A Fast Cauchy-Riemann Solver

By Michael Ghil* and Ramesh Balgovind**

Abstract. We present a solution algorithm for a second-order accurate discrete form of the inhomogeneous Cauchy-Riemann equations. The algorithm is comparable in speed and storage requirements with fast Poisson solvers. Error estimates for the discrete approximation of sufficiently smooth solutions of the problem are established; numerical results indicate that second-order accuracy obtains even for solutions which do not have the required smoothness. Different combinations of boundary conditions are considered and suitable modifications of the solution algorithm are described and implemented.

1. Introduction. Inhomogeneous Cauchy-Riemann equations appear naturally in many fluid-dynamical problems, as the divergence and the vorticity equations of a two-dimensional steady flow field \((u, v) = (u(x, y), v(x, y))\). The velocity components \(u, v\) are usually called in this context primitive variables, in contradistinction to the derived variables \(\psi, \zeta\) in the stream function-vorticity formulation of the flow equations (e.g., Roache [28]). In the latter formulation, the stream function \(\psi\) satisfies a Poisson equation; and computations with this formulation have greatly benefited from the rapid development of fast direct methods for the solution of Poisson’s equation, or Poisson solvers (Buneman [4], Buzbee, Golub and Nielson [5], Dorr [9], Fischer, Golub, Hald, Leiva and Widlund [12], Golub [18], Hockney [21], [22], Widlund [32]).

Working in the primitive variables, however, permits the treatment of more general flows. Indeed, either nondivergence or irrotationality of the flow are required in order to introduce a stream function \(\psi\) or a velocity potential \(\phi\), and obtain a Poisson equation for them. There are many situations of practical interest in which neither of these assumptions holds. Furthermore, the formulation of boundary conditions is often easier in terms of the primitive variables, by using physical considerations which arise naturally from the problem. On the other hand, a boundary condition on the vorticity \(\zeta\) for instance is at times hard to formulate (Langlois [23]); the construction of appropriate discrete versions of such a boundary condition is often even more difficult (Öliger and Sundström [27]). Hence, the desirability of simple, physically meaningful boundary conditions and, thus, of the use of primitive variables.

Lomax and Martin [24] have developed a fast Cauchy-Riemann solver and

Received April 10, 1978.


Key words and phrases. Fast direct solvers, Cauchy-Riemann equations, elliptic first-order systems, transonic flow.

*The work of this author was supported in part by NASA, Grant No. NSG-5130.
**The work of this author was supported in part by NASA, Grant No. NSG-5034.

© 1979 American Mathematical Society
0025-5718/79/0000-0058/$13.75

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
applied it to a quasilinear problem in aerodynamics ([24], [25]). Additional versions of
their solver are given in [26]. Our interest in the problem stems from a different applica-
tion, dealing with a two-dimensional version of the equations of dynamic meteorology
(Ghil [14], [15]). The solver we present also differs from those of [24], [26] in a num-
ber of ways: the boundary conditions and their numerical treatment, the decoupling of
$u, v$ and the reduction to a discrete Poisson problem, and finally the method of solution
of the resulting discrete Poisson equation are all different. We substantiate by numerical
results that the present solver is second-order accurate, even for solutions which do not
have the formally required degree of smoothness. The solvers of [24], [26] seem to be
only first-order accurate according to our numerical tests.

The intended application of our solver is to a fully nonlinear first-order system,
rather than a quasilinear one. This system is a generalization of a Monge-Ampère equation
[15], [17]. Eventually, we hope to apply the solver to cases where the nonlinear equa-
tions are of mixed elliptic-hyperbolic type [16], [17]. Preliminary results are encourag-
ing, and we expect to pursue the nonlinear problem in a future publication.

The organization of the article is the following: Section 2 contains the description
in continuous and then in discrete form of the model problem of which we seek a fast
solution. Section 3 contains the derivation and description of the solution algorithm.
Section 4 presents numerical results for test computations with the model problem. Sec-
tion 5 presents modifications of the model problem arising from changes in boundary
conditions. Section 6 contains a comparison of results with the solvers of [24], [26].
Section 7 gives conclusions and a discussion of possible extensions and generalizations.
Finally, Appendix A presents an error estimate for the method, and Appendix B con-
tains a listing of the basic program.

Acknowledgements. It is gratifying to acknowledge useful discussions with Profes-
sors Eugene Isaacson and Olof Widlund. Numerical calculations were performed in part
on the CDC 6600 of the Courant Mathematics and Computing Laboratory, New York
University under Contract EY-76-C-02-3077 with the U. S. Department of Energy.

2. The Model Problem.

The Differential Equations. We wish to study the fast numerical solution of the
elliptic system of two first-order linear equations in two independent variables,

\begin{align}
(2.1a) \quad & u_x + v_y = d(x, y), \\
(2.1b) \quad & u_y - v_x = e(x, y).
\end{align}

The dependent variables $u, v$ can be thought of as velocity components in the $x, y$ direc-
tions, respectively. In this interpretation $d, e$ are the divergence and vorticity of the
flow, which are assumed to be known. If $d \equiv 0 \equiv e$, (2.1) are the Cauchy-Riemann
equations, and $u, v$ are analytic. We are interested in the inhomogeneous case, $|d| + |e| \neq 0$,
and we concentrate on real-valued $d, e, u, v$, although the method is applicable with
minor changes to complex-valued functions as well.

We consider a rectangular domain $R$, taken without loss of generality to be $R =
\{(x, y): 0 < x < 2\pi, 0 < y < \pi\}$. The boundary conditions are that $v$ is given on the
lower and upper side of the rectangle,

(2.2a) \[ v = v^{(0)}(x), \quad y = 0, \]

(2.2b) \[ v = v^{(1)}(x), \quad y = \pi, \]

and that both \( u \) and \( v \) are periodic in the \( x \)-direction,

(2.3a) \[ u(x + 2\pi, y) = u(x, y), \]

(2.3b) \[ v(x + 2\pi, y) = v(x, y). \]

The Gauss divergence theorem implies that \( d(x, y) \), \( v^{(0)}(x) \), and \( v^{(1)}(x) \) have to satisfy

(2.3c) \[ \int_0^{2\pi} \int_0^\pi d(x, y) dx dy = \int_0^{2\pi} \{ v^{(1)}(x) - v^{(0)}(x) \} dx. \]

These conditions, together with (2.1), determine \( v \) completely and \( u \) up to an additive constant (compare Ghil [15]). The latter indeterminacy in the solution can be eliminated, for instance, by prescribing \( u \) at one arbitrary point \( (x_0, y_0) \) in the rectangle.

The boundary conditions we use are associated with a standard channel-flow problem in geophysical fluid dynamics (e.g., Elvius and Sundström [10], Gustafsson [20]), from which the nonlinear problem mentioned in Section 1 is derived (Ghil [15], [17]). Extensions to different boundary conditions and to irregular domains will be discussed in Sections 5 and 7.

**The Difference Equations.** The discretization of the problem we chose is to approximate the derivatives in (2.1) by finite differences. Let \( U, V, D, E \) stand for the mesh functions which approximate the continuous functions \( u, v, d, e \); and let \( h, k \) stand for the mesh size in the \( x, y \)-directions, respectively. It is natural to use centered differences to replace the corresponding derivatives in (2.1). We write

(\( \Delta \)) \[ U(x + \delta, y) - U(x - \delta, y) \approx 2\delta u_x(x, y), \]

\[ U(x, y + e) - U(x, y - e) \approx 2e u_y(x, y), \]

for \( u \) and similar formulas for \( v \); this yields a second-order accurate approximation of the derivatives and allows us to expect that in some adequate norm \( \| \cdot \| \),

\[ \| U - u \| + \| V - v \| = O(h^2) + O(k^2). \]

The use of centered differences in a straightforward manner, on an unstaggered mesh \( x_i = ih, y_j = jk \), leads, however, to the existence of spurious null vectors \( (U, V) \), i.e., to zero eigenvectors of the discrete matrix operator which approximates the differential Cauchy-Riemann operator. To avoid dealing with these null vectors and to obtain an invertible discrete matrix operator, we used a staggered mesh (Figure 1). Such a mesh, suggested already by Lomax and Martin [24], can be formulated for Eqs. (2.1) in a particularly efficient way.

Let \( u, v \) denote points at which \( U, V \) are defined, and let \( \cdot, \times \) (for the vector operators divergence and curl) denote the points at which the discrete versions (2.4a, b) (see below) of equations (2.1a, b) are written and at which \( D, E \) are defined. Thus,
u, v alternate on diagonals of the mesh and so do ·, × on diagonals parallel to and alternating with those of u, v; in other words, v, ×, u, ·, in clockwise direction, occupy the corners of an elementary mesh cell of area $hk/4$ (see Figure 1). No averaging of $U, V$ is necessary in writing (2.4a, b) on this staggered mesh. Furthermore, the boundary conditions (2.2) and the periodicity condition (2.3a, b) can be easily handled. Indeed, let $U, V$ be indexed independently, with $y = 0, \pi$ being horizontal $V$-lines corresponding to $V$-indices $j = 0, N$, and with $x = 0, 2\pi$ being vertical $V$-lines with $V$-indices $i = 1, M + 1$. Then the computational domain includes the points $((i - 1)h, jk)$, at which $V_{ij}$ is defined, and the points $((i - 1/2)h, (j - 1/2)k)$, at which $U_{ij}$ is defined; in other words, $V_{ij}$ approximates $v((i - 1)h, jk)$, $1 \leq i \leq M + 1, 0 \leq j \leq N$, and $U_{ij}$ approximates $u((i - 1/2)h, (j - 1/2)k), 1 \leq i \leq M, 1 \leq j \leq N$, with $h = 2\pi/M, k = \pi/N$.

