USING KRYLOV METHODS IN THE SOLUTION OF LARGE-SCALE DIFFERENTIAL-ALGEBRAIC SYSTEMS*

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

Abstract. In this paper we describe a new algorithm for the solution of large-scale systems of differential-algebraic equations. It is based in the integration methods in the solver DASSL, but instead of a direct method for the associated linear systems which arise at each time step, we apply the preconditioned GMRES iteration in combination with an Inexact Newton Method. The algorithm, along with those in DASSL, is implemented in a new solver called DASPK. We outline the algorithms and strategies used, and discuss the use of the solver. We develop and analyze some preconditioners for a certain class of DAE systems, and finally demonstrate the application of DASPK on two example problems.

1. Introduction.

This paper is concerned with the solution of large systems of differential-algebraic equations (DAEs). We write the system in the general form

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

where F, y, and y' are N-dimensional vectors, and a consistent set of initial conditions $y(t_0) = y_0$, $y'(t_0) = y'_0$ is given.[1] The starting point for this work is the solver DASSL [17, 3]. In that code, the linear systems which arise at each time step are solved with dense or banded direct linear system solvers. For large problems, this is highly restrictive. Instead we consider the preconditioned GMRES [18] iterative method. For large-scale systems including method of lines solution of partial differential equations in two and three dimensions, this method can be quite effective, combined with a suitable preconditioner.

A number of previous papers [14, 10, 15, 6] have dealt with the solution of large-scale systems of *ODEs*, $y' = f(t, y)$, via backward differentiation formulas for time stepping in combination with preconditioned Krylov methods for solving the linear systems at each time step. A code, LSODPK, based on these methods was developed by Brown and Hindmarsh, and is described in [6]. We are not aware of any previous work along these lines which is directed at DAEs.

In this paper we extend the work on preconditioned Krylov methods for ODEs to DAEs. While many of the considerations remain the same, the solution of DAEs by this approach introduces some additional questions and difficulties. In particular, a preconditioner is *always* needed for DAEs. Preconditioners can be developed for classes of DAE systems arising with a particular structure. Here we develop and analyze preconditioners for a certain class of stiff DAE systems.

A new code, DASPK, has been developed based on this approach. We begin in Section 2 by describing the code in some detail, outlining especially those strategies and considerations which differ from the ODE case. In Section 3 we describe the use of this code. In Section 4 we develop and analyze a class of preconditioners for DAEs arising from reaction-diffusion systems. Finally, in Section 5 we present some numerical experiments applying the new code and preconditioners to the solution of several large-scale problems.

* 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.

† Computing & Mathematics Research Division, L-316, Lawrence Livermore National Laboratory, Livermore, California 94550.

‡ Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. The work of this author was partially supported by ARO, contract number DAAL03-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center, and by ARO contract number DAAL03-92-G-0247 and DOE contract number DE-FG02-92ER25130.

[1] By a consistent set of initial conditions, for the systems under consideration we mean that (1.1) should be satisfied at the initial time. For a more complete description of what it means for an initial condition to be consistent, see [3].

2. Overview of the DASPK Algorithm. In DASPK, we have combined the time-stepping methods of DASSL with the preconditioned iterative method GMRES, for solving large-scale systems of DAEs of the form (1.1). Here we describe the algorithm.

2.1. Time-stepping. The underlying idea for solving DAE systems is due to Gear [13] and consists of replacing the solution and derivative in (1.1) by difference approximation, and solving 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

$$F\left(t_n, y_n, \frac{y_n - y_{n-1}}{h_n}\right) = 0 \tag{2.1}$$

where $h_n = t_n - t_{n-1}$. This equation is then solved at each time step using a modified Newton method,

$$y_n^{m+1} = y_n^m - \left(\frac{1}{h_n}\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}\right)^{-1} F\left(t_n, y_n^m, \frac{y_n^m - y_{n-1}}{h_n}\right) \tag{2.2}$$

where m is the iteration index. As in DASSL, DASPK uses the backward differentiation formulas (BDF) of orders 1 through 5 to approximate the derivative in (1.1). On every step, it chooses the order and stepsize based on the behavior of the solution. The integration methods and strategies for time-stepping are virtually identical to those in DASSL, and are described in detail in [3]. The equation to be solved on each time step is

$$F\left(t_n, y_n, \frac{\rho y_n}{h_n}\right) = 0 \tag{2.3}$$

where $\rho y_n = \sum_{i=0}^{k} \alpha_i y_{n-i}$ and α_i, $i = 0, 1, \ldots, k$ are the coefficients of the BDF method.

2.2. Nonlinear system solution. It is important to solve the nonlinear equation (2.3) efficiently. To simplify notation, we can rewrite this equation as

$$F(t, y, \alpha y + \beta) = 0, \tag{2.4}$$

where $\alpha = \alpha_0/h_n$ is a constant which changes whenever the stepsize or order changes, β is a vector which depends on the solution at past times, and t, y, α, β are evaluated at t_n. To simplify the discussion, we will sometimes refer to the above function as $F(y)$. Both DASPK and DASSL solve this equation by a modified version of Newton's method,

$$y^{m+1} = y^m - c\left(\alpha\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}\right)^{-1} F(t, y^m, \alpha y^m + \beta). \tag{2.5}$$

In the case of direct methods with DASPK, which is virtually identical to DASSL, the iteration matrix

$$A = \alpha\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y} \tag{2.6}$$

is computed and factored, and is then used for as many time steps as possible. By contrast, in the iterative methods option, a preconditioner matrix P, which is an approximation to A that leads to a cheap linear system solution, is computed and factored and used for as many time steps as possible. As we will see in the examples below, it is often possible to use a preconditioner over more steps than it would be possible to keep an iteration matrix in the direct option, because the iterative methods do the rest of the work in solving the system. One of the powerful features of the iterative approach is that it does not need to compute and store the iteration matrix A explicitly.[2] This is because the GMRES method, as we will see below, never actually needs this matrix explicitly. Instead, it requires only the action of A times a vector v. In DASPK, this matrix-vector product is approximated via a difference quotient on the function F in (2.4)

$$Av = F'(y)v \approx \frac{F(t, y + \sigma v, \alpha(y + \sigma v) + \beta) - F(t, y, \alpha y + \beta)}{\sigma}. \tag{2.7}$$

The GMRES algorithm requires products Av in which v is a vector of unit length (the norm is a weighted norm based on the user-defined error tolerances as described in Section 2.3.3) and y is the current iterate. In DASPK, σ is taken to be 1, as explained in Section 2.3.3. We note that, because y is current in (2.7), this amounts to taking a full Newton iteration in the iterative option of DASPK (rather than modified Newton, as in DASSL and the direct option of DASPK). In fact, for some highly nonlinear problems we have seen the iterative option in DASPK outperform the direct option in terms of time steps, corrector failures, etc., apparently for this reason.

In general, the value of α when A (or P) was last computed is different from the current value of α. We will denote the 'old' value of α by $\hat{\alpha}$. If α is too different from $\hat{\alpha}$ in the direct method, then (2.5) may not converge. The constant c in (2.5) is chosen to speed up the convergence in the direct method when $\alpha \neq \hat{\alpha}$, and is given by

$$c = \frac{2}{1 + \alpha/\hat{\alpha}} \tag{2.8}$$

For the iterative method, while the preconditioner may be based on an old $\hat{\alpha}$, the linear system to be solved is based on the current α, so in this case $c \equiv 1$.

The rate of convergence ρ of (2.5) is estimated whenever two or more iterations have been taken by

$$\rho = \left(\frac{||y^{m+1} - y^m||}{||y^1 - y^0||} \right)^{1/m} \tag{2.9}$$

(The norms are scaled norms which depend on the error tolerances specified by the user.) The iteration is taken to have converged when

$$\frac{\rho}{1-\rho} ||y^{m+1} - y^m|| < 0.33 \tag{2.10}$$

The basis for this test is that if the iteration is converging linearly at a rate ρ to y^*, then

$$\frac{\rho}{1-\rho} ||y^{m+1} - y^m|| \approx ||y^{m+1} - y^*||. \tag{2.11}$$

If $\rho > 0.9$ or $m > 4$, and the iteration has not yet converged, then the stepsize is reduced, and/or an iteration matrix based on current approximations to y, y', and α is formed, and the step is attempted again. If the difference between the predictor and the first correction is very small (for the direct solver, this is relative to roundoff error in y; for the iterative solver, it is relative to the accuracy requested in solving the linear system), the iteration is taken to have converged (because the initial correction is so close that it is impossible to get a good rate estimate).

[2] Depending on the preconditioner, it may need to compute and store a preconditioner matrix explicitly. However, this matrix is hopefully much cheaper to generate and to store than the actual iteration matrix.

