Preconditioned Iterative Methods for Sparse Linear Algebra Problems Arising in Circuit Simulation

William D. McQuain, Calvin J. Ribbens, Layne T. Watson, and Robert C. Melville

TR 92-07

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

March 16, 1992

PRECONDITIONED ITERATIVE METHODS FOR SPARSE LINEAR ALGEBRA PROBLEMS ARISING IN CIRCUIT SIMULATION

WILLIAM D. MCQUAIN[†], CALVIN J. RIBBENS[†], LAYNE T. WATSON[†], AND ROBERT C. MELVILLE[‡]

Abstract. The DC operating point of a circuit may be computed by tracking the zero curve of an associated artificial-parameter homotopy. It is possible to devise homotopy algorithms that are globally convergent with probability one for the DC operating point problem. These algorithms require computing the one-dimensional kernel of the Jacobian matri x of the homotopy mapping at each step along the zero curve, and hence the solution of a linear system of equations at each step. These linear systems are typically large, highly sparse, nonsymmetric and indefinite. Several iterative methods which are applicable to nonsymmetric and indefinite problems are applied to a suite of test problems derived from simulations of actual bipolar circuits. Methods tested include Craig 's method, GMRES(k), BiCGSTAB, QMR, KACZ (a row-projection method) and LSQR. The convergence rates of these methods may be improved by use of a suitable preconditioner. Several such techniques are considered, including incomplete LU factorization (ILU), sparse submatrix ILU, and ILU allowing restricted fill in bands or blocks. Timings and convergence statistics are given for each iterative method and preconditioner.

Key words. globally convergent, homotopy algorithm, homotopy curve tracking, iterative methods, nonlinear equations, preconditioned conjugate gradient, sparse matrix

AMS(MOS) subject classifications. 65F10, 65F50, 65H10, 68U20, 94C05

1. Introduction.

The cost and difficulty of producing a prototype of a proposed design for an integrated circuit provide substantial motivation for the development of accurate, efficient computer simulations of such designs. The mathematical models for common components of such circuits are nonlinear, and so the simulation of such integrated circuits requires solving large systems of nonlinear equations $F(x) = 0$, where $F : E^n \to E^n$ is C^2. Algorithms that solve such nonlinear systems through the use of an artificial-parameter homotopy mapping have been studied for some time [35]. Under reasonable hypotheses these algorithms are guaranteed to be convergent for almost all starting points [24].

The application of homotopy algorithms to a variety of problems has been considered in [29], [30] and [33]. The curve tracking that is inherent to homotopy algorithms requires solving a rectangular linear system whose coefficient matrix is the Jacobian matrix of the homotopy. For many applications, the Jacobian matrix of F is symmetric and positive definite, or nearly so. It is possible to take advantage of these properties when implementing the homotopy algorithm. However, for the DC operating point problem in circuit simulation, the Jacobian matrix of F will usually be nonsymmetric, indefinite, sparse, and unstructured. A detailed description of the DC operating point

[†] Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061-0106. The work of these authors was supported in part by Department of Energy Grant DE-FG05-88ER25068, National Science Foundation Grant CTS-8913198, and Air Force Office of Scientific Research Grant 89-0497.

[‡] AT & T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974-2070.

problem, and the justification of the application of homotopy methods to it, are given by Melville et al. [17], [24], [25].

Since the linear systems under consideration here arise in the context of homotopy curve tracking, §2 gives a brief summary of globally convergent homotopy theory. The various iterative methods are described in detail in §3, followed by the preconditioning strategies in §4. Section 5 presents the circuit simulation test problems. Numerical results are given and interpreted in §6, and §7 concludes.

2. Globally convergent homotopy algorithms.

Consider the problem of solving a nonlinear system of equations

$$F(x) = 0, \tag{1}$$

where $F : E^n \to E^n$ is a C^2 map defined on real n-dimensional Euclidean space E^n. Equation (1) may be solved by use of a suitable *homotopy*, a continuous mapping $H(\lambda, x)$ such that $H(0, x) = 0$ is easily solved and $H(1, x) = F(x)$. Given a solution of $H(0, x)$ as a starting point, a homotopy algorithm will attempt to track a curve in the zero set of $H(\lambda, x)$, terminating at a solution of $F(x) = 0$ when $\lambda = 1$. A globally convergent probability-one homotopy algorithm is based upon the construction of a homotopy whose zero curves are well behaved and reach a solution for almost all starting points. Such homotopies are easy to construct for Brouwer fixed point problems.

Let B be the closed unit ball in E^n, and let $f : B \to B$ be a C^2 map. The Brouwer fixed point problem is to solve $x = f(x)$. Define $\rho_a : [0, 1) \times B \to E^n$ by

$$\rho_a(\lambda, x) = \lambda(x - f(x)) + (1 - \lambda)(x - a), \tag{2}$$

where a is some point in the interior of B. The fundamental result [3] is that for almost all a in the interior of B, there is a zero curve $\gamma \subset [0, 1) \times B$ of ρ_a, along which the Jacobian matrix $D\rho_a(\lambda, x)$ has rank n, emanating from $(0, a)$, and reaching a point $(1, \bar{x})$, where $\bar{x}$ is a fixed point of f. Thus with probability one, picking a starting point $a \in \text{int}B$ and following γ leads to a fixed point $\bar{x}$ of f. An important distinction between standard continuation and modern probability-one homotopy algorithms is that, for the latter, λ is not necessarily monotonically increasing along γ. Indeed, part of the power of probability-one homotopy algorithms derives from the lack of a monotonicity requirement for λ.

The zero-finding problem (1) is more complicated. Suppose that there exists a C^2 map

$$\rho : E^m \times [0, 1) \times E^n \to E^n \tag{3}$$

such that

(a) the $n \times (m + 1 + n)$ Jacobian matrix $D\rho(a, \lambda, x)$ has rank n on the set

$$\rho^{-1}(0) = \{\, (a, \lambda, x) \mid a \in E^m, 0 \le \lambda < 1, x \in E^n, \rho(a, \lambda, x) = 0 \,\},$$

and for any fixed $a \in E^m$, letting $\rho_a(\lambda, x) = \rho(a, \lambda, x)$,

(b) $\rho_a(0, x) = \rho(a, 0, x) = 0$ has a unique solution x_0,

(c) $\rho_a(1, x) = F(x)$,

(d) $\rho_a^{-1}(0)$ is bounded.

Then for almost all $a \in E^m$ there exists a zero curve γ of ρ_a along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0, x_0)$, and reaching a zero $\bar{x}$ of F at $\lambda = 1$. γ does not intersect itself and is disjoint from any other zeros of ρ_a. The globally convergent homotopy algorithm is to pick $a \in E^m$ (which uniquely determines x_0), and then track the homotopy zero curve γ starting at $(0, x_0)$ until the point $(1, \bar{x})$ is reached.

There are many different algorithms for tracking the zero curve γ; the mathematical software package HOMPACK [32], [34] supports three such algorithms: ordinary differential equation-based, normal flow, and augmented Jacobian matrix. Small dense and large sparse Jacobian matrices require substantially different algorithms. For the circuit simulation problems considered in this paper, the Jacobian matrix $DF(x)$ is an invertible, nonsymmetric $n \times n$ matrix, possibly with diagonal blocks, but with largely unstructured nonzeros outside those blocks.

For practical computational reasons, it is convenient to reverse the order of λ and x, and write $\rho_a(x, \lambda)$. In this case the $n \times (n + 1)$ Jacobian matrix $D\rho_a(x, \lambda)$ has a dense last column and the sparse nonsymmetric $DF(x)$ as its leading $n \times n$ submatrix. If $F(x)$ is C^2, a is such that the Jacobian matrix $D\rho_a(x, \lambda)$ has full rank along γ, and γ is bounded, then the zero curve γ is C^1 and can be parameterized with respect to arc length s. Moreover, the unit tangent vector $(dx/ds, d\lambda/ds)$ to γ lies in the kernel of $D\rho_a$. Since curve tracking algorithms require tangent vectors to γ, it is necessary to compute the one-dimensional kernel of $D\rho_a$ at points along γ.

3. Iterative methods for nonsymmetric invertible systems.

Since the Jacobian matrix is $n \times (n + 1)$, the rectangular linear system $[D\rho_a(x, \lambda)]z = 0$ is converted to an equivalent square linear system of rank $N = n + 1$:

$$Ax = \begin{pmatrix} D_x\rho_a & D_\lambda\rho_a \\ w^t & d \end{pmatrix} x = \begin{pmatrix} 0 \\ c \end{pmatrix}, \tag{4}$$

by augmenting $D\rho_a$ with an additional row. The row $(w^t \quad d)$ was taken to be a standard basis vector e_k^t, where k was the index of the largest magnitude component of the unit tangent vector $\bar{y}$ to γ found at the previous step along γ. Other choices are possible, such as $(w^t \quad d) = \bar{y}^t$ or using $w = (D_\lambda\rho_a)$ and choosing d so as to make A nonsingular. All these choices were tested. Taking e_k^t for the augmenting row maintained the high sparsity of the Jacobian matrix, and yielded lower execution times, although slightly higher iteration counts. In the numerical experiments reported below, e_k^t was used as the augmenting row, and c was taken to be the max norm of the previous unit tangent vector to γ.

