21}c^1)/a_{22}. \end{array} \right. \tag{7.6}$$

The predator value c^2 above, determined by the equation $f_2(x, y, 0, c) = 0$, is an approximate quasi-steady state (QSS) value. The original DASPK solver has no difficulty with this problem, without further adjustment of the initial values. However, we expect that in a typical application of this type it is impractical to find such accurate initial values. So for our tests, we will prescribe a flat value

$$c^2 = c_0^{pred}$$

as the initial guess in the input Y array, and invoke the new algorithm for Initialization Problem 1. For the present problem parameters, the QSS values of c^2 at time $t = 0$ are all within 10% of 10^5 ($= -a_{21} \cdot 10/a_{22}$), and so we vary c_0^{pred} about 10^5.

We will report here only tests with the Krylov method (GMRES) option in DASPK, and as a preconditioner we use a product of a spatially-based factor and a reaction-based factor. In the notation of [9], this is given by

$$P_{SR} \equiv (I - c^{-1}S_Y)(c\bar{I}_1 - R_Y) \ . \tag{7.7}$$

Here R and S are (respectively) the reaction and diffusion terms of the right-hand side of the DAE system, so that the problem has the form $G(t, Y, Y') = \bar{I}_1 Y' - S - R$ ($\bar{I}_1$ is the identity matrix with 0 in place of 1 in positions corresponding to the components c^2). The spatial factor in P_{SR} consists of 5 Gauss-Seidel iterations, and the reaction factor uses difference quotient approximations for the diagonal blocks. For the DASPK input parameters relating to the Krylov method, default values were specified.[3]

In Table 7.1 below, we summarize the results of the DASPK tests with the new initialization algorithm incorporated in it. For each value of c_0^{pred} (with QSS denoting the values in (7.6)), the tabulated quantities are:

NNI0	=	number of Newton iterations in the initial condition calculation
NLI0	=	number of linear iterations in the initial condition calculation
NNI	=	total number of Newton iterations to complete the integration
NLI	=	total number of linear iterations to complete the integration
NRE	=	total number of residual evaluations to complete the integration.

The numbers NNI0 and NLI0 measure the cost of the initialization algorithm, while NNI, NLI, and NRE measure the total cost of solving the problem. Convergence (to correct values) in the initialization was achieved at a very reasonable additional cost for $.6 \cdot 10^5 \le c_0^{pred} \le 10^7$. Evidently, the convergence region for the initialization of this problem is strongly skewed to the high side, but does permit errors of at least 40% on the low side. In the case $c_0^{pred} = 10^4$, the algorithm converged, but to the value $c^2 = 0$, which corresponds to a solution that is valid but different from the one of interest here.

For comparison, when the initial condition calculation option was *not* selected, only the QSS initial values were successful, and in that case the total cost figures were NNI = 338, NLI = 371, NRE = 709. These are slightly larger than with the initialization, indicating that even the approximate QSS values from (7.6) are somewhat in error. The

[3] On the basis of experience with these tests, however, we have changed the default value of NRMAX, the maximum number of GMRES restarts, from 2 to 5.

c_0^{pred}	NNI0	NLI0	NNI	NLI	NRE	Notes
QSS	1	1	336	361	699	
10^4	5	12	-	-	-	predator $\to 0$
$.5 \cdot 10^5$	-	-	-	-	-	failed in I.C. calculation
$.6 \cdot 10^5$	5	46	341	417	762	
$.7 \cdot 10^5$	4	23	370	652	1025	
$.8 \cdot 10^5$	5	24	368	626	996	
$.9 \cdot 10^5$	4	17	367	618	987	
10^5	3	10	338	384	724	
10^6	8	22	342	383	727	
10^7	11	25	375	629	1006	

TABLE 7.1

Test results for new initialization algorithm on food web problem

unmodified DASPK solver, when run with its initial condition option on, was also unable to solve any case except the QSS initial values, and in that case the total costs were NNI = 366, NLI = 605, NRE = 971. The failed cases either halted in the initialization algorithm, or (when the initialization option was off) failed in the first time step with either repeated corrector convergence failures or repeated error test failures.