For the iterative methods, convergence tests such as (2.10) need to be justified, because the Newton iterates are not computed exactly but instead with a relatively large error which is due to solving the linear system inexactly. The test (2.10) can be justified, at least to some extent, by considering the Newton/GMRES method in the framework of the theory of Inexact Newton Methods [11]. In this framework, the Newton iteration for $F(y) = 0$, including errors r^m due to solving the linear system inexactly, is written as

$$F'(y^m)\delta y^m = -F(y^m) + r^m \tag{2.12a}$$
$$y^{m+1} = y^m + \delta y^m \tag{2.12b}$$

Theorem 2.2 in Brown and Hindmarsh [5] justifies a convergence test based on rate of convergence for the inexact Newton iteration, provided the residuals r^m satisfy $||r^m|| \leq \eta ||F(y^m)||$, for $\eta < 1$. However, as in the case of ODEs, we prefer to terminate the linear iteration based on a condition like $||r^m|| \leq \delta$. In this case, it can be argued heuristically as in [5] that the test is still justified, provided that $\delta << \epsilon/\lambda$, where ϵ is the tolerance for the final computed Newton iterate, $||y^{m+1} - y^*|| \leq \epsilon$, and $\lambda = ||(F')^{-1}(y^*)||$. Now, this is almost what we need, except that for most DAEs (and stiff ODEs), λ is likely to be quite large, which would seem to mandate a quite conservative test for the linear iteration. To see how the theory can be used to justify a less conservative test, multiply $F(y) = 0$, and hence (2.12a), by P^{-1}, where P is the preconditioner matrix described below. This changes nothing in terms of the Newton iterates or the final solution for y^*. Now, *assuming that the preconditioner is a good approximation to F'*, in the sense that $||P^{-1}F'|| \approx O(1)$, the theory and heuristic arguments[5], applied to $P^{-1}F$ instead of F, justify a rate of convergence based termination criteria for the Newton iteration, provided the *preconditioned* residuals for the linear iteration satisfy $||P^{-1}r^m|| \leq \delta$, where $\delta << \epsilon$. In DASPK, we take $\epsilon = .33$, and take $\delta = .05\epsilon$ as the default tolerance for solving the linear iteration (the constant δ/ϵ can be adjusted optionally by the user as described in Section 3). The norms used are weighted norms based on the user-defined error tolerances and are described in Section 2.3.2. We note that another desirable property of these tests is that they are invariant under scalings of the DAE (i.e. multiplying F on the left by some arbitrary matrix), provided the preconditioner has also been scaled accordingly.

In contrast to the ODE solvers LSODPK[6] and VODPK[8], which use preconditioned Krylov methods with left and/or right preconditioning, the DASPK solver allows only left preconditioning. The reason has to do with a basic difference between ODEs and DAEs. For a DAE system defined by $F(t, y, y') = 0$, the components of the vector F need not have any relation to those of y. For example, the two vectors need not have the same physical units in corresponding components. If a left preconditioner P_1 and a right preconditioner P_2 are allowed in the solution of the linear system $Ax = b$, where $A = F'$ and b is a value of $-F$, then the Krylov method in effect deals with the matrix $P_1^{-1}AP_2^{-1}$ and with residual vectors $P_1^{-1}r$ $(r = b - Ax)$, and performs a convergence test on weighted norms of those vectors. But consistent choices of P_1 and P_2 are possible, with $P_1P_2 \approx A$, for which $P_1^{-1}r$ does not have the same units as y. Then norms of the preconditioned residuals $P_1^{-1}r$ are meaningless as a measure of the quality of the current approximate solution vector x. In contrast, if $P_2 = I$ and P_1 is a consistent approximation to A, then $P_1^{-1}r$ has the same units as y in each component, and the convergence test in the Krylov algorithm, with the same weighted norm as used in the local error test, is completely consistent. Moreover, that convergence test is invariant under a change of scale in either the function F or the vector y (provided the absolute tolerances are rescaled consistently if y is). This consistency and scale invariance are not possible with preconditioning either on the right only or on both sides.

In DASPK, the iterative option *requires* the user to provide a preconditioner P. This is in part because the Newton iteration test, and hence ultimately the code reliability, is not justified without a halfway-reasonable preconditioner. It is also because *any nontrivial DAE needs a preconditioner.* Even a 'nonstiff' DAE needs a preconditioner, to approximate the Jacobian of the constraint matrix. Also, it is known that the iteration matrix for any nontrivial DAE becomes more and more ill-conditioned as the stepsize is reduced [3]. Therefore, the preconditioner may need to rescale; scalings for some common classes of DAEs are discussed in [3]. In the case that the DAE is really an ODE, with $F(t,y,y') = y' - f(t,y)$, the preconditioner could be taken as $P \approx \alpha * I$, although even for ODEs it is often better to provide a nontrivial preconditioner [6].

2.3. Linear system solution. Solving (2.5) requires the solution of a linear system

$$Ax = b \tag{2.13}$$

at each Newton iteration, where A is the $N \times N$ iteration matrix in (2.9), $x = y^{m+1} - y^m$ is an N-vector, and $b = -cF(t, y^m, \alpha y^m + \beta)$ is an N-vector.

In the direct option, this linear system is solved by either dense direct or banded direct Gaussian elimination with partial pivoting via LINPACK [12]. The iteration matrix is either provided by the user, or computed via finite difference quotients, as described in [3].

2.3.1. Description of GMRES algorithm. In the case of iterative methods, the linear system (2.13) is solved by the preconditioned GMRES iterative method [18]. Depending on the options chosen, the method may be either the complete or the incomplete GMRES method. It may be restarted.

GMRES is one of a class of *Krylov subspace projection methods* [19]. The basic idea of these methods is as follows. If x_0 is an initial guess for the solution, then letting $x = x_0 + z$, we get the equivalent system $Az = r_0$, where $r_0 = b - Ax_0$ is the initial residual. We choose $z = z_l$ in the *Krylov subspace* $K_l = \mathrm{span}\{r_0, Ar_0, \cdots, A^{l-1}r_0\}$. For the GMRES algorithm, z_l, hence $x_l = x_0 + z_l$ is specified uniquely by the condition

$$\|b - Ax_l\|_2 = \min_{x \in x_0 + K_l} \|b - Ax\|_2 \quad (= \min_{z \in K_l} \|r_0 - Az\|_2).$$

Here, $\|\cdot\|_2$ denotes the Euclidean norm.

GMRES uses the *Arnoldi process* [1] to construct an orthonormal basis of the Krylov subspace K_l. This results in an $N \times l$ matrix $V_l = [v_1, \cdots, v_l]$ and an $l \times l$ upper Hessenberg matrix H_l such that

$$H_l = V_l^T A V_l \text{ and } V_l^T V_l = I_l \quad (= l \times l \text{ identity matrix}).$$

If the vectors $r_0, Ar_0, \cdots, A^l r_0$ are linearly independent, so that the dimension of K_{l+1} is $l+1$, then the matrices $V_{l+1} = [v_1, \cdots, v_{l+1}]$ and $\bar{H}_l \in \mathbf{R}^{(l+1)\times l}$ defined by

$$\bar{H}_l = \begin{bmatrix} H_l \\ r^T \end{bmatrix}, \text{ where } r = (0, \cdots, 0, h_{l+1,l})^T \in \mathbf{R}^l$$

satisfy

$$AV_l = V_{l+1}\bar{H}_l.$$

Furthermore, letting $z = V_l y$, we find that $\|r_0 - Az\|_2 = \|\beta e_1 - \bar{H}_l y\|_2$, where $\beta = \|r_0\|_2$ and e_1 is the first standard unit vector in $\mathbf{R}^{l+1}$. The vector $y = y_l$ minimizing this residual is computed by performing a QR factorization of $\bar{H}_l$ using Givens rotations. Then the GMRES solution is $x_l = x_0 + V_l y_l$. As noted by Saad and Schultz [18], this QR factorization can be done progressively as each column appears, and one can compute the residual norm $\|b - Ax_l\|_2$ without computing x_l at each step. If the $\sin\theta$ elements of the Givens rotations are denoted s_j $(j = 1, \cdots, l)$, then one obtains

$$\|b - Ax_l\|_2 = \beta|s_1 \cdots s_l|. \tag{2.14}$$

The use of (2.14) leads to the following algorithm, in which l_{max} and δ are given parameters:

Algorithm 2.1 (GMRES)

```
1. Compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|r_0\|_2$ .
2. For $l = 1, \cdots, l_{max}$ do:
    (a) Form $Av_l$ and orthogonalize it against $v_1, \cdots, v_l$ via
            $w_{l+1} = Av_l - \sum_{i=1}^{l} h_{il}v_i, \quad h_{il} = (Av_l, v_i)$
            $h_{l+1,l} = \|w_{l+1}\|_2$
            $v_{l+1} = w_{l+1}/h_{l+1,l}$.
    (b) Update the QR factorization of $\bar{H}_l$ .
    (c) Use (2.14) to compute $\rho_l = \|r_0\|_2 \cdot |s_1 \cdots s_l| = \|b - Ax_l\|_2$ .
    (d) If $\rho_l \leq \delta$ , go to Step 3. Otherwise, go to (a).
3. Compute $x_l = x_0 + V_l y_l$, and stop.
```

In the above algorithm, if the test on ρ_l fails, and if $l = l_{max}$ iterations have been performed, then one has the option of either accepting the final approximation x_l or setting $x_0 = x_l$ and then going back to Step 1 of the algorithm. This last procedure has the effect of "restarting" the algorithm, and DASPK does such restarts when necessary to achieve convergence.

As l gets large, much work is required to make v_{l+1} orthogonal to all the previous vectors $v_1, \cdots, v_l$. One can propose an incomplete version of GMRES (denoted by IGMRES), which differs from Algorithm 2.1 only in that the sum in Step 2(a) begins at $i = i_0$ instead of at $i = 1$, where $i_0 = \max(1, l - p + 1)$. Details are given in [6].

Saad and Schultz [18] have given a convergence analysis of Algorithm 2.1 which shows that the GMRES iterates converge to the true solution of (2.13) in at most N iterations. We also note that Algorithm 2.1 may have breakdowns. If $w_{l+1} = 0$ in the Arnoldi process, then Saad and Schultz [18] have shown x_l is the exact solution of (2.13). This is also referred to as a happy breakdown. When $w_{l+1} \neq 0$, the matrix $\bar{H}_l$ has full column rank, and so the least squares problem for y_l can always be solved via the above QR factorization. (However, in some cases the approximation x_l may not be of much use. An example is given in [6] illustrating how GMRES can have a dramatic failure.)