The discrete equations are centered:

(2.4a) $(U_{i,j} - U_{i,j-1})/h + (V_{i,j} - V_{i,j-1})/k = D_{ij}$, $1 \leq i \leq M, 1 \leq j \leq N$,

(2.4b) $(U_{i,j+1} - U_{i,j})/h - (V_{i,j+1} - V_{i,j})/k = E_{ij}$, $1 \leq i \leq M, 1 \leq j \leq N - 1$;

here $D_{ij}$ approximates $d((i - 1)h, (j - 1/2)k)$, and $E_{ij}$ approximates $e((i - 1/2)h, jk)$. 

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
Notice that $\delta, \epsilon$ in (A) have been replaced after staggering by $h/2, k/2$, rather than by $h, k$. The boundary conditions become

\begin{align}
V_{i,0} &= \psi((i-1)h), \quad 1 \leq i \leq M, \\
V_{i,N} &= \psi((i-1)h), \quad 1 \leq i \leq M,
\end{align}

and the periodicity condition becomes

\begin{align}
U_{0,j} &= U_{M,j}, \quad 1 \leq j \leq N, \\
V_{1,j} &= V_{M+1,j}, \quad 0 \leq j \leq N.
\end{align}

Conditions (2.5), (2.6) leave $M(2N - 1)$ unknowns, to wit, $U_{ij}, 1 \leq i \leq M, 1 \leq j \leq N$, and $V_{ij}, 1 \leq i \leq M, 1 \leq j \leq N - 1$, while (2.4) yields $M(2N - 1)$ linear algebraic equations for them. We should expect from the situation in the continuous problem that the matrix of the linear system (2.4) has a one-dimensional null space. The resulting indeterminacy can be eliminated either by prescribing the value of $U_{ij}$ at an arbitrary index $(i_0, j_0)$, or by some alternative procedure which will arise naturally in Section 3; the necessary discrete compatibility condition will also be discussed there.

3. Solution Algorithm. Our plan shall be to rewrite (2.4)--(2.6) in a convenient block matrix form, to eliminate $V$, then to bring the remaining block matrix operating on $U$ to the form of a discrete five-point Laplacian, and finally to formulate a fast direct algorithm to solve for $U$. After this, $V$ obtains from $U$ by a straightforward, fast computation.

**Block Matrix Equation.** We start by rewriting the discrete linear system (2.4) in block matrix form. Let $U_j = (U_{1,j}, U_{2,j}, \ldots, U_{M,j})^*, V_j = (V_{1,j}, V_{2,j}, \ldots, V_{M,j})^*$, where $(\cdot)^*$ denotes (conjugate) transpose, so that $U_j, V_j$ are column vectors corresponding to a horizontal mesh line. $D_j$ and $E_j$ are introduced in similar fashion. Also let $\rho = k/h$. With this notation, (2.4) can be written as

\begin{align}
V_j &= V_{j-1} - TU_j + kD_j, \quad 1 \leq j \leq N, \\
U_{j+1} &= U_j - T^*V_j + kE_j, \quad 1 \leq j \leq N - 1;
\end{align}

here $\rho^{-1}T$ is the familiar **backward difference operator**

\begin{equation}
T = \rho \begin{bmatrix}
1 & -1 \\
-1 & 1 \\
\vdots & \vdots \\
-1 & 1
\end{bmatrix}_{M \times M},
\end{equation}

acting on the $M$-periodic vectors $U_j, V_j$ and $-\rho^{-1}T^*$ is the corresponding **forward** difference operator.
From (2.5) we have that $V_0$ and $V_N$ are known; accordingly, we redefine $D_1$ as $D_1 + k^{-1}V_0$ in the first one, and $D_N$ as $D_N - k^{-1}V_N$ in the last one of the equations (3.1a). After these changes of notation and a change of order, the vector equations (3.1) can be put into the block matrix form

\[
\begin{bmatrix}
N & N - 1 \\
N - 1 & \vdots & \vdots \\
N & \vdots & \vdots \\
1 & \vdots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
U_1 \\
U_{N-1} \\
U_N \\
V_1 \\
V_{N-1} \\
\end{bmatrix}
= \begin{bmatrix}
E_1 \\
E_{N-1} \\
D_1 \\
\end{bmatrix},
\]

where $I$ is the $M \times M$ identity matrix. This form is roughly analogous to writing (2.1a, b) in reversed order,

\[
\begin{bmatrix}
\partial/\partial y & -\partial/\partial x \\
\partial/\partial x & \partial/\partial y \\
\end{bmatrix}
\begin{bmatrix}
u \\
v \\
\end{bmatrix}
= \begin{bmatrix}
e \\
d \\
\end{bmatrix}.
\]

We notice that the matrix in (3.3) has only four nonzero diagonals of $M \times M$ blocks and that the blocks are at most scalar tridiagonal. Furthermore, all the nonzero entries are $\pm 1$ or $\pm \rho$. In the present form, however, we cannot take full advantage of the extreme sparsity and simplicity of the matrix in order to obtain a solution method comparable to fast Poisson solvers.

Before bringing (3.3) to a more advantageous form we remark that its matrix indeed has a one-dimensional null space: the sum of the $MN$ rows in the lower half of the matrix is zero. The corresponding compatibility condition that $\Sigma_{ij}D_{ij} = 0$ is the discrete counterpart of (2.3c).

Decoupling of $U$ and $V$. It is a well-known fact that each of the functions $u, v$ which satisfy (2.1) will also satisfy a Poisson equation obtained from (2.1) by eliminating the other dependent variable by cross-differentiation. This suggests the attempt to eliminate $V$ in system (3.3) in order to obtain a linear algebraic system for $U$ alone. The matrix of this system will be similar to a discrete Laplacian, and to it we shall be able to apply fast solution techniques.

The elimination of $V$ proceeds as follows. In system (3.3) add the $N$th block row to the $(N + 1)$st, then the $(N + 1)$st to the $(N + 2)$nd and so on. This discrete summation procedure is analogous to integration with respect to $y$ in (2.1a). The lower part of the system thus becomes
After this transformation, (3.3) can be written as

\[
\begin{pmatrix}
N \\
N-1
\end{pmatrix}
\begin{bmatrix}
T & I & 0 & \cdots & 0 \\
T & T & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
T & \cdots & T & 0 & 0
\end{bmatrix}
\begin{bmatrix}
U_N \\
V_1 \\
\vdots \\
V_{N-1}
\end{bmatrix}
= k
\begin{bmatrix}
D_1 \\
D_1 + D_2 \\
\sum_1^N D_j
\end{bmatrix}
\]

After this transformation, (3.3) can be written as

\[
\begin{pmatrix}
N \\
N-1
\end{pmatrix}
\begin{bmatrix}
P & Q \\
R & I_{N-1}
\end{bmatrix}
\begin{bmatrix}
U \\
V
\end{bmatrix}
= \begin{bmatrix}
E' \\
D' \\
k\sum_1^N D_j
\end{bmatrix}
\]

where \( U = (U_1^*, \ldots, U_N^*)^* \), \( V = (V_1^*, \ldots, V_{N-1}^*)^* \), \( I_{N-1} \) is the \( M(N-1) \times M(N-1) \) identity matrix, the definitions of \( P \), \( Q \) and \( E' \) are easily read from Eqs. (3.3), while those of \( R \) and \( D' \) follow from (3.4). In particular, note that \( D' \) and \( E' \) contain the factor \( k \).

From (3.5) it follows that

\[
\begin{align}
(3.6a) \quad & PU + QV = E', \\
(3.6b) \quad & RU + V = D',
\end{align}
\]

where we omit for the moment the last vector equation. Substituting \( V \) from (3.6b) into (3.6a), we obtain

\[
(3.7) \quad (P - QR)U = E' - QD',
\]

where

\[
(3.8) \quad QR = \begin{bmatrix}
TT^* \\
\vdots \\
TT^* & \cdots & TT^*
\end{bmatrix} \begin{array}{c} N-1 \\
0
\end{array}
\]

since \( Q \) is block diagonal with constant equal blocks and \( T^*T = TT^* \). Attaching now the last equation of (3.5) to (3.7) yields

\[
(3.9) \quad \begin{bmatrix}
P - QR \\
T & \cdots & T
\end{bmatrix} U = \begin{bmatrix}
E' - QD' \\
k\sum_1^N D_j
\end{bmatrix}
\]

or, changing sign in all but the last equation,
Let $S = TT^*$. Notice that

\[(3.10)\quad S = \rho(T + T^*)\]

or, in the familiar notation for tridiagonal matrices,

\[(3.10')\quad T = \rho(-1, 1, 0), \quad T^* = \rho(0, 1, -1), \quad S = \rho^2(-1, 2, -1);\]

this is a slight extension of standard notation: because of periodicity these matrices are actually circulants, rather than tridiagonal proper (cf. (3.2), (3.10)). Thus, $S$ corresponds to a central second difference, carrying further the analogy with the continuous case. However, (3.9') is still not in a form suitable for fast solution.

