Adaptive Linear Equation Solvers in Codes for Large Stiff Systems of ODEs

K. R. Jackson and W. L. Seward

Research Report CS-91-33
July 1991

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ADAPTIVE LINEAR EQUATION SOLVERS IN CODES FOR LARGE STIFF SYSTEMS OF ODES

K. R. JACKSON* AND W. L. SEWARD†

Abstract. Iterative linear equation solvers have been shown to be effective in codes for large systems of stiff initial-value problems for ordinary differential equations (ODEs). While preconditioned iterative methods are required in general for efficiency and robustness, unpreconditioned methods may be cheaper over some ranges of the interval of integration. In this paper, we develop a strategy for switching between unpreconditioned and preconditioned iterative methods depending on the amount of work being done in the iterative solver and properties of the matrix being solved. This strategy is combined with a "type-insensitive" approach to the choice of formula used in the ODE code to develop a method that makes a smooth transition between nonstiff and stiff regimes in the interval of integration. We find that, as expected, for some large systems of ODEs, there may be a considerable saving in execution time when the type-insensitive approach is used. If there is a region of the integration that is "mildly" stiff, switching between unpreconditioned and preconditioned iterative methods also increases the efficiency of the code significantly.

Key words. type-insensitive ODE code, iterative linear solver, preconditioning.

AMS(MOS) subject classifications. 65L05, 65F10.

1. Introduction. In recent years, there have been several investigations of the use of iterative linear equation solvers in codes for the numerical solution of large systems of stiff initial-value problems (IVPs) for ordinary differential equations (ODEs). See, for example, [5, 6, 8, 11]. Such work has established clearly the potential effectiveness of the combination of these methods, but there are many open questions concerning both the choice of iterative method and the way in which it interacts with strategies used in the ODE solver. Frequently, the iterative methods investigated have been of the "Krylov subspace" type, e.g., the conjugate gradient method, Orthomin [18] or GMRES [6]. In this paper, we also consider an iterative method of this type but our results should apply to a broader class since we are primarily concerned with the interaction between the iterative method and the other strategies of the ODE solver.

We study the numerical solution of large systems of stiff IVPs for ODEs of the form

$$y' = f(t, y), \qquad y(t_0) \quad \text{given.} \tag{1}$$

Due to the stiffness, such problems are usually discretized using an implicit numerical method, most often the Backward Differentiation Formulas (BDFs) [10]. We will concentrate on the BDFs, although many of our ideas apply to implicit numerical methods in general. Applied to (1), a k-step BDF has the form

$$y_n = \sum_{j=1}^{k} \alpha_{n-j} y_{n-j} + h_n \beta_n f(t_n, y_n). \tag{2}$$

To find y_n from (2), it is necessary to solve a system of equations of the form

$$F(y_n) = y_n - h_n \beta_n f(t_n, y_n) - \phi_n = 0, \tag{3}$$

* Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.
† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

where ϕ_n contains the terms in (2) that do not depend on y_n. In general, this system is nonlinear and is solved by Newton's method or some variant of it.

The usual form of Newton's method is

- select an initial guess y_n^0
- for $l = 0, 1, \ldots$
  - solve $F_y(y_n^l)\Delta_n^l + F(y_n^l) = 0$ for Δ_n^l
  - set $y_n^{l+1} = y_n^l + \Delta_n^l$

where $F_y(y_n^l) = I - h_n\beta_n \frac{\partial f}{\partial y}(y_n^l)$. It is common in ODE codes to use a chord- (or pseudo-) Newton method in which $F_y(y_n^l)$ is replaced by some approximation W_n^l, generally the Newton iteration matrix F_y (or some approximation to it) from some earlier timestep. If the system of ODEs (1) is large, the linear algebra cost of the *solve* step will be a major part of the cost of the integration. If W_n^l is sparse, though, an iterative linear algebra method may be an effective means of reducing this cost.

When an iterative linear equation solver is used in the *solve* phase of Newton's method, an *inexact Newton method* [9] is obtained. In general, such a method has the form

- select an initial guess y_n^0
- for $l = 0, 1, \ldots$
  - find Δ_n^l satisfying $F_y(y_n^l)\Delta_n^l + F(y_n^l) = d_n^l$ for d_n^l "sufficiently small"
  - set $y_n^{l+1} = y_n^l + \Delta_n^l$

This iteration converges at least linearly if

$$\|d_n^l\| \leq \eta_l \|F(y_n^l)\|, \tag{4}$$

where $0 \leq \eta_l \leq \eta < 1$. In the case where an iterative linear equation solver is used to find y_n^l, the residual d_n^l is the final residual of the linear iteration.

Brown and Hindmarsh [5] consider iterative linear equation solvers based on Arnoldi's method for use in the ODE code LSODE [12]. In their approach, the matrix-vector product required in the iterative method is approximated by a finite difference based on f. This approach avoids explicitly forming the iteration matrix W_n^l needed in Newton's method and hence has been referred to as a "matrix-free" method. These methods are very efficient in storage and, as expected, the iterative method converges quickly if the eigenvalues of W_n^l are clustered. When this is not the case, it is found that preconditioning is required for efficiency and robustness in the ODE solver. In [6], preconditioning strategies for use with the "matrix-free" approach are discussed. These strategies require formation and storage of at least part of W_n^l in general but are still very efficient in terms of storage. The authors find that, with preconditioning, a wider class of problems can be solved.

Chan and Jackson [8] also investigate both unpreconditioned and preconditioned Krylov-type iterative methods used with LSODE. The methods they use are not "matrix-free" — W_n^l is formed, stored and used explicitly to compute the matrix-vector products needed in the iterative method. The authors consider various ways of reducing the amount of time spent in forming and processing W_n^l. For example, if preconditioning is not being used, they show that it is not difficult to keep the factor $h_n\beta_n$ in W_n^l current, thus requiring fewer Jacobian evaluations for the integration.

Hence, we observe that, for a variety of reasons, unpreconditioned iterative methods can be cheaper to use than preconditioned ones in stiff ODE solvers. However, preconditioning appears necessary in general for efficiency and robustness. This observation led us to study an adaptive approach to switching preconditioning on and off. In particular, if the eigenvalues of the Newton iteration matrix W_n^l are clustered, which always occurs when h_n is small, preconditioning may not be necessary. In other cases, it is likely that the number of iterations of the linear equation solver can be reduced significantly by preconditioning.

Instead of not preconditioning at all, our method uses *diagonal scaling*, a cheap preconditioner described in more detail below, when this seems to be effective, but switches to a more powerful preconditioner when diagonal scaling appears to be ineffective. We expect that CPU time can be saved by avoiding unnecessary expensive preconditioning without sacrificing robustness since preconditioning is available when necessary.

The idea of switching preconditioning on and off combines naturally with the use of a "type-insensitive" ODE solver. A type-insensitive code switches between methods appropriate for nonstiff and stiff IVPs depending on the nature of the problem. This approach has been discussed by several authors for a variety of formulas. See, for example, [4, 13, 14, 15]. Petzold [14] developed a type-insensitive method based on the Adams formulas [10] and the BDFs. Her method starts integrating with the Adams formulas and monitors the timestep, the eigenvalues of the Jacobian (indirectly) and the region of absolute stability. It switches to the BDFs when it estimates that it can increase the stepsize by a factor of five or more by doing so. When integrating with the BDFs, the code continues to monitor the same information and will switch back to the Adams formulas if it expects to maintain the same stepsize after the switch.