2.3.2. Scaling and preconditioning.

Realistic DAE problems require the inclusion of scale factors, so that all vector norms become weighted norms in the problem variables. However, even the scaled iterative methods seem to be competitive only for a fairly narrow class of problems, namely ODEs characterized mainly by tight clustering in the spectrum of the system Jacobian. Thus for robustness, it is essential to enhance the methods further. As in other contexts involving linear systems, preconditioning of the linear iteration is a natural choice. In what follows, the use of scaling and preconditioning is reviewed.

Scaling

The user of DASPK must provide parameters that define error tolerances to be imposed on the computed solution. These are relative and absolute tolerances RTOL and ATOL such that the combination

$$w_i = RTOL_i \left|y^i_{n-1}\right| + ATOL_i$$

is applied as a scale factor for component y^i during the time step from t_{n-1} to t_n. Specifically, a weighted root-mean-square norm

$$\|x\|_{WRMS} = \left[N^{-1}\sum_1^N (x^i/w_i)^2\right]^{1/2}$$

is used on all error-like vectors. Thus if we define a diagonal matrix

$$D = \sqrt{N}\mathrm{diag}(w_1,\cdots,w_N),$$

we can relate this to an ℓ_2 norm:

$$\|x\|_{WRMS} = \|D^{-1}x\|_2.$$

Because D contains the tolerances, the local error test on a vector e of estimated local errors is simply $\|e\|_{WRMS} \leq 1$.

The linear system in (2.13) can be restated in scaled form in terms of $D^{-1}x = \tilde{x}$ and $D^{-1}b = \tilde{b}$. Likewise, the nonlinear system $F(y) = 0$ can be restated in a scaled form $\tilde{F}(\tilde{y}) = 0$.

We note that, while DASSL allows the user to replace the norm subroutine, DASPK does not allow this. This is because a scaled L_2 norm is needed in the implementation of the GMRES algorithm. However, DASPK does allow a user-replaceable subroutines to define the weights in the norm. (The default is to set the weights according to the tolerances RTOL and ATOL via EWT(I) = RTOL(I)*ABS(Y(I)) + ATOL(I)). We recommend that this be attempted only after careful thought and consideration.

Preconditioning

When a basic iteration fails to show acceptable convergence on a given problem, preconditioning is often beneficial, especially when the cause of the slow convergence can be identified with one or more parts of the problem which are (individually) easier to deal with than the whole problem. Generally, preconditioning in an iterative method for solving $Ax = b$ means applying the method instead to the equivalent system

$$(P^{-1}A)(x) = (P^{-1}b), \text{ or } \bar{A}x = \bar{b}, \tag{2.15}$$

where P is chosen in advance. The preconditioned problem is easier to solve than the original problem provided that (1) linear systems $Px = c$ can be solved economically, and (2) P is in some way close to A. Condition (1) is essential because carrying out the method on $\bar{A}x = \bar{b}$ clearly requires evaluating vectors of the form $P^{-1}c$, at the beginning of the iteration, during each iteration, and at the end. Condition (2) is less well-defined, but means simply that the convergence of the method for $\bar{A}x = \bar{b}$ should be much better than for $Ax = b$, because $\bar{A}$ is somehow close to the identity matrix (for which convergence is immediate).

It is essential that the scaling of the linear system (discussed above) be retained in the preconditioned methods. Since the scaling matrix D is based on the tolerance inputs to the ODE solver, D^{-1} can be thought of as removing the physical units from the components of x so that the components of $D^{-1}x$ can be considered dimensionless and mutually comparable. On the other hand, the matrix $A = \alpha\partial F/\partial y' + \partial F/\partial y$ is <u>not</u> similarly scaled, and so, because P is based on approximating A, the matrix

$$\bar{A} = P^{-1}A$$

is also not similarly (dimensionally) scaled. More precisely, it is easy to show that if the (i,j) elements of P each have the same physical dimension as that of A, i.e. the dimension of F_i/y_j, then the (i,j) element of $\bar{A}$ has the dimension of y_i/y_j. Similarly, for the vectors x and $\bar{b}$, the i^{th} component of each has the same physical dimension as that of y_i. It follows that the diagonal scaling D^{-1} should be applied to x and $\bar{b}$ in the same way that it was applied to x and b without preconditioning. Thus we change the system (2.15) again to the equivalent scaled preconditioned system

$$(D^{-1}\bar{A}D)(D^{-1}x) = (D^{-1}\bar{b}), \text{ or } \tilde{A}\tilde{x} = \tilde{b}. \tag{2.16}$$

Combining the two transformations, we have

$$\tilde{A} = D^{-1}P^{-1}AD,\ \tilde{x} = D^{-1}x,\ \tilde{b} = D^{-1}P^{-1}b. \tag{2.17}$$

2.3.3. Implementation details. In implementing the GMRES method in DASPK, many of the algorithmic issues that arise are the same as for the ODE case. We have carried over the treatment of these matters from LSODPK [6]. Below is a summary of those details.

- DASPK takes $x_0 = 0$, having no choice readily available that is clearly better.
- The scaling is incorporated in an explicit sense, storing vectors $\tilde{v}_i$ that arise in the method as it stands, rather than unscaled vectors $D\tilde{v}_i = v_i$.
- DASPK uses a difference quotient representation

$$Jv \approx [F(t, y + \sigma v, \alpha(y + \sigma v) + \beta) - F(t, y, \alpha y + \beta)]/\sigma.$$

- DASPK takes $\sigma = 1$ because $\|v\|_{WRMS} = 1$. Thus the perturbation vector v can be regarded as a small correction to y, since its WRMS norm ($= 1$) is a value that is accepted for local errors in y in the local error test.
- The modified Gram-Schmidt procedure is used for orthogonalizing basis vectors.
- DASPK handles breakdowns in the same same manner as in the basic algorithms.
- The convergence test constant δ used as a bound on the residuals $\|b - Ax_l\|_{WRMS}$ is taken to be $\delta = .05\epsilon$, where $\epsilon = .33$ is the tolerance on the nonlinear iteration in (2.10).

We can now state our algorithm for scaled preconditioned versions of the GMRES method. This is given for arbitrary x_0, for the sake of generality, and is denoted SPIGMR.

Scaled Preconditioned Incomplete GMRES (SPIGMR)

```
1. (a) $r_0 = b - Ax_0$; stop if $\|r_0\|_{WRMS} < \delta$.
   (b) $\tilde{r}_0 = D^{-1}P^{-1}r_0$, compute $\|\tilde{r}_0\|_2 = \|P^{-1}r_0\|_{WRMS}$, $\tilde{v}_1 = \tilde{r}_0/\|\tilde{r}_0\|_2$.
2. For $l = 1, 2, \cdots, l_{max}$, do:
   (a) Compute $\tilde{A}\tilde{v}_l = D^{-1}P^{-1}AD\tilde{v}_l$.
   (b) $\tilde{w}_{l+1} = \tilde{A}\tilde{v}_l - \sum_{i=i_0}^{l} \tilde{h}_{il}\tilde{v}_i$, where $i_0 = \max(1, l - p + 1)$ ,
       $\tilde{h}_{il} = (\tilde{A}\tilde{v}_l, \tilde{v}_i)$.
   (c) $\tilde{h}_{l+1,l} = \|\tilde{w}_{l+1}\|_2$, $\tilde{v}_{l+1} = \tilde{w}_{l+1}/\tilde{h}_{l+1,l}$.
   (d) Update QR factorization of $\tilde{H}_l = (\tilde{h}_{ij})$ (an $(l+1) \times l$ matrix).
   (e) Compute residual $\rho_l$ indirectly (by (2.14) in the complete case).
   (f) If $\rho_l < \delta$, go to Step 3; otherwise go to (a).
3. Compute $\|\tilde{r}_0\|_2 Q_l^T e_1 = (\bar{g}_l, g)^T$ , $\tilde{z} = \tilde{V}_l \bar{R}_l^{-1} \bar{g}_l$,  $x_l = x_0 + D\tilde{z}$.
```

3. Using DASPK. We have attempted to make DASPK as easy to use as possible, and also upward compatible with DASSL. However the use of iterative rather than direct methods requires more information from the user and a deeper understanding of both the application and the solution process, particularly in the choice and implementation of an effective preconditioner. The direct methods of DASSL are available as an option of DASPK, and the user may find them useful for the purposes of getting started or debugging by way of smaller test problems.

The actual names of the single and double precision Fortran versions of DASPK are SDASPK and DDASPK, respectively. The call sequence is described in detail in the initial source file prologue. But we summarize the main points below.

3.1. Getting started. To get started, DASPK 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.[3] Finding a consistent set of initial conditions for a given problem may not be trivial. There is an option in DASPK to compute the initial value of YPRIME, if the initial values of Y are known. For some problems, this method may require a good initial guess for YPRIME. In cases where not all the initial vector Y is known, a nonlinear equation solver for large-scale systems such as NKSOL [7] can be very effective. It is important that the error tolerances for such a solver be set to be quite stringent, especially in comparison to the tolerances specified in DASPK. Otherwise, DASPK may not be able to get past the initial step because of a difficulty in satisfying its error tolerances. We note that there has been some recent work addressing the specification of consistent initial conditions for general DAE problems [16], however at the time of this writing we are aware of no corresponding general purpose software.