7.2. Initialization Problem 2. In these tests, we specify the initial time derivatives y' to be 0, i.e., we are posing the steady-state problem for (7.1). Since we have no explicit time-dependence in the right-hand sides, once the consistent initial values are determined, the solution to the DAE problem is constant in time.

By comparison with Initialization Problem 1, the absence of the time derivative operator changes the basic nature of this type of problem considerably. The possibility of a singular Jacobian for the right-hand side function or multiple steady state solutions, neither of which causes much difficulty in a pure time integration problem, can cause considerable numerical difficulty in the time-independent problem. Moreover, the preconditioners devised in [9] for the DAE problem itself are much less useful here.

We first describe tests using the direct method. These specified a banded Jacobian, generated internally by difference quotients, where the two half-bandwidths are equal to $2L = 40$. For simplicity, the initial guesses for the discrete c^i values were taken to be spatially flat values with

$$c^1 = c_0^{prey} \quad \text{(given)} \quad , \quad c^2 = 10^4 c^1 \ . \tag{7.8}$$

Because the subsequent time integration is not an issue here, we stopped it at $t = 10^{-8}$. We performed tests for a variety of values of α and β, revealing, as in the case of Problem 1, a nontrivial region of convergence in each case. Table 7.2 below (upper half) gives the results for the case $\alpha = 50, \beta = 100$, where the tabulated counter NNI0 is defined as before. Convergence is achieved with no difficulty (always using the full Newton step) for (at least) the values $45 \le c_0^{prey} \le 100$. In an interval about $c_0^{prey} = 40$ and in the interval $c_0^{prey} \le 10$, the algorithm fails to find a solution. In an interval about $c_0^{prey} = 20$, it converges to an incorrect solution (that has negative values of c^i). In an interval about $c_0^{prey} = 30$, it converges to the correct solution, but with difficulty, in that the linesearch algorithm must choose values of $\lambda < 1$. For reference, we note that the true steady state values of c^1 in this case range from 9.9 to 66.

For this 2-D problem, using a Jacobian with the full bandwidth is quite costly. In an attempt to reduce costs, we also tested with half-bandwidths equal to 1, corresponding to an approximate Jacobian that ignores the diffusion terms. However, the results were completely unsuccessful. The resulting lumped tridiagonal approximate Jacobian is evidently too inaccurate.

In considering tests with the Krylov method for this problem, the choice of a preconditioner is problematical. In terms of the form $G = \bar{I}_1 Y' - S - R$, the true Jacobian for the steady state problem is $J = -S_Y - R_Y$. Since

the initialization algorithm sets $c = 0$ in any user-supplied preconditioner, the choice P_{SR} of (7.7) used for Problem 1 is undefined. We therefore use $P_R = -R_Y$, a block-diagonal matrix involving only the reaction Jacobian elements. We again tried a variety of values of α and β, and provided flat initial guesses (7.8). However, for the larger values of these parameters, it was found that convergence of the GMRES iteration was much slower than in the case of Initialization Problem 1. This is to be expected, since the diffusion terms contribute significantly to the system but are completely absent in the preconditioner. In order to achieve convergence, we therefore increased the Krylov subspace parameters over their default values, setting the maximum size of the Krylov subspace (MAXL) to 20, and the number of GMRES restarts allowed (NRMAX) to 19. This allows a total of 400 GMRES iterations on each linear system. For values of $c_0^{prey} \le 50$, the algorithm appears to fail, while for values of $60 \le c_0^{prey} \le 100$ (at least), it converges to the correct solution.

c_0^{prey}	linear method	NNI0	NLI0	Notes
10	direct	-	-	fails in I.C. calculation
20	direct	11	-	incorrect solution
30	direct	11	-	linesearch min $\lambda = 1/4$
40	direct	-	-	fails in I.C. calculation
50	direct	11	-	
60	direct	11	-	
80	direct	12	-	
50	Krylov	-	-	fails in I.C. calculation
60	Krylov	6	1349	
70	Krylov	6	708	
80	Krylov	6	551	
90	Krylov	6	457	
100	Krylov	6	444	