**Discrete Poisson Equation for $U$.** This form is now easily achieved. First, multiply the last block row by $T^*$. Then subtract the second block row from the first, the third from the second, and so on up to and including the last one; the new last row is obtained by adding all the new rows, from 1 to $(N - 1)$, to the last row. Finally, the new last row is taken as the first row of the system and all the signs are reversed, yielding

\[
\begin{bmatrix}
S + I & -I \\
-I & S + 2I & -I \\
& \ddots & \ddots & \ddots \\
& & -I & S + 2I & -I \\
& & & -I & S + I
\end{bmatrix}
\]

\[(3.11)\]

\[
\left[
\begin{array}{c}
T^*D_1 - E_1 \\
T^*(D_1 + D_2) - E_2 \\
\vdots \\
T^*\sum_{j=1}^{N-1} D_j - E_{N-1} \\
\sum_{j=1}^{N} D_j
\end{array}
\right] + k
\left[
\begin{array}{c}
B_1 \\
B_2 \\
\vdots \\
B_{N-1} \\
B_N
\end{array}
\right]
\]
the vector $\mathbf{B}$ is introduced for future notational convenience.

We notice that the matrix in (3.11) has almost, but not quite the form of a discrete Laplacian (compare for instance Buzbee, Golub and Nielson [5], for Laplacian with Dirichlet, Neumann and periodic boundary conditions). The matrix is symmetric and nonnegative semidefinite; in fact, $U_{ij} = \text{const}$ satisfies the homogeneous system. The system would be positive definite if any one diagonal element actually exceeded the sum of the off-diagonal elements in the same row (e.g., Collatz [6, pp. 43–47]). This immediately suggests that we make use of the possibility of prescribing $U_{i_0,j_0} = u_0$, i.e., that we increase the element at the position $Mj_0 + i_0$ along the diagonal by an arbitrary amount $\alpha > 0$, and add $\alpha u_0$ to the right-hand side of that equation at the same time. We shall call the system thus modified (3.11a) for future reference.

The system (3.11a) could now be solved by modifying any one of the different methods or combination of methods available for Poisson's equation, as reviewed for instance by Buzbee, Golub and Nielson [5], Dorr [9], or Widlund [32]. In this sense, the elimination of $V$ is analogous to a single step of odd/even reduction as proposed by Hockney [21], [22], although the role of this step is more crucial in the present context. We describe in the sequel the particular fast algorithm chosen to solve (3.11a).

Fast Direct Solution for $U$. Essentially, our algorithm is based on the one described in greater generality by Buzbee, Golub and Nielson [5] as matrix decomposition. It relies on the fact that the eigenvalues $\lambda_k$ and eigenvectors $\xi_k$ of $S$ are known, i.e., that

$$S\xi_k = \lambda_k \xi_k,$$

for

(3.12a) $\lambda_k = 2\rho^2(1 - \cos(2\pi(k - 1)/M)), \quad 1 \leq k \leq M,$

(3.12b) $\xi_k = M^{-1/2}(1, e^{-2\pi i(k-1)/M}, \ldots, e^{-2\pi i(M-1)(k-1)/M}), \quad 1 \leq k \leq M,$

where $i$ is the imaginary unit.

Our algorithm deviates, however, from general matrix decomposition because the first and last diagonal blocks in (3.11) differ from all the other blocks and because the matrix (3.11), without suitable modification, is singular. For the sake of simplicity, we shall describe the actual algorithm used directly in a self-contained manner.

Let $Q$ be the matrix whose columns are $\xi_1, \xi_2, \ldots, \xi_M$, and let $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_M)$. Then, with our normalization of $\xi_k$,

(3.13) $Q^*Q = I, \quad Q^*SQ = \Lambda.$

Introduce $\tilde{U}_j, \tilde{B}_j$ by

(3.14a) $U_j = Q\tilde{U}_j,$

(3.14b) $B_j = Q\tilde{B}_j, \quad 1 \leq j \leq N,$

where $B_j$ are the subvectors on the right-hand side of (3.11).
Clearly, (3.14) corresponds to a discrete Fourier transform in the \( x \)-direction. Notice in particular that

\[
\tilde{U}_{j,1} = \sum_{i=1}^{M} U_{i,j} / \sqrt{M},
\]

where \( \tilde{U}_{j,1} \) denotes the first component of the vector \( \tilde{U}_j \).

Premultiplying (3.11) by a block-diagonal matrix with all the diagonal blocks equal to \( Q^* \), and using (3.14), we obtain

\[
\begin{bmatrix}
\Lambda + I & -I \\
-I & \Lambda + 2I & -I \\
& \ddots & \ddots & \ddots \\
& & -I & \Lambda + 2I & -I \\
& & & -I & \Lambda + I
\end{bmatrix}
\begin{bmatrix}
\tilde{U}_1 \\
\tilde{U}_2 \\
\vdots \\
\tilde{U}_N
\end{bmatrix}
= \begin{bmatrix}
\tilde{B}_1 \\
\tilde{B}_2 \\
\vdots \\
\tilde{B}_N
\end{bmatrix}
\]

In order to turn the matrix in (3.15) from block-tridiagonal with diagonal blocks into a tridiagonal matrix, it suffices to reorder the components of \( \tilde{U} \) and of \( \tilde{B} \) by vertical lines rather than by Fourier transformed vectors of horizontal lines, i.e., let

\[
\tilde{U}_{k,j} = \tilde{U}_{j,k}, \quad \tilde{B}_{k,j} = \tilde{B}_{j,k}, \quad 1 \leq j \leq N, 1 \leq k \leq M.
\]

Here again the first subscript denotes the vector partition and the second denotes the component of the subvector; this notational convention is different from the mesh notation introduced in Section 2 and used in (3.1). Using the reordering (3.16), Eq. (3.15) becomes

\[
(3.17) \quad -A_k \hat{U}_k = \hat{B}_k, \quad 1 \leq k \leq M,
\]

with

\[
(3.18a) \quad A_k = \begin{bmatrix}
c_k & 1 \\
1 & a_k & 1 \\
& 1 & a_k & 1 \\
& & \ddots & \ddots & \ddots \\
& & & 1 & a_k & 1 \\
& & & & 1 & c_k
\end{bmatrix}_{N \times N},
\]

\[
(3.18b) \quad a_k = -2 - \lambda_k = 2 \rho^2 \left[ \cos(2\pi(k - 1)/M) - 1 \right] - 2,
\]

\[
(3.18c) \quad c_k = -1 - \lambda_k = 2 \rho^2 \left[ \cos(2\pi(k - 1)/M) - 1 \right] - 1, \quad 1 \leq k \leq M;
\]

here the \( c_k \) come from the first and last diagonal block in (3.15).

It is easy to check that each \( A_k \) is nonsingular, except the first, which has a one-dimensional null space. This reflects the fact that we have not yet utilized a side condition eliminating \( U = \text{const} \) as a null vector of (3.11), or of (3.15). As in the
A FAST CAUCHY-RIEMANN SOLVER

discussion of (3.11α), we can deal with the singularity of A1 by prescribing any component of \( \bar{U}_1 \), \( \bar{U}_{1,j_0} \) say; this means prescribing \( \bar{U}_{1,j_0} = \bar{U}_{0,1} = \Sigma_{i=1}^{M} u_{i,j_0}/\sqrt{M} \), rather than one given mesh value \( u_{0,j_0} \). Again we call the suitably modified systems (3.15α), (3.17α) and (3.18α).

The LU factorization of the tridiagonal matrices in (3.18α) is performed next, say \( A_k = L_k U_k \), storing the nontrivial entries of \( L_k \) and \( U_k \). The diagonal elements of \( L_k \) and the superdiagonal elements of \( U_k \) are identically 1, while the subdiagonal elements of \( L_k \) and the diagonal elements of \( U_k \) are reciprocals of each other. Hence only the diagonal elements of \( U_k \) in fact need to be stored.

Having computed \( \tilde{B} \) and thus \( \tilde{B} \), from the original data, we can solve for \( \tilde{U} \) and hence for \( U \). The computation of \( \tilde{B} \) from \( B \) and of \( U \) from \( \tilde{U} \) involves matrix multiplication by \( Q^* \) and by \( Q \), respectively. Since these correspond to a direct and an inverse Fourier transform, they can be performed efficiently by a fast Fourier transform (FFT) algorithm. The algorithm used was the one published by Cooley, Lewis and Welch [7], modified to operate in the real.

