SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 423–442, March 1991

ALGORITHMS FOR SPECIAL TRIDIAGONAL SYSTEMS*

RONALD F. BOISVERT†

Abstract. Algorithms for the solution of symmetric diagonally dominant tridiagonal systems of linear equations with constant diagonals are considered. Such systems occur, for example, when solving certain constant-coefficient elliptic partial differential equations by the Fourier method. In particular, the specialized LU factorization method of Malcolm and Palmer (SpLU), the cyclic reduction method of Hockney (CR), and the reversed triangular factorization method of Evans (RTF) are considered. An interesting property of the first two algorithms is that they may be terminated early for highly diagonally dominant systems. A new implementation of RTF that also has this property is presented, significantly reducing its operation count in many cases. The slightly perturbed systems that arise from problems with Neumann or periodic boundary conditions are also considered, with extensions given for SpLU to the periodic case and RTF to the Neumann case. Floating-point operation counts are given for each method, and the results of experiments on a single scalar processor are reported.

Key words. diagonally dominant matrices, Fourier method, LU factorization, Neumann boundary conditions, periodic boundary conditions, Poisson equation, reversed triangular factorization, symmetric matrices, Toeplitz matrices, tridiagonal matrices

AMS(MOS) subject classifications. 65F05, 65N20

1. Introduction.

Consider the $n \times n$ system of linear equations

$$T\mathbf{x} = \mathbf{g}, \tag{1}$$

where

$$T = \begin{bmatrix} a/\mu & b & & & & \rho b \\ b & a & b & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & b & a & b \\ \rho b & & & & b & a/\nu \end{bmatrix},$$

$b \neq 0$, $|a/b| > 2$, ρ is either zero or one, μ and ν are either one or two, and $\rho = 1$ implies $\mu = \nu = 1$. The coefficient matrix T has a very special form: it is tridiagonal (for $\rho = 0$), symmetric, diagonally dominant, and a Toeplitz matrix (for $\mu = \nu = 1$, i.e., has constant diagonals). The need to solve such systems arises, for example, when using finite difference methods to solve linear constant-coefficient boundary-value problems in various contexts [9], [13], [17]. In these cases, the constants μ, ν, and ρ are determined from the boundary conditions along opposite sides of the domain; see Table 1.

Our interest in these methods arises from the Fourier–tridiagonal method for the modified Helmholtz equation [2]. In the case of an $m \times n$ grid, for example, after preprocessing with fast Fourier transforms (FFTs), one obtains a set of m distinct systems of the form (1). One promising method of solving such systems on vector

* Received by the editors June 27, 1988; accepted for publication (in revised form) February 5, 1990.

† Center for Computing and Applied Mathematics, National Institute of Standards and Technology, Gaithersburg, Maryland 20899.

TABLE 1
Relationship of μ, ν, and ρ to boundary conditions.

Boundary Conditions	μ	ν	ρ
Dirichlet–Dirichlet	1	1	0
Dirichlet–Neumann	1	2	0
Neumann–Dirichlet	2	1	0
Neumann–Neumann	2	2	0
Periodic	1	1	1

TABLE 2
Floating-point operation counts for Dirichlet problems.

Method	Operations	Reference
Malcolm–Palmer	$5n + 2k$	[15]
Cyclic reduction	$7n$	[19]
Special LU factorization	$7n$	[19]
Gaussian elimination	$8n$	[3]
Reversed triangular factorization	$9n$	[4]
Toeplitz factorization	$9n$	[8]

processors is to apply a serial algorithm to the m systems in parallel. This leads naturally to the question: what is the best serial algorithm? In what follows we will be concerned with the performance of algorithms on a single scalar processor without further reference to vector or multiprocessor considerations.

A number of methods have been proposed for solving such systems in the Dirichlet case (most admit variants for other boundary conditions): Toeplitz factorization [1], [7], [8], [16]; cyclic reduction [12], [17], [19]; reversed triangular factorization [4], [5], [6]; and specialized variants of Gaussian elimination [15], [19]. Each of these requires $O(n)$ work. Such systems may also be solved using Fourier transforms, but this leads to an $O(n \log n)$ algorithm. Table 2 gives operation counts which have been cited for these methods.

In Toeplitz factorization methods one seeks an LU factorization of T in which L is a lower bidiagonal Toeplitz matrix and U is an upper bidiagonal Toeplitz matrix. This factorization does not exist in general, but suitable L and U may be found such that LU differs from T in only the upper left corner element. The Sherman–Morrison formula [10] may then be applied to adjust the solution obtained from these LU factors. Factorizations of the form $T = QQ^T$, where Q is an $n \times (n+1)$ bidiagonal Toeplitz matrix, also exist for suitably scaled T. The reversed triangular factorization technique is based upon the application of such factorizations. Both of these methods have the advantage that they require only $O(1)$ divisions and no additional vector storage.

The basis for cyclic reduction is the observation of Golub and Hockney [12] that, when $n + 1$ is a power of 2, one can eliminate all the odd-numbered unknowns in the even-numbered equations in a suitably scaled version of (1) by applying simple row operations. The resulting system for the even-numbered unknowns is of the same form as the original, and hence the process may be applied recursively until there are only a few unknowns left, in which case Gaussian elimination may be trivially applied. The remaining unknowns are then determined by a back-substitution process.

Gaussian elimination without pivoting for a symmetric tridiagonal matrix (called "Gaussian elimination" in Table 2) can be improved upon by noting that a suitably scaled T has a special LU factorization, where U has ones on the off-diagonal, L has ones on the diagonal, and the other elements of L and U are simply related. The method of Malcolm and Palmer is based upon this factorization.

It is interesting to note that if $|a/b| \gg 2$, i.e., the diagonal dominance of T is very strong, additional economies may be introduced in these methods when they are implemented in finite precision arithmetic. The method of Malcolm and Palmer relies upon the fact that the diagonal elements of U form a sequence which quickly (in k steps) becomes constant. This allows the factorization to be terminated early. Another form of early convergence has been observed for cyclic reduction [11], [12]. Here the diagonals of the successive tridiagonal systems are roughly squared in magnitude at each stage; when the ratio of off-diagonal to diagonal elements is less than the machine precision, then the system can be considered diagonal and back substitution can be started immediately. In this paper we show how similar economies may be introduced into the reversed triangular factorization method.

We begin by stating some pertinent facts about cyclic reduction. We then describe the method of Malcolm and Palmer in some detail, and give an extension to the periodic case. We then present new variants of the reversed triangular factorization method. Finally, we compare each of these techniques and present the results of computational experiments.

2. Cyclic reduction. In this section we summarize some of the properties of the cyclic reduction method as applied to the system (1) in order to facilitate comparison with the methods described in the latter sections.

We first consider the Dirichlet case. Scaling each equation in (1) by $1/b$ we arrive at a tridiagonal system with $\alpha \equiv a/b$ ($\alpha \geq 2$) on the diagonal and ones on the off-diagonals. For simplicity we first assume that $n = m - 1$, where $m = 2^k$. The first step in cyclic reduction is to multiply equations $2, 4, \cdots, m-2$ by $-\alpha$ and add the two adjacent equations to it. This decouples the even-numbered equations from the odd ones, yielding a reduced system of order $m/2 - 1$. Once the reduced system is solved the odd-numbered equations may be used to determine the odd-numbered unknowns by back substitution. The reduced equations have the same form as the original ones, except with diagonal elements $2 - \alpha^2$. Thus, the reduction process may be applied recursively until a system of order 1 remains. All unknowns may then be determined by applying back substitution recursively.

The algorithm may be summarized as follows.

$$
\begin{aligned}
&\alpha_{j+1} \leftarrow 2 - \alpha_j^2, \quad j = 1(1)k-1 \\
&\qquad g_i \leftarrow g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 2^j(2^j)m - 2^j, \quad j = 1(1)k-1 \\
&g_{m/2} \leftarrow g_{m/2}/\alpha_k \\
&\qquad g_i \leftarrow (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 2^{j-1}(2^j)m - 2^{j-1}, \quad j = k-1(-1)1.
\end{aligned}
$$

(Here we have taken $\alpha_1 = \alpha$ and $g_0 = g_m = 0$.) It is straightforward to show that this algorithm requires $7n$ floating-point operations (including the n operations required to initially scale the system). The cases of (1) with $\mu = 2$ and/or $\nu = 2$ may be handled similarly so that the operation counts remain the same in these cases.

