Solving Large ODE Systems Using a Reduced System Iterative Matrix Solver

by

Wendy L. Seward

Research Report CS-89-38
August, 1989

Solving Large ODE Systems Using a Reduced System Iterative Matrix Solver

W.L. Seward *

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

ABSTRACT

When the method of lines is used to solve time-dependent partial differential equations, the semidiscretization in space leads to a large stiff system of ordinary differential equations. A major part of the computational cost in solving such large systems is in the linear algebra, which leads naturally to the idea of using iterative matrix solvers in ODE software. This paper investigates the use of a powerful iterative method in the ODE solver package, SPRINT. The iterative method is based on a red-black re-ordering of the unknowns followed by a second-degree incomplete LU factorization and either conjugate gradient or Orthomin acceleration. This technique has a greater cost per iteration than some less powerful iterative methods but numerical tests indicate that the method is robust and more efficient than a direct solver for large problems. Comparisons of the iterative method and a well-known direct sparse matrix solver are made using the same ordinary differential equation software. For an ODE system arising from a three-dimensional model PDE problem, the iterative solver provides a considerable saving in execution time and storage over the direct solver. Two systems arising from two-dimensional PDEs are studied and it is shown that when the red-black re-ordering is effective the iterative solver can be cheaper than the direct solver even in 2D.

* This research was supported by the Natural Sciences and Engineering Research Council of Canada and the University of Toronto/University of Waterloo Information Technology Research Centre.

Key Words. stiff initial-value problem, iterative linear equation solver, re-ordering, ODE solver

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

Abbreviated title: Reduced System Iterative Solvers in ODE Codes

1. Introduction

Effective numerical methods for solving systems of ordinary differential equations (ODEs) are now widely available through the medium of robust efficient library software packages. These ODE codes (for example, routines in the IMSL or NAG libraries, the LSODE family (Hindmarsh (1980)) can solve a broad range of ODE problems. Because the software works well, users attempt to solve more and more problems with it; in particular, as computing power becomes cheaper, users attempt to solve larger and larger problems. This paper is concerned with the solution of large stiff systems of initial-value problems (IVPs) in ODEs, specifically problems of the form

$$y' = f(t,y(t)), \quad y(t_0) = y_0, \quad t > t_0. \tag{1.1}$$

Such systems may arise directly, for example in the modelling of electrical networks or chemical kinetics, but frequently occur as a result of a method-of-lines approach to solving time-dependent partial differential equations (PDEs). Since (1.1) is assumed to be stiff, its efficient solution requires the use of an implicit formula. For example, the backward differentiation formulas (BDF)

$$y_{n+1} = \sum_{j=1}^{k} \alpha_j y_{n+1-j} + \Delta t \beta_0 f(t_{n+1}, y_{n+1}), \tag{1.2}$$

are commonly used to solve stiff IVPs (Gear (1971)). The use of such an implicit formula leads to a system of nonlinear equations of the form

$$F(y_{n+1}) = y_{n+1} - \Delta t \beta_0 f(t_{n+1}, y_{n+1}) - \sum_{j=1}^{k} \alpha_j y_{n+1-j}, \tag{1.3}$$

which must be solved for y_{n+1}. Stiffness also forces the use of Newton's method (or one of its variants) to solve (1.3). Hence, the ODE code must solve linear systems of the form

$$W(y_{n+1}^l)(y_{n+1}^{l+1} - y_{n+1}^l) = -F(y_{n+1}^l), \tag{1.4}$$

where

$$W(y_{n+1}^l) = I - \Delta t \beta_0 \frac{\partial f}{\partial y}(t_{n+1}, y_{n+1}^l),$$

or is some approximation to this Newton iteration matrix. Forming and factorizing W is expensive when the system is large and ODE codes try to avoid these operations as much as possible, for example by using an out-of-date iteration matrix until convergence failures occur. Nonetheless, the time spent doing linear algebra remains a substantial proportion of the total integration time.

For large systems, this linear algebra cost would be prohibitively expensive except for the fact that the systems are usually sparse. This sparsity can be exploited both in forming W, which is often done numerically, and in factorizing it and codes that do so, using band matrix solvers and direct sparse matrix methods, are available (e.g. NAG library codes, GEARS of Sherman and Hindmarsh (1981)). Even these methods may not be efficient enough for the problems that users will want to solve in the future. For example, methods for discretizing PDEs are becoming more sophisticated, with the result that users are looking for black-box solutions of some PDE problems in two and three dimensions. For such very large systems, the use of iterative techniques for solving sparse linear systems can often provide a gain in efficiency over direct sparse matrix methods.

There is a large body of literature available on iterative methods in linear algebra and no attempt is made to survey it here. The particular question of using an iterative solver in ODE software packages has attracted attention in recent years. See, for example, Chan and Jackson (1986), Brown and Hindmarsh (1986) or Byrne (1989) and references therein. Basically, there are two major questions in this context:

1) What iterative methods are appropriate for the class or classes of problems that users might want to solve with an ODE package?

2) How do the ODE solver and the iterative method interact?

We wish to identify some points that fall into these two general areas. Under 1), there is a general consensus that the Krylov subspace methods are most appropriate and that the acceleration methods are more effective with preconditioning but the choice of preconditioner is an open question. The issue of re-ordering the unknowns also falls into this area.

One of the main points falling into the second area is the use of reduced storage methods (Brown and Hindmarsh (1986), Hindmarsh and Brown (1987)). In such methods, rather than computing the Jacobian matrix, which would normally be done by a finite-difference approximation, and storing it for subsequent use, the matrix-vector products (or directional derivatives) necessary for the iterative method are computed directly by differences. Other questions in this second area include the strategy for updating the Newton iteration matrix (assuming that a reduced storage method is not being used), detecting convergence of the iterative solver, an initial guess for the iterative solver, and the stepsize changing strategy in the ODE method. For a detailed discussion of these issues, see Chan and Jackson (1986) or Jackson (1987).

This paper is mainly concerned with the first question, although the work should be viewed as preliminary to investigating the second area. We report results of a study of the use of a particular iterative solver, WATSIT, together with the ODE package, SPRINT (Berzins and Furzeland (1985)). This iterative solver uses a powerful preconditioner and also offers the option of

re-ordering the unknowns in the matrix. Most investigations have used less powerful preconditioners and have not considered reordering the unknowns as part of the ODE software. (Of course, extensive studies of ordering strategies for PDE problems have been made.) Because this iterative solver uses a powerful preconditioner, it has a greater cost per iteration than many other strategies that have been studied as part of ODE codes. In this investigation, the iterative method is compared to using a direct sparse matrix solver already available with SPRINT and it is shown that in spite of the expense of the technique, the iterative solver can provide a considerable gain in efficiency for large problems. This is not a reduced-storage technique but there is still a saving in storage for sufficiently large problems. It is also shown that re-ordering the unknowns does not increase the execution time and may decrease it significantly, and also decreases the required storage.

Results from the general literature on iterative methods for PDE problems indicate that iterative techniques are nearly always more efficient than direct methods for solving problems in three dimensions. In this paper, a simple example is given that shows that this performance is apparently not affected when the PDE is discretized via the method of lines and solved by an ODE code. We focus on two reaction-diffusion type problems in two dimensions. In 2D, direct and iterative sparse matrix methods are generally found to be competitive and this is the case for one of the test problems. In the other case, though, the ordering strategy in the iterative solver is able to exploit the particular structure of the problem and the iterative method is shown to be considerably more efficient. Numerical results were obtained on a SUN 3/160 with a floating-point accelerator and on a MIPS M/120 RISComputer.

In the next section, the iterative solver and the ODE code are described and some issues of their interaction are discussed. In Section 3, numerical results are given, Section 4 contains a discussion of future work and concluding remarks are made in Section 5.

2. The Codes

2.1 The Iterative Solver

The iterative solver, WATSIT, has been developed at the University of Waterloo based on methods described in Behie and Forsyth (1984). 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 two choices for the acceleration method, either the conjugate gradient method or Orthomin. The acceleration is applied to a second degree incomplete LU factorization. The user also has the option to request a red-black re-ordering of the matrix

A. The acceleration methods will not be discussed in any more detail (for further information see, for example, Axelsson and Barker (1984)), but some further description of the other parts of the code is now given.