Summary and Programming Considerations. The solution algorithm developed in this section reduces in practice to the following:

(i) From the data of the problem \( D, E \), compute

\[
\begin{align*}
B_1 &= k(T^*D_1 - E_1), \\
B_j &= k(T^*D_j + E_{j-1} - E_j), \quad 2 \leq j \leq N - 1, \\
B_N &= k(T^*D_N + E_{N-1}),
\end{align*}
\]

where \( D_1, D_N \) include the boundary data \( V_0, V_N \), respectively. The number of operations is \( n_1 = 4MN.*** \) Notice that, after shifting of the blocks \( E_j \), \( B \) can overwrite \( E \), since \( E \) is no longer needed.

(ii) From the \( B_j \) above compute \( \tilde{B}_j \) (cf. (3.14b)) by

\[
\tilde{B}_j = Q^*B_j, \quad 1 \leq j \leq N,
\]

using an FFT, with \( n_2 = 2MN \log_2 M \) real operations. Although FFT algorithms exist which use \( MN \log_2 M \) operations for real transforms, we did not feel that in our application actual execution time would be much improved by the utilization of such an algorithm. In this step, \( \tilde{B} \) can be stored in the previous location of \( B \). Note that \( \tilde{B}_j \) is complex with pairwise complex conjugate components.

(iii) Solve (3.17α) for the \( \bar{U}_k \) by forward elimination and back substitution, using the stored entries of the \( LU \) factors. One solution, for fixed \( k \), requires \( 8N \) real operations, since the complex conjugate components of \( \bar{B}_j \) enter nonsymmetrically into the \( LU \) factorization of (3.17α), and we have to operate separately on the real and the imaginary parts of \( \tilde{B} \). However, the fact that the components of \( \tilde{U}_j \) are also complex

***All operation and storage counts are given to highest order. We do not distinguish between addition, multiplication and division, and we do not assume that accumulation of products is particularly efficient. Furthermore, we take \( h = k \), or \( \rho = 1 \).
conjugate allows us to solve only $M/2 + 1$ systems, so that the operation count for this step is $n_3 = 4MN$ real operations. Notice that no reindexing of $B$ as indicated by (3.16) is actually necessary, and that at this stage $\hat{U}$ can overwrite $\hat{B}$. The shifting of $B$ elements and $U$ elements required to use the $LU$ subroutine is relatively fast.

(iv) From the $\hat{U}_k$ computed in (iii) obtain $U$ (again with no need for reindexing) by carrying out (3.14a) with another application of the FFT, in $n_4 = 2MN \log_2 M$ real operations, and with $U$ stored instead of $\hat{U}$.

(v) Compute $V$ from $U$ obtained in (iv), using (3.1a). This requires $n_5 = 4MN$ operations. Notice that in this recursion $V$ does not depend on $U_N$, $D_N$, since $V_N$ is known. Using (3.1a) as a backward recursion would yield independence of $V$ on $U_1$, $D_1$, since $V_0$ is also known. This redundancy, however, is only apparent and computations with (3.1a) used as either forward or backward recursion yielded results with the same accuracy, which only depended on the accuracy of $U$.

The final operation count is

$$n = \sum_{k=1}^{s} n_k = 4MN(3 + \log_2 M),$$

to obtain both $U$ and $V$ at all the grid points at which they are defined.

The storage requirement for the $L$, $U$ factors is $s_1 = MN/2$, using again the fact that we only solve for half the number of components of $\hat{U}$; the storage requirement for the right-hand sides $D$, $E$, is $s_2 = 2MN$; and these locations are successively overwritten until $U$, $V$ are stored in them. Hence, the total storage requirement is $s = s_1 + s_2 = 5MN/2$.

We conclude this section with a diagram of the algorithm:

$$D_j, E_j, V_0, V_N \xrightarrow{\frac{E}{4MN}} B_j \xrightarrow{\text{FFT} \ 2MN \log_2 M} \hat{B}_j \xrightarrow{\text{R} \ 0} \hat{B}_k \xrightarrow{\text{LU} \ \frac{4MN}{L}} \hat{U}_k$$

$$\xrightarrow{\frac{R}{4MN}} \hat{U}_j \xrightarrow{\text{FFT} \ 2MN \log_2 M} U_j \xrightarrow{\frac{E}{4MN}} V_j,$$

here $\hat{E}$ stands for evaluations by linear recursion, FFT for an application of the Fast Fourier Transform, $R$ for reindexing, and $LU$ for factorization. The ranges of the indices are $1 \leq j \leq N$, $1 \leq k \leq M$, and the operation counts are given under the corresponding step of the algorithm.

4. Numerical Results. In this section we shall present results of test computations for model problem (2.1)–(2.3). The results will provide evidence for the short computation time required by the method, for its second-order accuracy and for its insensitivity to errors in the data.

The experiments consisted in evaluating analytically $d$, $e$ in (2.1) for known $u$, $v$, and then comparing the numerical solutions $U$, $V$ with the correct analytical $u$, $v$. The computations were carried out on an IBM 360/95 computer, with a FORTRAN IV, level $H$ compiler, the code optimization parameter OPT being set to the value OPT = 2. Double precision arithmetic was used throughout our numerical experiments. Test comparisons on an Amdahl 470V/6 computer and on a CDC 6600
gave entirely similar results; the program is currently being developed into a package and we expect to test it on other computers as well.

Initially, three different versions of the program were run. The first version solved the positive definite modification (3.11a) of the system (3.11) by Cholesky factorization. It is denoted by C in the tables, and given for orientation purposes only. This version required much longer computer time; moreover, results were less accurate than with the other two versions, since the larger number of arithmetic operations introduced more round-off error.

The second version utilized diagonalization by the discrete Fourier transform as given in (3.14) and the reindexing given in (3.16), but did not use the FFT in steps (ii) and (iv) of the algorithm. It is thus somewhat comparable to Hockney’s original method [21] and is also included for comparison purposes. Its numerical results were equal at least to within three significant digits to those of the third version and are listed separately only in Table III. This version is denoted by P in the tables.

**Table I**

CPU execution times for the versions C, P and F on IBM 360/95 with compiler optimization OPT = 2. The number of points is \(MN = M^2/2\).

<table>
<thead>
<tr>
<th>M</th>
<th>C</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>.07</td>
<td>.09</td>
<td>.01</td>
</tr>
<tr>
<td>32</td>
<td>.88</td>
<td>.65</td>
<td>.04</td>
</tr>
<tr>
<td>64</td>
<td>14.35</td>
<td>5.17</td>
<td>.13</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td>.51</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td>2.11</td>
</tr>
</tbody>
</table>

**Table II**

CPU execution times for version F with different optimization levels, OPT = 0, 1, 2, 3, on IBM 360/95, Amdahl 470V/6, and CDC 6600. The compiler used on the CDC computer was FTN 4.6; on the other two computers, a FORTRAN IV, level H compiler was used.

<table>
<thead>
<tr>
<th>M</th>
<th>360/95</th>
<th>470V/6</th>
<th>6600</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OPT/0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>0.02</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>32</td>
<td>0.08</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>64</td>
<td>0.35</td>
<td>0.20</td>
<td>0.13</td>
</tr>
<tr>
<td>128</td>
<td>1.40</td>
<td>0.88</td>
<td>0.51</td>
</tr>
<tr>
<td>256</td>
<td>5.91</td>
<td>3.70</td>
<td>2.11</td>
</tr>
</tbody>
</table>

The third version is the one actually described in the summary of Section 3 and listed in Appendix B; it represents the proposed method. This version is denoted by F in the tables.
Tables I and II contain timing results. These are essentially independent of the solution, and depend only on the method, or program version, and on the number of points. Table I compares CPU execution times for the three program versions C, P and F on the IBM 360/95 computer with compiler optimization OPT = 2. As the number of points increases, the advantages of version F become more and more obvious. We can also see clearly that the execution time for version F is proportional to the number of points MN.

Table II gives CPU execution times for version F with different optimization levels, OPT = 0, 1, 2, 3, on the IBM 360/95, Amdahl 470V/6, and CDC 6600 computers. This is meant to give an idea of the order of magnitude of running time improvements which could still be achieved by code optimization. We notice that timing with OPT = 3 is not superior to OPT = 2.

The other tables are organized as follows: the first column contains the known test solution (u, v), and for Table V only, its L_2 and L_\infty norms. The norms used for u are the continuous norms

\begin{align}
\|u\|_2^2 &= \frac{1}{2\pi^2} \int_0^\pi \int_0^{2\pi} u^2(x, y) \, dx \, dy,
\end{align}

\begin{align}
\|u\|_\infty &= \sup_{0 < x < 2\pi, 0 < y < \pi} |u(x, y)|,
\end{align}

and similarly for v. The corresponding L_2 and L_\infty norms for the vector (u, v) are defined in the obvious way, with u^2 + v^2 as the integrand in (4.1a) and \max\{ |u|, |v| \} as the maximand in (4.1b). These norms are given in order to compare the corresponding norms of the computational error with them.

The second column contains the version of the program used, specifically version C or version P/F. The third column contains the value of M, the number of points in the x-direction. All reported computations were carried out with equal mesh spacing in the x- and y-directions, i.e., h = k = 2\pi/M. Hence, N = M/2 always, and the total number of points equals M^2/2.