We need not have that m be a power of 2 in order to apply cyclic reduction. A simple modification is necessary to generalize this algorithm to any n, requiring only that we allow the diagonal element in the last equation to differ from the others at each step of the reduction. The asymptotic operation count of $7n$ is unchanged. If we also allow the diagonal element in the first equation to differ, we may generalize further to the case of equation (1) with $\rho = 0$. We state this algorithm in detail below. Here, n_j is the number of equations left after the jth reduction step, $\alpha_j^{(1)}$ is the diagonal element in the first equation before the jth reduction step, and $\alpha_j^{(n)}$ is the diagonal element in the last equation before the jth reduction step.

ALGORITHM 2.1. **Cyclic Reduction.**

0. *Preliminaries*

$$
\begin{array}{lll}
k & \leftarrow & \lfloor \log_2 n \rfloor \\
\alpha_1 & \leftarrow & \alpha \\
\alpha_{j+1} & \leftarrow & 2 - \alpha_j^2, \quad j = 1, 2, \cdots, k-1 \\
\alpha_1^{(1)} & \leftarrow & \alpha/\mu \\
\alpha_1^{(n)} & \leftarrow & \alpha/\nu \\
n_0 & \leftarrow & n \\
n_j & \leftarrow & \lfloor n_{j-1} \rfloor, \quad j = 1, 2, \cdots, k-1
\end{array}
$$

1. *Rescale*

$$g_i \leftarrow g_i/b, \quad i = 1, 2, \cdots, n$$

2. *Reduction*

For $j = 1, \cdots, k-1$ *do:*

$$
\begin{array}{lll}
g_{2^j} & \leftarrow & \alpha_j (g_{2^j - 2^{j-1}}/\alpha_j^{(1)} - g_{2^j}) + g_{2^j + 2^{j-1}} \\
\alpha_{j+1}^{(1)} & \leftarrow & 1 + \alpha_j/\alpha_j^{(1)} - \alpha_j^2 \\
g_i & \leftarrow & g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 2\cdot 2^j, 3\cdot 2^j, \cdots, (n_j - 1)2^j \\
i & \leftarrow & n_j 2^j \\
g_i & \leftarrow & \begin{cases} g_{i-2^{j-1}} - \alpha_j g_i + \alpha_j g_{i+2^{j-1}}/\alpha_j^{(n)}, & \textit{if } i + 2^{j-1} \le n \\ g_{i-2^{j-1}} - \alpha_j g_i, & \textit{otherwise} \end{cases} \\
\alpha_{j+1}^{(n)} & \leftarrow & \begin{cases} 1 - \alpha_j^2 + \alpha_j/\alpha_j^{(n)}, & \textit{if } i + 2^{j-1} \le n \\ 1 - \alpha_j \alpha_j^{(n)}, & \textit{otherwise} \end{cases}
\end{array}
$$

3. *Explicit solve*

If $n_{k-1} = 3$ *then:* (3×3 *case*)

$$
\begin{array}{lll}
g_{2^k} & \leftarrow & (g_{2^k} - g_{2^k - 2^{k-1}}/\alpha_k^{(1)} - g_{2^k + 2^{k-1}}/\alpha_k^{(n)})/(\alpha_k - 1/\alpha_k^{(1)} \\
& & -1/\alpha_k^{(n)}) \\
g_{2^k - 2^{k-1}} & \leftarrow & (g_{2^k - 2^{k-1}} - g_{2^k})/\alpha_k^{(1)} \\
g_{2^k + 2^{k-1}} & \leftarrow & (g_{2^k + 2^{k-1}} - g_{2^k})/\alpha_k^{(n)}
\end{array}
$$

Else: (2×2 *case*)

$$
\begin{array}{lll}
g_{2^k} & \leftarrow & (g_{2^k} - g_{2^k - 2^{k-1}}/\alpha_k^{(1)})/(\alpha_k^{(n)} - 1/\alpha_k^{(1)}) \\
g_{2^k - 2^{k-1}} & \leftarrow & (g_{2^k - 2^{k-1}} - g_{2^k})/\alpha_k^{(1)}
\end{array}
$$

Endif

4. *Back substitution*

For $j = k-1, k-2, \cdots, 1$ *do:*

$$
\begin{array}{lll}
g_{2^{j-1}} & \leftarrow & (g_{2^{j-1}} - g_{2^j})/\alpha_j^{(1)} \\
l & \leftarrow & 2^{j-1} + (\lfloor (n_j + 1)/2 \rfloor - 2)2^j \\
g_i & \leftarrow & (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 2^{j-1} + 2^j, 2^{j-1} + 2\cdot 2^j, \cdots, l \\
i & \leftarrow & \max(2^{j-1}, l) + 2^j \\
g_i & \leftarrow & \begin{cases} (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, & \textit{if } i + 2^{j-1} \le n \\ (g_i - g_{i-2^{j-1}})/\alpha_j^{(n)}, & \textit{otherwise.} \end{cases}
\end{array}
$$

For $|\alpha| \gg 2$ we see that $|\alpha_j| \approx \alpha^{2^{j-1}}$, and hence the algorithm, as stated, presents

TABLE 3
Effect of α on cyclic reduction operation count for $\epsilon = 2^{1-t}$.

	$t = 47$		$t = 23$	
α	k_c	Ops	k_c	Ops
2.001	10	$7.0n$	9	$7.0n$
2.010	9	$7.0n$	8	$7.0n$
2.100	7	$7.0n$	6	$6.9n$
2.500	6	$6.9n$	5	$6.8n$
4.000	5	$6.8n$	4	$6.7n$
10.000	4	$6.7n$	3	$6.4n$
50.000	4	$6.7n$	2	$5.8n$

the danger of floating-point overflow. However, as pointed out by Hockney [12], this can be turned to our advantage. If $|\alpha_j| > 1/\epsilon$, where ϵ is the machine epsilon, then the reduced system is diagonal to machine precision. At this point, the reduced system may be solved with $m/2^j$ multiplications and back substitution may be begun immediately. If the algorithm is stopped at step $j = k_c$, then the operation count is reduced to $(7 - 5/2^{k_c})n$. Table 3 shows the values of k_c and the resulting operation count for various values of α and ϵ. From the table it is clear that significant savings are obtained only when α is so large that the reduction may stop after the second step. The reason for this is that, since the work halved at each step of the reduction, most of the work is done in the first few steps.

We next consider the periodic case. Scaling (1) as before we obtain the same tridiagonal system modified only by the addition of ones in the $(1,n)$ and $(n,1)$ positions. Here, we assume that $n = m - 1$, where $m = 2^k + 1$. The basic reduction process is exactly as before, except that we reduce the odd equations first and use the last equation to help reduce the first one. After $k-1$ reduction steps we are left with a system of order 2. After this system is solved the remaining unknowns are again determined by back substitution.

The algorithm may be summarized as follows.

$$\begin{aligned}
\alpha_{j+1} &\leftarrow 2 - \alpha_j^2, \quad j = 1(1)k-1\\
g_i &\leftarrow g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 1(2^j)m - 2^j, \quad j = 1(1)k-1\\
g_{1+2^{k-1}} &\leftarrow (\alpha_k g_{1+2^{k-1}} - 2g_1)/(\alpha_k^2 - 4)\\
g_1 &\leftarrow (g_1 - 2g_{1+2^{k-1}})/\alpha_k\\
g_i &\leftarrow (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 1 + 2^{j-1}(2^j)m - 2^{j-1},\\
&\qquad j = k-1(-1)1.
\end{aligned}$$

(Here we have taken $\alpha_1 = \alpha$, $g_m = g_1$, and $g_{1-2^{j-1}} = g_{m-2^{j-1}}$.) Once again the algorithm is seen to require $7n$ floating-point operations. The diagonals of the successive systems are the same as in the Dirichlet case, and hence the same early stopping criterion may be applied, yielding an effective operation count of $(7 - 5/2^{k_c})n$.