Red-black ordering is well understood in the context of finite difference methods for PDEs (see, for example, Hageman and Young (1981)). Consider, for example, the heat equation on a rectangular domain in 2D,

$$u_t = \nabla^2 u\,, \quad (x,y) \in [0,1]\times[0,1], \quad t > 0.$$

Discretizing the Laplacian operator using the standard 5-point central difference scheme yields a system of ODEs of the form

$$U_{ij}' = \frac{1}{\Delta x^2}\Big[U_{i+1,j} - 2U_{i,j} + U_{i-1,j}\Big] + \frac{1}{\Delta y^2}\Big[U_{i,j+1} - 2U_{i,j} + U_{i,j-1}\Big],$$

where $U_{i,j} \approx u(x_i,y_j,t)$. Assuming an equidistant mesh with m intervals in both the x and y directions, then $x_i = i\Delta x$, $i = 0, \ldots, m$ and $y_j = j\Delta y$, $j = 0, \ldots, m$. Given Dirichlet boundary conditions, the ODE system will be of dimension $(m-1)^2$ and can be written in matrix form as $U' = A\,U$, where A is the appropriate matrix of size $(m-1)^2$. Applying the BDF (1.2) yields the completely discretized system

$$(I - \Delta t A)V_{n+1} = \sum_{j=1}^{k} \alpha_j V_{n+1-j},$$

where V_{n+1} is the solution of the completely discretized problem at time t_{n+1}. Let $W = (I - \Delta t A)$ (so that the linear system to be solved has the form $Wx = b$) and consider the case $m-1 = 3$. If the nodes of the mesh are numbered as in Fig. 1a, a natural ordering, then the 9x9 matrix W has the structure shown in Fig. 1b. Alternatively, the nodes can be numbered as in Fig. 2a (called a point red-black ordering in Hageman and Young (1981)) to obtain the matrix structure shown in Fig. 2b. The latter system can be written in block matrix form as

$$\begin{bmatrix} D_r & B \\ C & D_b \end{bmatrix} \begin{bmatrix} x_r \\ x_b \end{bmatrix} = \begin{bmatrix} b_r \\ b_b \end{bmatrix}, \tag{2.1}$$

where D_r and D_b are diagonal matrices of dimensions 5 and 4 respectively. Then to solve (2.1), one can first apply block Gaussian elimination to obtain

$$(D_b - CD_r^{-1}B)x_b = b_b - CD_r^{-1}b_r, \tag{2.2a}$$

$$x_r = D_r^{-1}(b_r - Bx_b). \tag{2.2b}$$

The reduced 4x4 system (2.2a) can be solved for x_b then x_r can be written down directly using (2.2b). This approach can yield savings in storage and execution time.

Given a matrix, rather than a grid, a similar re-ordering can be attempted. Rows of the matrix correspond to nodes in the finite difference grid in the previous example; we will continue the analogy and refer to nodes here. We use the rule that "red" nodes may only be connected to "black" nodes whereas black nodes may have arbitrary connections. Note that node (row) i is connected to node (row) j if there is a nonzero entry in element (i,j) or element (j,i) of the matrix. As a result of this rule for determining the "colours" of nodes, the submatrix corresponding to D_r in (2.1) will still be diagonal but that corresponding to D_b may now have arbitrary structure. The algorithm currently used in WATSIT to do the re-ordering begins by labelling the first node (the first row of the matrix) red. It then checks the second node for a connection to the first; if they are connected, then the second node must be labelled black, otherwise it is red. Similarly, the third node is tested for connections to the first two nodes and labelled appropriately and so on. This procedure allows all the rows of the matrix to be labelled but certainly there is no guarantee of finding the best ordering (where "best" in this case means the one with the largest number of red nodes). However, if the algorithm is applied to the matrix of Fig. 1b, the desired red-black ordering (Fig. 2b) is found.

Re-ordering strategies are an active topic of research in the numerical solution of PDEs and there are undoubtedly other orderings that will work better for some problems. While it is planned to pursue this topic further in future work, here it is shown that red-black re-ordering can be effective in ODE software.

The second degree ILU factorization is discussed in some detail by Behie and Forsyth (1983) and will be described here through a simple example. Consider the matrix structure shown in Fig. 3a, where a 0 denotes a zero entry and 1's are used to mark nonzero entries, labelling the original entries as "first degree". Applying one step of LU decomposition, i.e., operating on the first column of the matrix, yields the structure shown in Fig. 3b, where entries which have resulted from fill-in are denoted 2. (The structure shown is that of the L matrix in the lower triangle and that of the U matrix in the upper triangle.) The degree of the filled-in entry is the sum of the degrees of the entry being eliminated and the entry that causes the fill. Hence, after a second step of the decomposition, the structure shown in Fig. 3c is obtained, where third-degree entries have now arisen. In the second-degree ILU factorization, only the entries labelled 1 and 2 are retained as non-zero entries in the incomplete factorization. This structure of the ILU preconditioner is computed during a symbolic factorization stage, in which the algorithm only needs to consider eliminating first degree entries since eliminating a second or higher degree entry must result in fill of degree greater than 2.

When the user selects a red-black re-ordering of the matrix, the red nodes are eliminated exactly in the whole matrix, which is equivalent to computing $(D_b - CD_r^{-1}B)$ in (2.2a), then the incomplete factorization and acceleration are used to solve this reduced system for the black

nodes.

The WATSIT package consists of three main subroutines; one for symbolic factorization, which includes the re-ordering phase; one for numeric factorization; and one for the solve phase, i.e., the acceleration method. The user can call the numeric factorization routine repeatedly for matrices with different values but the same structure without calling the symbolic factorization again, and of course it is possible to solve for several different right-hand sides without repeating either factorization stage. The sparse matrix A is input as three arrays, in a format identical to that used in the Yale Sparse Matrix Package (Eisenstat et al. (1977)). That is, there is a real array which contains all the nonzero entries in the matrix, an integer array which indexes the columns of those nonzeroes and an integer array of row pointers. For further details, see the WATSIT User Guide (Kightley and Forsyth (1989)).

2.2 The ODE Solver

The ODE package used in this investigation is SPRINT (Berzins and Furzeland (1985)). This package was designed to solve systems of IVPs in ODEs and also systems of PDEs in one space dimension using the method of lines. The package is written in modular fashion and allows the user to select from two space discretizations, four time-stepping methods for IVPs 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 its modularity, the SPRINT package is ideal for testing various components of the complete method, such as the iterative solver, since it is possible to couple WATSIT to a sophisticated, fully-developed time-stepping scheme. The SPRINT package is also capable of solving problems large enough to make using an iterative solver attractive. While the SPRINT space discretization module is designed for one-dimensional PDE systems only, it is certainly possible for the user to semidiscretize a higher-dimensional problem into a system of ODEs and input the system to the time-stepping scheme. This has been done for the tests reported in this paper.

The particular ODE method used in this investigation is based on the BDF, shown in equation (1.2), and is similar in implementation to the wellknown code, LSODE (Hindmarsh (1980)). As previously mentioned, no changes have as yet been made to this routine to try and take advantage of the iterative solver.

2.3 Convergence Criteria for the Iterative Solver

One point of interaction between the BDF routine and WATSIT that must be addressed is convergence of the inner iteration, i.e., the iteration in the iterative solver. The choice of convergence criteria has a significant effect on the efficiency of the overall ODE code. Doing too many iterations leads, of course, to unnecessary expense. If too few iterations are done, two

situations can occur - either Newton's method will not converge or the error estimate computed by the time-stepping routine will be unacceptable. Either case can lead to a reduction in step-size that might have been avoided by doing more iterations of the iterative solver. It is necessary to decide both what quantity to measure and how small that quantity should be to consider that the iterative solver has converged to a sufficiently accurate solution.

WATSIT checks three conditions in deciding when to stop iterating in the solve phase:

i) A maximum number of iterations;

ii) A residual reduction of more than a value given by the user. That is, if r^0 is the initial residual and r^i is the residual after the i^{th} iteration, the code checks for

$$\frac{||r^i||_2}{||r^0||_2} \leq prec, \tag{2.3}$$

where *prec* is input by the user;

iii) The code also checks componentwise on the size of the iteration update. If x^i is the solution after the i^{th} iteration, the test is

$$|x_j^i - x_j^{i-1}| \leq w_j, \tag{2.4}$$

where $w = (w_1, w_2, \cdots, w_N)^T$ is input by the user.

If any one of these three criteria is met, the solve phase stops iterating. Since WATSIT preconditions on the right, the residual it calculates is the residual of the original unpreconditioned system.

In using WATSIT with the ODE solver, we have chosen $x^0 = 0$ as the initial guess to start the solve phase and the solver always does at least one iteration. The values used for w in the test (2.4) are the error tolerances input to the BDF routine, scaled by a specified factor. While trying out various values for *prec* and the scaling on w, the following observations were made.

Firstly, in the initial part of the domain of integration, where Δt is usually quite small, there is generally a large residual reduction after one iteration. When this occurred, the iterative solver stopped after one iteration due to the test (2.3) and this produced a solution of sufficient accuracy to allow the BDF routine to advance exactly as it would have done if a direct solver had been used.