We have combined the ideas from [14] with our adaptive approach to preconditioning. Our code starts integrating with the Adams formulas and switches to the BDFs as described in [14]. When this switch is made, the linear equation solver uses diagonal scaling only. Later the method may switch to a more expensive preconditioner described below. It can switch back to diagonal scaling and to the Adams formulas.

The basic iterative linear equation solver that we use is Orthomin [18]. Other reasonable choices include GMRES [5] and the stabilized version of the conjugate-gradient-squared (CGS) algorithm [17]. To focus on adaptive preconditioning, we chose to use one basic iterative solver only. We selected the well-known scheme Orthomin, in part because an effective implementation of it along with several preconditioners was readily available to us.

There are a variety of possible approaches for the "cheap" preconditioning. We could use no preconditioning at all, in which case a "matrix-free" method or an approach that updates $h_n\beta_n$ frequently might be very efficient. However, there may be hidden costs associated with such methods. For example, a "matrix-free" method requires one function evaluation per linear iteration. Forming the Newton iteration matrix W_n^l explicitly, on the other hand, is frequently a relatively inexpensive operation for a large sparse system of ODEs, costing a few function evaluations only.

In our numerical experiments, we chose to form W_n^l explicitly, according to the heuristics of the basic ODE solver, and to use diagonal scaling (that is, preconditioning by a diagonal matrix to make the diagonal elements of W_n^l all ones) as our cheap preconditioner. The improvement in performance when diagonal scaling is used can be significant even though the cost is low. Moreover, diagonal scaling often reduces the cost of the iterative solver since it is not necessary to multiply by the matrix diagonal in this case when computing matrix-vector products. So there is little point to omitting diagonal scaling if W_n^l is computed explicitly. Furthermore, diagonal scaling can be used with other approaches to forming W_n^l (see [5], for example, for a discussion of scaling with "matrix-free" methods), but we have not attempted any elaboration of our basic approach. For brevity, in the remainder of the paper, we occasionally refer to an iterative method with diagonal scaling only as an unpreconditioned method.

A more powerful preconditioner is switched on if diagonal scaling appears to be ineffective. The particular preconditioning strategies that we have studied are Level 0 and Level 1 Incomplete LU factorizations, denoted ILU(0) and ILU(1) respectively. The ILU(0) preconditioner has the same sparsity structure as the original matrix, while some fill-in is allowed in the ILU(1) preconditioner. We note that diagonal scaling is also used with the ILU preconditioners, as it is often found to improve their performance. These preconditioning strategies can be applied to any matrix and have been found to work well in solving time-dependent PDEs. See, for example, [1, 2]. We also develop criteria for switching these preconditioners off.

In the next section, we briefly describe the codes that were used in this investigation. Heuristics for identifying when to switch preconditioning on and off are discussed in §3. The use of adaptive preconditioning in type-insensitive codes is considered in §4. Test problems and numerical results are presented in both §3 and §4 to illustrate the performance of the techniques. We end with some conclusions in §5.

2. Codes Used in the Investigation. The ODE package used in this investigation was SPRINT [3]. This package is designed to offer to a user a range of methods for the numerical solution of systems of IVPs in ODEs. It allows the user to select from four time-stepping methods and three linear algebra packages to create the complete method appropriate for a particular problem. The three algebra routines are full, banded and sparse direct solvers. Because of this modularity, the SPRINT package is well-suited for testing the interaction of the iterative linear algebra solver with the ODE method, since it is possible to couple our iterative solver to a sophisticated, fully-developed time-stepping scheme.

SPRINT includes a space discretization module, but for one-dimensional PDE systems only. Consequently, the higher-dimensional PDE test problems used in this paper have been semidiscretized in space by hand. One motivation for our study of iterative linear solvers is current work being done with SPRINT to develop modules for spatial discretization of PDE systems in higher dimensions.

The particular ODE method used in this investigation is based on the BDFs and is similar in implementation to the well-known code LSODE [12]. A variant of LSODE, called LSODA, was written by Hindmarsh and Petzold to incorporate the ideas for a type-insensitive method described in [14]. Our type-insensitive ODE solver was developed by altering the SPRINT BDFs module corresponding to the changes made to evolve LSODE into LSODA.

The iterative solver, WATSIT, used in our study was developed at the University of Waterloo based on methods described in [1] and [2]. This code is designed to solve systems of linear equations of the form $Ax = b$ where the matrix A is large and sparse. The basic technique is an incomplete factorization scheme with acceleration. The user has several choices for both the acceleration method and the incomplete factorization. Possible acceleration methods include the conjugate gradient method, Orthomin [18], the conjugate-gradient-squared (CGS) algorithm [16] and a stabilized version of CGS [17]. As noted in §1, since we wish to focus on our adaptive strategy in this paper, we use Orthomin acceleration only in our numerical tests and have not encountered any test problems where it breaks down. The user has the choice of diagonal scaling only or preconditioning based on either Level 0 or Level 1 incomplete LU factorization as described earlier in this paper.

In Brown and Hindmarsh [5], convergence criteria for the iterative method are discussed in detail in the context of an ODE code. Both Brown and Hindmarsh [5] and Chan and Jackson [8] find that the residual reduction condition (4) is overly expensive — it forces linear iterations that do not seem to improve the accuracy of the ODE solution significantly. In [5], the authors propose and justify a strategy that accepts the result produced by the iterative linear solver if the residual is some small fraction of the tolerance level required on the solution of the nonlinear system. A similar strategy was developed in [8]. We adopt this approach and also use the same fraction that is suggested in [5], namely, 1/20. The residual is measured in the weighted norm used by the ODE solver. Since the acceleration methods used in WATSIT return the residual directly, we do not need to scale the matrix to obtain the weighted norm, as is done in [5]. The linear iteration is started with a zero vector as its initial guess and always does at least one iteration. If the maximum iteration count is reached without the desired reduction of the residual on the first Newton iteration, the code continues provided that the current residual is smaller than the initial one. Otherwise, a failure is signalled to the ODE solver and either the Newton iteration or the timestep is retried, according to the criteria in the ODE code for a failure of Newton's method.

All numerical results reported in this paper were computed in double precision on a MIPS M/120 RisComputer.

3. Adaptive Preconditioning. In this section, the adaptive choice of preconditioning method is discussed independently of the type-insensitive approach for solving stiff ODEs. The two ideas are combined in §4. To develop strategies for switching between preconditioning methods, we need to assess the relative cost of the preconditioners as well as the expected saving when the switch is made.

For a particular iterative method and implementation, it is straightforward to estimate the relative cost of preconditioning by counting floating point operations in the code. For simplicity, we estimate this operation count based on the number of elements per row of the matrix. If the original matrix has on average $nrow$ nonzero elements per row and the preconditioned matrix has on average $pnrow$ elements, then, for our implementation of Orthomin, the following table gives the costs for a matrix of dimension neq, where the cost of diagonal scaling is included in all cases.

Operation Counts For The Iterative Method.