The modification to this algorithm in the case where n is not a power of 2 does not appear to be as straightforward as the $\rho = 0$ case. In [18] Sweet presents a generalization analogous to that of the fast Fourier transform; unfortunately his generalization can show significant performance degradation when m has factors other than 2. We will not consider this case further.

In the next sections we will show how the diagonal dominance of (1) also leads to a reduction of operations in the special LU factorization and the reversed triangular factorization methods.

3. Special LU factorization. In [15], Malcolm and Palmer described how to reduce the number of operations of Gaussian elimination for symmetric diagonally dominant tridiagonal systems with constant diagonals. Their results also indicate how Neumann-type boundary conditions can be handled. In this section we first outline their results, and then extend them to the case of periodic boundary conditions.

3.1. Dirichlet and Neumann cases. We first consider (1) in the case $\rho = 0$. After scaling by $1/b$ we arrive at the equivalent problem

$$T\mathbf{x} = \mathbf{g}, \tag{2}$$

where

$$T = \begin{bmatrix} \alpha/\mu & 1 & & & & \\ 1 & \alpha & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & 1 & \alpha & 1 \\ & & & & 1 & \alpha/\nu \end{bmatrix},$$

$\alpha = a/b$, with $|\alpha| > 2$. We may then write $T = LDL^T$, where

$$L = \begin{bmatrix} 1 & & & & & \\ l_1 & 1 & & & & \\ & \cdot & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & l_{n-2} & 1 & \\ & & & & l_{n-1} & 1 \end{bmatrix}, \quad D = \begin{bmatrix} u_1 & & & & & \\ & u_2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & u_{n-1} & \\ & & & & & u_n \end{bmatrix}. \tag{3}$$

The u_i and l_i are defined by the recursion

$$\begin{aligned} u_1 &= \alpha/\mu \\ u_i &= \alpha - 1/u_{i-1}, \qquad i = 2, \cdots, n-1 \\ u_n &= \alpha/\nu - 1/u_{n-1} \\ l_i &= 1, \qquad i = 1, \cdots, n. \end{aligned}$$

After determining L and D the solution vector $\mathbf{x}$ may be found by solving the triangular systems $L\mathbf{y} = \mathbf{g}$ and $L^T\mathbf{x} = D^{-1}\mathbf{y}$ in turn. Note that the decomposition is uniquely represented by the n values l_i, $i = 1, 2, \cdots, n$.

In [15] it was observed that the sequence u_i converges to $u = [\alpha + \mathrm{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]/2$ (provided $\mu \neq \alpha^2$ and $\mu \neq \alpha/v$, where $v = [\alpha - \mathrm{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]/2$). Furthermore, it was shown that on a computer with precision[1] ϵ, the sequence u_i becomes constant for $i \geq k_u$, where

$$k_u = \left\lceil \frac{-\ln \epsilon}{2 \ln |u|} \right\rceil .$$

This leads to the following algorithm for the solution to (1) with $\rho = 0$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

[1] A computer with t-digit, base β arithmetic has precision β^{1-t}.

TABLE 4
Values of the convergence index k_u for $\epsilon = 2^{1-t}$.

α	k_u	
	$t = 47$	$t = 24$
2.001	505	253
2.010	160	80
2.100	51	26
2.500	24	12
4.000	13	7
10.000	7	4
50.000	5	3

ALGORITHM 3.1. **Special *LU* Factorization.**

0. *Preliminaries*
   $l \leftarrow 2/[\alpha + \operatorname{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]$
   $k \leftarrow \min(\lfloor \ln \epsilon / 2 \ln |l| \rfloor, n)$
1. *Rescale*
   $g_i \leftarrow g_i/b, \quad i = 1, 2, \cdots, n$
2. *Factorization*
   $l_1 \leftarrow \mu/\alpha$
   $l_i \leftarrow 1/[\alpha - l_{i-1}], \quad i = 2, 3, \cdots, \min(k, n-1)$
   $l_i \leftarrow l, \quad i = k+1, k+2, \cdots, n-1$
   $l_n \leftarrow 1/[\alpha/\nu - l_{n-1}]$
3. *Forward substitution*
   $g_i \leftarrow g_i - l_{i-1} g_{i-1}, \quad i = 2, 3, \cdots, n$
4. *Backward substitution*
   $g_n \leftarrow l_n g_n$
   $g_i \leftarrow l_i(g_i - g_{i+1}), \quad i = n-1, n-2, \cdots, 1$

Algorithm 3.1 requires $5n + 2\tilde{k}_u$ floating-point operations, where $\tilde{k}_u = \max(k_u, n)$, and a working storage of $\tilde{k}_u$ words. The ratio k_u/n becomes small quickly as $|\alpha|$ moves away from 2 (see Table 4), and hence the savings in using this algorithm can be significant in many cases. When $|\alpha|$ is near 2, or the early convergence of l_i is ignored, then $7n$ operations are required. In this case the algorithm is equivalent to Gaussian elimination with a straightforward modification to account for constant diagonals.

This algorithm also has certain advantages when the system (1) must be solved for many different right-hand side vectors $\mathbf{g}$. In this case, step 2 (the factorization), may be eliminated in all but the first solve. Each additional solve requires $5n$ operations. This amounts to a significant savings for weakly diagonally dominant systems. In the strongly diagonally dominant case the factorization step is nearly eliminated anyway, and the savings in repeat solves is only $2\tilde{k}_u$ operations.

The degenerate cases with $|\alpha| = 2$ may be handled as follows:

- $\mu = \nu = 1$: Set $k = n$ and apply steps 1–4 of the algorithm described above.
- $\mu = 2,\ \nu = 1$: Here $l_i = \operatorname{sgn}(\alpha)$ for $i = 1, 2, \cdots, n-1$; thus we may set $k = 1$ and apply steps 1–4 of the algorithm above. (Multiplications may be saved by separately considering the cases $l = 1$ and $l = -1$.)
- $\mu = 1,\ \nu = 2$: We may reduce this to the case $\mu = 2,\ \nu = 1$ by reordering the equations and unknowns (i.e., by processing the g_i in reverse order).

- $\mu = \nu = 2$: The matrix is singular.

3.2. Periodic case. We next consider (1) in the case $\mu = \nu = \rho = 1$. After scaling by $1/b$ this yields the problem

$$\left[\begin{array}{c|c} T & \mathbf{e} \\ \hline \mathbf{e}^T & \alpha \end{array}\right] \left[\begin{array}{c} \mathbf{x} \\ \hline x_n \end{array}\right] = \left[\begin{array}{c} \mathbf{g} \\ \hline g_n \end{array}\right], \tag{4}$$

where T is the matrix of (2) with $\mu = \nu = 1$, and $\mathbf{e}$ is the vector

$$\mathbf{e}^T = \left[\begin{array}{cccccc} 1 & 0 & \cdot & \cdot & \cdot & 0 \quad 1 \end{array}\right].$$

T is $(n-1) \times (n-1)$, and $\mathbf{e}$, $\mathbf{x}$, and $\mathbf{g}$ are of length $n-1$. Let L and D be matrices of the form (3) with $\mu = \nu = 1$. We then have the decomposition

$$\left[\begin{array}{c|c} T & \mathbf{e} \\ \hline \mathbf{e}^T & \alpha \end{array}\right] = \left[\begin{array}{c|c} L & 0 \\ \hline \mathbf{w}^T & 1 \end{array}\right] \left[\begin{array}{c|c} D & 0 \\ \hline 0 & u_n \end{array}\right] \left[\begin{array}{c|c} L^T & \mathbf{w} \\ \hline 0 & 1 \end{array}\right].$$