If we examine the preconditioned conjugate gradient algorithm, it is possible to see why this large residual reduction can occur. The linear system to be solved at the k^{th} Newton iteration is

$$(I - \Delta t \beta_0 J)x = b,$$

where $b = -F(y_{n+1}^k)$ and $y_{n+1}^{k+1} = y_{n+1}^k + x$. On the first iteration of the conjugate gradient acceleration, $x^1 = x^0 + \tau_0 r^0$, where

$$\tau_0 = \frac{r^{0T} C^{-1} r^0}{r^{0T}(I - \Delta t \beta_0 J) r^0},$$

C is the preconditioning matrix, and $r^0 = b - (I - \Delta t \beta_0 J)x^0 = b$. Since $C^{-1} \approx (I - \Delta t \beta_0 J)^{-1}$, then $C^{-1} \approx I + \Delta t \beta_0 J + O(\Delta t^2)$, and

$$\tau_0 \approx \frac{r^{0T}(I + \Delta t \beta_0 J) r^0}{r^{0T}(I - \Delta t \beta_0 J) r^0} + O(\Delta t^2) = \frac{r^{0T} r^0 (1 + \Delta t \gamma)}{r^{0T} r^0 (1 - \Delta t \gamma)} + O(\Delta t^2),$$

$$= 1 + O(\Delta t),$$

where $\gamma = \dfrac{r^{0T} \beta_0 J r^0}{r^{0T} r^0}$. Hence, $x^1 = x^0 + \tau_0 r^0 \approx b = -F(y_{n+1}^k)$, and

$$r^1 = b - (I - \Delta t \beta_0 J)x^1 \approx b - (I - \Delta t \beta_0 J)b = -\Delta t \beta_0 J b.$$

Finally,

$$\frac{||r^1||_2}{||r^0||_2} \approx \frac{||\Delta t \beta_0 J b||_2}{||b||_2} \leq \Delta t \beta_0 ||J||_2.$$

If Δt is small, then a large residual reduction may occur after one acceleration step. Chan and Jackson (1986) suggested taking $x^0 = -F(y_{n+1}^k)$, and then allowed their iterative solver to accept x^0, i.e., to do no iterations. With WATSIT, since the cost of a single iteration is cheap compared to the cost of preconditioning, we have chosen to set $x^0 = 0$ and do at least one iteration in all cases.

Secondly, it was observed that, for large values of Δt, the solve phase of the iterative solver often found the test (2.4) satisfied after one iteration and stopped. This criterion appears somewhat less reliable than the residual reduction. In many cases, the BDF code was able to use the computed solution but in some instances, a step that could have been accepted, according to the BDF code's behaviour with a direct solver, was rejected. This was observed for scaling factors on the BDF error tolerances ranging from 1 to 10^{-4}. Again, insight can be gained by examining the conjugate gradient acceleration. When Δt is large, the Newton iteration matrix $I - \Delta t \beta_0 J$ will be close to $-\Delta t \beta_0 J$ and $C^{-1} \approx \dfrac{-1}{\Delta t \beta_0} J^{-1}$. Hence,

$$\tau_0 \approx \frac{r^{0T}\left(\frac{-1}{\Delta t\,\beta_0}J^{-1}\right)r^0}{r^{0T}(-\Delta t\,\beta_0 J)r^0} = \frac{r^{0T}J^{-1}r^0}{(\Delta t\,\beta_0)^2\,r^{0T}Jr^0} = \frac{\delta}{\Delta t^2}.$$

Then $x^1 = x^0 + \tau_0 r^0 \approx \frac{\delta}{\Delta t^2}b$, and

$$r^1 = b - (I - \Delta t\,\beta_0 J)x^1 \approx b - (I - \Delta t\,\beta_0 J)\frac{\delta}{\Delta t^2}b = \left(1 - \frac{\delta}{\Delta t^2}\right)b + \frac{\beta_0\delta}{\Delta t}Jb.$$

In this case,

$$\frac{||r^1||_2}{||r^0||_2} \approx |1 - \frac{\delta}{\Delta t^2}| + \frac{|\beta_0\delta|}{\Delta t}||J||_2,$$

and it is not surprising that a large residual reduction is not observed. However, $x^1 - x^0 \approx \frac{\delta}{\Delta t^2}b$ and if Δt is large, this difference may be small enough to satisfy test (2.4).

Further investigation of the rejected steps showed that the most difficulty occurred when the BDF routine was attempting a large stepsize increase, say by a factor of 5 to 10. In this case, the only way to force the iterative solver to find an acceptable solution using test (2.4) was to make w several orders of magnitude smaller than the BDF error tolerances. This was a much smaller value of w than was required on most steps, and hence, its use on all steps forced the iterative solver to do more iterations than were really necessary.

A solution to this difficulty is motivated by the question of detecting convergence of Newton's method in the time-stepping routine. There, experience shows that, in order to estimate the rate of convergence of Newton's method, information from more than one iteration with the same iteration matrix is needed. (See, for example, Shampine (1980).) When the Newton iteration matrix is used over several steps, this information is saved from step to step but when the matrix is updated, many stiff solvers force two Newton iterations before testing for convergence. Along the same line, in our code, the iterative solver is forced to do at least two iterations whenever the stepsize changes. This second iteration was found to be sufficient to allow the BDF routine to make large stepsize increases in all cases that have been tested to date. It should be noted that a stepsize change does not necessarily correspond to an update of the Newton iteration matrix. Since SPRINT does force two Newton iterations after an update, the iterative solver did not seem to have difficulty in this phase.

This heuristic works well for the problems that have been tested. Based on experience gained from numerical experiments, we set $prec = 10^{-2}$, i.e., only a small residual reduction is required, which agrees with observations made by Gear and Saad (1983). Also, w is taken as 10^{-2} times the BDF error tolerances, which appears to be satisfactory on most steps. The

maximum number of inner iterations is currently 10.

These convergence criteria are different from the approaches used in other work. Chan and Jackson (1986), Brown and Hindmarsh (1986) and Hindmarsh and Brown (1987) compare the norm of the i^{th} residual, $||r^i||$, to the BDF error tolerances, scaled by a specified factor. Boulter and Carroll (1989) use a residual reduction test, i.e., test (2.3). We have made some comparisons between our strategy (using tests (2.3) and (2.4)) and a pure residual reduction test (test (2.3) only) and the results are inconclusive. Both strategies are efficient but neither has a clear advantage. The question of convergence criteria merits further investigation and the issue will likely by affected by the choice of x^0 and by any changes made to the time-stepping routine to take advantage of the iterative solver.

3. Numerical Results

In this section, results for the SPRINT/WATSIT code used to solve two sets of problems are reported. The first set consists of simply the convection-diffusion equation in three space dimensions. The second set is a pair of reaction-diffusion equations, and shows that the iterative solver can be very efficient for some two-dimensional problems and have little effect for others.

3.1 Convection-Diffusion Equation

The convection-diffusion equation on the unit cube is given by

$$u_t = \nabla^2 u + v \cdot \nabla u, \quad (x,y,z) \in [0,1]\times[0,1]\times[0,1].$$

The particular problem used in these tests is the same as that used by Chan and Jackson (1986): $v = (1,1,1)^T$, homogeneous Dirichlet boundary conditions and initial condition

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

The spatial derivatives are discretized using second-order central difference approximations on an equally-spaced grid of m intervals and the unknowns are ordered using a natural ordering. This semidiscretization yields a system of stiff ODEs of the form

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

where A is a constant nonsymmetric negative-real matrix of dimension $M = (m-1)^3$. The numerical solution of this problem was computed for $m-1 = 3$, 5, 7 and 9 on a SUN 3/160 with a floating-point accelerator. The code used was the BDF routine in SPRINT with an approximate Jacobian matrix and an absolute error tolerance of 10^{-3}.

The two direct solvers tested were the NAG Library sparse matrix solver and the Yale Sparse Matrix Package (YSMP). The SPRINT code already had an interface for each of these

packages. The four possible combinations of input ordering or red-black re-ordering and conjugate gradient or Orthomin acceleration in WATSIT are compared. Although the actual matrices encountered in the course of the integration will not be symmetric and positive definite, they will have a symmetric structure and hence, it is not unreasonable to test the conjugate gradient algorithm. For ease of reference in the remainder of the section, the combinations are labelled as follows:

NaCG = *input ordering, conjugate gradient acceleration*;

NaOr = *input ordering, Orthomin acceleration*;

RbCG = *red-black re-ordering, conjugate gradient acceleration*;

RbOr = *red-black re-ordering, Orthomin acceleration.*

In all cases, the course of the integration is the same, that is, the same stepsize sequence is used and the same numbers of function and Jacobian evaluations and Newton iterations are required. There is little difference in the performance of the four options in WATSIT, as shown in Table 1.

Table 1: Convection-Diffusion Problem - Iterative Solver

		NITER				Time			
m-1	NEWT	NaCG	NaOr	RbCG	RbOr	NaCG	NaOr	RbCG	RbOr
3	58	73	73	73	73	4.6	4.0	3.8	3.7
5	62	83	83	80	80	16.6	15.9	15.4	14.5
7	71	94	94	91	91	53.5	51.0	50.5	49.4
9	77	105	105	103	103	124.5	124.0	119.1	116.8

NEWT = *number of Newton iterations*

NITER = *number of inner iterations*

Time = *time in seconds on a SUN 3/160*

On average, the iterative solver required between 1.25 and 1.4 iterations per Newton iteration. The iterative solver does yield a considerable speed-up compared to the direct solvers for large values of m, as shown in Table 2. (Again, the time-stepping sequence is the same for all the solvers.)