3.2. Specifying the DAE. As in DASSL, the information about the function F in (1.1) is provided via 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. The call sequence of RES in the use of DASPK differs from that in using DASSL, in that it includes CJ, which is the scalar α of (2.4), for possible use in scaling.

3.3. Solution by direct methods. Solution to the linear system at each time step can be done with either the direct methods of DASSL or with the preconditioned GMRES method. For direct methods, specify INFO(12) = 0; the possiblities are the same as in DASSL. You can provide a subroutine JAC to evaluate the iteration matrix yourself, or else there is an option for DASPK to approximate the matrix via finite differences. The matrix needed is $(\mathrm{CJ})\partial F/\partial y' + \partial F/\partial y$, where CJ is a scalar ($= \alpha$) which is proportional to $1/h$ and is provided as input to JAC. In the direct case, the JAC routine must have the form

SUBROUTINE JAC (T, Y, YPRIME, PD, CJ, RPAR, IPAR)

The linear system is solved by either dense or banded Gaussian elimination by routines from LINPACK [12].

3.4. Solution by iterative methods. For the preconditioned GMRES method, specify INFO(12) = 1. You can specify some of the details in the linear system solution, such as MAXL (the number of iterations allowed), KMP (the number of vectors on which orthogonalization is done and EPLI (the convergence tolerance for the linear iteration). Defaults for these constants are: MAXL = MIN(5,NEQ), KMP = MAXL (this corresponds to complete GMRES iteration, rather than the incomplete form), and EPLI = 0.05. These defaults can be overridden by setting INFO(13) = 1 and following the instructions in the code documentation. Changing MAXL or KMP effects the amount of work storage required, as described in the code documentation.

[3] We note here that F is referred to as G in the actual code documentation.

With the GMRES method, the user must supply a subroutine PSOL which solves linear systems of the form $Px = b$ where P is the left preconditioner matrix. The subroutine PSOL has the form

```
SUBROUTINE PSOL( NEQ, T, Y, YPRIME, SAVR, WK, CJ,
     WGHT, WP, IWP, B, EPLIN, IER, RPAR, IPAR)
```

The right-hand side vector b is in the B array on input, and PSOL must return the solution vector x in B. The Y, YPRIME, and SAVR arrays contain the current values of Y, YPRIME, and the residual F, respectively. The preconditioner matrix P is an approximation to the iteration matrix $(\mathrm{CJ})\partial F/\partial y' + \partial F/\partial y$, where CJ is a scalar ($= \alpha$) which is proportional to $1/h$ and is provided as input to PSOL and to JAC.

For the purposes of DASPK there are two types of preconditioners, depending on whether information about the iteration matrix is saved from one iteration or timestep to the next. For example, in implementing an ILU preconditioner one would be likely to save information from one iteration/timestep to the next. To specify this type of preconditioner, set INFO(15) = 1; you will then need to supply a subroutine JAC. First, an approximate iteration matrix would be formed. Then, the ILU decomposition would be performed and stored. This work would all be done in the subroutine JAC. For the iterative methods option, the call to JAC has the form

```
SUBROUTINE JAC( RES, IRES, NEQ, T, Y, YPRIME, REWT,
     SAVR, WK, H, CJ, WP, IWP, IER, RPAR, IPAR)
```

The arrays WP and IWP are real and integer work arrays which you may use for communication between your JAC routine and your PSOL routine. For example, you might store the ILU factorization in WP. For this preconditioner, the subroutine PSOL solves the linear system by back-substitution using the saved matrix. Now, for our ILU example, since this type of preconditioner is relatively expensive, one would like to save it and use it over many iterations and even many time steps. The strategy that DASPK uses to decide when to re-evaluate the preconditioner matrix is the same as the strategy DASSL uses [3] to decide when to reevaluate its Jacobian, except that in addition a new preconditioner matrix is generated whenever there is a failure of the linear iteration.

The simpler type of preconditioner is one which does not make use of saved information. In this case, set INFO(15) = 0; then there is no need to supply a JAC routine. Examples of these types of preconditioners are diagonal scaling or matrix-free SSOR as described in [9].

One can get some idea about how well a given preconditioner is working by monitoring the following information: IWORK(16) contains the number of convergence test failures for the linear iteration, IWORK(20) contains the total number of linear iterations, IWORK(19) contains the total number of nonlinear iterations. One can compute the average dimension of the Krylov subspace by taking the ratio IWORK(20)/IWORK(19) of linear to nonlinear iterations. This gives an indication of how hard the iterative method is having to work to solve the part of the problem which is not being approximated well by the preconditioner. (Actually, this ratio can exceed the maximum dimension MAXL because of restarts; in fact up to 2*MAXL linear iterations are allowed per nonlinear iteration.)

3.5. Higher-index DAEs. As in DASSL, there is always the possibility in DASPK that the problem you have specified does not have a well-defined solution or is higher-index.

The index is a measure of the degree of singularity of the system, see [3]. Standard-form ODEs

$$y' = f(t, y)$$

are index-0, ODEs with constraints

$$\begin{aligned} y' &= f(x,y) \\ 0 &= g(x,y) \end{aligned}$$

are index-1 if $\partial g/\partial x$ is nonsingular. Problems of index higher than 1 can cause difficulties for numerical methods, and in particular for the stepsize and order selection mechanisms in DASSL. A higher-index problem in DASSL will usually cause failures of the error tests and Newton convergence test (this information is available in IWORK(14) and IWORK(15)), and we recommend printing it out routinely on any successful or unsuccessful termination from DASSL for diagnostic purposes). A surprising number of problems are higher-index; for example incompressible Navier-Stokes equations are index-2. It is possible to modify DASSL/DASPK to deal with higher-index systems, especially in the case of index-2. It is also possible to rewrite higher-index systems in lower-index forms which have the same analytical solution. For more information on this, see [3]. Either of these alternatives must be undertaken very carefully, because such modifications can sometimes effect the stability properties of the system or of the numerical method [2].

It is also possible to write problems in the form (1.1) for which there is no solution or no unique solution. If this is the case for your problem, what will probably happen is a termination with IDID = -8 (the matrix of partial derivatives is singular). In our experience, for large problems arising from the method of lines discretization of partial differential equations, this is usually due to an error in formulating the boundary conditions. In any case, if you get this message you should rethink your equations or else look for a bug.

4. Preconditioners for DAE Systems. The choice of preconditioner can be critical in the performance of DASPK, as in other settings. We discuss here an approach to forming preconditioners for problems that arise from the semi-discretization of certain partial differential equations, following work that was done in [6] for the ODE context. Specifically, consider a mixed system of reaction-diffusion equations in some spatial region, in which some of the species (some PDE variables) obey time-dependent (evolution) equations, while the rest obey time-independent (quasi-steady) equations. We assume that some finite difference spatial discretization has been performed on all the equations, and that the resulting DAE system has index 1.

Let M denote the number of spatial mesh points, p the number of evolutionary species, and q the number of quasi-steady species. For practical purposes, it is usually best to order the variables by mesh point and then by species index, i.e. $p+q$ variables at mesh point 1, followed by $p+q$ variables at mesh point 2, etc. However, for ease of presentation, consider the 'transpose' ordering—by species and then by mesh point. Thus if $u_i = (u_{i,1}, \ldots, u_{i,M})^T$ is the vector of species i variables at all mesh points, we take $y = (u_1, \ldots, u_p, u_{p+1}, \ldots, u_{p+q})^T$ as the DASPK vector. Each of the vectors $u_1, \ldots, u_p$ satisfies an ODE system in time, $u_i' = f_i(t,y)$, while each of $u_{p+1}, \ldots, u_{p+q}$ satisfies an algebraic system $0 = f_i(t,y)$. We assume that each f_i consists of a reaction term R_i with no spatial coupling, and a diffusion (or more general transport) term S_i that involves no interaction among the species. Thus the system is

$$\begin{aligned} u_i' &= f_i = R_i(t,y) + S_i(t,u_i) \quad (i \le p) \\ 0 &= f_i = R_i(t,y) + S_i(t,u_i) \quad (i > p), \end{aligned}$$

where we have $R_i = (R_{i,m})$ with $R_{i,m}$ depending only on the variables $u_{j,m}$ at mesh point m, and S_i depending on u_i but no other u_j. We can also write the DAE system as

$$0 = F(t,y,y') = (u_1' - f_1, \ldots, u_p' - f_p, -f_{p+1}, \ldots, -f_{p+q})^T. \tag{4.1}$$

If we let I_1 denote the identity matrix of order pM, and denote $f = (f_1, \ldots, f_{p+q})^T = R + S$, then the system Jacobian is

$$J = \alpha F_{y'} + F_y = \alpha \begin{pmatrix} I_1 & 0 \\ 0 & 0 \end{pmatrix} - \partial f/\partial y = \alpha \bar{I}_1 - \partial R/\partial y - \partial S/\partial y, \tag{4.2}$$

where $\bar{I}_1 \equiv \begin{pmatrix} I_1 & 0 \\ 0 & 0 \end{pmatrix}$.

For a preconditioner approximating J, there are two choices immediately available from this representation. First, if the reaction terms dominate the problem, then

$$P_R \equiv \alpha \bar{I}_1 - \partial R/\partial y \tag{4.3}$$

should be an effective choice. Here when the variables are ordered by mesh point and then species as suggested initially, P_R is a block-diagonal matrix with M blocks each of size $p+q$, and so is very economical to deal with. But if the transport terms dominate the problem, consider instead

$$P_S \equiv \alpha \bar{I}_1 - \partial S/\partial y. \tag{4.4}$$

In this case, with the present ordering, P_S is block-diagonal with $p+q$ blocks of size M. Depending on the particular geometry, discretization, and mesh point ordering, each $M \times M$ block of P_S should be amenable to standard solution methods for discrete elliptic PDE problems.