The relations which determine $\mathbf{w}$ and u_n are then

$$\begin{aligned} LD\mathbf{w} &= \mathbf{e}, \\ u_n + \mathbf{w}^T D\mathbf{w} &= \alpha, \end{aligned}$$

from which we get

$$\begin{array}{lcl} w_1 & = & l_1 \\ w_i & = & -w_{i-1}l_i, \quad i = 2, 3, \cdots, n-2 \\ w_{n-1} & = & (1 - w_{n-2})l_{n-1} \end{array}$$

and

$$u_n = \alpha - \sum_{j=1}^{n-1} u_j w_j^2 = \alpha - \left[w_1 - \sum_{j=2}^{n-2} w_{j-1}w_j + (1 - w_{n-2})w_{n-1} \right].$$

After determining L, D, and $\mathbf{w}$, the solution to (4) may be found by solving

$$L\mathbf{y} = \mathbf{g},$$

then setting

$$x_n = (g_n - \mathbf{w}^T\mathbf{y})/u_n,$$

and finally solving

$$L^T\mathbf{x} = D^{-1}\mathbf{y} - x_n\mathbf{w}.$$

Note that the decomposition is uniquely determined by the $2n-1$ values $l_n \equiv 1/u_n$ and $l_i,\ w_i,\ \ i = 1,2,\cdots,n-1$.

THEOREM 3.2. *The sequence* $w_1 = l_1,\ w_i = -w_{i-1}l_i,\ i > 1$ *converges to zero for* $|\alpha| > 2$.

Proof. In [15] it is shown that, if $|\alpha| > 2$, the sequence $|l_j|$ is strictly increasing and converges to $|l| < 1$. Thus,

$$|w_i| = \left|(-1)^{i+1}\prod_{j=1}^{i} l_j\right| = \prod_{j=1}^{i}|l_j| < |l|^i,$$

from which convergence to zero follows. □

The proof of the theorem gives a method of determining an upper bound for the value of i after which w_i is zero to machine precision. This occurs when $|l|^i = \epsilon$. Taking logarithms yields

$$i = k_w \equiv \left\lceil \frac{\ln\epsilon}{\ln|l|}\right\rceil = 2k_u,$$

where k_u is the convergence index of the sequence u_i. This, along with the convergence results for the sequence u_i, leads to the following algorithm for the solution to (1) with $\mu = \nu = \rho = 1$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

ALGORITHM 3.3. **Special *LU* Factorization (Periodic).**

0. *Preliminaries*
   $l \leftarrow 2/[\alpha + \mathrm{sgn}(\alpha)(\alpha^2-4)^{1/2}]$
   $k_u \leftarrow \min(\lfloor \ln\epsilon/2\ln|l|\rfloor, n-1)$
   $k_w \leftarrow \min(2k_u, n-2)$
1. *Rescale*
   $g_i \leftarrow g_i/b, \quad i = 1,2,\cdots,n$
2. *Factorization*
   $l_1 \leftarrow 1/\alpha$
   $l_i \leftarrow 1/[\alpha - l_{i-1}], \quad i = 2,3,\cdots,k_u$
   $l_i \leftarrow l, \quad i = k_u+1, k_u+2,\cdots,n-1$
   $w_1 \leftarrow l_1$
   $w_i \leftarrow -w_{i-1}l_i, \quad i = 2,3\cdots,k_w$
   $w_i \leftarrow 0, \quad i = k_w+1, k_w+2,\cdots,n-2$
   $w_{n-1} \leftarrow (1-w_{n-2})l_{n-1}$
   $l_n \leftarrow 1/[\alpha - w_1 + \sum_{j=2}^{k_w}(w_jw_{j-1}) - (1-w_{n-2})w_{n-1}]$
3. *Forward substitution*
   $g_i \leftarrow g_i - l_{i-1}g_{i-1}, \quad i = 2,3,\cdots,n-1$
   $g_n \leftarrow l_n(g_n - \sum_{j=1}^{k_w} w_jg_j - w_{n-1}g_{n-1})$
4. *Backward substitution*
   $g_{n-1} \leftarrow l_{n-1}(g_{n-1} - (1-w_{n-2})g_n)$
   $g_i \leftarrow l_i(g_i - g_{i+1}) - w_ig_n, \quad i = n-2, n-3,\cdots,1$

This algorithm requires $7n + 6\tilde{k}_w + 2\tilde{k}_u$ floating-point operations, where $\tilde{k}_w = \max(k_w, n)$. A working storage of length $2n$ is required (actually, only $\tilde{k}_u + \tilde{k}_w$ words are necessary). When $k_u \geq n$ the algorithm requires $15n$ operations.

In the case of multiple right-hand sides we may again eliminate step 2 in all but the first solve. Thus, subsequent systems may be solved with only $7n+2\tilde{k}_w$ operations. For weakly diagonally dominant systems this is equivalent to $9n$ operations, a significant savings over the $15n$ required for the first solve. In the strongly diagonally dominant case there is only a savings of $12\tilde{k}_w$ operations, which is insignificant except for very small n.

When $|\alpha| = 2$ the system is singular unless $\alpha = 2$ and n is odd. In the latter case we may set $k_u = k_w = n - 1$ and use steps 1–4 above.

4. Reversed triangular factorization. Each of the algorithms of §3 requires working storage of length $O(k_u)$. In contrast, a method proposed by Evans [4], [5], [6] requires no auxiliary vector storage and only $O(1)$ divisions. In this section we describe this method, extending it to the cases where $\mu = 2$ and/or $\nu = 2$, showing how the resulting algorithms may be implemented with the same types of economies as in the Malcolm–Palmer method.

4.1. Dirichlet and Neumann cases. We first consider (1) in the case $\rho = 0$. Choosing β such that $b/a = -\beta/(1+\beta^2)$ with $0 < |\beta| < 1$ and scaling (1) by $(1+\beta^2)/a$ yields the equivalent problem

$$Tx = g, \tag{5}$$

where

$$T = \begin{bmatrix} (1+\beta^2)/\mu & -\beta & & & & \\ -\beta & (1+\beta^2) & -\beta & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & -\beta & (1+\beta^2) & -\beta \\ & & & & -\beta & (1+\beta^2)/\nu \end{bmatrix}.$$

T has the factorization $T = QQ^T$, where Q is the $n \times (n+1)$ matrix

$$Q = \begin{bmatrix} \sigma & -\beta & & & & & \\ & 1 & -\beta & & & & \\ & & 1 & -\beta & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & \cdot & \cdot \\ & & & & & 1 & -\beta \\ & & & & & & 1 & -\gamma \end{bmatrix},$$

$\sigma^2 = (1+\beta^2)/\mu - \beta^2$, and $\gamma^2 = (1+\beta^2)/\nu - 1$. Thus, a solution to (5) may be determined by solving $Q\mathbf{y} = \mathbf{g}$ and $Q^T\mathbf{x} = \mathbf{y}$ in turn. Note that the factorization is determined simply by the scalars β, σ, and γ, and hence no auxiliary vector storage is required. (The vectors $\mathbf{y}$ and $\mathbf{x}$ may occupy the same storage.)