The iterative solver is also reasonably efficient in storage, as shown in Table 3. Certain caveats need to be made regarding this Table. Only one value is given for YSMP (in the table of real storage) since it increases its storage efficiency by writing integers and real values into

Table 2: Convection-Diffusion Problem - Solution Times

	Time		
m-1	RbCG	YSMP	NAG
3	3.8	3.5	4.2
5	15.4	18.1	22.0
7	50.5	102.2	124.8
9	119.1	448.6	582.3

the same locations. The NAG routines can run in less space, possibly at some expense in execution time. For the case $m-1 = 9$, it was possible to force the NAG routines to use only 94000 integer locations and 43000 real locations and the time required to solve the problem increased to only 591 seconds. The Orthomin variant in WATSIT will allocate real storage for 10 "orthomin" vectors if the storage is available (according to the user's dimension statements) even though it may never do 10 iterations in one solve. For example, for $m-1 = 9$, the RbOr variant does at most 3 iterations per solve and hence the real storage requirement could be reduced to 17059 real locations. For this problem, the red-black re-ordering strategy reduces the size of the system by one-half and consequently yields a significant saving in storage.

Table 3: Convection-Diffusion Problem - Storage Requirements

	Integer Storage			Real Storage					
m-1	Na	Rb	NAG	NaCG	NaOr	RbCG	RbOr	YSMP	NAG
3	545	405	1631	586	1136	418	688	751	750
5	2822	2192	11894	3056	5566	2300	3550	5204	6601
7	8144	6424	52838	8814	15684	6750	10180	20160	30631
9	17811	14161	176609	19252	33842	14872	22162	57516	106382

It is also interesting to examine the proportion of time that the ODE solver spends doing linear algebra. Results for the RbCG, YSMP and NAG solvers are given in Table 4.

The largest problem that has been solved to date had $m-1 = 24$, yielding a system of 13824 ODEs. This problem was solved on the SUN 3/160 in 64 minutes using the RbOr variant of WATSIT; extrapolating from the values in Table 2, the total time required by YSMP would be around 20 hours. However, this model problem is well-known to be amenable to solution using an iterative method for the linear algebra. The results given here show that the interface

Table 4: Convection-Diffusion Problem - Linear Algebra

	RbCG		YSMP		NAG	
m	Time	% of Total	Time	% of Total	Time	% of Total
3	0.9	24	0.7	17	1.0	24
5	4.1	27	7.1	39	9.6	44
7	14.5	29	68.9	67	77.0	62
9	34.9	29	371.9	83	438.6	75

to the ODE solver doesn't hinder the speed-up but the problem is not particularly challenging. Results are now presented for two problems which pose more challenge to the ODE solver.

3.2 Reaction-Diffusion Problems

Both PDEs discussed in this section are of reaction-diffusion type, modelling predator-prey systems, and are defined in two space dimensions. The first problem models the spread of rabies in foxes (Murray, Stanley and Brown (1986)) while the second is taken from Hindmarsh and Brown (1987), who also used it to test an iterative solver in an ODE code.

The rabies model has the form

$$\frac{\partial S}{\partial T} = (a-b)(1 - \frac{N}{K})S - \beta RS, \tag{3.1a}$$

$$\frac{\partial I}{\partial T} = \beta RS - [\sigma + b + (a-b)\frac{N}{K}]I, \tag{3.1b}$$

$$\frac{\partial R}{\partial T} = \sigma I - \alpha R - [b + (a-b)\frac{N}{K}]R + \nabla^2 R, \tag{3.1c}$$

where S, I and R represent concentrations of foxes in three stages of rabies and the remainder of the parameters are given in Murray et al. (1986). The solution of this problem takes the form of a travelling wave. The space derivative is discretized using central differences on an equally-spaced grid of m intervals and the unknowns are ordered by node first with a natural ordering of the nodes. An ODE system of $3(m+1)^2$ equations is obtained since the problem has Neumann boundary conditions. The structure of the system is shown in Fig. 4 for the case $m+1 = 3$. When the red-black re-ordering is applied to this matrix structure, the system is only reduced in size by one-third, that is, two-thirds of the rows are labelled "black". The results shown in Figs. 5 and 6 compare WATSIT to YSMP. These results were obtained on a MIPS M/120 RISComputer using grids of sizes $m+1 = 10, 20, 30, 40$, and 50. Again, the time-stepping scheme showed essentially the same behaviour regardless of the linear algebra. For this set of tests, YSMP is certainly the most efficient solver in terms of both execution time and storage. As the size of

the problem increases, the iterative solver is slowly becoming relatively more efficient but the problem will have to be very large before it does better than the direct solver. Extrapolating from the computed results suggests that the system would have to have around 5.5 million equations for the two solvers to take approximately the same time, corresponding to a grid with $m+1 = 1354$. The percentage of time spent in linear algebra remains almost constant at 53% to 54% for the RbOr variant over the 5 test cases and increases from 18% to 30% for YSMP.

In some sense, though, this is a relatively easy problem for the iterative solver, as demonstrated by the fact that the NaCG variant is almost as efficient as the RbOr option although there is no reason to expect that the matrix is positive definite. The conjugate gradient algorithm in WATSIT uses a symmetric scaling where the Orthomin method uses only diagonal scaling. Hence, CG costs an extra matrix-vector multiplication in this phase. In many cases, the higher cost per iteration of Orthomin would outweigh this additional cost of CG but the iteration cost has little impact here since the average number of iterations per solve is only 1.1. This extra multiplication combined with the larger system being solved in the NaCG variant explains its longer total time. The low number of iterations per solve is also an indication that the problem is "easy".

The YSMP code does so well on this problem because it is able to exploit the particular structure of the matrix. If the model (3.1) is modified to include diffusion terms in equations (3.1a) and (3.1b), the RbOr variant of the iterative solver is already faster than YSMP on a grid with $m+1 = 30$, taking 218 seconds compared to YSMP's 296 seconds. A corresponding change in relative storage requirements is also observed - YSMP needs 225736 real locations while RbOr uses 60683 integer and 100277 real locations.

The second reaction-diffusion problem, is one in which the iterative solver is able to exploit the structure effectively. This problem, taken from Hindmarsh and Brown (1987), is given by

$$\frac{\partial c^i}{\partial t} = c^i \left(b_i + \sum_j a_{ij} c^j\right) + d_i \nabla^2 c^i, \quad i = 1, \ldots, 2s,$$

where

$a_{ii} = -1, \quad i = 1, \ldots, 2s,$

$a_{ij} = -5\times10^{-7}, \quad i \leq s, \; j > s,$

$a_{ij} = 10^4, \quad i > s, \; j \leq s,$

$b_i = 1 + 50xy, \quad d_i = 1, \quad i \leq s,$

$b_i = -(1 + 50xy), \quad d_i = 0.05, \quad i > s.$

The problem is defined on the unit square with zero Neumann boundary conditions and initial condition

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

and time interval $0 \leq t \leq 10$ (time required to reach a steady state). In the following tests, the problem was discretized in space using standard central differences on an equally spaced mesh of m intervals and the same ordering as for problem (3.1), resulting in an ODE system of $M = 2s\,(m+1)^2$ equations with the structure shown in Fig. 7 (for $s = 2$, $m+1 = 3$). It should be noted that the $2s \times 2s$ diagonal blocks are two-cyclic (Hageman and Young (1981)) with the result that when the red-black re-ordering is applied, the size of the the system is reduced by one-half, that is, only half the nodes need to be labelled black. The ODE problem was solved using the BDF routine in the SPRINT package with a relative error tolerance of 10^{-2} and an absolute tolerance of 10^{-3}. The meshes used were of sizes $m+1 = 10$, 20, 30 and 40 and the results shown in Figs. 8 and 9 were obtained on a MIPS M/120 RISComputer.

Clearly, the iterative solver is much more efficient in time and storage than the direct solver in this case. Also, the red-black ordering is particularly effective, as can be seen by comparing the results for the RbOr and NaOr variants.

The conjugate gradient algorithm is not appropriate for this problem. When the RbCG variant was tested, the solve phase failed to converge in 10 iterations on a number of time steps. When the iterative solver fails to converge, the BDF routine will reduce Δt, which makes the Newton iteration matrix more nearly diagonally dominant. For this problem, reducing Δt does allow CG to converge but the BDF routine has to take many more time steps than required by the other constraints of the problem and code, as shown in Table 5. (Note that the time comparison includes failed steps while NSTEP is the number of successful steps only; hence, the difference in the ratios.)