Following [6], preconditioners based on the idea of operator splitting can be formed that combine both of the above simpler choices. They are

$$\begin{aligned} P_{SR} &\equiv (I - \alpha^{-1}\partial S/\partial y)(\alpha \bar{I}_1 - \partial R/\partial y), \quad \text{and} &(4.5)\\ P_{RS} &\equiv (I - \alpha^{-1}\partial R/\partial y)(\alpha \bar{I}_1 - \partial S/\partial y). &(4.6)\end{aligned}$$

The preconditioner P_{SR} represents the application of the reaction operator alone, followed by a correction involving the transport operator alone, and vice versa for P_{RS}. Each factor is much more efficiently treated than J itself, and so the same is true for P_{SR} and P_{RS}.

It is possible to do some analysis of the quality of these preconditioners, at least in the limit of small step size h, which corresponds to large α. Denoting

$$\begin{aligned} A &= \partial S/\partial y = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix} \\ B &= \partial R/\partial y = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \end{aligned}$$

and $\epsilon = \alpha^{-1}$, we have

$$\begin{aligned} J &= \epsilon^{-1}\bar{I}_1 - A - B, \\ P_R &= \epsilon^{-1}(\bar{I}_1 - \epsilon B), \\ P_S &= \epsilon^{-1}(\bar{I}_1 - \epsilon A), \\ P_{SR} &= \epsilon^{-1}(I - \epsilon A)(\bar{I}_1 - \epsilon B), \quad \text{and} \\ P_{RS} &= \epsilon^{-1}(I - \epsilon B)(\bar{I}_1 - \epsilon A). \end{aligned}$$

We begin with a lemma.

LEMMA 4.1. *Assume B_{22} is nonsingular. Define the matrix C by*

$$C \equiv \begin{pmatrix} B_{11} & B_{12} \\ -B_{22}^{-1}B_{21}B_{11} & -B_{22}^{-1}B_{21}B_{12} \end{pmatrix}. \tag{4.7}$$

Then for $\epsilon\|C\| < 1$, *we have*

$$(\bar{I}_1 - \epsilon B)^{-1} = \epsilon^{-1}X_{-1} + X_0 + \epsilon X_1 + \dots , \tag{4.8}$$

where

$$X_{-1} = \begin{pmatrix} 0 & 0 \\ 0 & -B_{22}^{-1} \end{pmatrix}$$
$$X_0 = \begin{pmatrix} I_1 & -B_{12}B_{22}^{-1} \\ -B_{22}^{-1}B_{21} & B_{22}^{-1}B_{21}B_{12}B_{22}^{-1} \end{pmatrix},$$

and $X_k = C^kX_0$ *for* $k \geq 1$.

Proof. We first note that

$$\begin{aligned} \bar{I}_1 - \epsilon B &= \begin{pmatrix} I_1 & 0 \\ 0 & \epsilon I_2 \end{pmatrix} \left[\begin{pmatrix} I_1 & 0 \\ -B_{21} & -B_{22} \end{pmatrix} - \epsilon \begin{pmatrix} B_{11} & B_{12} \\ 0 & 0 \end{pmatrix} \right] \\ &= \begin{pmatrix} I_1 & 0 \\ -\epsilon B_{21} & -\epsilon B_{22} \end{pmatrix} (I - \epsilon C), \end{aligned}$$

with C as given in (4.7). Thus, for $\epsilon\|C\| < 1$, we have

$$\begin{aligned} (\bar{I}_1 - \epsilon B)^{-1} &= (I - \epsilon C)^{-1} \begin{pmatrix} I_1 & 0 \\ -\epsilon B_{21} & -\epsilon B_{22} \end{pmatrix}^{-1} \\ &= (I + \epsilon C + \epsilon^2 C^2 + \cdots) \begin{pmatrix} I_1 & 0 \\ -B_{22}^{-1}B_{21} & -\epsilon^{-1}B_{22}^{-1} \end{pmatrix} \\ &\equiv (I + \epsilon C + \epsilon^2 C^2 + \cdots)(\epsilon^{-1}D + E). \end{aligned}$$

Hence,

$$(\bar{I}_1 - \epsilon B)^{-1} = \epsilon^{-1}D + (CD + E) + \sum_{k=1}^{\infty} \epsilon^k C^k (CD + E).$$

A quick calculation gives $X_{-1} = D$ and $X_0 = CD + E$, and this proves the lemma. □

We first consider the asymptotic behavior of the preconditioners P_R and P_{SR}. As will be seen below, the leading order term in the error for both preconditioners is the same. Hence, the order ϵ terms for both preconditioners are needed to compare their overall effectiveness.

THEOREM 4.2. *Assume that* B_{22} *is nonsingular. Then for* $\epsilon\|C\| < 1$, *we have*

$$P_R^{-1}J - I = \begin{pmatrix} 0 & 0 \\ 0 & B_{22}^{-1}A_{22} \end{pmatrix} + \epsilon \begin{pmatrix} -A_{11} & B_{12}B_{22}^{-1}A_{22} \\ B_{22}^{-1}B_{21}A_{11} & -B_{22}^{-1}B_{21}B_{12}B_{22}^{-1}A_{22} \end{pmatrix} + O(\epsilon^2). \tag{4.9}$$

Proof. First note that $J - P_R = -A$. Hence,

$$\begin{aligned} P_R^{-1}J - I &= -P_R^{-1}A \\ &= -\left(X_{-1} + \epsilon X_0 + O(\epsilon^2)\right) A \\ &= -X_{-1}A - \epsilon X_0 A + O(\epsilon^2). \end{aligned}$$

A simple calculation for $X_{-1}A$ and X_0A now gives the result. □

This result means that, for small h at least, to the extent that $B_{22}^{-1}A_{22}$ is small in norm, then so is the error matrix $P_R^{-1}J - I$, and P_R is a good approximation to J for the purposes of preconditioning. Here B_{22} represents the Jacobian of the reaction of the quasi-steady species relative to themselves, while A_{22} represents the transport operator for those species.

THEOREM 4.3. *Assume that B_{22} is nonsingular. Then for $\epsilon \cdot \min\{\|C\|, \|A\|\} < 1$, we have*

$$\begin{aligned} P_{SR}^{-1}J - I &= \begin{pmatrix} 0 & 0 \\ 0 & B_{22}^{-1}A_{22} \end{pmatrix} \\ &+\epsilon \begin{pmatrix} 0 & B_{12}B_{22}^{-1}A_{22} \\ B_{22}^{-1}A_{22}B_{21} & B_{22}^{-1}A_{22}(A_{22}+B_{22}) - B_{22}^{-1}B_{21}B_{12}B_{22}^{-1}A_{22} \end{pmatrix} + O(\epsilon^2). \end{aligned} \tag{4.10}$$

Proof. From the definition of P_{SR}, a simple calculation gives

$$P_{SR} = J + \begin{pmatrix} 0 & 0 \\ 0 & A_{22} \end{pmatrix} + \epsilon AB.$$

Hence,

$$P_{SR}^{-1}J - I = -P_{SR}^{-1}(\bar{A}_{22} + \epsilon AB),$$

letting $\bar{A}_{22} \equiv \mathrm{diag}(0, A_{22})$. From the definition of P_{SR}, it is apparent that $P_{SR} = (I - \epsilon A)P_R$. Hence, for $\epsilon \cdot \min\{\|C\|, \|A\|\} < 1$ and B_{22} nonsingular, we have

$$P_{SR}^{-1} = \left(X_{-1} + \epsilon X_0 + O(\epsilon^2)\right)\left(I + \epsilon A + O(\epsilon^2)\right).$$

Therefore,

$$\begin{aligned} P_{SR}^{-1}J - I &= -\left(X_{-1} + \epsilon X_0 + O(\epsilon^2)\right)\left(I + \epsilon A + O(\epsilon^2)\right)(\bar{A}_{22} + \epsilon AB), \\ &= -X_{-1}\bar{A}_{22} - \epsilon\left[X_0\bar{A}_{22} + X_{-1}\left(AB + \bar{A}_{22}^2\right)\right] + O(\epsilon^2). \end{aligned}$$

By an easy calculation, we find that $-X_{-1}\bar{A}_{22} = \mathrm{diag}(0, B_{22}^{-1}A_{22})$ and

$$-X_0\bar{A}_{22} - X_{-1}\left(AB + \bar{A}_{22}^2\right) = \begin{pmatrix} 0 & B_{12}B_{22}^{-1}A_{22} \\ B_{22}^{-1}A_{22}B_{21} & B_{22}^{-1}A_{22}(A_{22}+B_{22}) - B_{22}^{-1}B_{21}B_{12}B_{22}^{-1}A_{22} \end{pmatrix},$$

and this completes the proof. □

Comparing the upper left blocks in these two results, note that A_{11} *is* present in the order ϵ term in equation (4.9), and *is not* in equation (4.10). Thus, while the leading expression in the error for both preconditioners is the same, we can expect P_{SR} to be a better preconditioner when A_{11} is large (in some relative sense). This fact will be pointed out in the next section.

A result similar to (4.10) can be obtained for the preconditioner in which the two factors of P_{RS} are multiplied in the opposite order. The leading term in the error equation is the same, but the $O(\epsilon)$ term is not.