We first show how the element y_{n+1} may be computed. The remaining y_i are then determined by backward substitution and the x_i follow from a forward substitution. From $Q\mathbf{y} = \mathbf{g}$ we have

$$\begin{aligned} \sigma y_1 &- \beta^{n-1}\gamma y_{n+1} &= \textstyle\sum_{j=1}^{n} \beta^{j-1} g_j, \\ y_{n-k} &- \beta^k \gamma y_{n+1} &= \textstyle\sum_{j=n-k}^{n} \beta^{j-(n-k)} g_j, \quad 0 \le k \le n-2, \end{aligned} \tag{6}$$

while from $Q^T\mathbf{x} = \mathbf{y}$ we have

$$x_n = \sum_{k=0}^{n-2} \beta^k y_{n-k} + \beta^{n-1} y_1/\sigma, \tag{7}$$

$$-\gamma x_n = y_{n+1}. \tag{8}$$

Putting the expressions for y_{n-k} from (6) into (7), using (8), and simplifying gives

$$\left[1 + \frac{\gamma^2 \beta^{2(n-1)}}{\sigma^2} + \gamma^2 \sum_{k=0}^{n-2} \beta^{2k}\right] x_n = \frac{1}{\sigma^2} \sum_{j=0}^{n-1} \beta^{2(n-1-j)} \beta^j g_{n-j} + \sum_{j=0}^{n-2} \left(\sum_{k=0}^{n-2-j} \beta^{2k} \right) \beta^j g_{n-j}. \tag{9}$$

Using the summation formula for β^{2k} for $|\beta| < 1$ and simplifying further yields

$$\delta x_n = \sum_{j=0}^{n-2} \beta^j g_{n-j} + (c+1)\beta^{n-1} g_1 + c \sum_{j=n}^{2n-2} \beta^j g_{j-n+2}, \tag{10}$$

where $\delta = 1 - \beta^2 + \gamma^2(1 + c\beta^{2n-2})$, and $c = (1 - \beta^2 - \sigma^2)/\sigma^2$. This expression for x_n is a polynomial of degree $2n-2$ in β which may be stably and efficiently evaluated using Horner's rule.

Since $|\beta| < 1$ we have that $\beta^j \to 0$. If $|\beta|$ is sufficiently small, then many of the high-order terms in (10) will be zero to machine precision. This can be used to reduce the operation count for the computation of x_n. To determine when we can discard terms, first rewrite (10) as

$$x_n = \sum_{j=0}^{2n-2} c_j \beta^j,$$

where

$$\delta c_j = \begin{cases} g_{n-j}, & 0 \le j \le n-2 \\ (c+1) g_1, & j = n-1 \\ c g_{j-n+2}, & n \le j \le 2n-2. \end{cases}$$

We will discard terms that are small relative to the size of the result vector $\mathbf{x}$, i.e., for

$$\frac{|c_j \beta^j|}{||\mathbf{x}||} < \epsilon, \tag{11}$$

where $||\cdot||$ denotes the infinity norm. The following will be used to bound this quantity:

$$|c_j| \le \frac{2||\mathbf{g}||}{|\delta|},$$
$$||T|| = (1 + |\beta|)^2,$$
$$|\delta| \ge \frac{(1-\beta^2)^2}{2}.$$

TABLE 5
Values of the convergence index k_β for $\epsilon = 2^{1-t}$.

	k_β	
α	$t = 47$	$t = 24$
2.001	636	384
2.010	190	111
2.100	57	32
2.500	25	14
4.000	13	7
10.000	8	4
50.000	5	3

The first inequality follows from $|c| \leq 1$. Each requires that μ and ν be in the set $\{1, 2\}$. Using these in turn we have

$$\frac{|c_j \beta^j|}{||\mathbf{x}||} \leq \frac{2||\mathbf{g}|||\beta|^j}{|\delta|||\mathbf{x}||} \leq \frac{2||T|||\beta|^j}{|\delta|} = \frac{2(1+|\beta|)^2|\beta|^j}{|\delta|} \leq \frac{4|\beta|^j}{(1-|\beta|)^2}.$$

From this we see that a sufficient condition for (11) to hold is

$$|\beta|^j < (1-|\beta|)^2\epsilon/4.$$

Taking logs we find that this occurs when

$$j > 2k_\beta \equiv \log_{|\beta|}[(1-|\beta|)^2\epsilon/4].$$

Table 5 displays sample values of k_β; significant savings in computation time are possible when $k_\beta << n-1$.

This leads to the following algorithm for the solution to (5) with $\rho = 0$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

ALGORITHM 4.1. **Reversed Triangular Factorization.**

0. *Preliminaries*
   $\beta \leftarrow -\alpha(1 + [1 - 4/\alpha^2]^{1/2})/2$
   $\beta \leftarrow 1/\beta, \quad \text{if } |\beta| > 1$
   $\sigma \leftarrow [(1+\beta^2)/\mu - \beta^2]^{1/2}$
   $\gamma^2 \leftarrow (1+\beta^2)/\nu - 1$
   $c \leftarrow (1 - \beta^2 - \sigma^2)/\sigma^2$
   $\delta \leftarrow 1 - \beta^2 + \gamma^2(1 + c\beta^{2n-2})$
   $k \leftarrow \min(\lceil \log_{|\beta|}(1-|\beta|)^2\epsilon/4 \rceil, 2n-2)$
1. *Rescale*
   $s \leftarrow (1+\beta^2)/a$
   $g_i \leftarrow sg_i, \quad i = 1, 2, \cdots, n$
2. *Factorization*
   $x_n \leftarrow 0$
   $m \leftarrow k - n + 2$
   $x_n \leftarrow \beta x_n + g_i, \quad i = m, m-1, \cdots, 2$
   $x_n \leftarrow cx_n$
   $x_n \leftarrow \beta x_n + (c+1)g_1, \quad \text{if } k \geq n-1$
   $m \leftarrow \max(n-k, 2)$
   $x_n \leftarrow \beta x_n + g_i, \quad i = m, m+1, \cdots, n$
   $x_n \leftarrow x_n/\delta$

3. *Back substitution*
$g_n \leftarrow g_n - \gamma^2 x_n$
$g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1$
$g_1 \leftarrow g_1/\sigma^2$
4. *Forward substitution*
$g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2, 3, \cdots, n-1$
$g_n \leftarrow x_n$

This algorithm uses $5n + 4\tilde{k}_\beta$ floating-point operations, where $\tilde{k}_\beta = \max(k_\beta, n)$. If the early convergence of the series for x_n is not accounted for, or $k_\beta > n-1$, then this reduces to $9n$ floating-point operations.

The degenerate cases for $|\beta| = 1$ may be handled as follows:

- $\mu = \nu = 1$: In this case (9) reduces to

$$(n+1)x_n = \sum_{i=0}^{n-1} (n-i)\beta^i g_{n-i},$$

from which x_n may be computed using Horner's rule. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.) Steps 1, 3, and 4 of the algorithm may then be applied with $\sigma^2 = \gamma^2 = 1$.

- $\mu = 2$, $\nu = 1$: Here $\sigma = 0$, $\gamma = 1$, and the first column of Q drops out, leaving only forward and backward substitution phases. The algorithm is as follows. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

$$\begin{aligned}
&s \leftarrow 2/a \\
&g_i \leftarrow s g_i, \quad i = 1, 2, \cdots, n \\
&g_1 \leftarrow -\beta g_1 \\
&g_i \leftarrow -\beta(g_i - g_{i-1}), \quad i = 2, 3, \cdots, n-1 \\
&g_n \leftarrow g_n - g_{n-1} \\
&g_i \leftarrow -\beta(g_i - g_{i+1}), \quad i = n-1, n-2, \cdots, 1
\end{aligned}$$

- $\mu = 1$, $\nu = 2$: Here $\sigma = 1$, $\gamma = 0$, and the last column of Q drops out, leaving only backward and forward substitution phases. The algorithm is as follows. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

$$\begin{aligned}
&s \leftarrow 2/a \\
&g_i \leftarrow s g_i, \quad i = 1, 2, \cdots, n \\
&g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1 \\
&g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2, 3, \cdots, n
\end{aligned}$$

- $\mu = \nu = 2$: The matrix is singular.

4.2. Periodic case. We next consider (1) in the case $\mu = \nu = \rho = 1$. Scaling the system as before we then have the equivalent problem

$$Tx = \mathbf{g}, \tag{12}$$

where

$$T = \begin{bmatrix} (1+\beta^2) & -\beta & & & & -\beta \\ -\beta & (1+\beta^2) & -\beta & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & -\beta & (1+\beta^2) & -\beta \\ -\beta & & & & -\beta & (1+\beta^2) \end{bmatrix}.$$

T has the factorization $T = QQ^T$, where Q is the $n \times n$ matrix

$$Q = \begin{bmatrix} 1 & & & & & -\beta \\ -\beta & 1 & & & & \\ & -\beta & 1 & & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & -\beta & 1 \end{bmatrix}.$$

Note that the factorization is determined simply by the scalar β and hence no auxiliary vector storage is required. A solution to (12) may be determined by solving $Q\mathbf{y} = \mathbf{g}$ and $Q^T\mathbf{x} = \mathbf{y}$ in turn. The former system may be solved by forward substitution once y_1 is known, while the latter may be solved by backward substitution once x_n is known.