The remaining columns contain the absolute errors in (u - U, v - V). The norms used here are the discrete counterparts of the norms in the first column:

\begin{align}
I_2^2(U) &= \sum_{i=1}^M \sum_{j=1}^N U_{ij}^2 / MN,
\end{align}

\begin{align}
I_\infty(U) &= \max_{i=1}^M \max_{j=1}^N |U_{ij}|,
\end{align}

and similarly for V and (U, V). For simplicity, in the column headings, as well as in (4.2) above, we used I_2(U), instead of I_2(u - U), and so on. The grid values of (u, v) entering (4.2) are those identified in Section 2 and, therefore, are taken at different locations for u than for v.

Table III shows that for u, v which are combinations of linear and of trigonometric functions, and thus eigenvectors of the discrete operator in (3.11), the results are essentially exact to machine accuracy (in double precision arithmetic).
### Table III

Numerical results with versions C, P and F of the algorithm for a number of simple test cases. The labelling of rows and columns is explained in the text.

<table>
<thead>
<tr>
<th>N</th>
<th>$l_2(U)$</th>
<th>$l_2(V)$</th>
<th>$l_2(U,V)$</th>
<th>$l_2(U)$</th>
<th>$l_2(V)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>u = sin(y), v = y sin(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>8 2.6351 -14</td>
<td>2.5457 -15</td>
<td>1.9989 -14</td>
<td>2.8144 -14</td>
<td>6.6613 -15</td>
</tr>
<tr>
<td>16</td>
<td>2.4364 -13</td>
<td>3.2929 -14</td>
<td>1.7934 -13</td>
<td>2.5881 -13</td>
<td>7.6605 -14</td>
</tr>
<tr>
<td>32</td>
<td>2.3308 -12</td>
<td>3.1543 -13</td>
<td>1.6889 -12</td>
<td>2.4685 -12</td>
<td>6.7857 -13</td>
</tr>
<tr>
<td>64</td>
<td>2.1272 -11</td>
<td>2.6296 -12</td>
<td>1.5273 -11</td>
<td>2.2408 -11</td>
<td>5.5140 -12</td>
</tr>
<tr>
<td>16</td>
<td>5.2001 -16</td>
<td>5.0322 -16</td>
<td>5.1500 -16</td>
<td>1.4017 -15</td>
<td>1.5543 -15</td>
</tr>
<tr>
<td>64</td>
<td>1.4957 -14</td>
<td>2.5201 -15</td>
<td>1.1267 -14</td>
<td>3.8474 -14</td>
<td>3.2488 -14</td>
</tr>
<tr>
<td>F</td>
<td>8 2.7088 -16</td>
<td>3.4397 -16</td>
<td>3.0403 -16</td>
<td>6.7247 -16</td>
<td>6.6613 -16</td>
</tr>
<tr>
<td>16</td>
<td>6.3077 -16</td>
<td>3.3899 -16</td>
<td>5.1500 -16</td>
<td>1.5021 -15</td>
<td>1.6613 -16</td>
</tr>
<tr>
<td>64</td>
<td>1.4255 -15</td>
<td>2.2937 -15</td>
<td>1.0286 -14</td>
<td>3.8044 -14</td>
<td>7.7934 -15</td>
</tr>
</tbody>
</table>

| u = y sin(x), v = sin(y) |
| 16 | 1.7252 -14 | 2.7282 -14 | 2.0024 -14 | 3.0583 -14 | 5.8356 -14 |
| 32 | 1.4271 -13 | 1.9268 -13 | 1.6875 -13 | 2.9177 -13 | 5.0736 -13 |
| 64 | 1.0832 -12 | 1.5755 -13 | 1.3401 -12 | 2.5202 -12 | 4.2358 -12 |
| 64 | 2.1996 -14 | 1.7951 -13 | 1.2690 -13 | 7.9020 -13 | 2.0683 -12 |
| F  | 8 2.0791 -16 | 1.9369 -16 | 2.0194 -16 | 6.6613 -16 | 5.6899 -16 |
| 16 | 3.7749 -16 | 2.0176 -16 | 1.3785 -16 | 8.8818 -16 | 3.1225 -14 |
| 64 | 3.5572 -15 | 1.0348 -12 | 7.2592 -13 | 9.7145 -13 | 1.4948 -12 |

| u = y sin(x) + 0.001, v = y^4 + 0.001 |
| C  | 8 2.2678 -14 | 1.4387 -14 | 1.9560 -14 | 5.3290 -14 | 3.5527 -14 |
| 16 | 8.0236 -14 | 8.2059 -14 | 8.1092 -14 | 1.9540 -13 | 2.4158 -13 |
| 32 | 6.6952 -13 | 7.8598 -13 | 7.2820 -13 | 1.6165 -12 | 2.3129 -12 |
| P  | 8 1.1175 -14 | 8.9470 -15 | 1.0279 -14 | 3.6193 -14 | 2.1316 -14 |
| 16 | 1.1179 -14 | 1.5689 -14 | 1.3750 -14 | 5.4622 -14 | 1.0303 -13 |
| 32 | 4.1173 -14 | 1.3525 -13 | 9.8620 -14 | 2.5047 -13 | 1.4211 -12 |
| 32 | 8.2406 -14 | 7.6397 -13 | 5.3471 -13 | 1.8474 -13 | 1.4246 -12 |

Table IV presents results for slightly perturbed boundary conditions $v(0)(x)$, $v(1)(x)$, $u(x_0, y_0)$ and right-hand sides $d(x, y)$, $e(x, y)$. Notice that errors in the computed $V$ are bounded by the errors in the prescribed boundary data $v(0)$, $v(1)$ in the absence of other errors. Also observe that the $l_2$ error in $U$ is larger when $d$, $e$ are perturbed than when $v(0)$ and $v(1)$ are perturbed by a comparable amount.

Table V contains results for a number of more severe test cases, in which the discretization error is nonnegligible. For comparison, an additional last column is included in this table, which gives a theoretical error bound $e(u)$ on the $l_\infty$-discretization error in $u$. This bound is explained and proved in Appendix A; the values of the constants $a$, $b$, $\alpha$ and $\beta$ introduced there were chosen so as to optimize the bound, and the constants $C_{\alpha}$, $C_{\beta}$ in (A.14), (A.15) were computed explicitly. We observe that the numerical errors are indeed lower than the bound in all cases in which $u$, $v$ are sufficiently differentiable. In fact, the
**Numerical results with the proposed algorithm (version F) for problems with slightly perturbed boundary conditions $\psi^{(0)}(x), \psi^{(1)}(x), u(x_0, y_0)$ and right-hand sides $d(x, y), e(x, y)$. The subscript $(\cdot)_c$ stands for the correct values.**

<table>
<thead>
<tr>
<th>$u = \sin(x) \sin(y), \quad v = \sin(4x)$</th>
<th>$M$</th>
<th>$t_2(U)$</th>
<th>$t_2(V)$</th>
<th>$t_2(U,V)$</th>
<th>$t_c(U)$</th>
<th>$t_c(V)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 1.01 d_c \quad e = 1.01 e_c$</td>
<td>16</td>
<td>2.1270</td>
<td>7.0594</td>
<td>5.0665</td>
<td>4.1757</td>
<td>1.0958</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>5.1732</td>
<td>3.6540</td>
<td>1.2091</td>
<td>1.6793</td>
<td>2.6040</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.2952</td>
<td>9.0377</td>
<td>2.9790</td>
<td>5.2345</td>
<td>6.4284</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>3.2417</td>
<td>9.9953</td>
<td>7.4064</td>
<td>1.4537</td>
<td>1.6021</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>8.1071</td>
<td>2.4878</td>
<td>1.8473</td>
<td>3.8233</td>
<td>4.0020</td>
</tr>
</tbody>
</table>

| $v^{(0)} = 1.01 v^{(0)} \quad v^{(1)} = 1.01 v^{(1)}$ | $u(x_0, y_0) = 1.01 u(x_0, y_0)$ | $d = 1.01 d_c \quad e = 1.01 e_c$ | $u = \sin(x) \sin(y), \quad v = \sin(4x)$ | $M$ | $t_2(U)$ | $t_2(V)$ | $t_2(U,V)$ | $t_c(U)$ | $t_c(V)$ |
|----------------------------------------------------------|----------------------------------|----------------------------------|-----|-----------|-----------|-------------|-----------|-----------|
| $d = 1.01 d_c \quad e = 1.01 e_c$                       | 16  | 6.8127    | 1.4141    | 1.0866      | 1.0417    | 2.1951    |
| $v^{(0)} = 1.01 v^{(0)} \quad v^{(1)} = 1.01 v^{(1)}$ | 32  | 5.6760    | 8.1393    | 6.9773      | 1.0102    | 1.2814    |
| $d = 1.01 d_c \quad e = 1.01 e_c$                       | 64  | 3.6452    | 6.6998    | 6.0982      | 1.0028    | 1.0967    |
| $v^{(0)} = 1.01 v^{(0)} \quad v^{(1)} = 1.01 v^{(1)}$ | 128 | 5.4003    | 6.3252    | 5.8773      | 1.0009    | 1.0133    |
| $d = 1.01 d_c \quad e = 1.01 e_c$                       | 256 | 5.3874    | 6.2204    | 5.8127      | 1.0003    | 1.0007    |