TABLE 7.2
Test results for food web problem, Initialization Problem 2

In all of the cases tabulated, we compared the computed solution vector from the initialization algorithm with that from a more accurate solution with the direct method, integrated with tighter tolerances to $t = 100$, where it is virtually at steady state. All of the values from the Problem 2 tests had errors less than the tolerances imposed. For example, for $\alpha = 50, \beta = 100$, $c_0^{prey} = 70$, the maximum relative error observed was about $.49 \cdot 10^{-5}$.

Users of DASPK should be cautioned that Initialization Problem 2 is potentially more difficult than Initialization Problem 1, and that some extra effort may be necessary. In contrast with Problem 1 and with the time integration, the tests above indicate that a singular Jacobian or multiple solutions can occur, making convergence of the algorithm for Problem 2 more sensitive to the initial guess and to the quality of the approximate Jacobian J or preconditioner P. If the J or P used in the time steps is a good approximation only in the limit $c \to \infty$ ($h \to 0$), as was the case in the Problem 1 food web tests, a different preconditioner for the steady-state initialization problem (where $c = 0$) should be seriously considered. The user can easily determine within JAC and PSOL whether the preconditioner has been called for a steady-state initial condition calculation, because the parameter CJ will be equal to zero (it is nonzero in any other situation), and can then branch accordingly to the appropriate preconditioner.

7.3. Other preconditioners and general-purpose modules. This food web model has also been tested with a preconditioner based on sparse incomplete LU factorization. The results compare quite well with those reported above for the product preconditioner. With ILU preconditioning, the cost of each preconditioner evaluation is generally higher, but the number of linear and nonlinear iterations is lower.

Because this preconditioner is applicable quite generally, we have written a separate module that incorporates it. Likewise, we have written a banded preconditioner module for general use, and a module of tools for product preconditioners of the type described above for reaction-transport systems. These preconditioner modules are being distributed along with the DASPK solver.

REFERENCES

[1] R. Bachmann, L. Brüll, U. Pallaske and Th. Mrziglod, *On Methods for Reducing the Index of Differential Algebraic Equations*, Comput. Chem. Eng. 14 (1990), 1271-1273.

[2] P. I. Barton and C. C. Pantelides, *Modeling of Combined Discrete/Continuous Processes*, AIChE J. 40 (1994), 966-979.

[3] M. Berzins, P. M. Dew, and R. M. Furzeland, *Developing Software for Time Dependent Problems using the Method of Lines and Differential-Algebraic Integrators*, Appl. Numer. Math., (1988).

[4] K. Brenan, S. Campbell and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, second edition, SIAM, 1995.

[5] P. N. Brown, *Decay to Uniform States in Food Webs*, SIAM J. Appl. Math., 46 (1986), 376-392.

[6] P. N. Brown, *A Local Convergence Theory for Combined Inexact-Newton/Finite-Difference Projection Methods*, SIAM J. Numer. Anal., 24 (1987), 407–434.

[7] P. N. Brown and Y. Saad, *Hybrid Krylov Methods for Nonlinear Systems of Equations*, SIAM J. Sci. Comp., 11 (1990), 450–481.

[8] P. N. Brown and Y. Saad, *Convergence Theory of Nonlinear Newton-Krylov Algorithms*, SIAM J. Optimization., 4 (1994), 297–330.

[9] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, *Using Krylov Methods in the Solution of Large-Scale Differential-Algebraic Systems*, SIAM J. Sci. Comp., 15 (1994), 1467-1488.

[10] S. L. Campbell, *Consistent Initial Conditions for Linear Time Varying Singular Systems*, in Frequency Domain and State Space Methods for Linear Systems, Elsevier Science (North-Holland), Amsterdam, 1986.

[11] S. L. Campbell, *A Computational Method for General Higher-Index Nonlinear Singular Systems of Differential Equations*, Proc. 1988 IMACS Conference.