If instead we expect A to dominate B, we can consider the remaining two preconditioners P_S and P_{RS} in (4.4) and (4.6), respectively. The following theorem follows trivially along the same lines as above.

THEOREM 4.4. *Assume that A_{22} is nonsingular. Then for $\epsilon\|A_{11}\| < 1$, we have*

$$P_S^{-1}J - I = \begin{pmatrix} 0 & 0 \\ A_{22}^{-1}B_{21} & A_{22}^{-1}B_{22} \end{pmatrix} + O(\epsilon). \tag{4.11}$$

In addition, if $\epsilon \cdot \min\{\|A_{11}\|, \|B\|\} < 1$, then

$$P_{RS}^{-1}J - I = \begin{pmatrix} 0 & 0 \\ 0 & A_{22}^{-1}B_{22} \end{pmatrix} + O(\epsilon). \tag{4.12}$$

This result indicates that both P_S and P_{RS} are likely to be effective preconditioners when A_{22} dominates B_{21} and B_{22}. However, the fewer potential nonzeros in the leading order error term for P_{RS} suggests using it over P_S, all other things being equal. Moreover, if the two factors in P_{RS} are taken in the opposite order, one gets the same (larger) leading error term as for P_S, suggesting that the order of the factors in P_{RS} is to be preferred.

By departing from the operator splitting approach, it is possible to devise preconditioners P such that $P^{-1}J - I = O(\epsilon)$. However, the cost of applying P^{-1} is likely to be higher, and may be prohibitive on some problems. Along these lines, one can show the following result.

THEOREM 4.5. *Define* $P_1 = (\epsilon^{-1}\bar{I}_1 - \tilde{B})Q$, *where*

$$\tilde{B} \equiv \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & I_2 \end{pmatrix} \text{ and } Q \equiv \begin{pmatrix} I_1 & 0 \\ 0 & A_{22} + B_{22} \end{pmatrix}. \tag{4.13}$$

Then

$$P_1^{-1}J - I = O(\epsilon). \tag{4.14}$$

Next, define $P_2 = TP_1$, *where* $T = diag(I_1 - \epsilon A_{11}, I_2)$. *Then*

$$P_2^{-1}J - I = O(\epsilon). \tag{4.15}$$

In the first case, the error in the preconditioner itself is

$$P_1 - J = \begin{pmatrix} A_{11} & B_{12} - B_{12}(A_{22} + B_{22}) \\ 0 & 0 \end{pmatrix}.$$

In the second case, the factor T is designed to remove a block in the error matrix, so that

$$P_2 - J = \begin{pmatrix} 0 & B_{12} - B_{12}(A_{22} + B_{22}) \\ 0 & 0 \end{pmatrix} + O(\epsilon).$$

Thus one would expect P_2 to perform better than P_1 when A_{11} is large. The dominant cost of inverting P_1 and P_2 above is most likely to be in solving linear systems involving $A_{22} + B_{22}$. Note that this matrix will be nonsingular for index-1 systems.

Often the numerical behavior of DASSL and its variants on constrained systems of this type is improved if the constraint equations in the DAE system are scaled. For index-m DAEs, the condition number of the iteration matrix is $O(\alpha^m)$ [3, p. 144]. Scaling is a way to reduce the dependence of the conditioning of a DAE system on the stepsize. In the case of the reaction-diffusion system, this would mean applying a scale factor σ to the lower qM equations in the system (4.1). For semi-explicit index-1 problems, the scale factor $\sigma = \alpha$ is suggested [3, p. 145]. If we define a scaling matrix

$$S = \begin{pmatrix} I_1 & 0 \\ 0 & \sigma I_2 \end{pmatrix}, \tag{4.16}$$

then the scaled problem is $\bar{F} \equiv SF = 0$. The choice of preconditioner for this problem corresponding to P for the unscaled problem is $\bar{P} = SP$. It is easy to see that the Jacobian of $\bar{F}$ is $\bar{J} = SJ$, and so $\bar{P}^{-1}\bar{J} = P^{-1}J$. Thus scaling does not affect the analytical properties of the method, but it does have the potential to improve its numerical properties by way of improved conditioning (roundoff error growth) of $\bar{J}$ and $\bar{P}$.

If scaling is applied to the preconditioner P_{SR} of (4.5), we can rewrite $\bar{P}_{SR}$ as

$$\begin{aligned}\bar{P}_{SR} &= S(I-\alpha^{-1}A)(\alpha\bar{I}_1 - B) \\ &= S(I-\alpha^{-1}A)S^{-1}S(\alpha\bar{I}_1 - B) \\ &= (I-\alpha^{-1}SAS^{-1})(\alpha S\bar{I}_1 - SB).\end{aligned}$$

But $SAS^{-1} = A$ because A is block-diagonal, and $S\bar{I}_1 = \bar{I}_1$. Thus we have

$$\bar{P}_{SR} = (I-\alpha^{-1}A)(\alpha\bar{I}_1 - SB). \tag{4.17}$$

This means that the addition of scaling to this preconditioner can be implemented simply by scaling the lower blocks of the B terms in the second factor of P_{SR}.

The result is not as simple if scaling is applied to the preconditioner P_{RS} of (4.6). Since $SBS^{-1} \neq B$, we obtain only

$$\bar{P}_{RS} = (I-\alpha^{-1}SBS^{-1})(\alpha\bar{I}_1 - SA). \tag{4.18}$$

Thus, in addition to scaling the lower blocks of the A in the second factor, we must scale the off-diagonal blocks of B to σB_{21} and $\sigma^{-1}B_{12}$ in the first factor.

5. Examples.

In this section we will present several examples which illustrate how the method is applied and how well it works on various types of problems.

5.1. Two-dimensional heat equation.

The first test problem is a two-dimensional heat equation given by

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{5.1}$$

on a square $0 \le x, y \le 1$, with homogeneous boundary conditions ($u = 0$ on the boundary) and initial condition profile $u(x,y,0) = 16x(1-x)y(1-y)$. We used a uniform Cartesian product mesh with M internal points in each direction, with spacings $\Delta x = \Delta y = 1/(M+1)$. For

$$y_{jk} \approx u(j\Delta x, k\Delta y), \quad (0 \le j,k \le M+1)$$

the discrete problem is

$$\left\{\begin{array}{rcll} y'_{jk} &=& (\Delta x)^{-2}(y_{j+1,k} + y_{j-1,k} + y_{j,k+1} + y_{j,k-1} - 4y_{jk}), & 1 \le j,k \le M \\ 0 &=& y_{jk} \quad \text{otherwise} & \end{array}\right. \tag{5.2}$$

Thus the total size of the system is NEQ $= (M+2)^2$.

The DASPK vector y consists of the y_{jk} ordered first by j, then by k. The DAE system has the form

$$0 = F(t,y,y') = Ey' - By$$

where E is the identity matrix with 0 in place of 1 for boundary indices, and B has the usual 5-stripe structure. Because of the way the boundary conditions are formulated, the problem is a DAE, not an ODE.

The Newton matrix required in DASPK is

$$A = (\mathrm{CJ})F_{y'} + F_y = (\mathrm{CJ})E - B.$$

The preconditioner P was obtained by a finite-difference approximation to the iteration matrix, where the differencing was done as though the matrix were tridiagonal. The bandwidth was assumed to be three, and columns whose indices differ by three are differenced simultaneously, as in DASSL [3]. This produces a tridiagonal approximation to the iteration matrix, at a cost of four F evaluations, but it is not the tridiagonal part of the original matrix. Instead, the elements of the original matrix which do not lie in the tridiagonal band have been lumped, via the differencing, into the tridiagonal band. The preconditioner was computed and factorized in a subroutine JAC as described in Section 3.4, and the back-substitution was done in subroutine PSOL.

We solved the problem with the error tolerances RTOL = 0, ATOL = 10^{-3}, for three cases: $M = 5$, $M = 10$, $M = 20$, with both the direct and iterative options of DASPK. For the direct option, the iteration matrix was approximated by the tridiagonal matrix described above; hence it is a relatively poor but cheap approximation to the actual iteration matrix. Solution errors in all cases were comparable. The results are given in Table 5.1, where F denotes the number of function evaluations, PE is the number of preconditioner evaluations (in the direct method, this is the number of evaluations of the approximate iteration matrix), PS is the number of preconditioner solves (backsubstitutions), NLI is the number of nonlinear (modified Newton) iterations, LI is the total number of linear iterations, AVL is the average number of Krylov iterations per Newton iteration (NLI/LI), and NCF is the number of nonlinear convergence test failures. (There were no convergence test failures for the linear iteration.)

Method	M	Steps	F	PE	PS	NLI	LI	AVL	NCF
Iterative	5	45	220	17	169	87	82	0.94	0
Direct	5	98	513	102	0	207	0	0.00	28
Iterative	10	47	280	18	226	91	135	1.48	0
Direct	10	671	4443	985	0	1488	0	0.00	324
Iterative	20	51	449	17	398	100	298	2.98	0
Direct	20	1779	12111	2651	0	4158	0	0.00	878

TABLE 5.1
Test results for example 1

The results indicate that the direct method (which gives identical results to DASSL), is having some trouble converging the Newton iteration. This is not unexpected due to the fact that the approximation we are using to the iteration matrix is not very good. The iterative method, using (as a preconditioner) the same matrix approximation as the direct method, but in addition using GMRES to give a more accurate solution to the linear system, has no trouble achieving convergence in either the linear or nonlinear iterations, and gives a solution much more efficiently.