This paper is primarily concerned with solving the linear system (4). The coefficient matrix A will be invertible, nonsymmetric, unstructured, and highly sparse. Available iterative methods offer a number of advantages when dealing with a large, sparse, unstructured, linear system. The coefficient matrix A is usually needed only to perform a few matrix-vector multiplications at each iteration, and that is generally cheap for a sparse problem. In addition, the iterative methods considered avoid generating any matrix fill-in, and thus require the storage of only a small number of auxiliary vectors at each iteration. Some of the leading iterative methods applicable to our class of problems are described in the following subsections.

Let Q be an invertible matrix of the same size as A; then the linear system $Ax = b$ is equivalent to

$$\tilde{A}x = Q^{-1}Ax = Q^{-1}b = \tilde{b}. \tag{5}$$

The use of such an auxiliary matrix is known as *preconditioning*. Ideally, the preconditioned linear system (5) requires less computational effort to solve than does $Ax = b$, due to a reduction in the number of iterations required to achieve convergence. The choice of an effective preconditioning matrix Q is often tied to some special structure or other property of the coefficient matrix A. Since the Jacobian matrices arising in the circuit simulation problems considered here lack symmetry and are almost certainly not positive definite, the selection of an effective preconditioner is nontrivial. Several choices are discussed in Section 4.

3.1. Craig's method. Craig's method [12] applies the conjugate gradient algorithm to the problem

$$AA^tz = b, \qquad x = A^tz,$$

without using either z or AA^t directly. Given a starting point x_0, at the k-th iteration Craig's method determines x_k so that $\|e_k\|_2 = \|x - x_k\|_2$ is minimized over the translated space

$$x_0 + \langle A^tr_0, A^t(AA^t)r_0, \cdots, A^t(AA^t)^{k-1}r_0\rangle,$$

or equivalently

$$e_k \perp \langle A^tr_0, A^t(AA^t)r_0, \cdots, A^t(AA^t)^{k-1}r_0\rangle. \tag{6}$$

The error norm $\|e_k\|_2$ is guaranteed to decrease at each iteration, so progress is assured. However, the rate of convergence of Craig's method depends upon the condition number of the matrix AA^t, and for that reason an effective preconditioner is desirable.

Craig's method (with preconditioning matrix Q) is:

```
choose $x_0$, $Q$
set $r_0 = b - Ax_0$;
    $\tilde{r}_0 = Q^{-1}r_0$;
    $p_0 = A^tQ^{-t}\tilde{r}_0$;
for $i = 0, 1, \ldots$ until convergence do
begin
    $a_i = \frac{(\tilde{r}_i, \tilde{r}_i)}{(p_i, p_i)}$;
    $x_{i+1} = x_i + a_ip_i$;
    $\tilde{r}_{i+1} = \tilde{r}_i - a_iQ^{-1}Ap_i$;
    $b_i = \frac{(\tilde{r}_{i+1}, \tilde{r}_{i+1})}{(\tilde{r}_i, \tilde{r}_i)}$;
    $p_{i+1} = A^tQ^{-t}\tilde{r}_{i+1} + b_ip_i$;
end
```

With preconditioning, Craig's method requires, at minimum, storage of 5 vectors of length N, and each iteration requires two preconditioning solves, two matrix-vector products, two inner products and three SAXPY operations.

3.2. GMRES. At the k-th iteration, GMRES [21] computes x_k in the translated space

$$x_0 + \langle r_0, Ar_0, \cdots, A^{k-1}r_0 \rangle$$

to minimize the residual norm $\|r_k\|_2 = \|b - Ax_k\|_2$, or equivalently to guarantee

$$r_k \perp \langle Ar_0, A^2 r_0, \cdots, A^k r_0 \rangle. \tag{7}$$

In contrast to Craig's method, the rate of convergence of GMRES does not depend on the condition number of A. Let P_k be the space of complex polynomials of degree k or less, and for any bounded set S of complex numbers and $p_k \in P_k$, define $\|p_k(z)\|_S = \sup_{z\in S}|p_k(z)|$. Let Λ_ϵ be the set of *ϵ-pseudo-eigenvalues* of A: all complex numbers z which are eigenvalues of some matrix $A + E$ with $\|E\| \leq \epsilon$. Let L be the arc length of the boundary of Λ_ϵ. For arbitrary invertible A and $\epsilon > 0$, the residual norms must satisfy [18]:

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \frac{L}{2\pi\epsilon} \inf_{\substack{p_k\in P_k \\ p_k(0)=1}} \|p_k(z)\|_{\Lambda_\epsilon}.$$

Speaking loosely then, the rate of convergence of GMRES depends on the spectrum of A. Also in contrast to Craig's method, the reduction of the residual norms is not necessarily strictly monotonic.

GMRES (with preconditioning matrix Q) is:

```
choose $x_0, Q$
set $r_0 = b - Ax_0$;
    $\tilde{r}_0 = Q^{-1} r_0$;
    $v_1 = \frac{\tilde{r}_0}{\|\tilde{r}_0\|_2}$;
for $m = 1, 2, \ldots$ until convergence do
begin
    for $j = 1$ to $m$ do
        $h_{j,m} = (Q^{-1}Av_m, v_j)$;
    $\tilde{v}_{m+1} = Q^{-1}Av_m - \sum_{j=1}^{m} h_{j,m} v_j$;
    $h_{m+1,m} = \|\tilde{v}_{m+1}\|_2$;
    $v_{m+1} = \tilde{v}_{m+1} \,/\, h_{m+1,m}$;
    Find $y_m$ to minimize $\big\| \|\tilde{r}_0\|_2 e_1 - \bar{H}_m y \big\|_2$ where $\bar{H}_m$ is described in [21];
    $x_m = x_0 + \sum_{j=1}^{m} (y_m)_j v_j$;
end
```

The m-th iteration of GMRES requires one preconditioning solve, one matrix-vector product, a least-squares solution and $O(m)$ SAXPY operations and inner products. Unfortunately, GMRES also requires the retention of a potentially large number of vectors of length N. In order to control the storage requirements, the algorithm is frequently used in a truncated or restarted form, GMRES(k), in which the inner loop is limited to k iterations, for some $k \ll N$, and the algorithm is restarted unless the residual norm has been reduced to an acceptable size. The price of this restarting is that the residual norms may not converge to zero, but may stagnate at some positive number.

3.3. BiCGSTAB. At the k-th iteration, the biconjugate gradient method BiCG [5], [18] computes x_k in the same translated space as GMRES, but does not attempt to choose x_k so as to minimize the residual norm. Instead, x_k is chosen to satisfy the orthogonality condition

$$r_k \perp \langle \tilde{r}_0, A^t\tilde{r}_0, \cdots, (A^t)^{k-1}\tilde{r}_0 \rangle, \tag{8}$$

where $\tilde{r}_0$ is chosen so that $(r_0, \tilde{r}_0) \neq 0$. BiCG cannot achieve convergence in fewer iterations than GMRES, but may require less time since BiCG iterations are significantly cheaper than those of GMRES. The BiCG iterates are computed using three-term recurrences, and each iteration of BiCG requires a constant number of vector operations and a fixed amount of storage, in contrast to GMRES. The conjugate gradients squared (CGS) algorithm [23] reorganizes the BiCG algorithm to eliminate multiplications involving A^t and increase the rate of convergence by up to a factor of 2.

Since BiCG and CGS do not minimize the residual norm on each iteration, their convergence can be irregular [18], [26]. BiCGSTAB [26] is a recent modification of the BiCG algorithm that attempts to achieve both faster and smoother convergence than BiCG. The iterates in the BiCGSTAB method are also obtained with three-term recurrences, so the storage and complexity advantages of BiCG are not lost.

BiCGSTAB is:

```
choose $x_0$
set $r_0 = b - Ax_0$;
    $\rho_0 = \alpha = \omega_0 = 1$;
    $v_0 = p_0 = 0$;
choose $\tilde{r}_0$ such that $(r_0, \tilde{r}_0) \neq 0$;
for $i = 1, 2, 3, \ldots$ until convergence do
begin
    $\rho_i = (\tilde{r}_0, r_{i-1})$;        $\beta = (\rho_i/\rho_{i-1})(\alpha/\omega_{i-1})$;
    $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$;
    $v_i = Ap_i$;                   $\alpha = \rho_i/(\tilde{r}_0, v_i)$;
    $s = r_{i-1} - \alpha v_i$;             $t = As$;
    $\omega_i = (t, s)/(t, t)$;
    $x_i = x_{i-1} + \alpha p_i + \omega_i s$;
    if $x_i$ is accurate enough then quit;
    $r_i = s - \omega_i t$;
end
```

Each iteration of BiCGSTAB requires two matrix-vector products, four inner products and five SAXPY operations. If preconditioning is used, then two preconditioning solves are also required.