Scheme	Initialization	k iterations
no pre	$(2nrow + 4)neq$	$(2nrow + 11)neq\,k + 3neq\,k(k-1)$
pre	$(2pnrow + 4)neq$	$(4pnrow + 12)neq\,k + 3neq\,k(k-1)$

For example, for the heat equation $u_t = u_{xx} + u_{yy} + u_{zz}$ in three space dimensions and a centered finite difference spatial discretization, $nrow = 7$, for ILU(0), $pnrow = 7$ and for ILU(1), $pnrow = 13$. Using these values, we can calculate the relative costs for different iteration counts shown in Table 1. We note that the iterations with ILU(0) preconditioning are only about 40% more expensive than those that use only diagonal scaling. Of course, the ILU-preconditioned iterations incur the extra cost of doing

TABLE 1
Relative cost of iterative methods applied to the heat equation

Iterations	Scaling	ILU(0)	ILU(1)
1	43 neq	58 neq	94 neq
2	74 neq	104 neq	164 neq
3	111 neq	156 neq	240 neq
4	154 neq	204 neq	322 neq
5	203 neq	278 neq	410 neq

the ILU factorization, $O(pnrow^2 \cdot neq)$ floating point operations. This cost is roughly that of doing a few linear iterations. Since we are using a chord-Newton iteration and the factorization is only done when found necessary by the ODE solver, this cost is small compared to the total cost of linear iterations. We do not include here the cost of forming the matrix W_n^l since this is done in all cases. If the cheap preconditioning strategy also avoided the formation of W_n^l, then forming W_n^l would be an additional cost of using the more powerful preconditioner.

In one single solve of a linear system, a preconditioned method will be cheaper than an unpreconditioned one if the number of linear iterations is reduced sufficiently by preconditioning. The necessary reduction can be shown clearly, as in the tables above. Over the course of the integration of an ODE, the trade-off is more involved. The basic strategy for a general-purpose ODE solver must be to use a preconditioned iterative method for robustness. By substituting an unpreconditioned iteration in some cases, we hope to reduce the total cost of linear algebra but expect to see the total number of linear iterations increase, as a few expensive iterations are replaced by a larger number of cheaper ones.

We address the question of switching from diagonal scaling to the more powerful preconditioner first. Diagonal scaling is used at the start of the integration, since most ODE codes begin time-stepping with a small stepsize and it is usual to find an initial region of the integration where the stepsize remains small due to the accuracy requirement. The progress of the integration is then monitored with the aim of switching to a more powerful preconditioner when diagonal scaling appears to require too many iterations. An obvious strategy for assessing when the diagonally-scaled method is using "too many" iterations is to wait until the iteration fails to meet the convergence criterion in the maximum number of iterations. Frequently, this maximum number is set fairly small — for example, Brown and Hindmarsh [5] found 5 iterations to work well. In this case, switching preconditioning on when the diagonally-scaled iteration "fails" is not unreasonable. In our tests, we have used a maximum value of 10 iterations to try to assess the switching strategy independently of convergence failures. We have found that the methods are more efficient if preconditioning is switched on when the diagonally-scaled method is taking only a few (e.g., 2 to 4) iterations per solve.

It turns out to be straightforward to assess the number of linear iterations that the method uses per solve. We have observed in practice that the number of linear iterations per Newton iteration tends to remain constant over several Newton iterations. That is, typically only one linear iteration per Newton iteration is required at the start of an integration. This number increases to two at some point as the stepsize increases, remains steady at two for a while, then increases to three, and so on. Consequently, it is possible to estimate the number of linear iterations per solve by taking the average over the last few Newton iterations.

We have not attempted to use any theoretical results to predict the reduction in the number of linear iterations when preconditioning is applied. Such results are available for some preconditioners and iterative methods but they give upper bounds on the number of iterations required to reduce the residual to some fraction of its initial size. Since our convergence criterion, discussed in §2, only requires reducing the residual to a fixed value and since we expect to use only a few iterations per solve, the theoretical results are not particularly useful here.

When a differential equation yields linear systems with clustered eigenvalues, unpreconditioned and diagonally-scaled iterative methods can be very effective. In [5], the authors use a reaction-diffusion system with two species in three space dimensions as a test problem. The differential equations have the form

$$\frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c^1, c^2), \quad i = 1, 2, \tag{5}$$

$$d_1 = 0.05, \quad d_2 = 1.0$$

$$f^1(c^1, c^2) = c^1(b_1 - a_{11}c^1 - a_{12}c^2), \quad f^2(c^1, c^2) = c^2(b_2 - a_{21}c^1 - a_{22}c^2),$$

$$a_{11} = 10^6, \ a_{12} = 1, \ a_{21} = 10^6 - 1, \ a_{22} = 10^6,$$

$$b_1 = b_2 = (1 + \alpha xyz)(10^6 - 1 + 10^{-6}).$$

The equations are defined on the unit cube with $t \in [0, 10]$, homogeneous Neumann boundary conditions and initial conditions

$$\begin{aligned} c^1(x, y, z, 0) &= 500 + 250\cos(\pi x)\cos(3\pi y)\cos(10\pi z), \\ c^2(x, y, z, 0) &= 200 + 150\cos(10\pi x)\cos(\pi y)\cos(3\pi z). \end{aligned}$$

The authors point out that, as $t \to \infty$, the solution approaches a steady state which is given roughly by the asymptotic solution of the problem without diffusion, namely,

$$c^1 = (1 - 10^{-6})(1 + \alpha xyz), \quad c^2 = 10^{-6}(1 + \alpha xyz).$$

The spatial derivatives are discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $2(m+1)^3$ ODEs. Near the steady state, the interaction terms dominate the Jacobian and the dominant part of the spectrum is clustered in the interval -10^6 to $-10^6(1+\alpha)$. Brown and Hindmarsh [5] point out that, consequently, unpreconditioned iterative methods are expected to work well unless α is large. It does not necessarily follow that diagonal scaling will work well, but, in this case, it does since all eigenvalues of the scaled matrix are of moderate size. In [5], the ODE problem is solved using a relative error tolerance of 10^{-6} and an absolute error tolerance of 10^{-8}. We use the same tolerances here, with the result that the code takes a very small stepsize (starting around 10^{-9}) through the transient region, which lasts to about $t = 0.5$. There is then an abrupt transition to the steady-state region, during which the stepsize increases by factors of two to five several times in rapid succession. The diagonally-scaled method encounters difficulty once the transition to the steady-state region starts. Numerical results are shown in Tables 2 and 3 for the case $m = 9$ (2000 equations) and $\alpha = 0$. The maximum number of linear iterations was set to 10.

Table 2 shows the number of function evaluations (NFCN), Jacobian evaluations (NJAC), Newton iterations (NNWT), linear iterations (NLIN) and the time in seconds (TIME) required to integrate the problem without adaptation of the preconditioning, using either diagonal, ILU(0) or ILU(1) preconditioning. Table 3 shows the effect of switching from diagonal scaling to ILU(0) or ILU(1) preconditioning during the integration. Each column corresponds to the number of linear iterations being used by the diagonally-scaled method when the switch was made. This number was calculated by looking at the average number of iterations over the last four Newton iterations. The column labelled MAXIT indicates that the switch was made because the diagonally-scaled method failed to converge in the maximum number of iterations. The criterion used in the program was to switch if the code took an average of eight linear iterations over the past four Newton iterations, but a convergence failure occurred in each case before this condition was met. The row of the table labelled TSWI shows the elapsed CPU time when the code switched on the preconditioning.