Table 5: Hindmarsh and Brown Example - Comparing RbOr and RbCG

Ratio RbCG/RbOr		
m+1	Time	NSTEP
10	6.5	3.1
20	5.6	3.0

The average number of iterations in the RbOr solve phase is greater for this problem than previous examples, varying from 1.6 to 2.5 as the number of mesh intervals increases. For the NaOR variant, this ratio varies from 2.1 to 3.3. The percentage of time spent doing linear algebra increases from 47% to 66% for the RbOr method and from 57% to 90% for the YSMP code. The time-stepping sequences are no longer identical for all the linear algebra variants although the difference between RbOr and YSMP is small (one time step and one Jacobian evaluation for the case $m+1 = 40$.)

The problem solved by Hindmarsh and Brown (1987) had $s = 10$ on a grid with $m+1 = 12$. The resulting system of ODEs, of size 2880, was solved using a BDF routine very similar to that in SPRINT with a relative error tolerance of 10^{-6} and an absolute tolerance of 10^{-8}. Their iterative solver used a reduced storage technique, GMRES acceleration and a number of different preconditioners. This same case has been solved using YSMP and the RbOr variant of WATSIT and the results shown in Table 6 were obtained on the MIPS M/120.

Table 6: Hindmarsh and Brown Example

Code	NSTEP	NFCN	NJAC	NEWT	NITER	Time(min)	Integer	Real
YSMP	319	1478	39	371		71		682757
RbOr	316	1503	40	368	487	20	134954	166611

NSTEP = *number of integration steps*

NFCN = *number of function evaluations*

NJAC = *number of Jacobian evaluations*

NEWT = *number of Newton iterations*

NITER = *number of inner iterations*

Except for time and storage, these results are close to those obtained by Hindmarsh and Brown. It is not possible to compare execution times since their computations were done on a Cray-1. For the different preconditioners, Hindmarsh and Brown found ratios of NITER to NEWT ranging from 1.27 to 2.42. Hence, the value for RbOr of 1.32 represents a relatively small number of inner iterations. They observed that over the time interval $4 \leq t \leq 10$, their ratio was 5, which was the maximum number of inner iterations allowed per Newton iteration in their code; for the RbOr code, the ratio in this interval is around 3. The storage requirement of the Hindmarsh and Brown code is much less than the amount needed by RbOr, by a factor of about 5. This makes clear the advantage of the reduced storage technique but it should be noted that their preconditioners used some problem-dependent information to achieve efficiency in time. The

SPRINT/WATSIT combination requires no problem-dependent information beyond that required to use a direct solver.

4. Discussion

For the problems that have been tested to date, using red-black re-ordering in the iterative solver never increases the cost of the integration and leads to a significant saving in execution time compared to not re-ordering for some problems, e.g., the Hindmarsh and Brown example. The test problems all arose as ODE systems through the spatial discretization of PDEs using finite difference methods on regular grids. In coding the ODE system for input to SPRINT, the unknowns were ordered within each node first in the case of systems of PDEs then by a natural ordering of the nodes. For such problems, red-black ordering is the obvious choice and one would expect that it should improve performance for sufficiently large systems. It is encouraging to see that it does not increase the cost of solving the smaller systems.

The question of ordering for other classes of problems is less clear. For some problems, there may not even be a "natural" ordering. For example, when a PDE on an arbitrary domain is discretized using the finite element method, a particular node in the mesh may be connected to any number of other nodes in random "directions". In general, ODE systems arising directly from modelling large physical systems would also not be expected to exhibit regular patterns of connections. In these cases, the questions of "natural" ordering and re-ordering are linked - the ideal would be to choose the "natural" ordering so that a re-ordering could be applied to obtain a significantly smaller system that was as well-conditioned as possible. In addition, one would like these strategies to be as problem-independent as possible.

Even in the case of finite difference space discretization, it may be possible to do better than red-black ordering for systems of PDEs. In this case, when there are multiple unknowns per node, it can be more efficient to re-order in a block fashion, i.e., just re-order the nodes, then apply the iterative solver in block fashion, too. See, for example, Behie and Forsyth (1984). These questions of ordering strategies will be pursued in future work.

The second-degree ILU preconditioning is more expensive in storage and time than other techniques such as DKR preconditioning, for example. The more expensive technique is used in the expectation that fewer iterations of the acceleration method will be required and that the acceleration method will converge for a wider class of problems. The results in this paper show that very few iterations are required per solve for the class of problems tested and that, in spite of the overhead of the method, there is a significant speed-up over the direct approach. Future avenues of research include comparing the second-degree ILU preconditioner to other strategies in order to obtain more information about which preconditioners are appropriate for particular classes of problems and about the trade-off between overhead cost and number of iterations.

Acceleration methods are currently the best understood part of iterative solvers and hence one expects the least improvement to come from changing the acceleration. For the test problems used to date, the average number of iterations per solve is so small that no advantage in execution time has been seen for the conjugate gradient method even for symmetric positive definite matrices (a discretization of the heat equation, for example). Its only advantage seems to be in the reduced storage it requires. It is possible that the GMRES or Conjugate-Gradient-Squared algorithms might prove more robust than Orthomin but no difficulties with Orthomin have been encountered yet. Four different acceleration techniques are considered by Boulter and Carroll (1989) in the context of ODE software and all are found to be effective but no comparisons are made among them.

5. Conclusion

The purpose of this investigation was to consider the use of a powerful iterative solver in solving large ODE systems and it is felt that the results that have been obtained are encouraging. In spite of the expense of the red-black re-ordering and second-degree ILU preconditioning, significant savings in execution time have been demonstrated. The SPRINT/WATSIT combination also seems to be robust in that, for all the problems tested, no particular problem-dependent information was given to the iterative solver, yet the ODE code was not affected by the change in linear algebra. The interaction between the iterative and ODE solvers has been considered only in the convergence criteria for the inner iteration. It is planned to address this matter more fully in future work, and expected that this will lead to improved performance when using the iterative solver. The robustness of the technique suggests that powerful preconditioning methods may enable a single iterative solver to be used for a wide class of problems.

References

O. Axelsson and V.A. Barker, *Finite Element Solution of Boundary Value Problems: Theory and Computation,* Academic Press, Orlando, 1984.

G.A. Behie and P.A. Forsyth, Comparison of fast iterative methods for symmetric systems, IMA J. Numer. Anal. 3(1983), pp. 41-63.

G.A. Behie and P.A. Forsyth, Incomplete factorization methods for fully implicit simulation of enhanced oil recovery, SIAM J. Sci. Stat. Comput. 5(1984), pp. 543-561.

M. Berzins and R.M. 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, Technical Report TNER.85.058, Thornton Research Centre, Shell Research Ltd., Thornton, U.K., 1985.

B. Boulter and J. Carroll, Application of iterative linear solvers in stiff codes for PDEs, Report, School of Mathematical Sciences, Dublin City University, Dublin, Ireland, 1989.

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

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

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), pp. 378-417.

S.C. Eisenstat, M.C. Gursky, M.H. Schultz and A.H. Sherman, Yale sparse matrix package I: The symmetric codes, Tech. Rep. 112, Yale sparse matrix package II: The nonsymmetric codes, Tech. Rep. 114, Dept. Computer Science, Yale Univ., New Haven, CT, 1977.

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

C.W. Gear and Y. Saad, Iterative solution of linear equations in ODE codes, SIAM J. Sci. Stat. Comput. 4(1983), pp. 583-601.

L.A. Hageman and D.M. Young, *Applied Iterative Methods,* Academic Press, New York, 1981.

A.C. Hindmarsh, LSODE and LSODI, Two new initial value ordinary differential equation solvers, ACM SIGNUM Newsletter, December issue, 1980, pp. 10-11.

A.C. Hindmarsh and P.N. Brown, Reduced-storage techniques in the numerical method of lines, *Advances in Computer Methods for Partial Differential Equations VI,* R. Vichnevetsky and R.S. Stepleman, eds., IMACS, 1987, pp. 355-362.

K.R. Jackson, The use of Krylov subspace methods in codes for large stiff systems of ODEs, *Advances in Computer Methods for Partial Differential Equations VI,* R. Vichnevetsky and R.S. Stepleman, eds., IMACS, 1987, pp. 452-459.

J.R. Kightley and P.A. Forsyth, WATSIT iterative sparse matrix solver: user's guide, Dept. Computer Science, Univ. of Waterloo, Waterloo, Ont., 1989.

J.D. Murray, E.A. Stanley and D.L. Brown, On the spatial spread of rabies in foxes, Proc. Roy. Soc. Lond. B 229(1986), pp. 111-150.

L.F. Shampine, Implementation of implicit formulas for the solution of ODEs, SIAM J. Sci. Stat. Comput. 1(1980), pp. 103-118.

A.H. Sherman and A.C. Hindmarsh, GEARS: a package for the solution of sparse, stiff ordinary differential equations, *Electrical Power Problems: The Mathematical Challenge,* A. Erisman, K. Neves and M.H. Dwarakanath, eds., SIAM, Phildelphia, 1981, pp. 190-200.