3.4. QMR. The *quasi-minimal residual* method (QMR) [8] is based on a modification of the classical nonsymmetric Lanczos algorithm [16]. Given two starting vectors, v_1 and w_1, the Lanczos algorithm uses three-term recurrences to generate sequences $\{v_i\}_{i=1}^{L}$ and $\{w_i\}_{i=1}^{L}$ of vectors satisfying, for $m = 1, \ldots, L$,

$$\begin{aligned} \operatorname{span}\{v_1, \ldots, v_m\} &= \langle v_1, Av_1, \cdots, A^{m-1}v_1\rangle, \\ \operatorname{span}\{w_1, \ldots, w_m\} &= \langle w_1, A^t w_1, \cdots, (A^t)^{m-1}w_1\rangle, \end{aligned} \tag{9}$$

and

$$w_i^t v_j = d_i \delta_{ij}, \text{with } d_i \neq 0, \text{ for all } i, j = 1, \ldots, L, \tag{10}$$

where δ_{ij} is the Kronecker delta. It is possible that the classical Lanczos algorithm can break down, where v_m and w_m are orthogonal and nonzero. The look-ahead Lanczos [6] algorithm attempts to avoid such failure by generating different sequences of vectors v_i and w_i, imposing a block biorthogonality condition in place of (10). These look-ahead Lanczos vectors also satisfy three-term recurrences, so the computational cost is comparable to that of the classical Lanczos algorithm.

At the k-th iteration, QMR computes an iterate x_k in the same translated space as GMRES. If the initial residual r_0 is taken as one of the starting vectors v_1 for the look-ahead Lanczos method, then the residual at the k-th iteration satisfies

$$r_k = V^{(k+1)}\left(\|r_0\|_2 e_1 - H_e^{(k)} z\right), \tag{11}$$

where the columns of the matrix $V^{(k+1)}$ are the right (look-ahead) Lanczos vectors $v_1, v_2, \ldots, v_{k+1}$, and $H_e^{(k)}$ is a $k \times k$ block tridiagonal matrix augmented with a row of the form ρe_k^t. In equation (11), the vector z is the unique minimizer of

$$\left\| \|r_0\|_2 e_1 - H_e^{(k)} z \right\|_2, \tag{12}$$

which can be found with considerably less work than would be needed to minimize the residual norm, since the matrix $V^{(k+1)}$ will not usually be unitary. This choice of z gives an r_k minimal with respect to the norm $\|r\| = \|\alpha\|_2$, $r = V^{(k+1)}\alpha$, and satisfying (8) in a generalized sense. The next iterate is then given by

$$x_{k+1} = x_0 + V^{(k+1)} z.$$

3.5. LSQR. The algorithm LSQR [19] uses a bidiagonalization scheme due to Golub and Kahan [10] to solve both least squares problems and consistent systems of linear equations. At the k-th iteration, LSQR computes x_k in the translated space

$$x_0 + \langle A^t r_0, (A^t A)A^t r_0, \cdots, (A^t A)^{k-1} A^t r_0\rangle$$

to minimize the residual norm $\|r_k\|_2 = \|b - Ax_k\|_2$. The iterates x_k are analytically the same as those produced when the conjugate gradient iteration is applied to the normal equations, and the residual norm is guaranteed to decrease monotonically.

At each iteration, the residual satisfies the equation

$$r_k = U^{(k+1)} \left(\|r_0\|_2 e_1 - B^{(k)} z \right), \tag{13}$$

where $U^{(k+1)}$ is orthogonal and $B^{(k)}$ is bidiagonal. Thus the residual norm may be minimized by finding z to minimize

$$\left\| \|r_0\|_2 e_1 - B^{(k)} z \right\|_2 . \tag{14}$$

The special form of $B^{(k)}$ allows the computation of the next iterate x_{k+1} as an update of x_k, requiring only two matrix-vector products, four SAXPY operations, and two vector norm calculations.

A description of the LSQR algorithm follows. The scalars α_i and β_i are chosen to normalize the corresponding vectors v_i and u_i. So the assignment $\beta_1 u_1 = b$ implies the calculations $\beta_1 = \|b\|_2$ and $u_1 = (1/\beta_1)b$.

```
set $x_0 = 0$;   $\beta_1 u_1 = b$;   $\alpha_1 v_1 = A^t u_1$;
    $w_1 = v_1$;   $\bar{\phi}_1 = \beta_1$;   $\bar{\rho}_1 = \alpha_1$;
for $i = 1, 2, 3, \ldots$ until convergence do
begin
    $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$;   $\alpha_{i+1} v_{i+1} = A^t u_{i+1} - \beta_{i+1} v_i$;
    $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$;   $c_i = \bar{\rho}_i / \rho_i$;   $s_i = \beta_{i+1} / \rho_i$;
    $\theta_{i+1} = s_i \alpha_{i+1}$;   $\bar{\rho}_{i+1} = -c_i \alpha_{i+1}$;
    $\phi_i = c_i \bar{\phi}_i$;   $\bar{\phi}_{i+1} = s_i \bar{\phi}_i$;
    $x_i = x_{i-1} + (\phi_i / \rho_i) w_i$;   $w_{i+1} = v_{i+1} - (\theta_{i+1} / \rho_i) w_i$;
end
```

3.6. KACZ. There are a number of iterative projection methods which are suitable for large, sparse, nonsymmetric linear systems [2]. Kaczmarz [14] proposed an iterative method in which each equation is viewed as a hyperplane so that the solution is simply the point of intersection of the hyperplanes. The initial guess is then projected onto the first hyperplane, the resulting point is projected onto the second hyperplane, and so on. This approach applies equally well if the hyperplanes are defined by blocks of equations. Partition A into m blocks of rows as

$$A^t = [A_1, A_2, \ldots, A_m],$$

and partition b conformally. The orthogonal projection of a vector x onto the range of A_i is given by $P_i x = A_i (A_i^t A_i)^{-1} A_i^t x$. Let $Q_u = (I - P_m) \cdots (I - P_1)$, $\hat{b}_i = A_i (A_i^t A_i)^{-1} b_i$ for $1 \le i \le m$, and

$$b_u = \hat{b}_m + (I - P_m)\hat{b}_{m-1} + \cdots + (I - P_m) \cdots (I - P_2)\hat{b}_1 .$$

Then for $k \ge 0$ the Kaczmarz iterates are given by

$$x_{k+1} = Q_u x_k + b_u .$$

This method will converge for any system with nonzero rows, even if the system is rectangular, singular or inconsistent. Unfortunately the rate of convergence depends on the spectral radius of the iteration matrix Q_u, and may be arbitrarily slow. In order to improve the rate of convergence, Björck and Elfving [1] proposed the row projection method KACZ, in which the forward sweep through the rows of A is followed with a backward sweep, effectively symmetrizing the iteration matrix. The general formulation of the KACZ method involves a relaxation parameter ω; for a number of reasons the best choice for ω is generally 1, and that value was used in the experiments reported in this paper. A full discussion of this method may be found in [2].

Let $AA^t = L + D + L^t$, where L is block lower triangular and D is block diagonal. Let $\bar{b} = A^t(D+L)^{-t}D(D+L)^{-1}b$ and $Q = (I - P_1)(I - P_2)\cdots(I - P_m)^2\cdots(I - P_2)(I - P_1)$. Then for $k \geq 0$ the KACZ iterates are given by

$$x_{k+1} = Qx_k + \bar{b}.$$

4. Preconditioning strategies. The rate of convergence of the iterative methods considered here depends upon the spectrum and condition number of the coefficient matrix. For the problems in the test suite, many of the Jacobian matrices along the homotopy zero curve γ have condition numbers on the order of 10^6 to 10^9. Moreover, many of the Jacobian matrices have a significant number of eigenvalues far from one. As a result, the performance of each of the methods would be expected to suffer. It may be possible to improve the spectrum or conditioning through use of a judiciously chosen preconditioning matrix Q.