TABLE 2
Reaction-Diffusion Equation in 3D — No Switching

	Diagonal	ILU(0)	ILU(1)
NFCN	1490	1466	1486
NJAC	71	67	68
NNWT	617	641	649
NLIN	1005	737	712
TIME	320	322	352

TABLE 3
Reaction-Diffusion Equation in 3D — Switching

	ILU(0)				ILU(1)			
	2	4	6	MAXIT	2	4	6	MAXIT
NFCN	1402	1390	1402	1417	1403	1404	1416	1416
NJAC	65	64	65	66	65	65	66	66
NNWT	601	601	601	604	602	603	603	603
NLIN	733	773	796	848	679	743	773	814
TSWI	217	243	253	263	217	243	256	266
TIME	278	280	284	295	280	281	288	293

From these tables, we see that there is a definite advantage to using the combination of the diagonally-scaled and preconditioned iterative methods, as most of the CPU time is spent with the diagonally-scaled iteration. This reflects the fact that 75% to 90% of the work of the integration, measured in function evaluations, Jacobian evaluations, or nonlinear or linear iterations is done using that diagonally-scaled method. There is little to choose between switching criteria in terms of total time, suggesting that there is a small range of the integration in which the diagonally-scaled and preconditioned methods are equally expensive.

In the following tables, we show some numerical results obtained using the heat equation in three space dimensions as a test problem. It is well-known that the eigenvalues of the associated Jacobian matrix are widely spread without clustering and that unpreconditioned and diagonally-scaled iterative methods are generally inefficient. As described in [8], the PDE has the form

$$u_t = u_{xx} + u_{yy} + u_{zz}, \tag{6}$$

on the unit square with homogeneous Dirichlet boundary conditions, $t \in [0, 10.24]$ and initial condition

$$u(0, x, y, z) = 64x(1-x)y(1-y)z(1-z).$$

The spatial derivatives were discretized using a centered second-order finite difference discretization on a mesh of m equal subdivisions, resulting in a system of $(m-1)^3$ ODEs. In this example, $m = 16$ (3375 equations). The maximum number of linear iterations was set to 10. The absolute error tolerance in the ODE solver was set to 10^{-4} and no relative error control was used. Table 4 shows the results without adaptation of the preconditioner and Table 5 shows the effect of switching preconditioners. The number of linear iterations was calculated as the average over the last four Newton iterations.

Table 4
Heat Equation — No Switching

	Diagonal	ILU(0)	ILU(1)
NFCN	356	373	326
NJAC	19	20	17
NNWT	117	122	113
NLIN	400	200	140
TIME	130	107	94

Table 5
Heat Equation — Switching

	ILU(0)				ILU(1)			
	2	4	6	MAXIT	2	4	6	MAXIT
NFCN	357	358	369	372	373	360	369	373
NJAC	19	19	20	20	20	19	20	20
NNWT	118	119	118	121	122	121	118	122
NLIN	189	244	320	359	170	212	306	340
TSWI	22	48	90	100	22	48	90	100
TIME	98	106	118	127	102	103	117	125

The most efficient way to solve this particular problem is to use the ILU(1) preconditioner over all iterations. This seems to be due, at least in part, to accuracy considerations related to the use of the diagonally-scaled iterative solver. Following Brown and Hindmarsh [5], the solution of the linear system is accepted if

$$\|d_n^l\| \leq \epsilon,$$

where d_n^l is the residual at the last linear iteration. With two different iterative methods, it often happens that one method just meets this criterion while the other returns a solution with a much smaller residual. This difference in accuracy affects not only the acceptance of the Newton iteration but also the error control strategy and stepsize selection in the ODE solver. In the tables, we note that, as expected, the total number of linear iterations increases as we use the diagonally-scaled method for a longer period. The other statistics are more variable because of this effect of the accuracy of the linear solver.

For the ILU(0) preconditioning, the approaches that switch when the diagonally-scaled method is taking two or four iterations per solve are no worse than using preconditioning on all steps. Switching always decreases performance with the ILU(1) preconditioning.

For both the reaction-diffusion problem (5) and the heat equation (6), we have tested the effect of averaging the number of linear iterations over the last six and the last eight Newton iterations. Changing this parameter has little effect on the step at which preconditioning is switched on.

Based on the results from these two test problems and additional numerical experiments, it seems reasonable to use the following strategy to switch from a diagonally-scaled method to a preconditioned one:

- Find the average number of linear iterations over the past four Newton iterations.
- Switch on preconditioning when the average is four or greater, and definitely when the diagonally-scaled iteration fails to converge in the maximum number of iterations.

This approach yields a significant improvement in performance sometimes; for those problems where it is less efficient, it does not degrade performance very much. The particular value of *four* linear iterations obviously depends on the relative costs of the preconditioning strategies and hence is dependent on the problem class but does seem appropriate for the PDE problems that we have tested.

For either the reaction-diffusion equation (5) or the heat equation (6), we observe that the number of linear iterations per solve is reduced when preconditioning is first switched on. As the integration continues with the preconditioned method, the number of iterations per solve increases again, along with the timestep, as the solution of the differential equation approaches a steady state. This is typical behaviour for many time-dependent systems, but problems exist for which phases similar to the initial transient recur. For such problems, it is reasonable to consider switching preconditioning off. We have developed a test to switch preconditioning off based on both the amount of work being done by the linear solver and the character of the iteration matrix. As the primary test, we require that the iterative solver takes one iteration only per solve over the last several Newton iterations. Around 15 to 20 Newton iterations seems appropriate since we wish to avoid cases where, say, the error estimate has forced an anomalously small stepsize over a few steps. This also avoids switching preconditioning off again immediately after switching it on.

A quantity relevant to convergence of a Krylov-subspace type iterative method is the *spectral ratio* — the ratio of the largest to the smallest eigenvalues of the matrix or, for clustered eigenvalues, the ratio of largest to smallest eigenvalues within each cluster. We use Gerschgorin circles to estimate the clustering of the eigenvalues of the Newton iteration matrix. We need consider one cluster only since all the Gerschgorin circles of the diagonally-scaled matrix will have center $(1,0)$. We also observe that the matrices $D^{-1}W_n^l$, $W_n^l D^{-1}$ and $D^{-1/2}W_n^l D^{-1/2}$ are similar, hence have the same eigenvalues, but need not have the same Gerschgorin circles. ($D = \mathrm{diag}(W_n^l)$ is the diagonal scaling matrix.) When the Newton iteration matrix is formed by the ODE solver, the Gerschgorin circles for these three matrices are calculated by both rows and columns, giving six estimates of the eigenvalue range. If all the circles for a single case (particular scaling and row or column calculation) lie in the same half plane, we call the ratio of the extreme real axis intercepts of the Gerschgorin circles the *Gerschgorin ratio* of the matrix. We take the minimum of our six calculated Gerschgorin ratios and use it to approximate the associated spectral ratio. When this value is small, it is reasonable to expect that an unpreconditioned iterative method will work well.

As the theory suggests, we have found for all our test problems that, at the start of the integration, the eigenvalues of the diagonally-scaled iteration matrix are all contained in one small cluster centered at $(1,0)$. As the stepsize increases, the cluster usually expands. If any circle crosses the axis into the left half plane, we consider the Gerschgorin ratio to be infinite. Here, we expect that the diagonally-scaled iterative method will have difficulty and therefore we do not switch preconditioning off.