It is straightforward to show that

$$(1-\beta^n)y_1 = g_1 + \sum_{j=1}^{n-1} \beta^j g_{n+1-j}, \tag{13}$$

$$(1-\beta^n)x_n = y_n + \sum_{j=1}^{n-1} \beta^j y_j. \tag{14}$$

When $|\beta| << 1$ many of the high-order terms in β will be zero to machine precision, which can be used to reduce the operation count for the computation of y_1 and x_n as before. Let d_j and e_j be the coefficients of β^j in (13) and (14), respectively. We then have

$$\frac{|d_j\beta^j|}{||\mathbf{y}||} \le \frac{||\mathbf{g}|||\beta|^j}{(1-\beta^n)||\mathbf{y}||} \le \frac{||Q|||\beta|^j}{1-\beta^n},$$

and

$$\frac{|e_j\beta^j|}{||\mathbf{x}||} \le \frac{||\mathbf{y}|||\beta|^j}{(1-\beta^n)||\mathbf{x}||} \le \frac{||Q^T|||\beta|^j}{1-\beta^n}.$$

Using $||Q|| = ||Q^T|| = 1 + |\beta|$ we find that both of these quantities are bounded by

$$\frac{(1+|\beta|)|\beta|^j}{1-\beta^n} \le \frac{(1+|\beta|)|\beta|^j}{1-\beta^2} = \frac{|\beta|^j}{1-|\beta|}.$$

Thus, terms in 13 and (14) may be safely discarded for

$$j > k_p \equiv \log_{|\beta|}[(1-|\beta|)\epsilon].$$

TABLE 6
Values of the convergence index k_p for $\epsilon = 2^{1-t}$.

α	k_p	
	$t = 47$	$t = 24$
2.001	1119	614
2.010	343	184
2.100	106	55
2.500	48	25
4.000	25	13
10.000	14	8
50.000	9	5

Sample values of the convergence index k_p are given in Table 6.

This leads to the following algorithm for the solution to (1) with $\mu = \nu = \rho = 1$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

ALGORITHM 4.2. **Reversed Triangular Factorization (Periodic).**

0. *Preliminaries*
   $\beta \leftarrow -\alpha(1 + [1 - 4/\alpha^2]^{1/2})/2$
   $\beta \leftarrow 1/\beta, \quad \textit{if } |\beta| > 1$
   $\delta \leftarrow 1 - \beta^n$
   $k \leftarrow \min(\lceil \log_{|\beta|}(1 - |\beta|)\epsilon \rceil, n-1)$
1. *Rescale*
   $s \leftarrow (1 + \beta^2)/a$
   $g_i \leftarrow s g_i, \quad i = 1, 2, \cdots, n$
2. *Determine* y_1
   $y_1 \leftarrow g_{n-k+1}$
   $y_1 \leftarrow \beta y_1 + g_i, \quad i = n-k+2, n-k+3, \cdots, n$
   $g_1 \leftarrow (\beta y_1 + g_1)/\delta$
3. *Forward substitution*
   $g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2, 3, \cdots, n$
4. *Determine* x_n
   $x_n \leftarrow g_k$
   $x_n \leftarrow \beta x_n + g_i, \quad i = k-1, k-2, \cdots, 1$
   $g_n \leftarrow (\beta x_n + g_n)/\delta$
5. *Backward substitution*
   $g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1$

This algorithm requires $5n + 4\tilde{k}_p$ floating-point operations, where $\tilde{k}_p = \max(k_p, n)$. If the early convergence of the series for y_1 and x_n are not accounted for, or $k_p > n$, then this reduces to $9n$ operations, which is larger than for the algorithm of §3.1. However, when $k_p << n$, the operation count is considerably better.

When $|\alpha| = 2$ the system is singular unless $\alpha = 2$ and n is odd. In the latter case we may set $k_p = n - 1$ and use steps 1–5 above. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

5. Comparison of methods. In this section we compare the methods described in the previous sections and present the results of some computational experiments.

Table 7 summarizes the floating-point operation counts for the methods of §§2, 3, and 4. Also included are operation counts for Gaussian elimination without pivoting

TABLE 7
Floating-point operation counts.

Method	$\rho = 0$	$\rho = 1$
Gaussian elimination	$8n$	$18n$
Cyclic reduction	$7n-5n/2^{k_c}$	$7n-5n/2^{k_c}$ †
Special LU factorization	$5n+2\tilde{k}_u$	$7n+6\tilde{k}_w + 2\tilde{k}_u$
Reversed triangular factorization	$5n+4\tilde{k}_\beta$	$5n+4\tilde{k}_p$

† For $n = 2^k$ only.

TABLE 8
Floating-point operation counts for repeat solves.

Method	$\rho = 0$	$\rho = 1$
Gaussian elimination	$5n$	$9n$
Special LU factorization	$5n$	$7n + 2\tilde{k}_w$

for a symmetric tridiagonal system (e.g., the algorithm used by LINPACK's SPTSL [3]). In the periodic case the algorithm is modified in a straightforward way to account for the extra nonzeros in the last row and column of the matrix.

We first consider the case $\rho = 0$. For highly diagonally dominant systems ($|\alpha| \gg 2$) we have that $\tilde{k}_u$ and $\tilde{k}_\beta$ are $O(1)$, in which case the special LU factorization and reversed triangular factorization appear best. Each method deteriorates in performance as $|\alpha| \to 2$, with cyclic reduction being the least affected. In the worst case $k_c = \log_2 n$, and $\tilde{k}_u = \tilde{k}_\beta = n$, in which case cyclic reduction and the special LU factorization appear to be best.

We next consider the periodic case $\rho = 1$. For highly diagonally dominant systems we have that $\tilde{k}_p$, $\tilde{k}_w$, and $\tilde{k}_u$ are $O(1)$, in which case the reversed triangular factorization is most efficient. Once again, each method deteriorates in performance as $|\alpha| \to 2$, with cyclic reduction being the least affected. In the worst case $k_c = \log_2 n$ and $\tilde{k}_p = \tilde{k}_u = \tilde{k}_w = n$, in which case cyclic reduction appears to be best.

We reemphasize that these conclusions are only valid for computations on a scalar processor. Cyclic reduction is preferred for solving single tridiagonal systems on vector computers because vector operations may be carried out during most of the computation [14]. The other algorithms described here are inherently recursive.

Of the methods considered, only the special LU factorization and Gaussian elimination can take advantage of the case when many systems of the form (1) must be solved, each differing only in the right-hand-side vector $\mathbf{g}$. After the first solve, the factorization portion of these algorithms may be eliminated. Doing this we arrive at methods with operation counts given in Table 8. We first consider the case $\rho = 0$. The methods of Table 8 appear to be best when $|\alpha| \approx 2$, although they do not appear to be significantly better than a full solve using reversed triangular factorization in the highly diagonally dominant case. In the periodic case reversed triangular factorization always seems preferable.

The operation counts that we have given are valid only for large n. In addition, they ignore all overhead required to implement the methods. The following numerical experiment is designed to assess the performance of these methods in practice.