[12] R. S. Dembo, S. C. Eisenstat and T. Steihaug, *Inexact Newton Methods*, SIAM J. Numer. Anal., 19 (1982), 400-408.

[13] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[14] A. Kröner, W. Marquardt and E. D. Gilles, *Computing Consistent Initial Conditions for Differential-Algebraic Equations*, Comput. Chem. Eng. 16 Suppl. (1992), 131-138.

[15] R. Lamour, *A Shooting Method for Fully-Implicit Index-2 Differential-Algebraic Equations*, preprint, Humboldt-Universitat zu Berlin, 1994.

[16] B. Leimkuhler, L. R. Petzold and C. W. Gear, *Approximation Methods for the Consistent Initialization of Differential-Algebraic Equations*, SIAM J. Numer. Anal. 28 (1991), 205-226.

[17] C. Majer, W. Marquardt and E. D. Gilles, *Reinitialization of DAEs after Discontinuities*, Comput. Chem. Eng. 19 Suppl. (1995), 507-512.

[18] C. C. Pantelides, *The Consistent Initialization of Differential-Algebraic Systems*, SIAM J. Sci. Stat. Comp. 9 (1988), 213-232.

[19] C. C. Pantelides, *Speedup - Recent Advances in Process Simulation*, Comput. Chem. Eng. 12 (1988), 745-755.

[20] Y. Saad and M. H. Schultz, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM J. Sci. Stat. Comp. 7 (1986), 856-869.

[21] B. Simeon, C. Führer and P. Rentrop, *Differential-Algebraic Equations in Vehicle System Dynamics*, Surveys on Math. for Industry 1 (1991), 1-37.

8. Appendix: Structure and Organization of DASPK. The revised DASPK solver consists of 32 subprograms, excluding the required LINPACK and BLAS routines and user-supplied subroutines.

The following is a complete list of the subordinate routines in DDASPK, the double precision version of DASPK, with a brief description of each. For the single precision version, the names begin with S instead of D, and the single precision version of XERRWD is XERRWV.

Subordinate Routines in the DDASPK Package

DDASIC computes consistent initial conditions.
DYYPNW updates Y and YPRIME in linesearch for initial condition calculation.
DDSTP carries out one step of the integration.
DCNSTR/DCNST0 check the current solution for constrant violations.
DDAWTS sets error weight quantities.
DINVWT tests and inverts the error weights.
DDATRP performs interpolation to get an output solution.
DDWNRM computes the weighted root-mean-square norm of a vector.
DDASID driver to initialize Y and YPRIME using direct linear methods.
DNSID solves for initial values by modified Newton and direct linear methods.
DLINSD carries out linesearch algorithm for initial condition calculation (direct case).
DFNRMD computes WRMS norm of preconditioned residual in I.C. calculation (direct case).
DNEDD nonlinear equation driver for direct linear system methods.
DMATD assembles the iteration matrix (direct case).
DNSD solves a nonlinear system by modified Newton and direct linear methods.
DSLVD interfaces to linear system solver (direct case).
DDASIK driver to initialize Y and YPRIME using Krylov iterative linear methods.
DNSIK solves for initial values by Newton and Krylov linear methods.
DLINSK carries out linesearch algorithm for initial condition calculation (Krylov case).
DFNRMK computes WRMS norm of preconditioned residual in I.C. calculation (Krylov case).
DNEDK nonlinear equation driver for iterative linear system methods.
DNSK solves a nonlinear system by inexact Newton and Krylov linear methods.
DSLVK interfaces to linear system solver (Krylov case).
DSPIGM solves a linear system by SPIGMR algorithm.
DATV computes matrix-vector product in Krylov algorithm.
DORTH performs orthogonalization of Krylov basis vectors.
DHEQR performs QR factorization of Hessenberg matrix.
DHELS finds least-squares solution of Hessenberg linear system.
DGEFA, DGESL, DGBFA, DGBSL (from LINPACK) solve dense or banded linear systems.
DAXPY, DCOPY, DDOT, DNRM2, DSCAL are Basic Linear Algebra (BLAS) routines.
D1MACH provides the unit roundoff of the machine.
XERRWD handles error messages.