Van der Pol's equation (see, for example, [14]) is often used as an example of an ODE system in which transients recur throughout the integration. This second-order equation is frequently rewritten as system of two first-order ODEs. We have extended it to a large system by using the two ODEs as the reaction terms in a reaction-diffusion PDE system, as follows.

$$\frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c^1, c^2), \quad i = 1, 2, \tag{7}$$

$$d_1 = 0.05, \quad d_2 = 1.0\ ,$$

$$f^1(c^1, c^2) = c^2, \quad f^2(c^1, c^2) = \eta(1 - (c^1)^2)c^2 - c^1,$$

on the unit square with $\eta = 100$, homogeneous Neumann boundary conditions and initial conditions

$$\begin{aligned} c^1(x, y, 0) &= 1 - \cos(\pi x)\cos(2\pi y), \\ c^2(x, y, 0) &= 1 + 2\cos(2\pi x)\cos(\pi y). \end{aligned}$$

The spatial derivatives were discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $2(m+1)^2$ ODEs.

We solved this problem with $m = 10$ subdivisions, $t \in [0, 1000]$, relative and absolute error tolerances of 10^{-6} and 10^{-4} respectively and various switching strategies. As noted earlier, to be conservative about switching preconditioning off, it seems most appropriate to observe the number of linear iterations over a larger number of Newton iterations than is used in the test to switch it on. We require that the linear solver has taken only one iteration per solve over the last 16 Newton iterations before we consider switching preconditioning off. If we use this criterion only, and no information about the iteration matrices, the code switches preconditioning on 14 times and off 13 times in the interval $[0, 1000]$.

To use information about the matrix, a bound on the Gerschgorin ratio is chosen. If the smallest Gerschgorin ratio is less than the bound, then we switch preconditioning off if the requirement on number of iterations per solve is also met. We solved Van der Pol's equation again with the Gerschgorin bound set to two and observed the same number of switches as above. Comparing results from the two tests shows that the switches occur at about the same times in the integration, a satisfactory result since they should be triggered by the behaviour of the continuous system rather than being artifacts of the code. However, we observe that, without matrix information, the code switches preconditioning off as much as one hundred timesteps earlier than when matrix information is used. The earlier switch seems satisfactory in the sense that the diagonally-scaled method experiences no difficulties after the switch, converging in one or two iterations. This result suggests that using a larger Gerschgorin bound might be appropriate, a conclusion supported by tests using values of four and eight that also yield satisfactory behaviour of the switching strategy. With the value eight, we observe the same results as when no matrix information is used — that is, the condition on number of iterations controls the switch. There is no significant difference in total solution time in any case, including a comparison to doing the whole integration with preconditioning (ILU(1) preconditioning was used in all tests). This is due to the small size of the ODE system. We note that the Gerschgorin ratio in the right half plane grew as large as 10^3 and that some circles extended into the left half plane on several Jacobian evaluations.

With $m = 20$ subdivisions (882 equations), $t \in [0, 500]$ and the same error tolerances, the integration time using ILU(1) preconditioning on all iterations was 742 seconds, with 452 seconds spent in linear algebra. The time when switching was used with the Gerschgorin bound set to four was 722 seconds, including 435 seconds of linear algebra. The code switched preconditioning on seven times and off six times. Similar results were obtained when the Gerschgorin bound was set to two.

As the mesh is refined, the diffusion term in the PDE causes the ODE system to become stiffer. The results shown in Table 6 are for a grid with $m = 40$ subdivisions (3362 equations), $t \in [0, 500]$ and the same error tolerances. The code switched ILU(1) preconditioning on if the diagonally-scaled method was taking four iterations per solve over the last four Newton iterations, and off if the ILU(1)-preconditioned method took only one iteration per solve over the last 16 Newton iterations. The Gerschgorin bound (G.B.) used in each test is given at the top of the column. Here, the most

TABLE 6

Van der Pol Equation — Switching on and off

	No switches	G.B. 2	G.B. 4
NFCN	7447	7967	8129
NJAC	341	376	375
NNWT	3956	4118	4290
NLIN	9496	10474	11214
TIME	4365	4569	4597

efficient solution technique is to use ILU(1) preconditioning on all steps. With the Gerschgorin bound set to two, preconditioning was switched on four times and off three times; with bound four, the code switched preconditioning on seven times and off six times. The grid size is sufficiently small that the diffusion terms keep the ODE stiff and preconditioning on all steps is the most efficient technique.

For the Van der Pol equation, we expect to encounter recurring transients during the integration, causing the ODE solver to use a small timestep to meet the accuracy requirement. Due to this small timestep, all the eigenvalues of the unscaled Newton iteration matrix will be of moderate size and will occur in one single cluster. Checking the Gerschgorin circles of the unscaled matrix confirms that in the tests above, whenever the code switched preconditioning off, the circles of the unscaled matrix made up one small cluster. This is not the case for the following test problem, used in [5] and [7]. This PDE system, also of reaction-diffusion type, models ozone production in the stratosphere and has the following form.

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z}\left(K_v(z)\frac{\partial c^i}{\partial z}\right) - V_h \frac{\partial c^i}{\partial x} + R^i(c^1, c^2, t), \quad i = 1, 2, \tag{8}$$

$$K_h = 4 \cdot 10^{-6},\ K_v(z) = 10^{-8} e^{z/5},\ V_h = 0.01,$$

$$R^1(c^1, c^2, t) = -k_1 c^1 - k_2 c^1 c^2 + k_3(t) \cdot 7.4 \cdot 10^{16} + k_4(t) c^2,$$

$$R^2(c^1, c^2, t) = k_1 c^1 - k_2 c^1 c^2 - k_4(t) c^2,$$

$$k_1 = 6.031,\ k_2 = 4.66 \cdot 10^{-16},$$

$$k_3(t) = \begin{cases} \exp[-22.62/\sin(\pi t/43200)] & \text{for } t < 43200 \\ 0 & \text{otherwise} \end{cases}$$

$$k_4(t) = \begin{cases} \exp[-7.601/\sin(\pi t/43200)] & \text{for } t < 43200 \\ 0 & \text{otherwise} \end{cases}$$

The problem is posed for $x \in [0, 20]$, $z \in [30, 50]$ and $t \in [0, 86400]$ with homogeneous Neumann boundary conditions and initial conditions

$$c^1(x, z, 0) = 10^6 \alpha(x)\beta(z),\ \ c^2(x, z, 0) = 10^{12}\alpha(x)\beta(z),$$

$$\begin{aligned} \alpha(x) &= 1 - (0.1x - 1)^2 + (0.1x - 1)^4/2, \\ \beta(z) &= 1 - (0.1z - 4)^2 + (0.1z - 4)^4/2. \end{aligned}$$

If the differential equations are discretized using central differences for both the diffusion and convection terms, as is done in [5] and [7], the eigenvalues of the Jacobian are found in two clusters, as discussed in [5].

The problem was solved with $m = 9$ subdivisions (200 equations) and relative and absolute error tolerances of 10^{-5} and 10^{-3} respectively. With the Gerschgorin bound set to two, the code switches preconditioning on at $t = 13340$, then off at $t = 42064$ and completes the integration successfully using the diagonally-scaled method. Checking the Gerschgorin circles of the unscaled matrix reveals that when preconditioning is switched off, the circles make up two small clusters although the ratio of the extreme limits of all the circles is over 100. In this case, the diagonal scaling has mapped these two clusters into a single small circle. With the bound set to four, the code switches preconditioning on then off twice between $t = 13340$ and $t = 35002$, finishing the integration with the diagonally-scaled method. It is less efficient than when the bound is set to two.