The preconditioned iteration matrix $Q^{-1}A$ in (5) is not usually formed explicitly; the sparsity of A provides no guarantee that $Q^{-1}A$ will not be relatively dense. Thus the extra work required for preconditioning lies in the computation of matrix-vector products involving Q^{-1}, and preconditioning will be effective when the cost of these matrix-vector products is outweighed by the reduction in the number of iterations required to achieve convergence. Use of preconditioning also requires additional storage. For a sparse problem, this typically amounts to one extra array to store the elements of Q and another to store the associated indices, or roughly the same amount of storage needed for the matrix A. The preconditioning schemes that were examined were selected in an attempt to balance the density of the matrix Q against the desirable property that Q^{-1} approximate A^{-1}. Throughout the rest of this paper it is assumed that the diagonal of A contains no zeros. This assumption is always valid for the circuit models under consideration.

4.1. ILU preconditioning. The first preconditioner considered is the incomplete LU factorization (ILU) [13] of the coefficient matrix A. Let Z be a subset of the set of indices $\{(i,j) \mid 1 \leq i,j \leq N, i \neq j\}$, typically where A has structural zeros. Let $\bar{Z}$ be the complement of Z. Then the ILU factorization of A is given by $Q = LU$ where L and U are lower triangular and unit upper triangular matrices, respectively, satisfying

$$\begin{cases} L_{ij} = U_{ij} = 0, & \text{if } (i,j) \in Z; \\ Q_{ij} = A_{ij}, & \text{if } (i,j) \in \bar{Z}. \end{cases}$$

Taking the ILU factorization of A as the preconditioning matrix allows the preservation of the exact sparsity pattern in A, and so permits the arrays storing Q and A to share the same array of indices.

4.2. Sparse submatrix preconditioning. The Jacobian matrices considered in this paper have substantial numbers of off-diagonal entries that are very small in magnitude relative to the majority of nonzeros. This suggests constructing an inexpensive preconditioner by ignoring some of these relatively small values.

Let S be a matrix formed from A by taking the diagonal of A and a percentage of the largest off-diagonal elements of A, and zero elsewhere. The ILU factorization, Q, of S may be used as a preconditioner for A. We refer to this as ILUS preconditioning. This is similar to the ILUT(k) preconditioner [8], [20] but does not allow any fill-in. Taking the percentage of off-diagonal entries retained to be 100 yields the usual ILU preconditioner. Retaining a smaller percentage of the off-diagonal entries may produce a superior preconditioner; i.e., the condition number or spectrum of $Q^{-1}A$ may be better than if ILU preconditioning were used. In any case, the increased sparsity of the ILUS matrix Q will reduce the amount of work necessary to apply the preconditioner. If the ILUS preconditioner yields an iteration matrix whose conditioning or spectrum is only slightly worse than with ILU preconditioning, then use of ILUS preconditioning may reduce execution time, even though more iterations are required to achieve convergence.

Assuming that the percentage of off-diagonal entries to be retained is specified, the matrix S may be extracted with work proportional to $|\bar{Z}|$. The off-diagonal entries of A must be examined to determine a threshold value to serve as a lower bound for the retained off-diagonal entries. This may be done by using an order statistics algorithm [22], rather than a full sort of the off-diagonal entries. The nonzero part of the matrix S then consists of the diagonal of A and all off-diagonal entries of A whose magnitudes equal or exceed the threshold. The actual density of S may differ from the specified value, since there may be many equal elements in A. The Jacobian matrices to which this scheme was applied typically had $5N$ to $6N$ nonzeros, so the cost of computing the threshold was not large, so long as the value of the threshold did not need to be recomputed too often. The ILU factorization of S may be computed in the usual manner.

If we proceed naively, there is little difference between the use of ILU and ILUS preconditioning. Consider the application of an iterative method with ILUS preconditioning to a sequence of Jacobian matrices along a homotopy zero curve. The value of the threshold may be determined from the first Jacobian matrix. For each Jacobian matrix A, we extract a sparse matrix S from A in the manner described above, compute the ILU factorization Q of S, and perform preconditioned iterations of the method until it converges to a solution of (5). ILUS preconditioning potentially requires increased storage, since A and Q cannot share array indices.

However, the entries of A may vary considerably in magnitude as the zero curve γ is tracked. If the value of the threshold obtained from the initial Jacobian matrix is used along the entire zero curve, the actual percentage of off-diagonal entries retained will also vary. It is therefore desirable to consider the actual percentage of off-diagonal entries retained at each step, and recompute the threshold whenever the actual percentage retained differs excessively from the target percentage. In the following discussion, this variant will be called *threshold-adaptive* ILUS or simply ILUSt.

Care must be taken to avoid recomputing the threshold too often—experiments indicate that allowing the actual percentage retained to vary 10% or 15% from the target percentage produces good performance. In the experiments reported here, the threshold was not recomputed unless the actual percentage retained differed from the target by at least 15%.

Experiments with threshold-adaptive ILUS indicate that using a denser matrix S will generally produce lower iteration counts. However, use of a relatively sparser matrix S may result in smaller execution times, so long as the increase in the iteration count is modest. This suggests adjusting the target percentage upward or downward according to the number of iterations required for convergence in the previous step along the zero curve. In this way, a sparse preconditioner can automatically be used for relatively easy problems and a denser one for relatively hard problems. In the following discussion, this variant will be called *hybrid-adaptive* ILUS or simply ILUSh. Given a desired iteration count, a current target percentage, a corresponding threshold value, and the actual percentage of off-diagonal entries retained at the previous step along γ, ILUSh preconditioning is implemented as follows:

```
if $|actual\_pct - target\_pct| > tol_1$ then
    recompute the threshold.
Extract sparse matrix $S$ from $A$.
Compute ILU factorization $Q$ of $S$.
for $iteration\_count := 1$ step 1 until convergence do
    perform preconditioned iteration and increment $iteration\_count$.
$iteration\_ratio := iteration\_count/iteration\_limit$.
if $(iteration\_ratio < tol_2)$ or $(iteration\_ratio > tol_3)$ then
    reset $target\_pct$.
```

The values of tol_2 and tol_3, which determine when the target percentage will be reset, should not be too close to 1. Experiments indicate that taking $tol_2 = 0.75$ and $tol_3 = 1.5$ produces good results, and those values were used in the numerical experiments reported below. The experiments also indicate that the target percentage is reset far more often than the threshold is recomputed, i.e., when the target percentage is reset, the actual percentage of off-diagonal elements retained still falls within an acceptable range. A reset of the target percentage requires only a few flops in each step along γ, so these unnecessary resets are acceptable.

The experiments with threshold-adaptive ILUS indicate that setting the target percentage too low can result in a dramatic increase in the iteration count. Therefore a lower bound of 50% was established for the target percentage in the experiments reported below. Two schemes for adjusting the target percentage were examined. The first was a simple averaging scheme:

```
if $(iteration\_ratio < tol_2)$ then
    $target\_pct := (target\_pct + minimum\_pct)/2$;
else if $(iteration\_ratio > tol_3)$ then
    $target\_pct := (target\_pct + 100)/2$;
```

while the second used a proportional adjustment:

```
if ($iteration\_ratio < tol_2$) then
    $target\_pct := target\_pct$
            $- (target\_pct - minimum\_pct) * (1 - iteration\_ratio)$
else if ($iteration\_ratio > tol_3$) then
    $target\_pct := target\_pct + (100 - target\_pct) * (1 - 1/iteration\_ratio)$
```

Tests indicated that the proportional adjustment scheme held a slight advantage, and the experiments reported here used that approach.

The iteration limit represents a target for the iteration count on each step along the zero curve. In order to adjust expectations to experience, it is necessary to recompute the iteration limit in order to reflect the number of iterations actually required for convergence. In the codes tested, whenever the target percentage was reset, the iteration limit was reset by averaging it with the iteration count from the previous step. The effect of choosing different initial values for the iteration limit was examined, and the choice was found to not influence results significantly. In the experiments reported here, the iteration limit was initialized to $N/2$.

Finally, an initial target percentage must be specified. The hybrid-adaptive ILUS codes were tested using starting percentages ranging from 100% down to 40%, in increments of 5%. While there was some variation in the results, taking an initial value of 100% produced results that were generally as good as for any other initial value. In the absence of a reason to do otherwise, it seems reasonable to initialize the target percentage to 100%, and that was done in the experiments reported below.

4.3. ILU preconditioning with limited fill. The Jacobian matrices for some of the test problems have a pattern of 6×6 blocks along a portion of the diagonal and relatively unstructured nonzeros elsewhere. The diagonal blocks (which correspond to the internal variables in a particular transistor model) are structured, nearly symmetric, and approximately 78% full. Most of the nonzeros in the Jacobian matrices occur within these blocks, so it is possible that allowing fill to occur within these blocks when the ILU factorization Q is computed will result in a Q^{-1} that is a better approximation to A^{-1}. In the following discussion, this scheme is referred to as *block-fill* ILU or simply ILUB.