On the following pages are block diagrams showing the call paths within DDASPK, including the user-supplied routines (which appear repeatedly, for ease of presentation). Figure 1 is an overall diagram, while Figure 2 shows the initial condition calculation module.

DASPK Overall Block Diagram

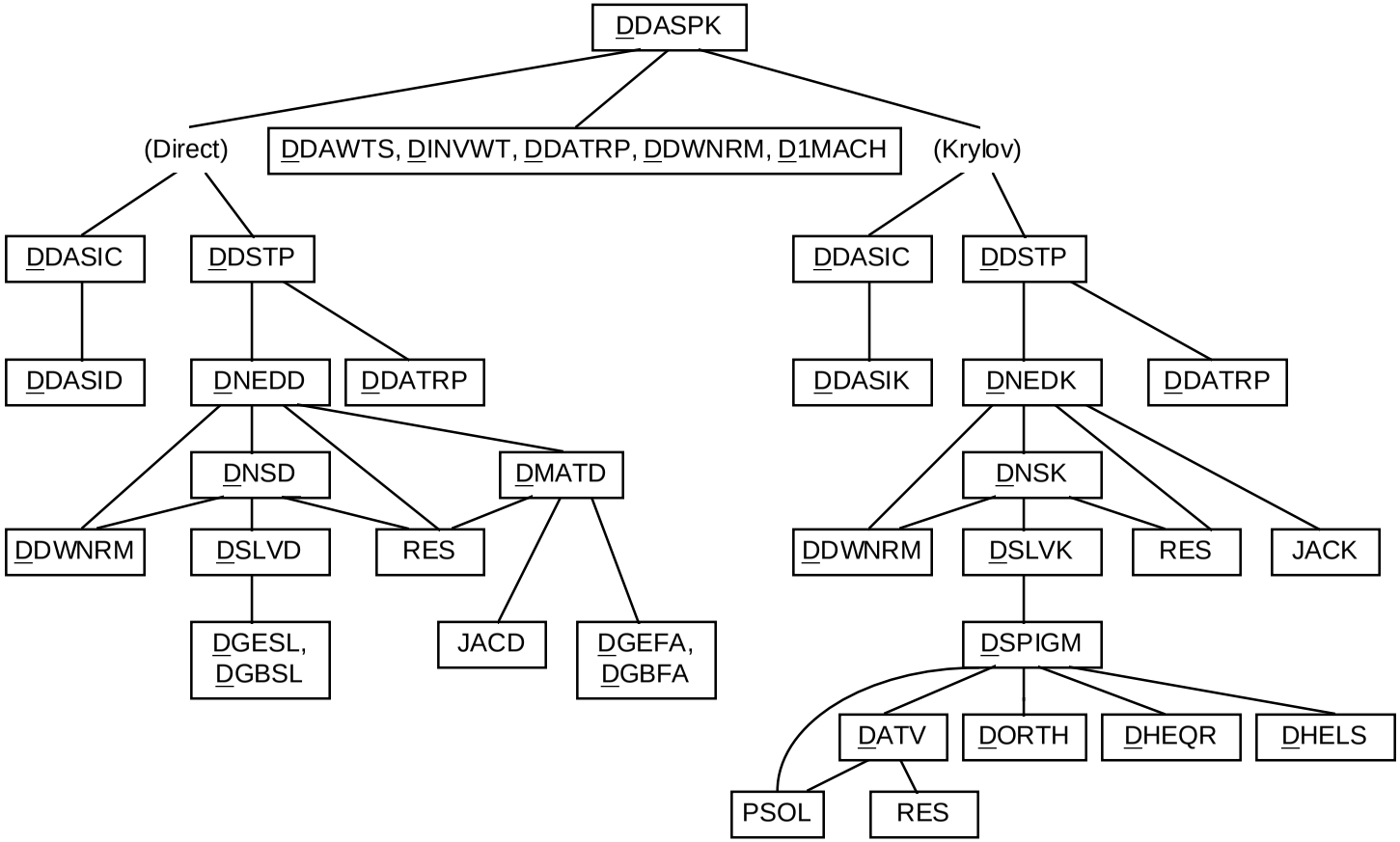

Fig. 1. Calls to BLAS and error message handler are not shown. Names with underlined D are double precision versions; the single precision version has S.