5.2. Multi-species food web problem. The next problem is a model of a multi-species food web [4], in which mutual competition and/or predator-prey relationships in a spatial domain are simulated. Here we consider a model with s species, where species $s/2+1, \cdots, s$ (the predators) have infinitely fast reaction rates:

$$\left\{ \begin{array}{l} \frac{\partial c^i}{\partial t} = f_i(x,y,t,c) + d_i(c^i_{xx} + c^i_{yy}) \quad (i = 1,2,\cdots,s/2), \\ \quad 0 = f_i(x,y,t,c) + d_i(c^i_{xx} + c^i_{yy}) \quad (i = s/2+1,\cdots,s), \end{array} \right. \tag{5.3}$$

with

$$f_i(x,y,t,c) = c^i(b_i + \sum_{j=1}^{s} a_{ij}c^j). \tag{5.4}$$

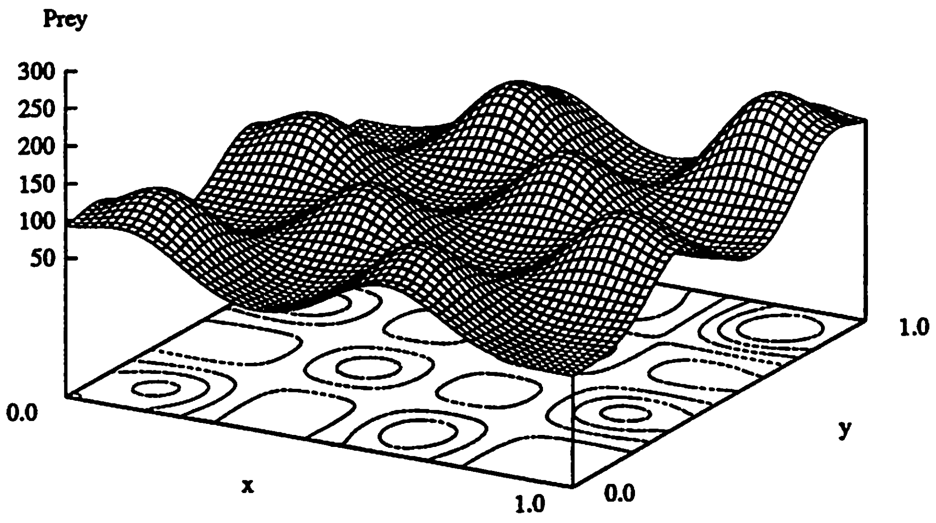


FIG. 5.1. *Problem 2. Plot of prey species at steady-state ($\beta = 300$).*

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 for a simple model of p prey and p predator species ($s = 2p$), arranged in that order in the vector c. We take the various coefficients to be as follows:

$$\left\{\begin{array}{l} a_{ii} = -1 \text{ (all } i) \\ a_{ij} = -0.5 \cdot 10^{-6} \ (i \le p, j > p) \\ a_{ij} = 10^4 \ (i > p, j \le p) \end{array}\right. \tag{5.5}$$

(all other $a_{ij} = 0$),

$$\left\{\begin{array}{l} b_i = (1 + \alpha xy + \beta \sin(4\pi x)\sin(4\pi y)) \ (i \le p) \\ b_i = -(1 + \alpha xy + \beta \sin(4\pi x)\sin(4\pi y)) \ (i > p) \end{array}\right. \tag{5.6}$$

and

$$\left\{\begin{array}{l} d_i = 1 \ (i \le p) \\ d_i = .05 \ (i > p). \end{array}\right. \tag{5.7}$$

The domain is the unit square $0 \le x, y \le 1$, and $0 \le t \le 10$. The boundary conditions are of Neumann type (zero normal derivatives) everywhere. The coefficients are such that a unique stable equilibrium is guaranteed to exist when α and β are both zero, and time derivatives appear in the equations for species $s/2 + 1, \cdots, s$ [4]. Empirically, for (5.3) a stable equilibrium appears to exist when α and β are positive, although it may not be unique. In this problem we take $\alpha = 50$ and various values of β. The steady-state solution for the prey with $p = 1$ and $\beta = 300$ is represented in Figure 5.1. The plot for the predator is identical except for a different scale on the vertical axis.

The initial conditions used for this problem are taken to be simple peaked functions that satisfy the boundary conditions and very nearly satisfy the constraints, given by

$$\begin{aligned} c^i &= 10 + i[16x(1-x)y(1-y)]^2 \quad (i = 1, \cdots, s/2) \\ c^i &= -(b_i + \sum_{j=1}^{s/2} a_{ij}c^j)/a_{ii} \quad (i = s/2 + 1, \cdots, s). \end{aligned}$$

The PDE system (5.3) (plus boundary conditions) was discretized with central differencing on an $M \times M$ mesh, much the same as for the first example. The resulting DAE system has size NEQ $= sM^2$.

This problem is of the form treated in Section 4, with $q = p$. Based on the ideas there, three different preconditioners were constructed. First, because the interaction terms f_i contain very large coefficients, we expect that P_R, given by (4.3), may be effective. Second, we consider the product P_{SR}, given by (4.5), and the same product taken in the opposite order, which we denote P'_{SR}. The inverse of the factor $(I - \alpha^{-1}\partial S/\partial y)$ representing the spatial part of the Jacobian is not computed exactly, but only approximated by a fixed number (namely 5) of Gauss-Seidel iterations on the corresponding linear system.

To begin with, we compare the direct and Krylov solution modes for the case $\beta = 100$, $p = 1$, and $M = 20$. In the direct case, the Jacobian is treated as a banded matrix with half-bandwidths equal to $sM = 40$. In the Krylov case, we use the preconditioner P_{SR}. In order to measure global errors, a run in the direct mode was first made with tolerances RTOL = ATOL equal to 10^{-9}, and for each subsequent run with looser tolerances, the differences ΔY between the current and accurate solution were taken at times $t = 10^{-7}, 10^{-4}, 10^{-1}, 3, 6, 9, 10$. A weighted global error measure was computed as WGE $= max[|\Delta Y_i|/(|Y_i| + 1)]$, the maximum being over i and t. For the direct solution mode with RTOL = ATOL = 10^{-5}, the value of WGE is $2.5 \cdot 10^{-5}$. For the Krylov solution mode, values of RTOL = ATOL of 10^{-5}, 10^{-6}, and 10^{-7} gives WGE values of $1.4 \cdot 10^{-4}$, $4.3 \cdot 10^{-5}$, and $4.9 \cdot 10^{-6}$, respectively. Thus the Krylov run with 10^{-6} tolerances produces nearly the same accuracy as the direct run with 10^{-5} tolerances. Running on a Cray Y/MP computer, the cost of the two runs is nearly the same: 23.8 CPU sec in the direct case, and 23.3 CPU sec in the Krylov case. However, the storage requirements are far lower in the Krylov case, because of the use of a banded Jacobian in the direct case and a block-diagonal matrix in the Krylov case. For this pair of runs, the total number of real and integer words of work space is 104,910 in the direct case, and 16,931 in the Krylov case, a factor of 6.2 lower. For a finer mesh or larger number of species, the relative savings in storage in storage would be even greater, and the cost advantage for the Krylov mode would also be greater.

Next, we show the results of DASPK runs in the Krylov mode for various values of β and M, still with $p = 1$, $s = 2$, and with tolerances of 10^{-5}. The runs were made on a Cray Y/MP. Table 5.2 gives, for each choice of β, M, and preconditioner, the total number of steps, the average number of Krylov iterations per Newton iteration AVL, and the CPU run time RT in sec. For $\beta = 100$ or 300, it is clear that P_R is inferior to the product preconditioners, whereas for $\beta = 1000$ it is superior. As expected, the less expensive preconditioner P_R always results in a higher average number of linear iterations per nonlinear iteration. The product order P_{SR} proved to perform better than P'_{SR} in all cases for which it was run.

One disturbing pattern in Table 5.2 is the unexpected growth of total cost as a function of M for fixed choice P_{SR} of preconditioner. We expect costs to grow as M^2, but for $\beta = 100$ the jump in run time as M is changed from 20 to 40 is anomalous, as is that for $\beta = 300$ and $M = 40$ and 60. While we have no complete explanation for this, we note that the higher costs are accompanied by a higher frequency of preconditioner evaluation, and suspect that they are related to errors in the value of $B = \partial R/\partial y$ incurred by evaluating this part of the Jacobian as infrequently as possible. Furthermore, by an analysis extending that in Section 4, it can be shown that relative errors in B are magnified in the error matrices $P_R^{-1}J - I$ and $P_{SR}^{-1}J - I$, by roughly 10^4, the coefficient in the lower left block of the array (a_{ij}).

The question of whether or not to apply scaling to the constraint equations arose in some of the tests on this problem, when the performance of the solver seemed to degrade at late times t. In some (but not all) such cases, scaling the constraints by α, as described in Section 4, improved

β	M	Prec.	Steps	AVL	RT
100	20	P_R	874	5.07	82.0
100	20	P_{SR}	198	1.32	14.9
100	40	P_{SR}	314	2.76	154
100	60	P_{SR}	320	2.69	350
300	20	P_R	989	4.88	90.2
300	20	P_{SR}	192	1.46	15.3
300	20	P'_{SR}	213	2.02	22.2
300	40	P_{SR}	200	1.15	54.6
300	40	P'_{SR}	226	1.56	75.2
300	60	P_{SR}	237	2.01	205
1000	20	P_R	188	1.94	8.9
1000	20	P_{SR}	219	1.49	17.7
1000	40	P_R	189	2.20	38.8
1000	40	P_{SR}	220	1.32	64.2
1000	60	P_R	205	2.86	113
1000	60	P_{SR}	198	1.61	147

TABLE 5.2
Test results for example 2

the performance, by as much as 20% in run time. But for some cases the run time with scaling was larger than without it, with no apparent pattern for the cases in the two categories. For example, for the case $\beta = 100$, $M = 20$ with preconditioner P_{SR}, the run time is 14.9 sec without the rescaling and 12.2 sec with it. But the run with P_R on the same problem (the first case in Table 5.2) ran even slower with rescaling (92 sec vs 82 sec).