Errors become very nearly equal to the bound in some such cases (e.g., $u = y \sin(x), v = y^2 \sin(x)$), which seems to indicate that the bound is rather close to being sharp. The theoretical bound provides also an indication of error magnitude even in some cases where the data are not sufficiently differentiable (e.g., $u = y^4 \sin(8x), v = x(2\pi - x) \sin(8y); u = v = x(\pi - x)(2\pi - x) \sin(y)$).
TABLE IV (Continued)

<table>
<thead>
<tr>
<th>M</th>
<th>$L_2(U)$</th>
<th>$L_2(V)$</th>
<th>$L_2(U,V)$</th>
<th>$L_\infty(U)$</th>
<th>$L_\infty(V)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>6.7163  -3</td>
<td>3.5965  -3</td>
<td>5.3998  -3</td>
<td>6.6605  -2</td>
<td>2.7103  -2</td>
</tr>
<tr>
<td>16</td>
<td>3.2180  -2</td>
<td>7.0895  -2</td>
<td>5.3831  -2</td>
<td>7.2440  -2</td>
<td>1.7657  -1</td>
</tr>
<tr>
<td>64</td>
<td>7.4420  -3</td>
<td>4.5354  -3</td>
<td>6.4849  -3</td>
<td>4.1426  -2</td>
<td>1.8729  -2</td>
</tr>
<tr>
<td>128</td>
<td>6.3894  -3</td>
<td>3.7044  -3</td>
<td>5.0505  -3</td>
<td>5.2501  -2</td>
<td>2.7606  -2</td>
</tr>
<tr>
<td>16</td>
<td>3.8439  -2</td>
<td>8.0665  -2</td>
<td>6.1843  -2</td>
<td>1.1296  -1</td>
<td>1.5208  -1</td>
</tr>
<tr>
<td>64</td>
<td>7.5512  -3</td>
<td>5.1109  -3</td>
<td>6.4665  -3</td>
<td>4.6641  -2</td>
<td>1.8363  -2</td>
</tr>
<tr>
<td>128</td>
<td>6.8369  -3</td>
<td>3.7503  -3</td>
<td>5.5256  -3</td>
<td>5.3952  -2</td>
<td>2.7601  -2</td>
</tr>
<tr>
<td>16</td>
<td>1.1961  -1</td>
<td>5.7240  -1</td>
<td>9.5706  -1</td>
<td>3.6459  -1</td>
<td>1.5927  -1</td>
</tr>
<tr>
<td>32</td>
<td>2.9682  -1</td>
<td>4.2061  -1</td>
<td>2.3461  -1</td>
<td>1.0109  -1</td>
<td>4.1008  -1</td>
</tr>
<tr>
<td>64</td>
<td>7.3896  -2</td>
<td>3.3020  -2</td>
<td>5.7534  -2</td>
<td>2.6792  -1</td>
<td>1.0273  -1</td>
</tr>
<tr>
<td>128</td>
<td>1.9594  -2</td>
<td>8.9053  -3</td>
<td>1.5258  -2</td>
<td>1.0993  -1</td>
<td>3.2782  -2</td>
</tr>
<tr>
<td>256</td>
<td>8.3894  -3</td>
<td>5.3044  -3</td>
<td>7.0244  -3</td>
<td>8.3911  -2</td>
<td>4.8979  -2</td>
</tr>
<tr>
<td>16</td>
<td>2.2889  -1</td>
<td>9.2522  -1</td>
<td>1.7871  -1</td>
<td>5.1558  -1</td>
<td>2.8077  -1</td>
</tr>
<tr>
<td>32</td>
<td>5.3549  -1</td>
<td>2.4141  -1</td>
<td>4.2463  -1</td>
<td>1.7679  -1</td>
<td>6.7504  -1</td>
</tr>
<tr>
<td>64</td>
<td>2.3150  -1</td>
<td>5.7032  -2</td>
<td>1.0401  -1</td>
<td>4.9607  -1</td>
<td>1.6534  -1</td>
</tr>
<tr>
<td>128</td>
<td>3.4841  -2</td>
<td>1.3094  -3</td>
<td>2.6397  -2</td>
<td>1.6413  -1</td>
<td>4.1227  -2</td>
</tr>
</tbody>
</table>

The formal truncation error of the finite-difference scheme, in the interior as well as on the boundary, is $O(h^2 + k^2)$; hence, we expect that the scheme will produce second-order accurate results. A good way of testing this numerically is by computing the ratio

$$
\frac{\| U^{(2h, 2k)} - u \|}{\| U^{(h, k)} - u \|},
$$

and similar quantities for $v$ and $(u, v)$; here $\| \|$ is either the $L_2$ or the $L_\infty$ norm of (4.2). For sufficiently small $h$, $k$ and twice continuously differentiable $(u, v)$ these ratios should be very close to 4 if the method is indeed second-order accurate, and it should be close to 2 if the method is first-order accurate.

The indicated ratios are computed and entered in additional rows in Table V; the entry in column 3 indicates the values of $M$ and $2M$, rather than of $h$ and $h/2$, to which the norms whose ratio was taken correspond. The interesting result is that for those cases tested the method seems to have second-order accuracy even when $v$ is merely continuous and $u$ once continuously differentiable and that it is first-order accurate when both $u$ and $v$ are merely continuous. The continuity of $u$, $v$ is given in standard notation as a column to the left of the usual first column; i.e., $u$, $v \in C^0$, $C^1$, $C^2$, ..., $C^\infty$, indicate the number of continuous derivatives of $u$, $v$. Continuity is understood for $u$, $v$ and their derivatives as extended $x$-periodic functions.

†A more detailed discussion of accuracy appears in Appendix A.
the method necessitates for given numerical accuracy. The results are given here as relative errors, \( \|u - U\|_2/\|u\|_2 \), rather than as absolute errors, \( \|u - U\|_2 \), and similarly for \( v \) and \( f \). It seems that roughly 4 points per wavelength will give \( 10^{-1} \) relative error, 8 points will give \( 5 \times 10^{-2} \), and 16 will give \( 10^{-2} \). We notice again that if \( u \) oscillates less than \( v \), the error in \( u \) will be considerably smaller than that in \( v \).

Table VI contains a study of the number of grid points per wavelength which the method necessitates for given numerical accuracy. The results are given here as relative errors, \( L_2(u - U)/\|u\|_2 \), rather than as absolute errors, \( L_2(u - U) \), and similarly for \( v \) and for \( l_w \). It seems that roughly 4 points per wavelength will give \( 10^{-1} \) relative error, 8 points will give \( 5 \times 10^{-2} \), and 16 will give \( 10^{-2} \). We notice again that if \( u \) oscillates less than \( v \), the error in \( u \) will be considerably smaller than that in \( v \).
These conclusions are also supported by some of the results in Table V. It is interesting that experiments with solutions containing odd wave numbers give results which are only slightly worse than those for even wave numbers, if at all; in other words, using $M, N$ which are powers of 2 is not detrimental to accuracy, even when odd wave numbers are present in the solution.
### TABLE V (Continued)

<table>
<thead>
<tr>
<th>( \psi )</th>
<th>( M )</th>
<th>( \psi(0) )</th>
<th>( \psi(1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u = x(2\pi-x)\sin(y) )</td>
<td>( v = x^2(2\pi-x)^2\sin(y) )</td>
<td>( \psi(u) )</td>
<td>( \psi(v) )</td>
</tr>
<tr>
<td>( u = x(\pi-x)\sin(y) )</td>
<td>( v = 0 )</td>
<td>( \psi(u) )</td>
<td>( \psi(v) )</td>
</tr>
<tr>
<td>( u = x(2\pi-x)\sin(y) )</td>
<td>( v = \pi \sin(x) )</td>
<td>( \psi(u) )</td>
<td>( \psi(v) )</td>
</tr>
</tbody>
</table>

In particular, these results also show that the present solver would perform very well on linearized versions of the original geophysical fluid dynamic problem we were interested in ([16], [17]). We shall return to this point in Section 7.

### 5. Changes in Boundary Conditions.

In Section 2 we have formulated the model problem (2.1)–(2.3) which motivated this study. The algorithm of Section 3 has obvious applications to many other situations; it is of interest, therefore, to consider a number of different boundary conditions which could be associated with the Cauchy-Riemann equations (2.1) in a rectangle.

We shall assume throughout this section that the boundary conditions on \( y = 0, \pi \) are still (2.2), i.e., \( v \) is prescribed there as \( v^0(x) \) and \( v^1(x) \), respectively. The two different combinations of boundary conditions we consider explicitly are: (1) that \( u \) is given on the left boundary of the rectangle and \( v \) is given on the right boundary, and (2) that \( u \) is given on both vertical sides.