DDASIC Block Diagram

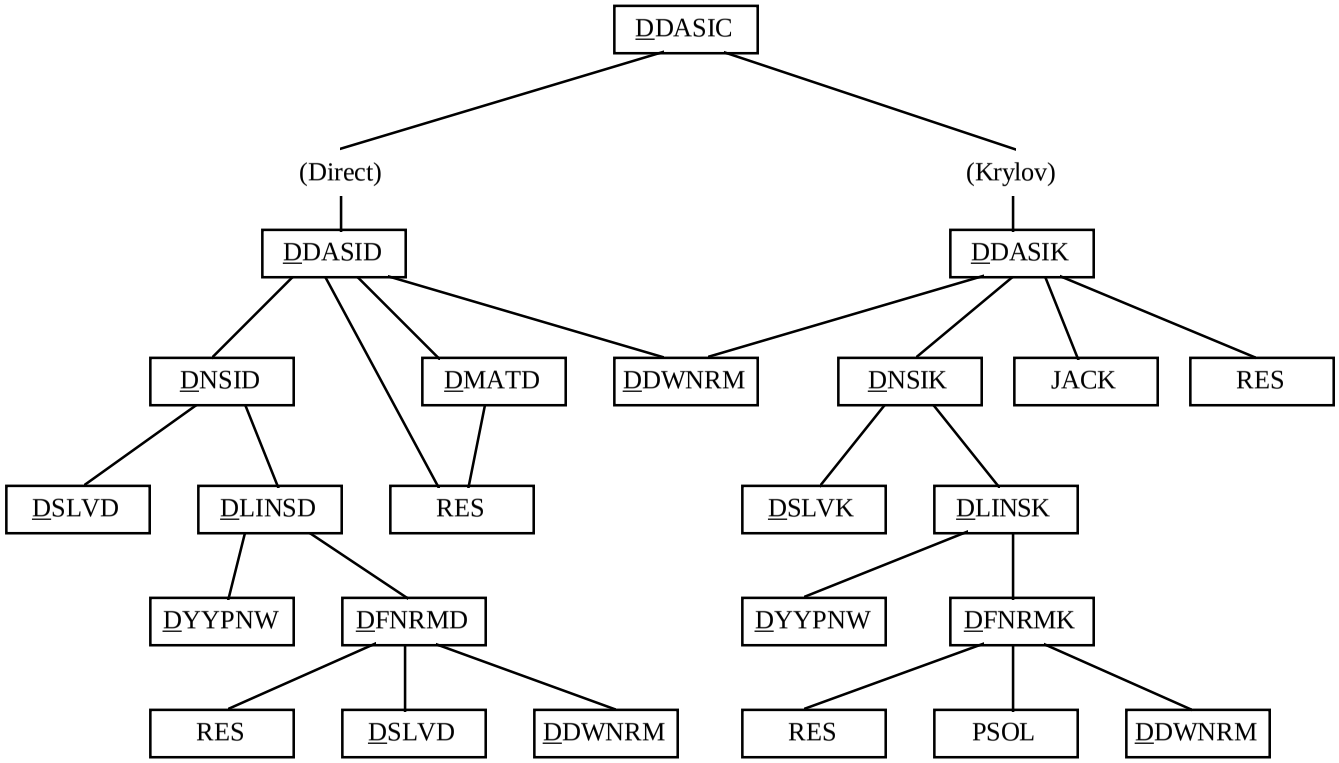

Fig. 2. Block diagram of initial condition calculation module in DASPK. Names with underlined initial D are double precision versions; the single precision version has an initial S. Some subordinate routine paths, given in Fig. 1, are not repeated here.