The results in Table 7 are obtained with a discretization having $m = 19$ subdivisions (800 equations). The ODE solver has some difficulty with the integration, probably due to the central difference discretization since it does not occur when upstream differencing is used for the convection term. With the Gerschgorin bound set to either two or four, the code switches preconditioning on five times and off five times between $t = 7734$ and t around 42700; the switches occur at slightly different times depending on the bound. When preconditioning is switched off at $t = 42700$, the Gerschgorin circles of the unscaled Newton iteration matrix form two small clusters. The earlier switches occur because the timestep is small enough, often after a failed step, to force all the circles of the unscaled matrix into one cluster. It is not clear if these switches should be made; the timestep tends to increase rapidly after a failed step and the possible higher accuracy of the more powerful preconditioner might be useful.

TABLE 7
Ozone Production Model — 800 equations

	No switches	G.B. 2	G.B. 4
NFCN	6001	6267	6179
NJAC	297	320	310
NNWT	3552	3606	3612
NLIN	3552	4985	4865
TIME	654	657	650

The results in Table 8 are obtained with $m = 39$ subdivisions (3200 equations). When the Gerschgorin bound is set to two, the code switches preconditioning on at $t = 7216$ then off at $t = 42321$. With bound four, there are four on-off switches between $t = 7216$ and $t = 42293$ and the integration again finishes using the diagonally-scaled method. In this case, it is more efficient to switch preconditioning off when possible.

TABLE 8
Ozone Production Model — 3200 equations

	No switches	G.B. 2	G.B. 4
NFCN	7713	7686	7766
NJAC	307	318	314
NNWT	5182	5065	5173
NLIN	5182	7490	7874
TIME	4559	4113	4168

Overall, it seems that our strategy for switching preconditioning off does detect problem information correctly, i.e., nonstiffness of the Van der Pol oscillator. Also, the diagonal scaling is able to exploit the clustering of the eigenvalues in Equation (8). Computing the six different Gerschgorin ratios can have an effect on when the switches occur. For the Van der Pol equation, all six values tend to be close on all Jacobian evaluations. For Equation (8), where the components are of widely differing magnitudes, the six values can be very different particularly when they are large. When a Gerschgorin bound of two was used to switch preconditioning off, we found that the ratios were close when a switch occurred. In contrast, with a bound of four, some switches would not have occurred if all six values had not been computed. Any strategy should certainly be conservative about making a switch. A bound of two seems a better choice, although the value four yields better results in some cases.

4. Adaptive Preconditioning in Type-Insensitive Codes. In this section, we insert our strategies for adaptive preconditioning into the type-insensitive ODE solver described in §2. We report the results of numerical experiments with several test problems. In these tests, the maximum number of linear iterations is always set to 10. Preconditioning is switched on if the diagonally-scaled method took four iterations per solve over the last four Newton iterations and off if the Gerschgorin bound (2, unless otherwise specified) is met and only one iteration per solve was required over the last 16 Newton iterations.

In the tables, the preconditioning strategy (either ILU(0) or ILU(1)) and number of equations are shown in the caption. The column labels have the following meanings:

- No switch: BDF method; ILU preconditioning on all iterations.
- Switch: BDF method; switching from diagonal scaling to ILU preconditioning.
- Type-ins: Type-insensitive method; ILU preconditioning on all iterations.
- Combined: Type-insensitive method; switching from diagonal scaling to ILU preconditioning.

4.1. Reaction-diffusion equation in three space dimensions. Equation (5) was solved on a number of different grids with $\alpha = 100$ and $t \in [0, 100]$. All other problem and method parameters are the same as in §3. When $\alpha = 100$, the eigenvalues of the Jacobian are somewhat more widely spread than in the tests reported in §3. Extending the range of integration from $t = 10$ to $t = 100$ simply causes the code to take one large last step, of order 100 itself. This large step poses no problem for the iterative solver in this case.

TABLE 9
ILU(0) preconditioning — 2000 equations

	No switch	Switch	Type-ins	Combined
NFCN	1414	1335	1448	1444
NJAC	65	60	44	44
NNWT	613	596	350	346
NLIN	653	726	391	449
TIME	298	262	237	226

We always find that the method that combines the type-insensitive approach with adaptive preconditioning is the most efficient, saving 18 to 24% of execution time with ILU(0) preconditioning and 26 to 31% with ILU(1) preconditioning. Without adaptive preconditioning, the ILU(0) method is more efficient, but when preconditioning is applied only when necessary, it is as efficient to use the more powerful ILU(1) approach.

4.2. Ozone production model. In §3, equation (8) was discretized using centered differences for both the diffusion and convection terms, following [5] and [7].

Table 10
ILU(1) preconditioning — 2000 equations

	No switch	Switch	Type-ins	Combined
NFCN	1414	1334	1439	1444
NJAC	65	60	42	44
NNWT	613	595	365	346
NLIN	631	697	379	437
TIME	328	259	260	227

Table 11
ILU(0) preconditioning — 5488 equations

	No switch	Switch	Type-ins	Combined
NFCN	1365	1378	1278	1270
NJAC	62	64	44	44
NNWT	602	589	400	392
NLIN	674	722	465	530
TIME	869	798	725	670

Table 12
ILU(1) preconditioning — 5488 equations

	No switch	Switch	Type-ins	Combined
NFCN	1365	1378	1290	1270
NJAC	62	64	44	44
NNWT	602	589	412	392
NLIN	643	698	445	501
TIME	966	801	763	667

Table 13
ILU(0) preconditioning — 16000 equations

	No switch	Switch	Type-ins	Combined
NFCN	1401	1363	1568	1527
NJAC	64	60	43	42
NNWT	612	624	372	343
NLIN	744	849	511	556
TIME	2895	2698	2541	2365

Table 14
ILU(1) preconditioning — 16000 equations

	No switch	Switch	Type-ins	Combined
NFCN	1413	1362	1568	1539
NJAC	65	60	43	43
NNWT	612	623	372	343
NLIN	694	786	443	505
TIME	3158	2668	2662	2342

Since the convection coefficient $V_h = 0.01$ while the diffusion coefficients have magnitude 10^{-6} to 10^{-4}, this problem is convection-dominated and it is reasonable to use upstream differencing for the convection term. This discretization leads to a Jacobian matrix that is better conditioned than when centered differences are used and allows the code to take much larger timesteps. We do not expect that preconditioning will be switched off once it has been turned on. Upstream differencing was used in computing the results in Tables 15 and 16. The equations were discretized on a grid with 41 mesh points in each direction, giving $\Delta x = \Delta z = 0.5$ and a system of 3362 equations. As before, an absolute error tolerance of 10^{-3} and a relative tolerance of 10^{-5} were used.

TABLE 15
ILU(0) preconditioning — upstream differences — 3362 equations

	No switch	Switch	Type-ins	Combined
NFCN	1508	1566	1533	1375
NJAC	78	81	70	62
NNWT	705	731	599	523
NLIN	954	1083	826	773
TIME	677	696	652	574

TABLE 16
ILU(1) preconditioning — upstream differences — 3362 equations

	No switch	Switch	Type-ins	Combined
NFCN	1276	1614	1359	1350
NJAC	61	86	57	59
NNWT	647	729	557	528
NLIN	837	1004	800	775
TIME	613	709	618	590

Again, it is found that the method that combines the type-insensitive approach with adaptive preconditioning is the most efficient, although the saving is not as significant as for Equation (5). The method with ILU(1) preconditioning on all steps does well, taking fewer function evaluations than any other approach. As mentioned previously, this seems to indicate that there is an advantage in accuracy to using ILU(1) preconditioning. Adaptive preconditioning alone degrades efficiency although it does well in combination with the type-insensitive method. With the combined method, the switch from the Adams formulas to the BDFs occurs at step 93, $t = 3.93$, then preconditioning is switched on at step 140, $t = 1142$. With adaptive preconditioning only, the switch occurs later at step 189, $t = 2167$.