In Figs. 1–4 we present timing data for cyclic reduction (CR, long dash), the special LU factorization method (SpLU, short dash), the reversed triangular factor-

FIG. 1. *Timing of algorithms in Dirichlet case for small* n (*solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

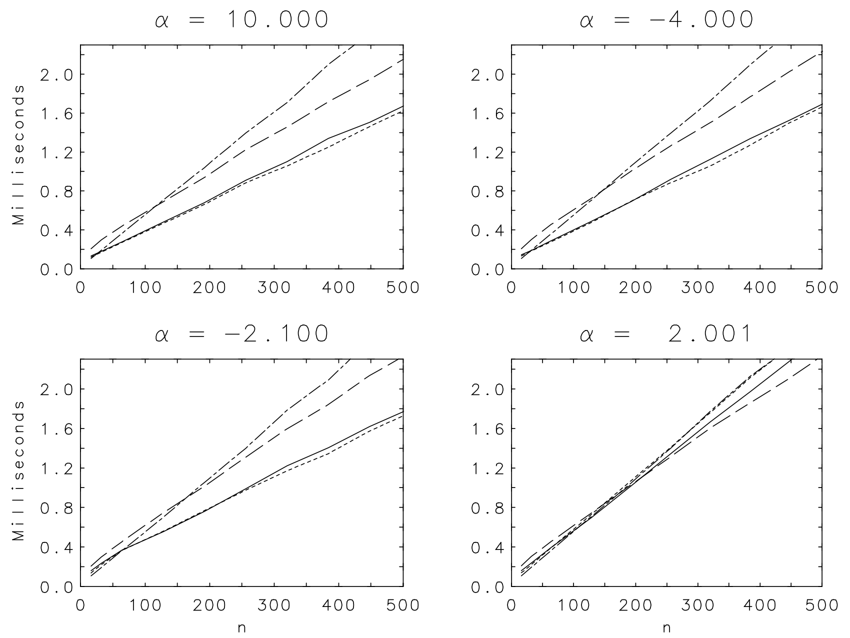

FIG. 2. *Timing of algorithms in Dirichlet case for large* n (*solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

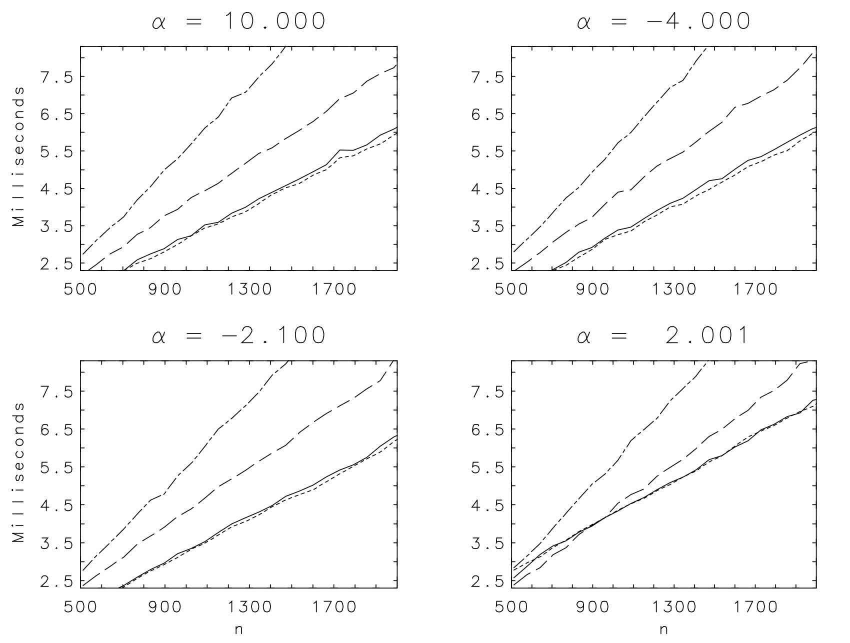

FIG. 3. *Timing of algorithms in periodic case for small* n (*solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

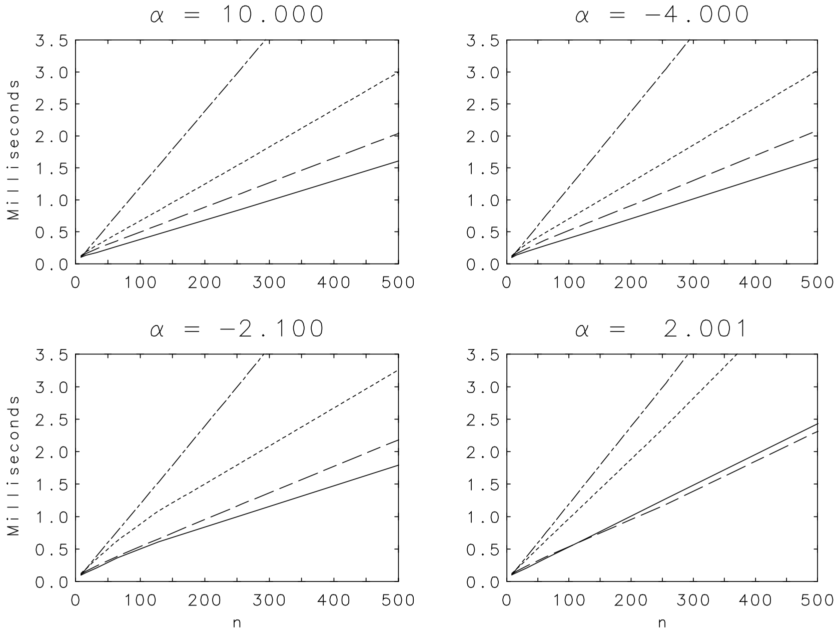

FIG. 4. *Timing of algorithms in periodic case for large* n (*solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

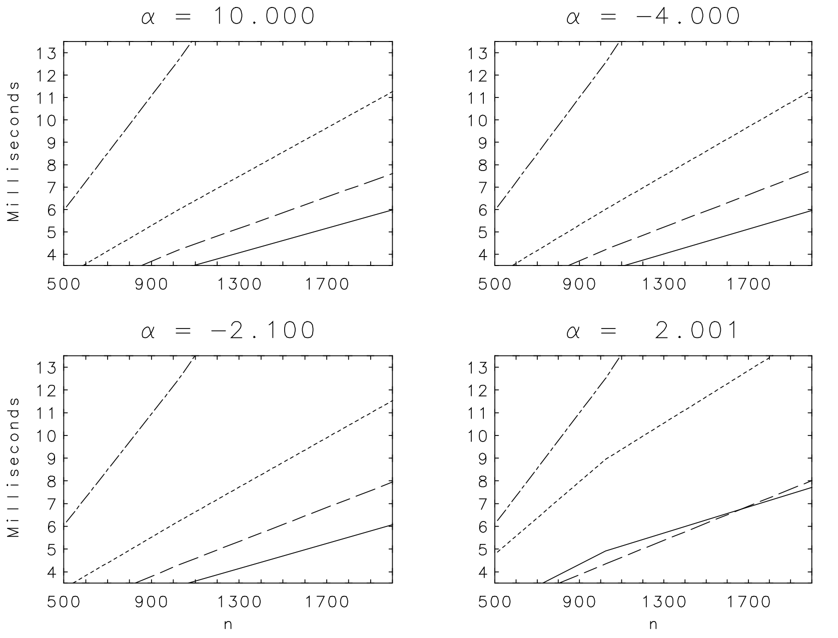

TABLE 9
Timing of algorithms on Cyber 855 (NOS). *Entries are* milliseconds : megaflops.