The implementation of ILUB preconditioning only requires the insertion into the sparse data structure of additional cells corresponding to the empty cells in the 6×6 blocks. The ILU factorization may then be computed in the usual manner. Since allowing fill within the diagonal blocks changes the structure of the matrix, the ILUB preconditioning matrix Q cannot share an index array with A. Thus the ILUB scheme will require more storage than either ILU or ILUS. Moreover, the increased density of Q implies an increase in the cost of applying the preconditioner.

A number of alternative preconditioning schemes that also use an ILU factorization with limited fill were considered. The presence of the diagonal blocks suggests that allowing fill to occur within bands encompassing the blocks might produce a more effective preconditioner. This scheme, called *band-fill* ILU, was implemented in a manner similar to that used for ILUB. Tests with Craig's method and GMRES(k) using band-fill ILU preconditioning demonstrated occasional slight reductions in the average number of iterations needed for convergence (compared to ILUB preconditioning). However, the considerable increase in density led to higher execution times in every case. Limited experiments were also conducted with a hybrid-adaptive variation of the band-fill ILU preconditioner, in which small entries outside the bands were discarded. The results indicated that this approach was likely to increase iteration counts and execution times substantially. A variation of ILUB preconditioning, in which the smallest off-diagonal entries lying outside the diagonal blocks are discarded, is also possible. In view of the performance of the hybrid-adaptive band-fill ILU preconditioner, that ILUB variation was not pursued. One could also use as a preconditioner the LU factorization of the matrix consisting of the entries of the Jacobian matrix that lie on the diagonal or within the diagonal blocks. Again, limited experiments showed these preconditioners to be relatively ineffective. All these minor variations add no substantially different information from the ILUB preconditioner, and results for them are not included in §6.

5. Numerical analysis of electronic circuit equations.

Computer simulation is a necessary adjunct to the design of integrated circuits. The high cost of design iterations, combined with market pressure in the chip business place a high premium on design methodologies which produce working silicon as quickly as possible. Computer simulation techniques provide important performance information to a designer without the expense and delay of building a prototype circuit.

A necessary component of almost any simulation task is the computation of a DC operating point. This point is used as the initial value for an ODE solver, or can be used to compute an approximate linearized model of the circuit for so-called "small signal" analysis.

The sharply nonlinear behavior of certain semiconductor elements makes the computation of an operating point difficult. This section describes a formulation of the DC operating point problem using globally convergent homotopy methods. Section 6 presents performance data for a suite of bipolar integrated circuits taken from current industrial designs.

5.1. Equation formulation and device models.

The DC operating point problem is typically formulated as a system of n equations in n unknowns to be solved for zero, i.e., $F(x) = 0$, where $F : E^n \to E^n$. The unknowns are node voltages and branch currents and the equations represent the application of Kirchoff's current and voltage laws at various points in the circuit. Different circuit components (resistor, transistor, diode, etc.) impose linear and nonlinear constraints among the unknowns.

Values of n range up to 100,000. For large n, the Jacobian matrix of F is very sparse, with only a few nonzero elements per row. In present fabrication technologies, circuit components are arranged in the form of a graph which is almost planar, which puts a limit on the density of interconnections between circuit elements, leading to a sparse Jacobian matrix.

If a circuit contains only the familiar two-terminal elements—voltage source, current source, resistor and diode—it is possible to formulate equations so that the resulting Jacobian matrix is symmetric positive definite. However, the introduction of any *coupling* element like a transistor or controlled source destroys the symmetry of the matrix. Thus, except for rather special cases, circuit equations lead to large, sparse nonsymmetric matrix problems. In some cases, a circuit element like a transistor is represented as a small "subcircuit", which is installed in place of every transistor in the network. This policy results in a replication of structurally identical submatrices throughout the overall system Jacobian matrix. This structure can be used to advantage during the solution of linear systems involving the Jacobian matrix.

Of course, electronic devices operate in a fashion which is mathematically smooth, although device modeling subroutines might not always reflect this. Semiconductor physics often analyzes the behavior of a circuit element by considering qualitatively different "regions" of behavior. For example, the operation of a diode is classified as "reverse biased" or "forward biased". Equations describing the operation of the device within one region of operation are naturally C^∞ smooth, however, transitions between regions are problematic. A simple method of achieving smooth modeling is to sample the operation of the device at various voltages and currents, then fit a spline through the sample points. Such *table models* are popular and allow a model for a device to be built before the detailed physics of the element is known. However, each different setting of the parameters of the device requires a new table. Also, in some cases it is useful to compute *sensitivity information*, which shows how the response of the circuit element changes with a perturbation in one or more of its parameters. This kind of analysis is easier with so-called *analytical models*, which represent the behavior of the device with equations derived from an understanding of the semiconductor physics of the device. These equations are symbolic expressions involving the device parameters, voltages and currents. Thus, it is possible to compute exact analytic derivatives of voltage and current with respect to device parameters. There is a difficulty with analytic models, however, in getting smooth transitions between regions of operation. In bad cases, the analytic equations are not even continuous across region boundaries. Even when formulated as smooth functions, device modeling functions can still present extremely sharp nonlinearities. This is particularly true for bipolar devices because of the exponential nature of the *pn*-junction equation.

5.2. Homotopies for circuit equations. Homotopy (continuation) methods allow reliable solution of DC operating point equations. However, the performance of such methods is sensitive to the manner in which the homotopy parameter λ is introduced into the equations. The standard embedding $H(x, \lambda) = \lambda F(x) + (1 - \lambda)(x - a)$ is widely used in continuation work, but our computational experience shows that this is a very poor embedding for circuit equations. Generally speaking, good performance can be obtained only by pushing the continuation parameter down *into the device model equations*. This requires minor modifications to the device model code, but the results are well worth the effort [17]. Consider, for example, the classic Ebers-Moll model for a bipolar transistor [9]. Suppose the forward and reverse current gains (α_F, α_R) are multiplied by λ. The result is a somewhat artificial *variable gain* transistor. The random perturbation required by the probability-one homotopy theory is accomplished by connecting a *leakage circuit* from each node of the circuit to ground. This circuit consists of a conductance in series with a random value voltage source. At $\lambda = 0$, each transistor reduces to a pair of diodes, and a damped Newton scheme is able to solve the start system $\rho_a(x, 0) = 0$ quickly. As λ approaches one, the value of the leakage conductance goes to zero, thus disconnecting the leakage elements from the circuit. Moreover, the transistors are restored to a full-gain condition at $\lambda = 1$, so the solution to the homotopy equations at $\lambda = 1$ is an exact DC operating point of

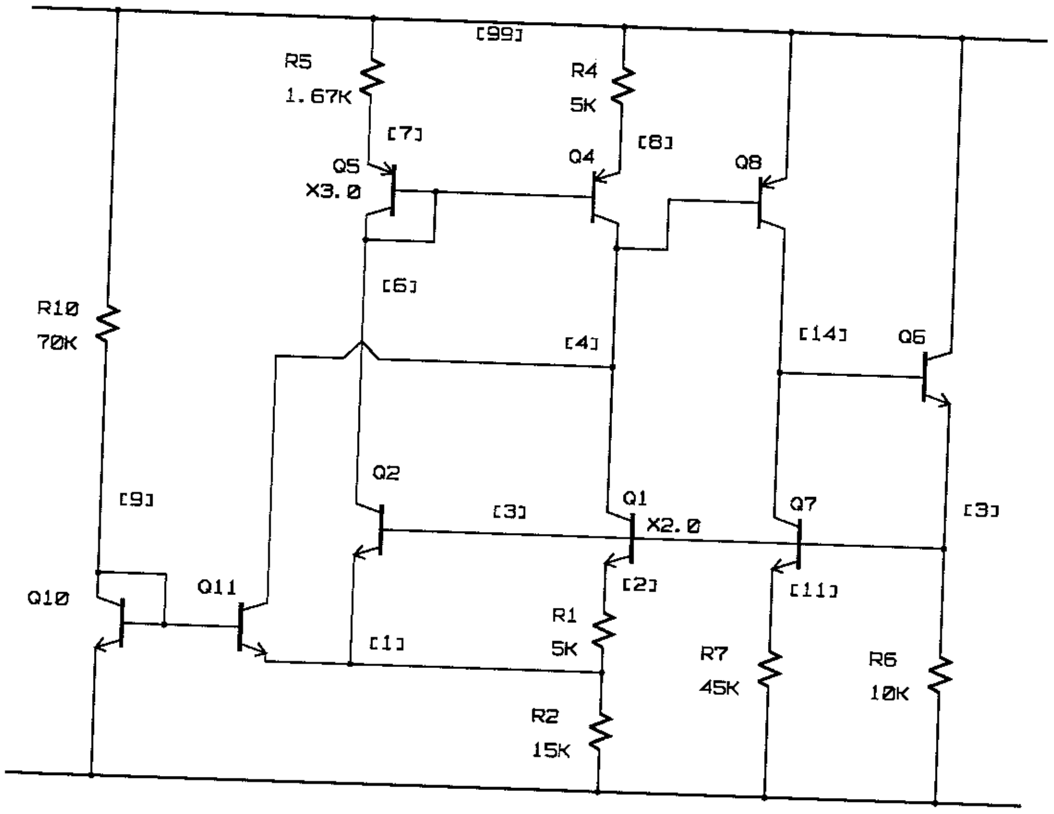

FIG. 1. *Circuit diagram for the circuit* vref.

the network. The performance results reported below are based on this variable gain homotopy map.