It is clear that if \( u \) is given on all sides, the problem should be formulated as Poisson's equation for \( v \) with Dirichlet boundary conditions; similarly if \( u \) is given on all the sides, a Dirichlet problem for \( u \) is more suitable. A moment's thought will show that the two situations we shall discuss can easily be transformed into a considerable number of others, by reflections or by interchanging the roles of \( x \) and \( y \), and of \( u \) and \( v \). In fact, all situations in which \( u \) as well as \( v \) are prescribed on some of the sides of the rectangle can be handled by slight modifications of the algorithms we present, yielding second-order accurate numerical solutions.
### Table VI

Numerical results indicating the resolution (mesh points per wavelength) required by the proposed discretization procedure and solution algorithm (versions C and F) to obtain prescribed accuracy.

| u = y sin(x) , v = y² sin(x) | M | t₂(U)/|u|₂ | t₂(V)/|v|₂ | t₁(U)/|u|₁ | t₁(V)/|v|₁ |
|-----------------------------|---|------|------|------|------|
| C 64                        | 6.6212 -4 | 1.3611 -4 | 8.6548 -4 | 8.6792 -5 |
| F 64                        | 5.1956 -4 | 1.3611 -4 | 6.9789 -4 | 8.6788 -5 |

| u = y sin(4x) , v = y² sin(4x) | F 64 | 8.444 -3 | 4.849 -3 | 1.47 -2 | 3.737 -3 |
| u = y sin(8x) , v = y² sin(8x) | F 64 | 3.1358 -2 | 2.721 -2 | 5.59 -2 | 1.88 -2 |
| u = y sin(64x) , v = y² sin(64x) | F 64 | 1.9450 +2 | 4.4180 +1 | 2.0070 +2 | 3.00 +1 |
| 128 | 8.0699 -1 | 4.4600 -3 | 5.6440 -1 | 1.42 -3 |
| 256 | 1.1422 -1 | 1.0867 -1 | 2.0238 -1 | 1.0347 -1 |

| u = y sin(x) sin(y) , v = sin(4x) sin(4y) | C 64 | 4.1000 -4 | 6.5580 -3 | 4.4520 -4 | 4.6550 -3 |
| u = y sin(4x) sin(y) , v = sin(8x) sin(8y) | C 64 | 3.0328 -4 | 1.7716 -3 | 3.3952 -4 | 2.0102 -3 |
| u = y sin(8x) sin(y) , v = sin(16x) sin(16y) | F 64 | 2.5786 -4 | 1.7716 -3 | 9.8728 -4 | 2.0102 -5 |
| u = y sin(16x) sin(y) , v = sin(32x) sin(32y) | F 64 | 1.0997 -1 | 1.2101 -1 | 2.0570 -2 | 5.1980 -2 |
| u = y sin(32x) sin(y) , v = sin(64x) sin(64y) | F 64 | 6.0271 -2 | 2.3781 -1 | 5.2271 -1 | 2.085 -1 |
| 128 | 8.0577 -1 | 1.2376 -1 | 5.4615 -1 | 1.5496 -1 |
| 256 | 1.1064 -1 | 1.1180 -1 | 7.6292 -2 | 1.5496 -1 |

| u = y⁴ sin(x) , v = x(2π-x) sin(y) | F 64 | 1.592 +1 | 6.956 +2 | 5.234 | 9.36 +2 |
| u = x(2π-x) sin(y) , v = y⁴ sin(x) | F 64 | 3.394 -2 | 5.270 -3 | 4.104 -2 | 3.0079 -2 |
| 128 | 1.5906 +1 | 2.6429 +2 | 5.3360 | 4.6962 -3 |
| 256 | 1.1070 -1 | 4.514 -2 | 5.5526 -1 | 8.1543 -2 |

| u = x(2π-x) sin(y) , v = x⁴(2π-x) sin(y) | C 64 | 1.121 -2 | 1.2937 -3 | 9.4755 -3 | 4.694 -4 |
| u = x(2π-x) sin(y) , v = x⁴(2π-x) sin(y) | F 64 | 4.9081 -3 | 1.2937 -3 | 9.4755 -3 | 4.694 -4 |
| u = x⁴(2π-x)² sin(y) , v = x⁴(2π-x)² sin(y) | F 64 | 1.785 -2 | 1.2937 -3 | 9.4755 -3 | 4.694 -4 |
| u = x⁴(2π-x)² sin(y) , v = x⁴(2π-x)² sin(y) | F 64 | 1.121 -2 | 1.2937 -3 | 9.4755 -3 | 4.694 -4 |
| u = x⁴(2π-x)² sin(y) , v = x⁴(2π-x)² sin(y) | F 64 | 1.785 -2 | 1.2937 -3 | 9.4755 -3 | 4.694 -4 |
| u = x⁴sin(3x) , v = x(2π-x) sin(3y) | F 64 | 5.2079 -2 | 1.1205 -1 | 3.6800 -2 | 1.6137 -1 |
| u = y⁴ sin(3x) , v = x(2π-x) sin(3y) | F 64 | 1.2851 -2 | 2.6395 -2 | 9.6328 -3 | 4.1549 -2 |
| u = y⁴ sin(5x) , v = x(2π-x) sin(5y) | F 64 | 3.0223 -2 | 6.4495 -3 | 2.4339 -3 | 1.0409 -2 |
| u = y⁴ sin(7x) , v = x(2π-x) sin(7y) | F 64 | 7.9989 -4 | 1.5969 -3 | 6.0900 -4 | 2.6059 -3 |
| u = y⁴ sin(9x) , v = x(2π-x) sin(9y) | F 64 | 1.9993 -4 | 3.9748 -4 | 1.5225 -4 | 6.5159 -4 |
| u = y⁴ sin(11x) , v = x(2π-x) sin(11y) | F 64 | 1.6604 -1 | 2.7604 -1 | 1.0791 -1 | 4.2728 -1 |
| u = y⁴ sin(13x) , v = x(2π-x) sin(13y) | F 64 | 3.8882 -2 | 5.9866 -2 | 3.1351 -2 | 9.6724 -2 |
| u = y⁴ sin(15x) , v = x(2π-x) sin(15y) | F 64 | 9.5680 -3 | 1.4350 -2 | 7.8489 -3 | 2.3872 -2 |
| u = y⁴ sin(17x) , v = x(2π-x) sin(17y) | F 64 | 2.3826 -3 | 3.5362 -3 | 1.9585 -3 | 6.0154 -3 |
| u = x(2π-x)² sin(y) , v = x(2π-x)² sin(y) | F 64 | 5.9508 -4 | 8.7915 -4 | 4.9629 -4 | 1.5020 -3 |
| u = y⁴ sin(3x) , v = x(2π-x) sin(3y) | F 64 | 3.7455 -1 | 5.7315 -1 | 2.7813 -1 | 6.7758 -1 |
| u = y⁴ sin(5x) , v = x(2π-x) sin(5y) | F 64 | 8.0104 -2 | 1.0879 -1 | 6.5915 -2 | 1.6856 -2 |
| u = y⁴ sin(7x) , v = x(2π-x) sin(7y) | F 64 | 1.3931 -2 | 1.9621 -2 | 1.6605 -2 | 4.0480 -2 |
| u = y⁴ sin(9x) , v = x(2π-x) sin(9y) | F 64 | 7.4843 -1 | 1.4103 -1 | 5.451 -1 | 1.396 -1 |
| u = y⁴ sin(11x) , v = x(2π-x) sin(11y) | F 64 | 1.3931 -1 | 1.7668 -1 | 1.7074 -1 | 2.3986 -1 |
| u = y⁴ sin(13x) , v = x(2π-x) sin(13y) | F 64 | 2.5673 -2 | 3.9733 -2 | 2.8390 -2 | 6.3575 -2 |
| u = y⁴ sin(15x) , v = x(2π-x) sin(15y) | F 64 | 8.0160 -3 | 9.6458 -3 | 7.1369 -3 | 1.5598 -2 |
| u = y⁴ sin(17x) , v = x(2π-x) sin(17y) | F 64 | 1.9963 -3 | 2.3893 -3 | 1.7829 -3 | 3.8817 -3 |

We proceed now with the description of the algorithm for the two cases mentioned.

**Case 1. u Given on the Left Side.** The rectangular domain is now taken as $R_1 = \{(x, y): -\pi/2 < x < 2\pi, 0 < y < \pi\}$. This is merely done for notational convenience, so as to leave Figure 1 unchanged. The boundary conditions are first (2.2),
which we repeat here as

\begin{align}
(5.1a) & \quad v = v^{(0)}(x), \quad y = 0, \\
(5.1b) & \quad v = v^{(1)}(x), \quad y = \pi,
\end{align}

and also

\begin{align}
(5.2a) & \quad u = u^{(0)}(y), \quad x = -h/2, \\
(5.2b) & \quad v = v^{(1)}(y), \quad x = 2\pi.
\end{align}

Thus Eqs. (5.2) replace the periodicity conditions (2.3a, b). Eqs. (2.1), together with (5.1), (5.2) completely determine \( u, v, \) and \( u \) need not and should not be prescribed any more at an interior point of \( R_1. \)

The difference equations are still (2.4), which we repeat for convenience as

\begin{align}
(5.3a) & \quad \left( U_{i,j} - U_{i-1,j} \right)/h + (V_{i,j} - V_{i,j-1})/k = D_{i,j}, \quad 1 \leq i \leq M, 1 \leq j \leq N, \\
(5.3b) & \quad \left( V_{i,j+1} - U_{i,j} \right)/k - (V_{i+1,j} - V_{i,j}) = E_{i,j}, \quad 1 \leq i \leq M, 1 \leq j \leq N - 1.
\end{align}

The boundary conditions become

\begin{align}
(5.4a) & \quad V_{i,0} = v^{(0)}((i - 1)A), \quad 1 \leq i \leq M, \\
(5.4b) & \quad V_{i,N} = v^{(1)}((i - 1)A), \quad 1 \leq i \leq M,
\end{align}

and

\begin{align}
(5.5a) & \quad U_{0,j} = u^{(0)}((j - 1/2)B), \quad 1 \leq j \leq N, \\
(5.5b) & \quad V_{M+1,j} = v^{(1)}((j)B), \quad 0 \leq j \leq N.
\end{align}

Hence, there are \( MN \) interior \( U \)-values to be determined, and \( M(N - 1) \) interior \( V \)-values, or \( M(2N - 1) \) unknowns altogether. Eqs. (5.3) yield \( M(2N - 1) \) linear algebraic equations; we shall see that these equations are actually independent and determine \( U, V \) completely.