List of Figures

Legend for Figures 5 and 6

x YSMP

□ RbOr

○ RbCG

△ NaOr

\+ NaCG

Legend for Figures 8 and 9

x YSMP

□ RbOr

△ NaOr

```
 7 ¦ 8 ¦ 9
-----------
 4 ¦ 5 ¦ 6
-----------
 1 ¦ 2 ¦ 3
```

Figure 1a

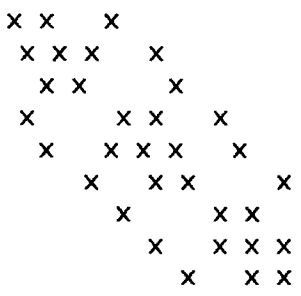


Figure 1b

```
 4 ¦ 9 ¦ 5
-----------
 7 ¦ 3 ¦ 8
-----------
 1 ¦ 6 ¦ 2
```

Figure 2a

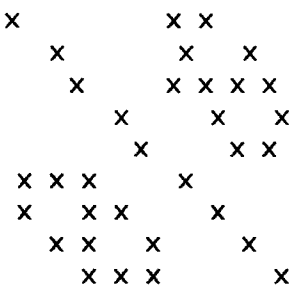


Figure 2b

1	1	0	1	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0
1	0	0	1	1	0	1	0	0	
0	1	0	1	1	1	0	1	0	
0	0	1	0	1	1	0	0	1	
0	0	0	1	0	0	1	1	0	
0	0	0	0	1	0	1	1	1	
0	0	0	0	0	1	0	1	1	

Figure 3a

1	1	0	1	0	0	0	0	0	0
1	1	1	2	1	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0
1	2	0	1	1	0	1	0	0	
0	1	0	1	1	1	0	1	0	
0	0	1	0	1	1	0	0	1	
0	0	0	1	0	0	1	1	0	
0	0	0	0	1	0	1	1	1	
0	0	0	0	0	1	0	1	1	

Figure 3b

1	1	0	1	0	0	0	0	0	0
1	1	1	2	1	0	0	0	0	0
0	1	1	3	2	1	0	0	0	0
1	2	3	1	1	0	1	0	0	
0	1	2	1	1	1	0	1	0	
0	0	1	0	1	1	0	0	1	
0	0	0	1	0	0	1	1	0	
0	0	0	0	1	0	1	1	1	
0	0	0	0	0	1	0	1	1	

Figure 3c

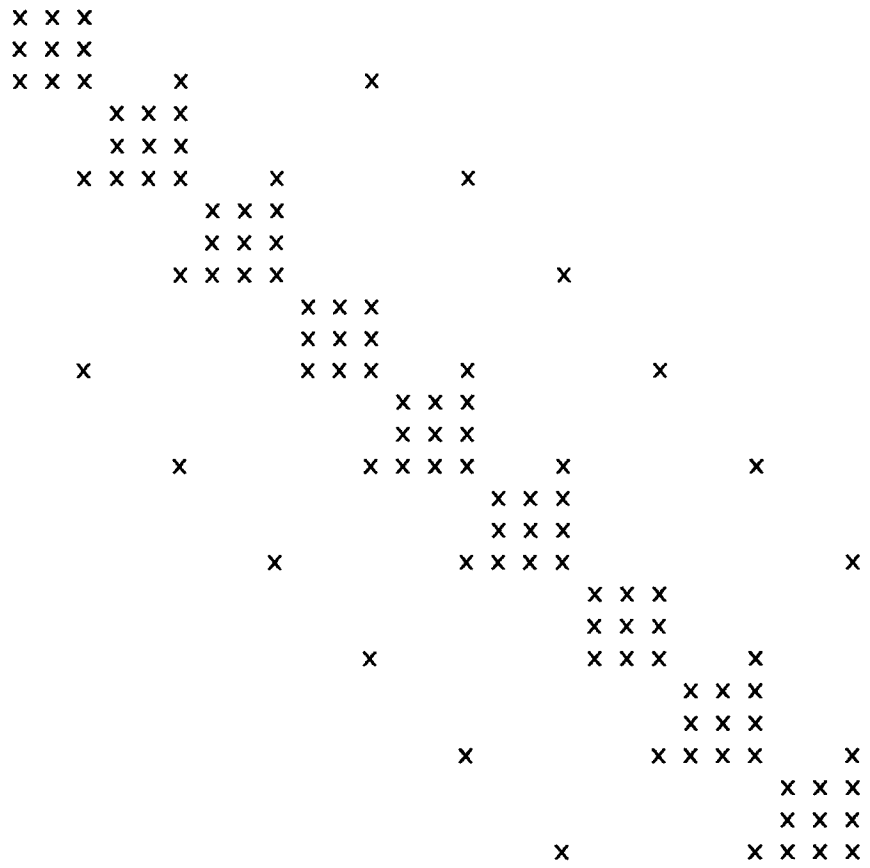
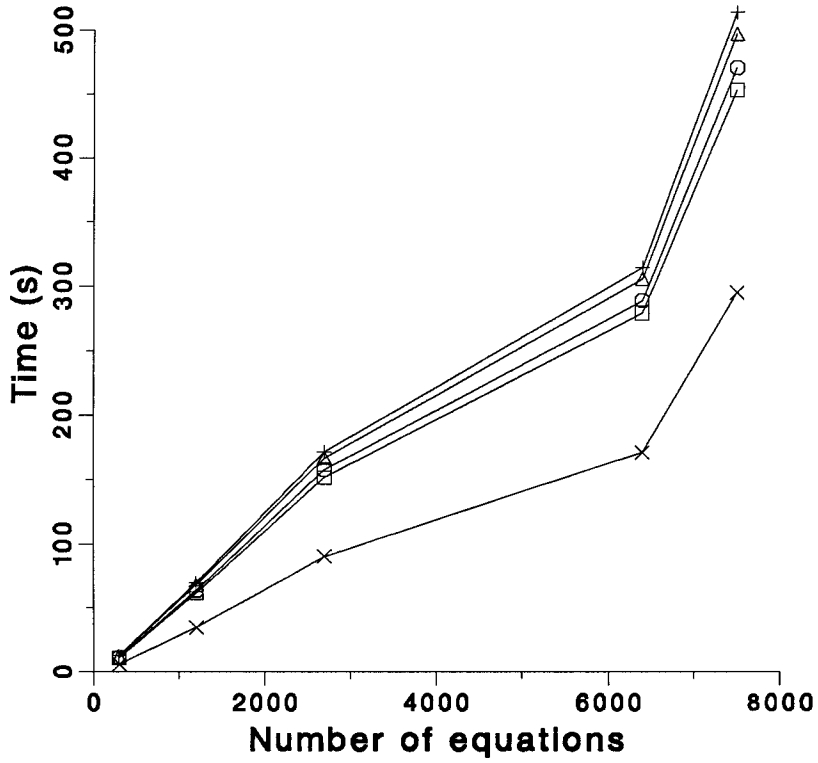