In Table 17, we show results computed with the type-insensitive and combined methods and a central difference discretization of all terms, for comparison with results given in §3. (Those results are also included in this table.) In this case, the switching method and the combined method have comparable cost. In the switching method, preconditioning is turned on at $t = 7216$ then off at $t = 42321$. In the combined method, the BDFs are selected as above at $t = 3.93$, step 93; preconditioning is turned on at $t = 5837$ and off at $t = 42360$.

4.3. Van der Pol Equation. We use the modified Van der Pol Equation (7) to investigate the interaction between turning preconditioning on and off and switching

TABLE 17
ILU(1) preconditioning — centered differences — 3200 equations

	No switch	Switch	Type-ins	Combined
NFCN	7713	7686	7933	7668
NJAC	307	318	311	296
NNWT	5182	5065	5157	5016
NLIN	5182	7490	5157	7347
TIME	4559	4113	4863	4146

from the BDFs back to the Adams formulas. As we noted in §3, on a grid with $m = 40$ subdivisions (3362 equations) the ODE system is quite stiff due to the effect of the diffusion terms regardless of the behaviour of the reaction terms. The results shown in Table 18 were computed using the same problem and method parameters as in §3. G.B. denotes the Gerschgorin bound.

TABLE 18
ILU(1) preconditioning — 3362 equations

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	7447	7967	8129	7840	7521	8054
NJAC	341	376	375	353	334	359
NNWT	3956	4118	4290	4015	3890	4169
NLIN	9496	10474	11214	9608	9073	11198
TIME	4365	4569	4597	4485	4267	4639

In the "switching" code, preconditioning is first turned on at time $t = 0.025$, step 65. Since the total integration takes around 4000 steps, this is very early in the integration. With the Gerschgorin bound set to two, preconditioning is switched on 4 times and off 3 times; with bound four, the code switches preconditioning on 7 times and off 6 times during the integration. The type-insensitive method is less effective than the code that uses ILU(1) preconditioning throughout, somewhat surprisingly since the type-insensitive code switches from the Adams formulas to the BDFs at time $t = 0.0076$, step 85 and never switches back to the Adams formulas. The combined method with Gerschgorin bound two turns preconditioning on at $t = 0.026$, step 115, and never switches it off. With the bound set to four, the combined method makes the same number of switches as the "switching" code, at essentially the same times in the integration.

This problem is best solved with ILU(1) preconditioning. The same set of tests were run using ILU(0) preconditioning and the same relative performances were observed. The total time to complete the integration was about 1000 seconds greater in all cases.

To investigate the effect of switching between the Adams formulas and the BDFs when the number of equations is large, the diffusion coefficients were reduced from 0.05 and 1 to 0.005 and 0.1. The numerical results using ILU(0) preconditioning are given in Table 19, those using ILU(1) preconditioning in Table 20. Other parameters of the problem and method are the same as in §3.

With ILU(0) preconditioning, the "switching" method and the combined method

Table 19
ILU(0) preconditioning — 3362 equations — small diffusion

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	9503	9363	9579	9918	9930	10129
NJAC	414	398	408	275	262	275
NNWT	5264	5288	5402	3171	2941	2990
NLIN	7964	8517	8792	5889	5849	6129
TIME	4216	4048	4073	3492	3343	3391

turn preconditioning on 7 times and off 6 times for both choices of Gerschgorin bound. The type-insensitive method selects the BDFs for the first time at $t = .709$, step 1285, where the total integration still takes around 4000 steps. Following the initial formula change, there are three subsequent switches to the Adams formulas then back to the BDFs. The combined method makes four subsequent switches between Adams formulas and BDFs. Three of these switches are at essentially the same values of t as the type-insensitive method but there is one additional switch. There are two switches between preconditioning and diagonal scaling where the Adams formulas are not selected. A switch to the Adams formulas is always preceded by switching preconditioning off. This order is due only to the switching strategies already described — the code could allow a switch directly from the BDFs with preconditioning to the Adams formulas. In fact, the switch from preconditioning to diagonal scaling typically precedes the switch from the BDFs to the Adams formulas by about 80 to 100 timesteps, indicating that the method detects a region of "mild" stiffness in the transition from a stiff regime to a transient regime.

Table 20
ILU(1) preconditioning — 3362 equations — small diffusion

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	9519	9605	9612	10049	10041	9966
NJAC	417	411	413	288	275	266
NNWT	5250	5398	5385	3302	2954	2939
NLIN	6547	7472	7761	4703	4570	4934
TIME	4028	3879	3853	3360	3132	3096

When ILU(1) preconditioning is used, the "switching" method and combined method still turn preconditioning on 7 times and off 6 times for both choices of Gerschgorin bound. The type-insensitive method of course selects the BDFs for the first time at the same timestep as in the ILU(0) case, then there are four subsequent switches to the Adams formulas then back to the BDFs. The combined method makes the same switches, with either Gerschgorin bound. A switch to the Adams formulas is always preceded by switching preconditioning off; the switches occur at essentially the same times for ILU(0) preconditioning.

The strategy for formula selection appears to be robust with respect to effects of the linear algebra, as one would hope. Also, the switches between diagonal scaling and preconditioning always occur at about the same t values, indicating that the switching strategy is, in fact, detecting properties of the ODE system. It is still difficult to choose a value for the Gerschgorin bound based on these tests. As noted in §3, the value two appears preferable since it gives a saving whenever one is achieved by switching preconditioning and does better when a saving is not achieved.

4.4. A "Dense" Two-dimensional Problem. The system of PDEs defined by (9) below is used as a test problem in [6]. Although this problem is similar in nature to (5) — both are reaction-diffusion systems modelling a predator-prey interaction — it poses a slightly different challenge to the iterative methods since it includes a large number of species, making the associated Jacobian matrix relatively dense.

$$\frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c), \quad i = 1, 2, \ldots, s, \tag{9}$$

where $c = (c^1,\ c^2, \ldots, c^s)^T$, $p = s/2$

$$\begin{aligned}
d_i &= 1, \quad i = 1, 2, \ldots, p, \\
d_i &= 0.05, \quad i = p+1, \ldots, s, \\
f^i(c) &= c^i(b_i + \sum_{j=1}^{s} a_{ij} c^j), \\
a_{ii} &= 1, \forall i \\
a_{ij} &= -0.5 \cdot 10^{-6}, \ i \le p,\ j > p, \\
a_{ij} &= 10^4, \ i > p,\ j \le p, \\
a_{ij} &= 0, \text{ for all other } i \text{ and } j; \\
b_i &= (1 + \alpha x y z), \ i \le p, \\
b_i &= -(1 + \alpha x y z), \ i > p.
\end{aligned}$$

The system (9) is defined on the unit square with $t \in [0, 10]$. The problem has homogeneous Neumann boundary conditions and initial conditions

$$c^i(x, y, 0) = 10 + i[16x(1-x)y(1-y)]^2, \quad 1 \le i \le s.$$

Here, we take $s = 20$ ($p = 10$) and $\alpha = 50$. The steady state solution is spatially inhomogeneous and the Jacobian is highly nonsymmetric at equilibrium [6].