5.3. Circuit simulation test problems. The test problems consist of real circuits studied and used by scientists at AT&T Bell Laboratories. Some of the problems are described in complete detail (with circuit diagrams) in [17], and the data for all of them is available from the authors or *netlib*. The test data were obtained by solving each problem with HOMPACK, using a direct method to compute the kernel of the Jacobian matrix at each step along the zero curve, and writing the nonzero entries of the Jacobian matrices to a file. The iterative methods considered here were then applied to each sequence of Jacobian matrices. The sparse Jacobian matrices were stored in a variation of the compressed sparse row storage scheme [36]. The nonzero matrix entries were stored by rows in a linear array, with a parallel array of column indices. The starting indices for each row were also stored in a second integer array. The rows, and column indices for each row, were assumed to be in order within the linear arrays.

Table 1 gives the test problem names (matching those in [17]), dimension N, number of nonzeros NZ, and the number of Jacobian matrices NJ. The column labeled DB indicates whether the Jacobian matrices for each problem had diagonal blocks. Fig. 1 shows one of the circuits from which the test problems were derived. This circuit, known as the Brokaw voltage reference circuit [17], is a fairly well known circuit. The sparsity pattern for the Jacobian matrices corresponding to the circuit bgatt is shown in Fig. 2; note the diagonal blocks corresponding to the transistor subcircuits mentioned earlier.

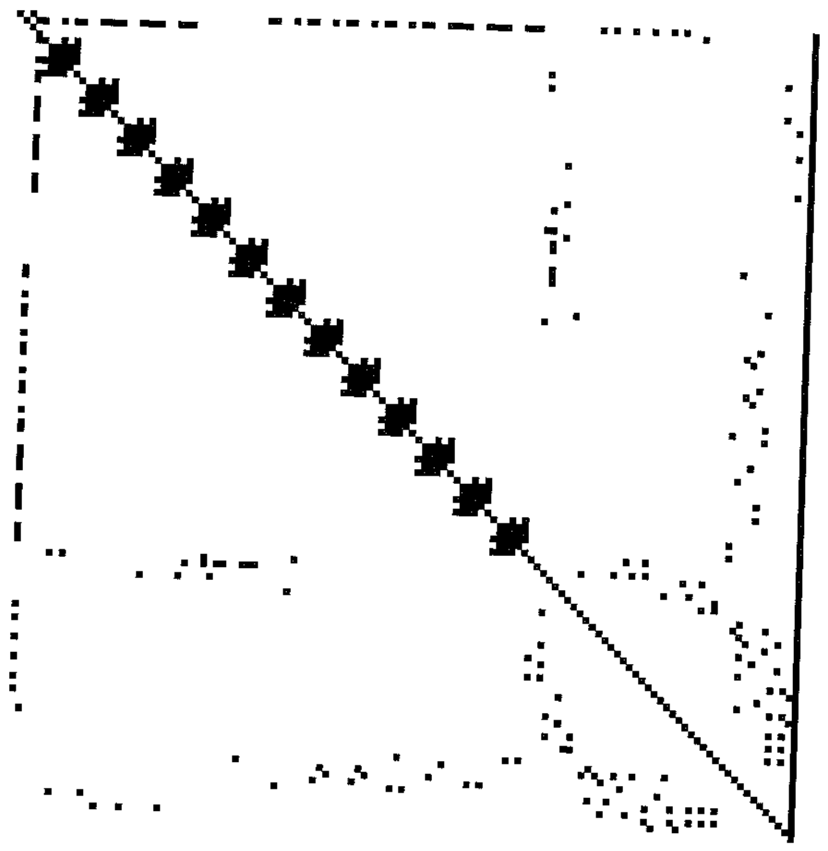

FIG. 2. *Jacobian matrix sparsity pattern corresponding to the circuit* bgatt.

TABLE 1

Problem	N	NZ	NJ	DB
rli13b	31	120	118	N
ups01a	59	342	25	Y
vref	67	411	13	Y
bgatt	125	782	14	Y
is7a	468	2871	5	N
is7b	1854	10849	5	Y

6. Results. Each iterative method was applied to each of the test problems. When applied without preconditioning, none of the methods achieved convergence in $5N$ iterations except on the problem of order 31; and even for this problem, the iteration counts and execution times achieved without preconditioning were two or three times larger than those achieved with any of the preconditioning schemes.

The row projection method KACZ consistently required substantially more time than any of the other methods. For example, KACZ required about 8 times as long to solve the problem of order 31 as did Craig's method. Thus KACZ was not tested with the full range of preconditioners, nor are numerical results for KACZ given in the tables. However, it must be noted that a sequential implementation of KACZ was used, and the row projection methods are perhaps best suited to a parallel environment.

TABLE 2

Craig's method with ILUSt preconditioning			
N	avg%	min, max, avg	time
31	100	1, 32, 17	4.83
	90	2, 32, 17	4.75
	80	2, 58, 19	5.14
	70	3, 64, 21	5.44
	60	3, 56, 21	5.22
	50	3, 60, 22	5.25
	41	4, 93, 30	6.88
	31	7, 95, 33	7.20
	25	7, 115, 35	7.53
59	100	28, 76, 42	5.93
	78	36, 97, 57	7.31
	70	36, 88, 57	7.05
	65	37, 87, 57	6.89
	59	37, 108, 58	6.91
	50	44, 116, 69	7.74
	41	107, 228, 163	17.94
67	100	75, 134, 89	7.50
	75	82, 132, 95	7.22
	65	81, 137, 96	7.07
	55	81, 133, 96	6.81
	46	82, 123, 96	6.57
	43	93, 142, 103	7.13
125	100	156, 359, 232	38.79
	80	221, 535, 331	51.24
	70	221, 556, 337	50.09
	64	221, 570, 340	49.39
	57	234, 619, 398	56.27

GMRES(k) with ILUSt preconditioning			
N	avg%	min, max, avg	time
31 (k=6)	100	1, 27, 11	2.41
	90	1, 27, 11	2.32
	80	1, 35, 13	2.46
	70	1, 42, 13	2.55
	60	1, 39, 13	2.36
	50	1, 46, 14	2.53
	41	1, 75, 20	3.32
	31	5, 113, 28	4.43
	26	5, 136, 34	5.22
59 (k=7)	100	12, 26, 17	1.72
	78	14, 35, 22	2.00
	70	14, 35, 22	1.93
	65	14, 35, 22	1.87
	59	14, 48, 24	1.94
	50	21, 61, 33	2.57
	41	*	
67 (k=20)	100	58, 115, 94	6.21
	75	72, 111, 92	5.61
	65	70, 111, 92	5.41
	55	70, 108, 90	5.14
	50	59, 108, 88	4.96
	49	72, 120, 93	5.23
125 (k=11)	100	52, 81, 68	7.97
	80	63, 175, 106	11.48
	70	64, 175, 106	11.01
	60	70, 220, 102	10.21
	55	83, 209, 114	11.16

Each of the other iterative methods was applied in combination with each of the preconditioning schemes to each of the test problems, if appropriate. The remainder of this section presents and discusses the numerical results obtained. Section 6.1 discusses the performance of the threshold-adaptive preconditioner. Section 6.2 discusses the relative performance of ILU, ILUSh and ILUB preconditioning. Section 6.3 compares the overall performance of the various methods.

In the tables that follow, avg% refers to the average percent of off-diagonal elements of the Jacobian matrices along γ retained in S. The minimum, maximum, and average number of iterations along the homotopy zero curve γ are shown, and the CPU time is in seconds on a DECstation 3100. The reported times are the medians of times obtained on from three to five runs. All code is double precision, since that is the default for the homotopy software HOMPACK. An asterisk denotes convergence failure at some point along the zero curve γ. In particular, a method was deemed to have failed if, at any step along γ, it required more than $5N$ iterations to converge. Convergence was construed to mean a relative residual less than 100 times machine epsilon; such high accuracy is frequently crucial for homotopy curve tracking [17].

6.1. Performance of ILUSt preconditioning. In order to begin to understand the potential of sparse submatrix preconditioners, the performance of the ILUSt preconditioner was tested with Craig's method and GMRES(k). Table 2 summarizes the results for the test problems rli13b ($N = 31$), ups01a ($N = 59$), vref ($N = 67$), and bgatt ($N = 125$), respectively. In testing GMRES(k), the value of k was chosen as the smallest for which convergence was achieved for relatively sparse preconditioners. Using ILU preconditioning, it was possible to solve each problem with a slightly smaller value of k.