Figure 4

FIGURE 5


Time (s)
0
100
200
300
400
500
0
2000
4000
6000
8000
Number of equations

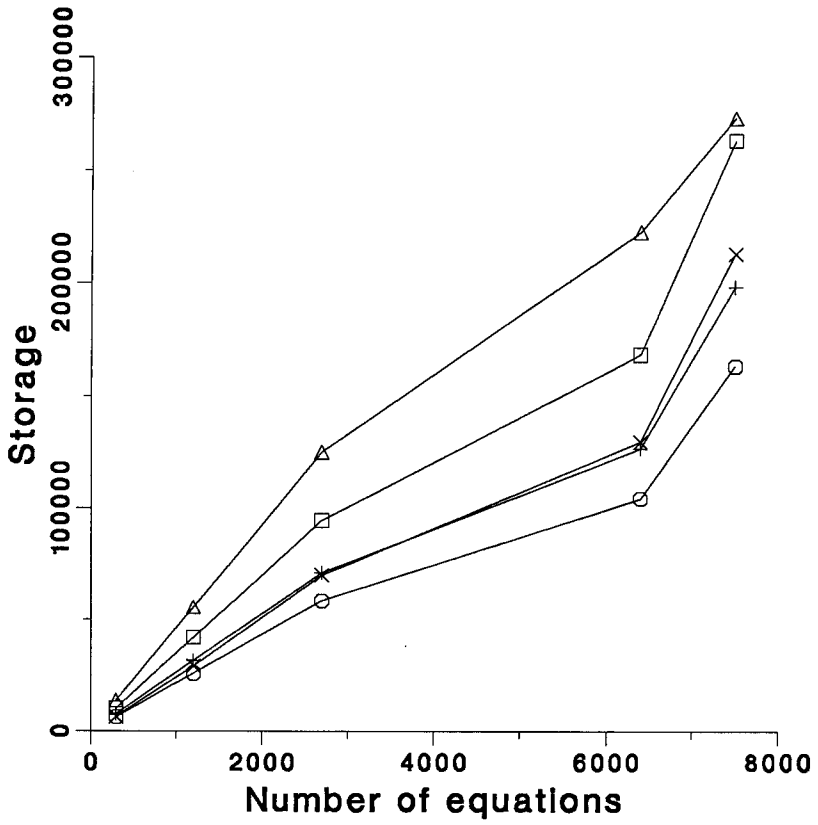
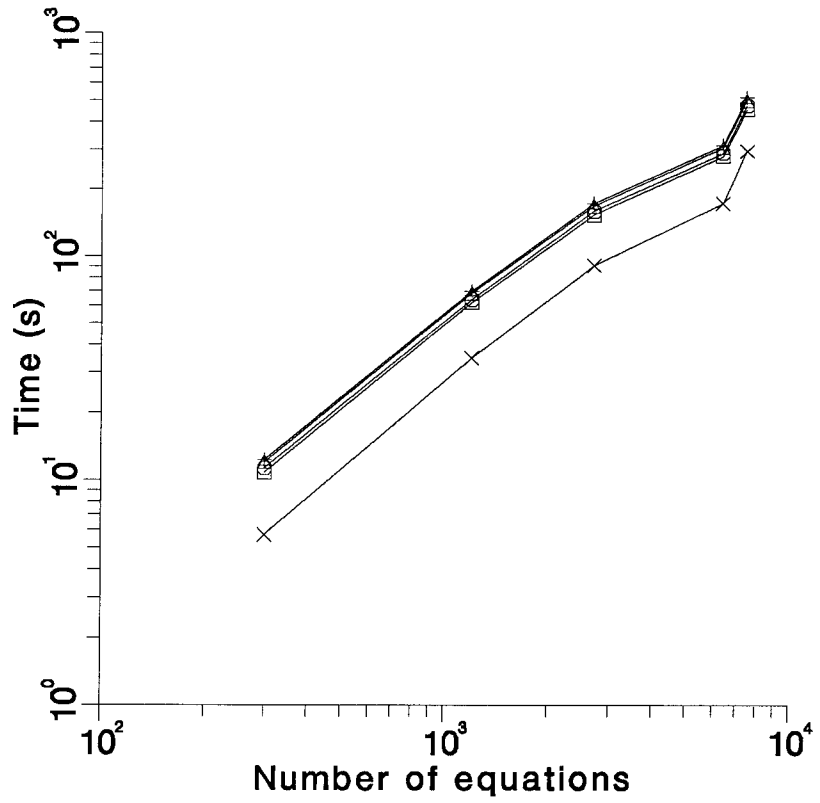
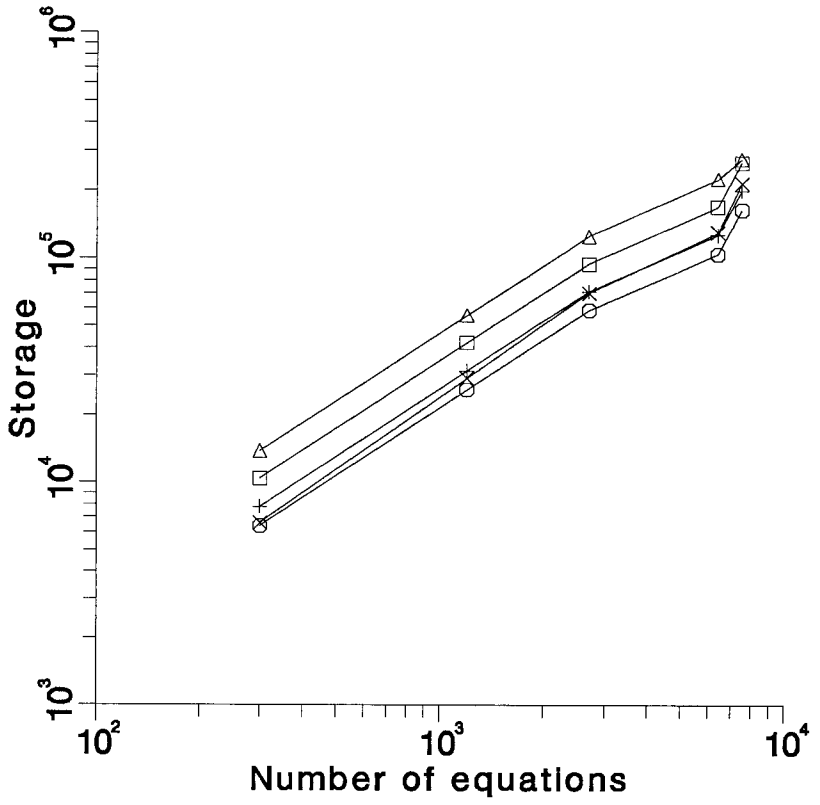

Storage
0
100000
200000
300000
0
2000
4000
6000
8000
Number of equations

FIGURE 6


Time (s)
10^0
10^1
10^2
10^3
10^2
10^3
10^4
Number of equations


Storage
10^3
10^4
10^5
10^6
10^2
10^3
10^4
Number of equations

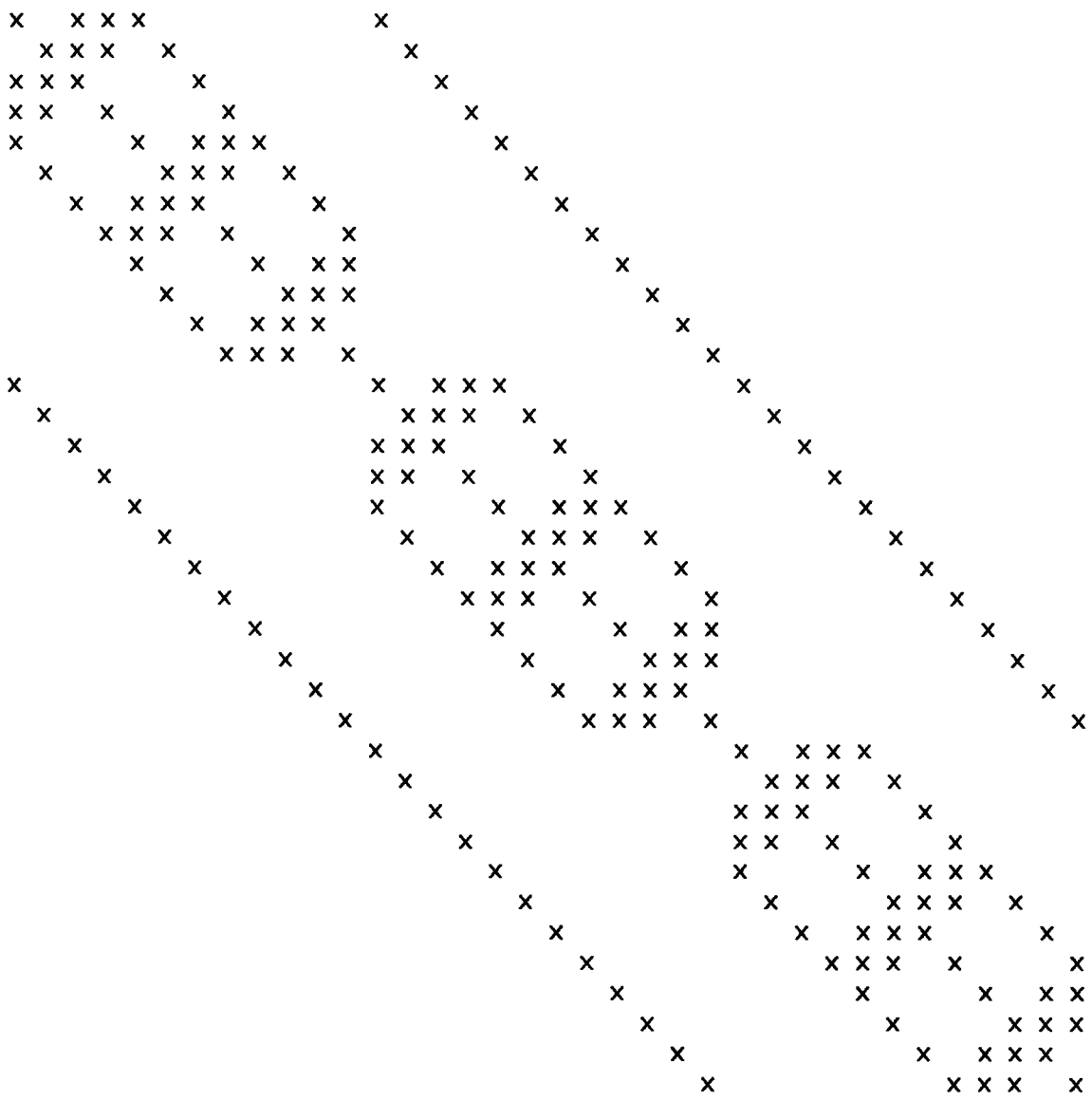
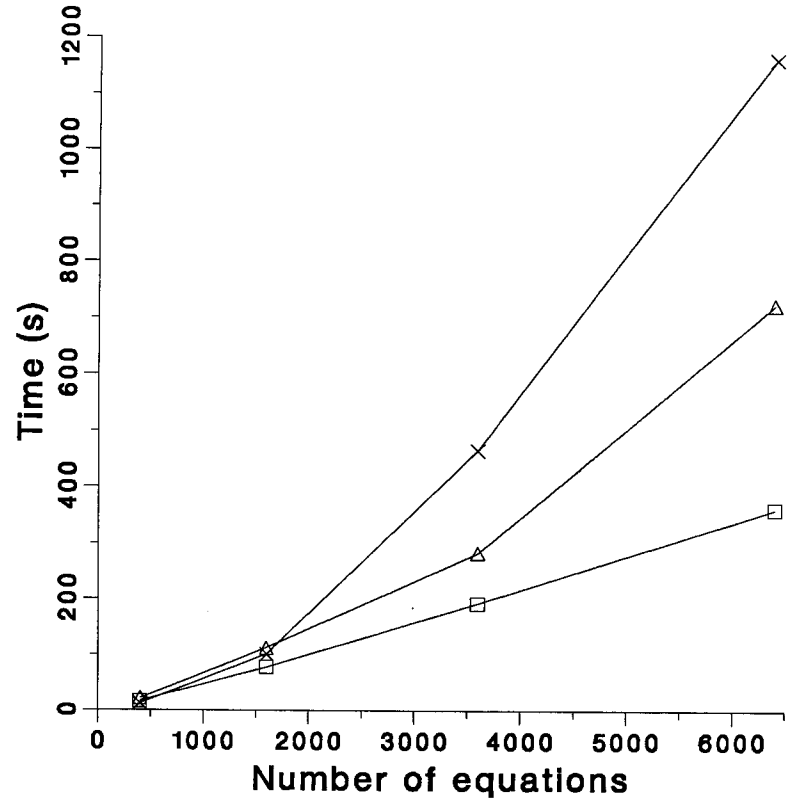