Finally, we present the results for one other case, the largest one in this test set: $\beta = 1000$, $M = 60$, $p = 7$, for which the problem size is NEQ = 50,400. The run with preconditioner P_R and tolerances of 10^{-5} (as before) ran to completion in 471 sec on the Cray Y/MP, in 215 steps, with AVL = 2.75.

6. Acknowledgement. Most of the programming and debugging work in generating the DASPK solver from DASSL was done by Clement Ulrich. We are very grateful for his efforts.

REFERENCES

[1] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quart. J. Appl. Math., 9 (1951), 17-29.

[2] U. Ascher and L. Petzold, *Stability of computational methods for constrained dynamics systems*, to appear, SIAM J. Sci. Comput.

[3] K. Brenan, S. Campbell and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elsevier 1989.

[4] P. N. Brown, *Decay to uniform states in food webs*, SIAM J. Appl. Math., 46 (1986), 376-392.

[5] P. N. Brown and A. C. Hindmarsh, *Matrix-free methods for stiff systems of ODEs*, SIAM J. Numer. Anal. 23 (1986), 610-638.

[6] P. N. Brown and A. C. Hindmarsh, *Reduced storage matrix methods in stiff ODE systems*, J. Appl. Math & Comp. 31, (1989), 40-91.

[7] Peter N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Stat. Comput. 11 (1990), 450-481.

[8] George D. Byrne, *Pragmatic experiments with Krylov methods in the stiff ODE setting*, in *Computational Ordinary Differential Equations*, J. R. Cash (Ed.), Oxford University Press, 1992.

[9] T. F. Chan and K. R. Jackson, *Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms*, SIAM J. Sci. Stat. Comput. 5 (1984), 533-542.

[10] T. F. Chan and K. R. Jackson, *The Use of Iterative Linear Equation Solvers in Codes for Large Systems of Stiff IVPs for ODEs*, SIAM J. Sci. Stat. Comput. 7 (1986), 378-417.

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

[12] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK User's Guide*, SIAM, 1979.

[13] C. W. Gear, *Simultaneous numerical solution of differential/algebraic equations*, IEEE Trans. on Circuit Theory, CT-18, No. 1, (1971), 89-95.

[14] C. W. Gear and Y. Saad, *Iterative Solution of Linear Equations in ODE Codes*, SIAM J. Sci. Stat. Comput. 4 (1983), 583-601.

[15] K. R. Jackson, *The Use of Krylov Subspace Methods in Codes for Large Stiff Systems of ODEs*, in Advances in Computer Methods for Partial Differential Equations - VI, R. Vichnevetsky and R. S. Stepleman (Eds.), IMACS, New Brunswick, NJ, 1987, pp. 452-459.

[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] L. Petzold, *A description of DASSL: A differential/algebraic system solver*, in *Scientific Computing*, R. S. Stepleman et al. (eds.), North-Holland (Vol. 1 of IMACS Trans. on Scientific Computation), (1983), 65-68.

[18] 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.

[19] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp. 37 (1981), 105-126.

7. Appendix: DASPK Structure.

The DASPK solver consists of 22 subprograms, not counting the required LINPACK and BLAS routines or the user-supplied subroutines.

The following is a complete list of the subordinate routines in DDASPK, the double precision version of DASPK, excluding user-supplied routines, with a brief description of each. The names for the single precision version begin with S instead of D.

Subordinate routines used by DDASPK

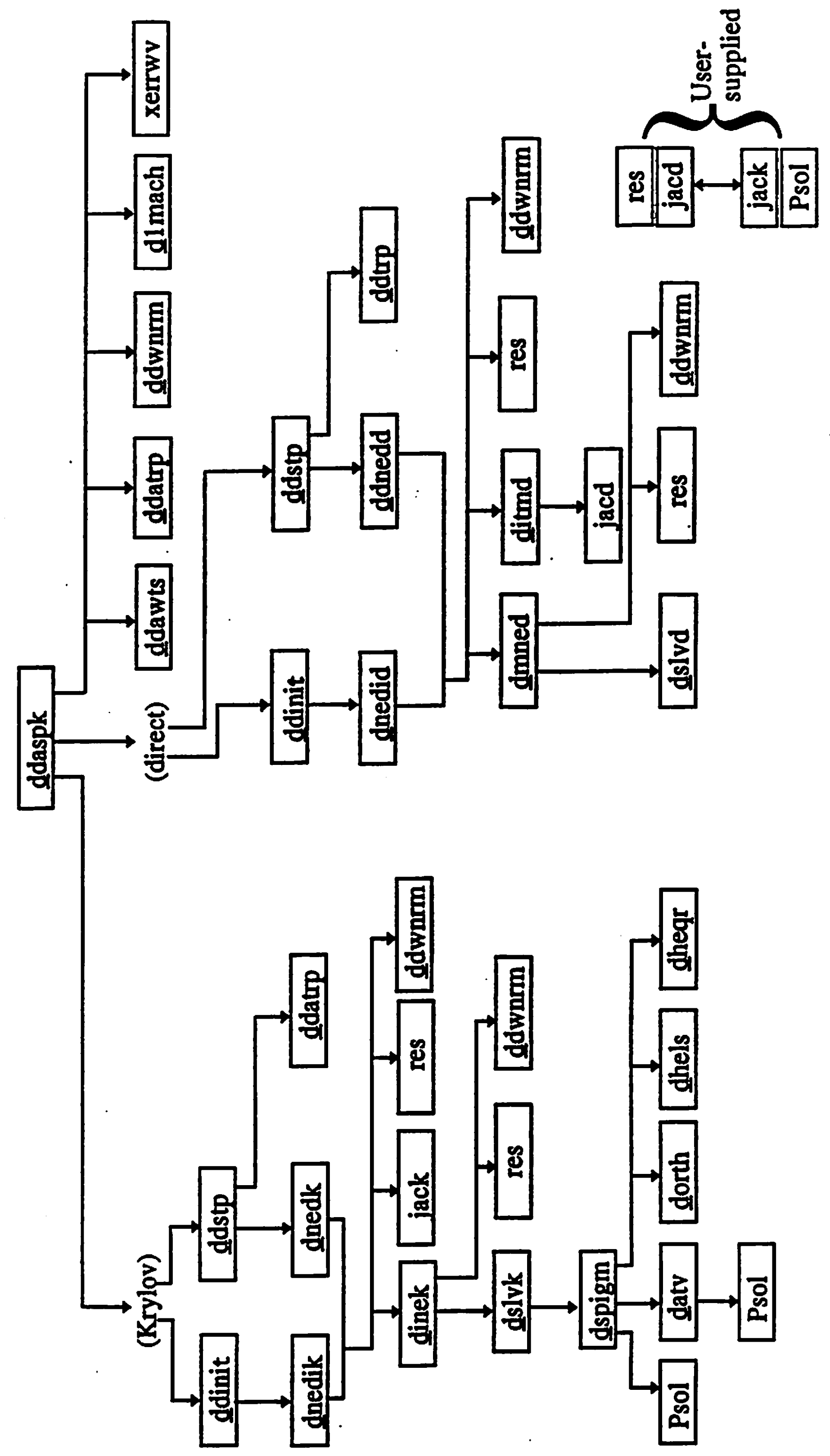
DDINIT	computes the initial YPRIME vector.
DDSTP	carries out one step of the integration.
DDAWTS	sets error weight quantities.
DDATRP	performs interpolation to get an output solution.
DDWNRM	computes the weighted root-mean-square norm of a vector.
D1MACH	provides the unit roundoff of the computer.
XERRWV	handles error messages.
DNEDID	nonlinear equation driver to initialize YPRIME by using direct linear system solver methods. Interfaces to Newton solver (direct case).
DNEDD	nonlinear equation driver for direct linear system solver methods. Interfaces to Newton solver (direct case).
DITMD	assembles the iteration matrix (direct case).
DMNED	solves the associated nonlinear system by modified Newton iteration and direct linear system methods.
DSLVD	interfaces to linear system solver (direct case).
DNEDIK	nonlinear equation driver to initialize YPRIME by using iterative linear system solver methods. Interfaces to Newton solver (iterative case).
DNEDK	nonlinear equation driver for iterative linear system solver methods. Interfaces to Newton solver (iterative case).
DINEK	solves the associated nonlinear system by Inexact Newton iteration and (linear) Krylov iteration.
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 are LINPACK routines for solving linear systems (dense or band direct methods).

DAXPY, DCOPY, DDOT, DNRM2, DSCAL are Basic Linear Algebra (BLAS) routines.

Below is a block diagram showing the call paths within DDASPK, including the user-supplied routines (which appear repeatedly, for ease of presentation).

DASPK Block Diagram


ddaspk
xerrwv
d1mach
ddwnrm
ddatrp
ddawts
(direct)
ddinit
ddstp
ddtrp
dnedid
dnedd
dmned
ditmd
res
jacd
dslvd
(Krylov)
dnedik
dnedk
dinek
jack
dslvk
dspigm
Psol
datv
dorth
dhels
dheqr
User-supplied