For each method, using a slightly sparser preconditioner produced a slight reduction in execution time for the problem of order 31, even though the iteration count increased. For the problem of order 67, using a substantially sparser preconditioner with GMRES(k) actually reduced the iteration count. For each method, the execution time on that problem decreased significantly as the density of the preconditioner decreased, until the percentage of off-diagonal entries retained dropped below 50%. However, it is possible that the ILUS preconditioner is far less effective than ILU preconditioning on some problems, as the results on the second and fourth test problems show. Using the ILUSt preconditioner with a target percentage of 100% results in somewhat higher times than using the ILU preconditioner, since the ILUSt preconditioner requires separate storage for column indices and hence some additional data movement.

The results obtained using ILUSt preconditioning suggested that the use of a sparse preconditioner could improve performance in some cases. However, there was no obvious way to determine in advance what target percentage would produce the best performance. Moreover, comparing the performance of ILU and ILUSt preconditioning at each step along γ indicated that a denser preconditioner usually produced a lower iteration count when the Jacobian matrix was badly conditioned, but that a sparser preconditioner usually reduced execution time if the Jacobian matrix was not badly conditioned. The hybrid-adaptive preconditioning scheme was devised to provide a practical implementation of these ideas.

6.2. ILU, ILUSh and ILUB preconditioning. The results obtained using each iterative method in conjunction with ILU, ILUSh and ILUB preconditioning are presented in Table 3. The Jacobian matrices for the problems of order 31 and 468 do not have diagonal blocks, so the ILUB preconditioner is not appropriate for those problems. That is indicated in the table by the annotation NA.

The ILUB preconditioner was applicable to sixteen combinations of problem and method in which convergence was achieved. In terms of iteration counts, the ILUB preconditioner did as well as or better than either ILU or ILUSh preconditioning for every combination of problem and method, except when BiCGSTAB was applied to the problem of order 67. The increased cost of applying the denser ILUB preconditioner led to the highest execution time in five cases. In three cases the execution time using ILUB preconditioning was more than 10% greater than the best achieved with either of the other preconditioners. However, ILUB preconditioning reduced execution times by 10%, versus ILU, in two cases and by 75% in a third case. Execution times using ILUB preconditioning ranged from 25% to 119% of those achieved with the other preconditioners.

There were twenty-four combinations of problem and method in which convergence was achieved with ILUSh preconditioning. In eleven of those combinations, the number of iterations needed for convergence was so large that 100% of the off-diagonal

Table 3

	ILUSh			ILU		ILUB	
N	avg%	min, max, avg	time	min, max, avg	time	min, max, avg	time
				Craig's method			
31	73	1, 33, 17	4.60	1, 32, 17	4.78	NA	
59	100	28, 76, 42	5.84	28, 76, 42	5.46	28, 72, 37	5.24
67	100	75, 134, 89	7.40	75, 134, 89	6.87	68, 127, 82	6.80
125	100	156, 359, 232	38.49	156, 359, 232	35.32	134, 270, 187	31.79
468		*		*		NA	
1854		*		*		*	
				LSQR			
31	87	1, 30, 16	5.65	1, 30, 16	4.93	NA	
59	100	24, 53, 32	4.91	24, 53, 32	4.49	24, 50, 29	4.73
67	100	69, 89, 77	6.96	69,89, 77	6.36	63, 83, 70	6.68
125	100	147, 276, 195	34.52	147, 276, 195	32.63	126, 216, 155	29.24
468		*		*		NA	
1854		*		*		*	
				GMRES(k)			
31	84	1, 24, 11	2.42	1, 27, 11	2.36	NA	
59	71	12, 35, 22	1.96	12, 26, 17	1.70	7, 26, 16	1.81
67	100	58, 115, 94	6.36	58, 115, 94	6.19	20, 20, 20	1.55
125	100	52, 81, 68	8.11	52, 81, 68	7.93	53, 86, 67	8.37
468	84	107, 323, 162	33.60	106, 323, 161	33.20	NA	
1854	100	839, 1038, 926	1072.94	839, 1038, 926	1066.29	748, 1079, 898	1069.77
				BiCGSTAB			
31	66	1, 21, 9	2.51	1, 18, 8	2.56	NA	
59	66	8, 19, 14	1.86	8, 16, 11	1.80	8, 17, 11	2.04
67	100	35, 97, 59	4.97	35, 97, 59	4.96	27, 114, 64	5.86
125	71	23, 42, 35	5.37	22, 39, 28	5.12	21, 35, 26	5.18
468	78	72, 124, 98	23.07	67, 124, 87	20.97	NA	
1854		*		*		*	
				QMR			
31	78	2, 14, 9	4.59	2, 14, 9	4.72	NA	
59	69	10, 22, 16	3.30	10, 17, 13	3.15	10, 17, 12	3.38
67	100	27, 45, 33	3.66	27, 45, 33	3.68	24, 43, 32	4.12
125	75	28, 44, 34	6.86	26, 38, 29	6.34	25, 32, 27	6.32
468	80	77, 182, 113	32.62	77, 182, 104	30.69	NA	
1854		*		*		*	

entries of the Jacobian matrix were retained for the preconditioner, essentially reducing the ILUSh scheme to ILU preconditioning. The overhead in the ILUSh code is apparent in the timings for those cases. Overall, the ILUSh preconditioner was associated with the largest execution time in sixteen of the twenty-four problem/method combinations, including seven of the thirteen cases in which a sparse preconditioner was selected. ILUSh preconditioning produced the lowest execution time in only two cases, both involving the smallest test problem. Even in those cases the reduction in time was less than 4%, when compared to the best time achieved with the same method and ILU preconditioning. In six combinations of problem and method, the time required for convergence with ILUSh preconditioning was at least 10% greater than the best time achieved with the other preconditioners.

There were also twenty-four problem/method combinations in which convergence was achieved with ILU preconditioning. There were no cases in which the ILU preconditioner was either clearly best or worst among the three preconditioners tested. ILU preconditioning was associated with the lowest execution time in fourteen cases. In seven cases, the time required using the ILU preconditioner was second lowest, and within 5% of the best time for that problem/method combination. In only three cases did the ILU preconditioner correspond to an execution time more than 10% greater than the best time achieved with the other schemes.

6.3. Comparison of the iterative methods. Aside from KACZ, Craig's method and LSQR were clearly the slowest of the methods examined. Craig's method and LSQR were the only methods which failed on the problem of order 468. Craig's method did reduce the error norm at each iteration, as expected, but the rate of convergence was extremely slow, and the iteration limit of $5N$ was exceeded before convergence. LSQR managed a similar reduction of the residual norm, but again convergence was very slow. It is worth noting that LSQR, in combination with each preconditioner, solved the problems of order 59, 67, and 125 in significantly less time than Craig's method.

QMR did not produce the lowest overall execution time on any problem. However, with all three preconditioners, execution times with QMR were lower than for GMRES(k) on the problems of order 125 and 468. QMR with ILUSh and ILU preconditioning was also faster than GMRES(k) on the problem of order 67. QMR, with each preconditioner, was also faster than BiCGSTAB on the problem of order 67. The implementation of QMR that was used here requires the specification of an upper bound m on the number of look-ahead Lanczos vectors retained ($m = 5$ was used here). Changing the parameter m had little effect on the iteration count, and increasing m increased the execution time because of the overhead for the $m \times m$ blocks.

BiCGSTAB with ILU preconditioning achieved significantly better times on the problems of order 125 and 468 than any other combination of method and preconditioner. In addition, BiCGSTAB was only slightly slower than GMRES(k) on the problems of order 31 and 59. In almost every case, BiCGSTAB required fewer iterations to solve the problems of orders 31, 59, 125 and 468, when compared to the other methods using the same preconditioner.

The results for the six problems shown in Table 3 for GMRES(k) were obtained using $k = 6$, 7, 20, 11, 25 and 40, respectively. For smaller values of k, GMRES(k) failed to converge at some point along the zero curve if a relatively sparse preconditioner was used. Larger values of k can make a dramatic difference, but not always. For example, GMRES(20) with ILU preconditioning takes 6.19 seconds on the problem of size 67, but GMRES(25) takes only 1.79 seconds. Similar sensitivity to k occurs for the structural mechanics problems in [13]. The problem of order 67 was solved in considerably less time using GMRES(k) with ILUB preconditioning than with any other combination of method and preconditioner. The problems of order 31 and 59 were solved in slightly less time with GMRES(k) than any other method. Moreover, GMRES(k) was the only method which solved the largest problem.