$\alpha = 10.0$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.1 : 0.8	0.1 : 1.2	0.2 : 0.7	0.2 : 1.0	0.1 : 1.0	0.2 : 1.5
64	0.4 : 1.0	0.3 : 1.2	0.3 : 1.3	0.4 : 1.4	0.4 : 1.2	0.5 : 1.1	0.3 : 1.4	0.8 : 1.5
256	1.2 : 1.4	0.9 : 1.5	0.9 : 1.4	1.4 : 1.5	1.1 : 1.6	1.6 : 1.2	0.8 : 1.6	3.1 : 1.5
1024	4.2 : 1.6	3.2 : 1.6	3.2 : 1.6	5.7 : 1.4	4.2 : 1.6	6.0 : 1.2	3.3 : 1.6	12.7 : 1.5
2048	8.2 : 1.7	6.1 : 1.7	6.3 : 1.6	11.6 : 1.4	7.8 : 1.8	11.5 : 1.3	6.1 : 1.7	26.9 : 1.4
$\alpha = -4.0$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.1 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.3 : 1.2	0.3 : 1.3	0.4 : 1.4	0.4 : 1.1	0.5 : 1.2	0.3 : 1.4	0.8 : 1.5
256	1.3 : 1.4	0.9 : 1.5	0.9 : 1.5	1.4 : 1.5	1.1 : 1.6	1.6 : 1.2	0.9 : 1.6	3.0 : 1.5
1024	4.4 : 1.6	3.3 : 1.6	3.4 : 1.5	5.7 : 1.4	4.2 : 1.7	6.0 : 1.2	3.3 : 1.6	12.6 : 1.5
2048	8.1 : 1.7	6.1 : 1.7	6.2 : 1.7	11.5 : 1.4	7.9 : 1.8	11.6 : 1.3	6.1 : 1.7	26.9 : 1.4
$\alpha = -2.1$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.2 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.4 : 1.2	0.4 : 1.5	0.4 : 1.4	0.4 : 1.2	0.6 : 1.5	0.4 : 1.6	0.8 : 1.5
256	1.3 : 1.4	1.0 : 1.4	1.0 : 1.5	1.4 : 1.5	1.2 : 1.5	1.8 : 1.4	1.0 : 1.7	3.1 : 1.5
1024	4.4 : 1.6	3.3 : 1.6	3.3 : 1.6	5.6 : 1.5	4.3 : 1.7	6.2 : 1.3	3.4 : 1.6	12.5 : 1.5
2048	8.6 : 1.7	6.4 : 1.6	6.5 : 1.6	11.8 : 1.4	8.1 : 1.8	11.8 : 1.3	6.2 : 1.7	26.8 : 1.4
$\alpha = 2.001$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.2 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.4 : 1.1	0.4 : 1.5	0.4 : 1.4	0.4 : 1.2	0.6 : 1.5	0.4 : 1.6	0.8 : 1.5
256	1.3 : 1.4	1.4 : 1.3	1.3 : 1.7	1.4 : 1.5	1.2 : 1.5	2.4 : 1.6	1.3 : 1.8	3.1 : 1.5
1024	4.5 : 1.6	4.4 : 1.4	4.3 : 1.8	5.7 : 1.4	4.4 : 1.6	9.0 : 1.6	4.9 : 1.9	12.5 : 1.5
2048	8.6 : 1.7	7.3 : 1.5	7.3 : 1.7	11.6 : 1.4	8.2 : 1.7	14.9 : 1.4	7.9 : 1.9	26.9 : 1.4

† implementation restricted to $n = 2^k$.

ization method (RTF, solid line), and Gaussian elimination (GE, long-short dash) as a function of n for various values of α in the Dirichlet and periodic cases. (The results for other boundary conditions are similar to the Dirichlet results.) Times are given in milliseconds on a Cyber 180/855 (NOS). Each algorithm was written in standard Fortran and compiled with the highest level of optimization (FTN5,OPT=3). The problems were generated with solutions $\mathrm{x} = 1$; the accuracy of the computed solutions for each method were comparable. All tests are for a single right-hand side. Data were gathered for $n = 16, 32, 64(64)2048$ in the Dirichlet case and for $n = 2^k$, for $k = 4, 5, \cdots, 11$ in the periodic case. A subset of these data is presented in Table 9, where entries are given in the form *millisec : megaflops*. The megaflop entries provide a check on the operation counts; since the computations are done on a scalar processor, these rates should converge with n and be roughly the same for each method.

The data are generally consistent with the timing formulas of Table 7. SpLU and RTF are the most sensitive to the value of $|\alpha|$, while CR is the least: the degradation of SpLU and RTF from the largest to smallest $|\alpha|$ for $n = 2048$ was about 15 percent in the Dirichlet case and 23 percent in the periodic case; the degradation of CR was only 5 percent in both cases. In the Dirichlet case the best methods were SpLU and RTF, whose performance was nearly identical. As $|\alpha|$ approached 2 their superiority to CR decreased: for $n = 2048$ they were 26 percent faster at the largest $|\alpha|$, and only 15 percent faster at the smallest $|\alpha|$. At $\alpha = 2.001$ CR was actually slightly faster for n in the range 200–900. In the periodic case RTF performed best; however, its superiority to cyclic reduction decreased as $|\alpha|$ approached 2: for $n = 2048$ it was 22 percent faster at the largest $|\alpha|$, and only 4 percent faster at the smallest $|\alpha|$. Once again, at $\alpha = 2.001$ CR was actually slightly faster for intermediate n, i.e., 128, 256, 512, and 1024. Note that an implementation of CR for arbitrary n would execute more slowly than the one tested here.

We conclude that, on a single scalar processor, SpLU and RTF are best for Dirichlet/Neumann problems and RTF is best for periodic problems, although their superiority over CR decreases as the system becomes less diagonally dominant. The difference between the methods is most apparent in the periodic case. In the nonperiodic, weakly diagonally dominant case, the SpLU is preferred when multiple solves for different right-hand sides are required.

REFERENCES

[1] M. D. BAKES, *An alternative method of solution of certain tri-diagonal systems of linear equations*, Comput. J., 7 (1964), pp. 135–136.

[2] R. F. BOISVERT, *A fourth-order accurate Fourier method for the Helmholtz equation in three dimensions*, ACM Trans. Math. Software, 13 (1987), pp. 221–234.

[3] J. J. DONGARRA, C. B. MOLER, J. R. BUNCH, AND G. W. STEWART, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[4] D. J. EVANS, *An algorithm for the solution of certain tridiagonal systems of linear equations*, Comput. J., 15 (1972), pp. 356–359.

[5] ———, *Fast* ADI *methods for the solution of linear parabolic partial differential equations involving* 2 *space dimensions*, BIT, 17 (1977), pp. 486–491.

[6] ———, *On the solution of certain Toeplitz tridiagonal linear systems*, SIAM J. Numer. Anal., 17 (1980), pp. 675–680.

[7] D. J. EVANS AND C. D. V. FORRINGTON, *Note on the solution of certain tri-diagonal systems of linear equations*, Comput. J., 5 (1963), pp. 327–328.

[8] D. FISHER, G. GOLUB, O. HALD, C. LEIVA, AND O. WIDLUND, *On Fourier-Toeplitz methods for separable elliptic problems*, Math. Comp., 28 (1974), pp. 349–368.

[9] I. GLADWELL AND I. WAIT, *A Survey of Numerical Methods for Partial Differential Equations*, Clarendon Press, Oxford, U.K., 1979.

[10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[11] D. HELLER, *Some aspects of the cyclic reduction algorithm for block triangular linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484–496.

[12] R. W. HOCKNEY, *A fast direct solution of Poisson's equation using Fourier analysis*, J. Assoc. Comput. Mach., 12 (1965), pp. 95–113.

[13] H. B. KELLER, *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, London, 1963.

[14] J. J. LAMBIOTTE AND R. G. VOIGT, *The solution of tridiagonal linear systems on the* CDC *Star*-100 *computer*, ACM Trans. Math. Software, 1 (1975), pp. 308–329.

[15] M. A. MALCOLM AND J. PALMER, *A fast method for solving a class of tridiagonal linear systems*, Comm. ACM, 17 (1974), pp. 14–17.

[16] D. J. ROSE, *An algorithm for solving a special class of tridiagonal systems of linear equations*, Comm. ACM, 12 (1969), pp. 234–236.

[17] P. N. SWARZTRAUBER, *The methods of cyclic reduction, Fourier analysis, and the* FACR *algorithm for the discrete solution of Poisson's equation on a rectangle*, SIAM Rev., 19 (1977), pp. 490–501.

[18] R. A. SWEET, *A generalized cyclic reduction algorithm*, SIAM J. Numer. Anal., 11 (1974), pp. 506–520.

[19] O. B. WIDLUND, *On the use of fast methods for separable finite difference equations for the solution of general elliptic problems*, in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 121–131.