Figure 7

FIGURE 8


Time (s)
0
200
400
600
800
1000
1200
0
1000
2000
3000
4000
5000
6000
Number of equations

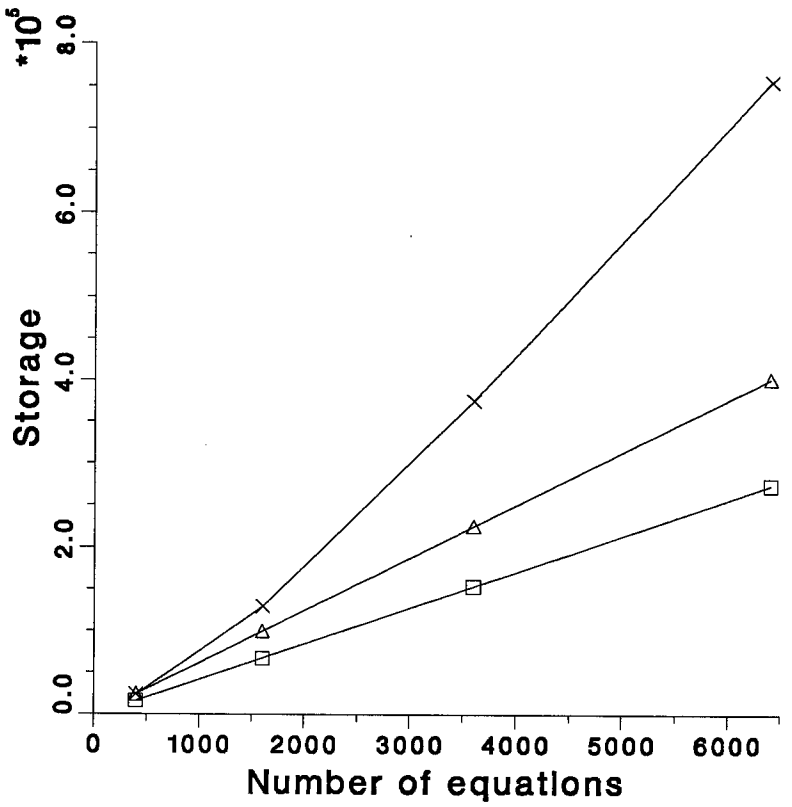
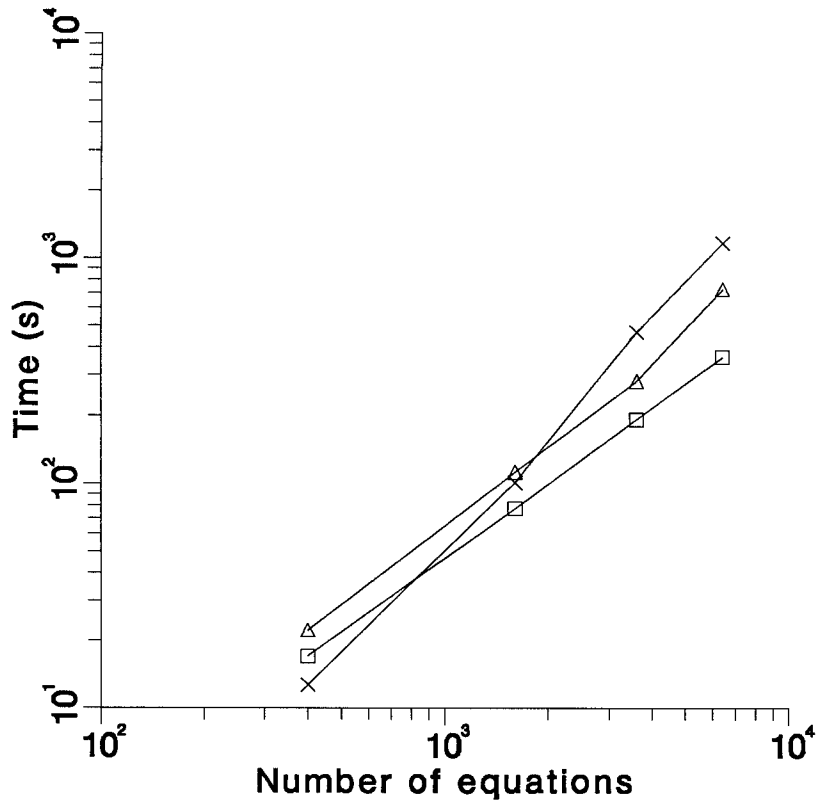
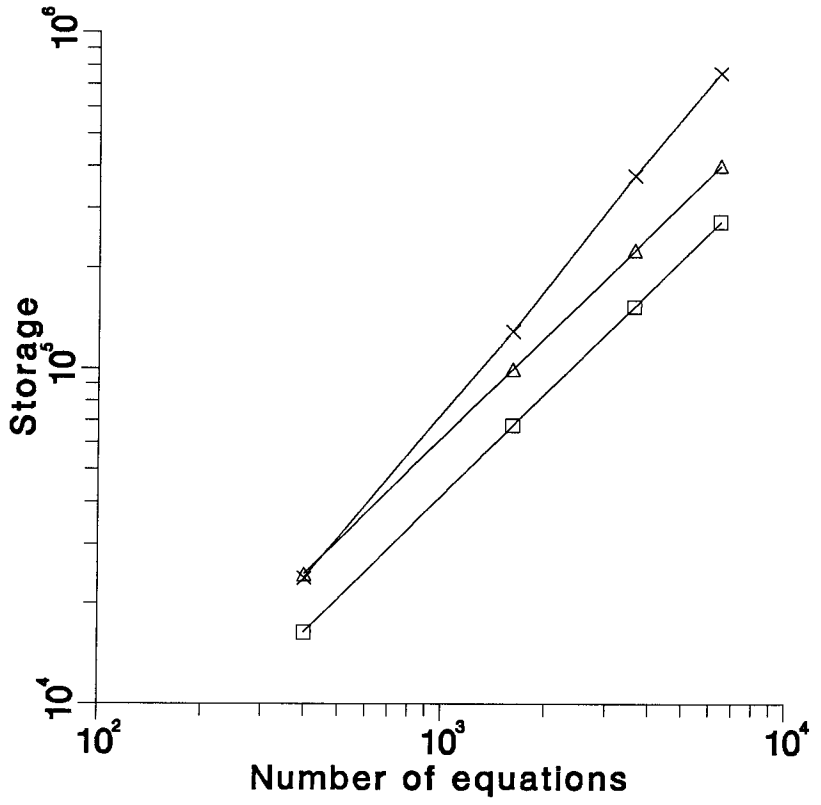

*10^5
Storage
0.0
2.0
4.0
6.0
8.0
0
1000
2000
3000
4000
5000
6000
Number of equations

FIGURE 9


Time (s)
10^1
10^2
10^3
10^4
10^2
10^3
10^4
Number of equations


Storage
10^4
10^5
10^6
10^2
10^3
10^4
Number of equations