6.4. Low rank perturbations. As suggested by Fig. 2, the Jacobian matrices in the test suite can be decomposed as $A = B+E$, where B is symmetrically structured (but not symmetric) and E is a matrix of low rank. Depending on the spectrum or conditioning of the matrix B compared to A, it may be advantageous to use B as the iteration matrix and apply the Sherman-Morrison formula to account for the low rank correction E. This approach has been used effectively in a number of applications [13], and was considered for the circuit simulation problems examined here. Unfortunately there was little difference between the condition numbers and spectra of the Jacobian matrices A and their symmetrically structured components B. Limited experiments with Craig's method and GMRES(k) did not show a significant advantage in overall performance and the approach was abandoned.

Whether an examination of the spectrum of the iteration matrix does, in fact, yield much useful information regarding the convergence rate of GMRES(k) is doubtful. For example, the eigenvalues of the eleventh Jacobian matrix for the vref circuit ($N = 67$) were computed after explicitly applying the ILU and ILUB preconditioners. The two spectra are only a Hausdorf distance of 0.18 apart, and have over fifty values in common. The performance of GMRES(k) on these two matrices is, however, strikingly different. Taking $x_0 = 0$, GMRES(20) converges in only 20 iterations if the ILUB preconditioner is used, but requires 275 iterations when ILU preconditioning is used. Clearly, the spectrum of the iteration matrix alone does not predict the performance of GMRES(k).

7. Conclusions.

Although the results are mixed and the interactions between the specific problem, the preconditioner, and the iterative method are complicated, some well supported conclusions can be drawn.

- None of the preconditioning schemes appears to be substantially better in terms of robustness or efficiency. Convergence on each problem was either achieved with every applicable preconditioner or not achieved at all. Considering execution times, the ILUSh preconditioner was clearly the least effective. The fact that ILUB preconditioning produced a spectacularly low execution time with GMRES(k) on the problem of order 67 must be balanced against the fact the ILUB preconditioning was associated with the highest execution time for five problem/method combinations. In contrast, ILU preconditioning produced the highest time in only three cases involving the smallest test problem, which was not suitable for ILUB preconditioning. Moreover, in those three cases ILU preconditioning exhibited only a slight disadvantage when compared with ILUSh. If only the three fastest methods are considered, there was only one combination of problem and method in which ILUB preconditioning led to substantially faster convergence than ILU. In summary, ILU preconditioning should be preferred over either ILUSh or ILUB preconditioning; the fancier methods are not worth the extra complexity.

• Of the iterative methods considered here, GMRES(k) and BiCGSTAB are the best for linear systems with large, sparse, nonsymmetric, indefinite, unstructured coefficient matrices, typified by the circuit simulation problems here. GMRES(k) achieved the lowest execution time on four of the six test problems, and solved the largest problem on which all the other iterative methods failed. However, the speed of GMRES(k) requires a substantial increase in storage, compared to the other iterative methods considered here. Given the extreme sparseness of the Jacobian matrices, typically between $5N$ and $6N$ nonzeros, GMRES(k) required extra storage that was as much as six or seven times that required for storing the Jacobian matrix. If memory usage is a concern, then BiCGSTAB offers an attractive alternative to GMRES(k). Using substantially less storage, BiCGSTAB solved the problems of order 31 and 59 in only slightly more time than required by GMRES(k). BiCGSTAB also achieved the lowest execution times overall on the problems of order 125 and 468. And although GMRES(k) solved the problem of order 67 in roughly 31% of the time required by BiCGSTAB, that required the retention of 20 N-vectors.

• Virtually all well known classes of applicable iterative methods have been considered here, and none have acceptable performance for linear systems with large, sparse, nonsymmetric, indefinite, unstructured coefficient matrices arising in circuit simulation. The best algorithms were BiCGSTAB and GMRES(k), but BiCGSTAB failed on the large problem is7b ($N = 1854$), and GMRES(k) requires an unpredictable and unacceptably large value of k to converge. To emphasize just how bad these iterative methods are (for circuit simulation problems, at least), using a proprietary ordering algorithm of AT&T, a direct stable LU factorization *never* generates more than $5N$ fill elements, and takes an order of magnitude less CPU time than GMRES(k) with $k \gg 5$. Improvements in preconditioning and algorithms are certain, but the gauntlet has been laid down for iterative methods on the type of linear systems considered here.

REFERENCES

[1] A. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[2] R. Bramley and A. Sameh, *Row projection methods for large nonsymmetric linear systems*, Tech. Report No. 957, Center for Supercomputing Research and Development, Univ. Illinois-Urbana, IL, January 1990.

[3] S. N. Chow, J. Mallet-Paret, and J. A. Yorke, *Finding zeros of maps: homotopy methods that are constructive with probability one*, Math. Comput., 32 (1978), pp. 887–899.

[4] C. DeSa, K. M. Irani, C. J. Ribbens, L. T. Watson, and H. F. Walker, *Preconditioned iterative methods for homotopy curve tracking*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 30–46.

[5] R. Fletcher, *Conjugate gradient methods for indefinite systems*, Proc. of the Dundee Biennial Conference on Numerical Analysis, Springer-Verlag, New York, 1975, pp. 73–89.

[6] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, Part I*, Tech. Report 90.45, RIACS, NASA Ames Research Center, Moffett Field, CA, November 1990.

[7] ———, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, Part II*, Tech. Report 91.09, RIACS, NASA Ames Research Center, Moffett Field, CA, April 1991.

[8] R. W. Freund and N. M. Nachtigal, *QMR: A quasi-minimal residual method for non-hermitian linear systems*, Numer. Math., to appear.

[9] I. Getreu, *Modeling the Bipolar Transistor*, Tektronix Inc., Beaverton, OR, 1976, pp. 9–23.

[10] G. H. Golub and W. Kahan, *Calculating the singular values and pseudoinverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205–224.

[11] G. H. Golub and C. F. van Loan, *Matrix Computations, Second Ed.*, Johns Hopkins University Press, Baltimore, 1989.

[12] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 409–436.

[13] K. M. Irani, M. P. Kamat, C. J. Ribbens, H. F. Walker, and L. T. Watson, *Experiments with conjugate gradient algorithms for homotopy curve tracking*, SIAM J. Optim., 1 (1991), pp. 222–251.

[14] S. Kaczmarz, *Angenäherte auflösung von systemen linearer gleichungen*, Bull. Intern. Acad. Polon. Sci. Class A., (1939), pp. 355–357.

[15] C. Kamath and A. Sameh, *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Computing, 9 (1988/1989), pp. 291–312.

[16] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 33–53.

[17] R. C. Melville, Lj. Trajković, S.-C. Fang, and L. T. Watson, *Globally convergent homotopy methods for the DC operating point problem*, Tech. Report TR-90-61, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1990.

[18] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, *How fast are nonsymmetric matrix iterations?*, Preliminary Proceedings of the Copper Mountain Conference on Iterative Methods, April 1990.

[19] C. C. Paige and M. A. Saunders, *LSQR: an algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Soft., 8 (1982), pp. 43–71.

[20] Y. Saad, *SPARSKIT: a basic tool kit for sparse matrix computations*, Tech. Report 90.20, RIACS, NASA Ames Research Center, Moffett Field, CA, May 1990.

[21] Y. Saad and M. H. Shultz, *GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[22] R. Sedgewick, *Algorithms, Second Ed.*, Addison-Wesley, New York, 1988.

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

[24] Lj. Trajković, R. C. Melville, and S.-C. Fang, *Passivity and no-gain properties establish global convergence of a homotopy method for DC operating points*, Proc. IEEE Int. Symp. on Circuits and Systems, New Orleans, LA, May, 1990, pp. 914–917.

[25] ———, *Finding DC operating points of transistor circuits using homotopy methods*, Proc. IEEE Int. Symp. on Circuits and Systems, Singapore, 1991.

[26] H. F. Walker, *Implementations of the GMRES method*, Comput. Phys. Comm., 53 (1989), pp. 311–320.

[27] ———, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 152–163.

[28] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.

[29] L. T. Watson, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.

[30] ———, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 394–401.

[31] ———, *Numerical linear algebra aspects of globally convergent homotopy methods*, SIAM Rev., 28 (1986), pp. 529–545.

[32] ———, *Globally convergent homotopy methods: a tutorial*, Appl. Math. Comput., 31BK (1989), pp. 529–545.

[33] ———, *A survey of probability-one homotopy methods for engineering optimization*, Tech. Report TR-90-47, Dept. of Computer Sci., VPI&SU, Blacksburg, VA, 1990.

[34] L. T. Watson, S. C. Billups, and A. P. Morgan, *HOMPACK: A suite of codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software, 13 (1987), pp. 281–310.

[35] L. T. Watson and D. Fenner, *Chow-Yorke algorithm for fixed points or zeros of C^2 maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.

[36] Z. Zlatev, *Computational Methods for General Sparse Matrices*, Kluwer Acad. Pub., 1991.