The spatial derivatives are discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $s(m+1)^2$ ODEs. Following [6], we take $m = 11$, which gives an ODE system of 2880 equations. This problem is solved using a relative error tolerance of 10^{-6} and an absolute error tolerance of 10^{-8}. Numerical results for ILU(0) and ILU(1) preconditioning are given in Tables 21 and 22 respectively.

The approaches that use ILU(0) preconditioning are always more efficient than those using ILU(1) preconditioning, due to the density of the Jacobian matrix, which has 42,240 nonzero entries, as does the ILU(0) preconditioner, while the ILU(1) preconditioner has 112,840 nonzeros. Hence, even though the code using ILU(1) preconditioning generally uses fewer function and Jacobian evaluations, nonlinear and linear iterations, the total CPU time is greater. In both cases, the combined method is the most efficient.

Brown and Hindmarsh [6] identify the nonstiff transient region for this problem as roughly $0 \le t \le 10^{-3}$. Our strategy for switching on preconditioning also picks

TABLE 21
ILU(0) preconditioning — 2880 equations

	No switch	Switch	Type-ins	Combined
NFCN	1614	1515	1388	1330
NJAC	43	40	31	29
NNWT	395	380	203	201
NLIN	780	755	655	621
TIME	507	449	416	384

TABLE 22
ILU(1) preconditioning — 2880 equations

	No switch	Switch	Type-ins	Combined
NFCN	1451	1592	1257	1173
NJAC	38	42	27	24
NNWT	372	401	184	186
NLIN	586	638	397	496
TIME	570	507	422	418

out this region. While the switch from the Adams formulas to the BDFs takes place at $t = 2.88 \cdot 10^{-5}$, the switch from diagonal scaling to preconditioning takes place at $t = 2.45 \cdot 10^{-3}$ when the "switching" code is used and at $t = 2.76 \cdot 10^{-3}$ in the combined code.

The ratio of the number of linear iterations to the number of Newton iterations (AVDIM) is reported in [6]. This value provides a rough comparison of the work being done by the iterative method in the various approaches. For problem (9), it is also interesting to compare the work done in the nonstiff and stiff regimes, which we characterize by the intervals $[0, 1]$ and $[1, 10]$ for convenience. Values are reported in Table 23. As expected, AVDIM increases when diagonal scaling is used and also when the type-insensitive method is used. The iterative solver works considerably harder in the stiff regime. The values for AVDIM found here are similar to those reported in [6].

TABLE 23
AVDIM — ratio of linear to Newton iterations

		No switch	Switch	Type-ins	Combined
ILU(0)					
	$[0, 1]$	1.82	1.90	2.88	2.97
	$[1, 10]$	5.17	5.10	5.72	4.53
ILU(1)					
	$[0, 1]$	1.48	1.53	1.99	2.56
	$[1, 10]$	5.10	4.22	4.58	4.25

5. Conclusions. We have developed a strategy for the adaptive choice of preconditioning when an iterative linear solver is used in an ODE code. This strategy successfully identifies characteristics of the problem and is able to switch preconditioning on or off according to those characteristics. This approach combines naturally with the use of a type-insensitive ODE solver. For the test problems considered, we found that, when the type-insensitive approach yields a saving in execution time, the combined method increases the saving. There are problems for which the adaptive approach is not effective — in these cases, it appears that the type-insensitive approach is not appropriate either. As a simple rule of thumb, the combined method is effective when both the Adams formulas and the diagonally-scaled preconditioning are used over a significant percentage of steps.

There are a variety of ways of adapting the preconditioner that one might consider. Possible choices for the "cheap" preconditioner were discussed in the introduction. Another idea is to adapt through various more powerful preconditioners, say ILU(0) to ILU(1), possibly to a direct solution method if the ODE appears very difficult to solve. However, the difference in cost of the various preconditioning methods is not that great. As we noted earlier, for example, ILU(0) preconditioning is only about 40% more expensive than diagonal scaling on many problems derived from PDEs. This suggests that little additional saving would be achieved by this incremental approach. A "cheap" preconditioner that was effective over a longer range would be more useful. Switching to a direct method is possible if the problem size is not too great or if a sparse direct method does not suffer too much fill-in.

Both the ILU(0) and ILU(1) preconditioners worked well for these test problems. They have the advantage of being "black-box" strategies that can be applied to any matrix. The choice between them depends on the amount of fill-in generated by the ILU(1) factorization and how difficult the ODE is to solve. As we saw for the last test problem, even though ILU(1) preconditioning can be significantly more efficient in terms of the number of linear iterations, the cost per iteration may make ILU(0) preferable. We note that the choice of the maximum number of iterations can have a significant effect on the overall performance of the code. In a method such as Orthomin or GMRES, the amount of storage restricts the choice of this maximum. (Although restarting can be used, it is not clear how effective it is.) If the ODE solver fails and reduces the stepsize because the iterative method did not converge, this can have a ripple effect throughout the rest of the integration, significantly increasing the total number of steps and amount of work. The maximum should be chosen as large as possible consistent with the storage available.

REFERENCES

[1] A. Behie and P. Forsyth, *Comparison of fast iterative methods for symmetric systems*, IMA J. Numer. Anal., 3 (1983), pp. 41–63.

[2] ———, *Incomplete factorization methods for fully implicit simulation of enhanced oil recovery*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 543 – 561.

[3] M. Berzins and R. Furzeland, *A user's manual for SPRINT - a versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 - Algebraic and ordinary differential equations*, Tech. Rep. TNER.85.058, Thornton Research Centre, Shell Research Ltd., Thornton, U.K., 1985.

[4] ———, *An adaptive method for the solution of stiff and non-stiff differential equations*, tech. rep., School of Computer Studies, Leeds University, Leeds, UK, 1990.

[5] P. Brown and A. Hindmarsh, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numer. Anal., 23 (1986), pp. 610–638.

[6] ———, *Reduced storage matrix methods in stiff ODE systems*, J. Appl. Math. Comp., 31 (1989), pp. 40–91.

[7] G. Byrne, *Pragmatic experiments with Krylov methods in the stiff ODE setting*, in Proc. IMA Conference on Computational Ordinary Differential Equations, 1989, J. Cash and I. Gladwell, eds., London, 1989, Oxford University Press.

[8] T. Chan and K. 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), pp. 378 – 417.

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

[10] C. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, N.J., 1971.

[11] C. Gear and Y. Saad, *Iterative solution of linear equations in ODE codes*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 583 – 601.

[12] A. Hindmarsh, *LSODE and LSODI, two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, (1980), pp. 10–11.

[13] S. Norsett and P. Thomsen, *Switching between modified Newton and fix-point iteration for implicit ODE-solvers*, BIT, 26 (1986), pp. 339–348.

[14] L. Petzold, *Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 136 – 148.

[15] L. Shampine, *Type-insensitive ODE codes based on extrapolation methods*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 635 – 644.

[16] P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric systems*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 36 – 52.

[17] H. van de Worst and P. Sonneveld, *CGSTAB, a more smoothly converging variant of CGS*, Tech. Rep. 90 – 50, Delft University of Technology, Delft, Netherlands, 1990.

[18] P. Vinsome, *Orthomin, an iterative method for solving sparse sets of simultaneous linear equations*, in Society of Petroleum Engineers of AIME, Paper SPE 5729, 1976.