It turns out to be more convenient in this case to form column vectors \( U_i, V_i \) from the values of \( U, V \) along vertical mesh lines; in Section 3 vectors \( U_j, V_j \) were formed along horizontal mesh lines. Thus

\[ U_i = (U_{i,1}, U_{i,2}, \ldots, U_{i,N})^*, \quad V_i = (V_{i,1}, V_{i,2}, \ldots, V_{i,N-1})^*; \]

in particular \( U_i, V_i \) have now different lengths, the \( U_i \)'s being \( N \)-vectors, while the \( V_i \)'s are \( (N - 1) \)-vectors. In a similar fashion, \( D_i \) is an \( N \)-vector, while \( E_i \) is an \( (N - 1) \)-vector of values along the corresponding vertical mesh lines.

With this notation, (5.3) becomes

\begin{align}
(5.6a) & \quad U_i - U_{i-1} + TV_i = hD_i, \quad 1 \leq i \leq M, \\
(5.6b) & \quad V_{i+1} - V_i + U_i = hE_i, \quad 1 \leq i \leq M;
\end{align}
here $T$ is an $N \times (N - 1)$ matrix of rank $(N - 1)$,

$$T = \rho \begin{bmatrix} 1 \\ -1 \\ \vdots \\ -1 \\ 1 \\ -1 \end{bmatrix},$$

and we define $\rho = h/k$. In (5.6a) $U_0$ is known from (5.5a), and in (5.6b) $V_{M+1}$ is known from (5.5b). Hence it is convenient to redefine $D_i$ as $D_i + h^{-1}U_0$, and $E_M$ as $E_M - h^{-1}V_{M+1}$. Furthermore, we redefine $D_{i,0}$ as $D_{i,0} + k^{-1}V_{i,0}$ and $D_{i,N}$ as $D_{i,N} - k^{-1}V_{i,N}$. After these changes of notation (5.6) can be given the block matrix form

$$\begin{bmatrix} M \\ M \end{bmatrix} T \left[ \begin{array}{c} T \\ \vdots \\ T \end{array} \right] = \begin{bmatrix} M \\ M \end{bmatrix} \begin{bmatrix} I_N \\ -I_N \end{bmatrix} \begin{bmatrix} V_1 \\ \vdots \\ V_M \\ U_1 \\ \vdots \\ U_M \end{bmatrix} = h \begin{bmatrix} D_1 \\ \vdots \\ D_M \\ E_1 \\ \vdots \\ E_M \end{bmatrix},$$

where $I_L$ is the $L \times L$ identity matrix.

The decoupling of $U$ and $V$ in this case proceeds as follows. In the upper half of system (5.8) we add the first block row to the second, then the second to the third, and so on, until the new $(M - 1)$st row is added to the $M$th. We obtain a new system, which we write in condensed form as

$$\begin{bmatrix} I \\ Q \\ R \end{bmatrix} \begin{bmatrix} V \\ U \end{bmatrix} = h \begin{bmatrix} F \\ E \end{bmatrix},$$

where

$$P = \begin{bmatrix} T & \vdots & \vdots & \vdots & T \\ \vdots & T & \vdots & \vdots & \vdots \\ \vdots & \vdots & T & \vdots & \vdots \\ T & \vdots & \vdots & T & T \end{bmatrix}$$
\[ Q = \begin{bmatrix} -I_{N-1} & I_{N-1} \\ \vdots & \ddots & \ddots \\ -I_{N-1} & \cdots & I_{N-1} \\ -I_{N-1} & \cdots \\ \end{bmatrix}, \]

\[ R = \begin{bmatrix} T^* \\ \vdots \\ T^* \end{bmatrix}, \]

\[ F = \begin{bmatrix} D_1 \\ D_1 + D_2 \\ \vdots \\ \sum_{1}^{M} D_i \end{bmatrix}, \]

and \( I \) is the \( MN \times MN \) identity matrix. From (5.9) we have

\[ (5.11a) \quad U = hF - PV, \]

\[ (5.11b) \quad (Q - RP)V = h(E - RF), \]

with

\[ (5.11c) \quad RP = \begin{bmatrix} S \\ S \\ \vdots \\ S \end{bmatrix}, \]

here \( S = T^*T \) is an \((N-1) \times (N-1)\) matrix,

\[ (5.12) \quad S = \rho^2 \begin{bmatrix} 2 & -1 \\ -1 & 2 & -1 \\ \vdots & \ddots & \ddots & \ddots \\ -1 & \cdots & -1 \\ -1 & 2 \end{bmatrix}. \]

Notice that \( S \), in contradistinction to \( S \) of Section 3, is nonsingular. We shall return to it later.

Written out explicitly, (5.11b) becomes after a change of sign,
A FAST CAUCHY-RIEMANN SOLVER

(5.13) \[
\begin{bmatrix}
S + I & -I \\
S & S + I & -I \\
\vdots & \ddots & \ddots & \ddots \\
S & \ldots & S & S + I \\
\end{bmatrix}
\]

\[V = h
\begin{bmatrix}
T*D_1 - E_1 \\
T*(D_1 + D_2) - E_2 \\
\vdots \\
T* \sum_M D_1 - E_M \\
\end{bmatrix}
\]

where \( I \) is now the \((N - 1) \times (N - 1)\) identity. This system can be brought into block-tridiagonal form simply by subtracting the \(i\)th row from the \((i + 1)\)st, starting from the top. This produces the system

(5.14) \[
\begin{bmatrix}
S + I & -I \\
-I & S + 2I & -I \\
\vdots & \ddots & \ddots & \ddots \\
-I & \ldots & S + 2I & -I \\
-I & S + 2I \\
\end{bmatrix}
\]

\[V = h
\begin{bmatrix}
T*D_1 - E_1 \\
T*D_2 + E_1 - E_2 \\
\vdots \\
T*D_M + E_{M-1} - E_M \\
\end{bmatrix}
\]

We see that the decoupling resulted in this case in the elimination of \( U \), rather than of \( V \). Equation (5.14) is rather similar to (3.11). Some of the differences have already been pointed out; as a result of these differences, the matrix of (5.14) is nonsingular. It can be brought to scalar tridiagonal form by diagonalizing \( S \) and then re-indexing. We shall comment on the fast solution of (5.14) further at the end of this section, together with the fast solution of the matrix equation obtained in the second case we wish to discuss.

**Case 2. \( u \) Given on Both Vertical Sides.** We fit the grid to the rectangular domain so that \( R_2 = \{(x, y): -\pi/2 < x < 2\pi - \pi/2, 0 < y < \pi\} \). The boundary conditions are (5.1) on the horizontal sides of the rectangle, and

(5.15a) \[ u = u(0)(y), \quad x = -\pi/2, \]

(5.15b) \[ u = u(1)(y), \quad x = 2\pi - \pi/2, \]

on the vertical sides. Equations (2.1), (5.1) and (5.15) determine \( u \) and \( v \) completely, subject to the requirement that the data \( d(x, y), u(0)(y), u(1)(y), v(0)(x), v(1)(x) \) satisfy the Gauss divergence theorem:

(5.16) \[
\int_{R_2} d(x, y) \, dx \, dy = \int_{-\pi/2}^{2\pi - \pi/2} \{v(1)(x) - v(0)(x)\} \, dx \\
+ \int_0^\pi \{u(1)(y) - u(0)(y)\} \, dy.
\]

The difference equations are (5.3), with (5.3b) only being written for \( 1 \leq i \leq M - 1, 1 \leq j \leq N - 1 \). The boundary conditions for the mesh variables are (5.4) and

(5.17a) \[ U_{0,j} = u(0)((j - 1/2)k), \quad 1 \leq j \leq N, \]

(5.17b) \[ U_{M,j} = u(1)((j - 1/2)k), \quad 1 \leq j \leq N. \]
Hence, there remain \((M - 1)N\) U-values to be determined, and \(M(N - 1)\) \(V\)-values, i.e., \(2MN - M - N\) unknowns. In (5.3) we have \(MN\) \(D\)-equations and \((M - 1)(N - 1)\) \(E\)-equations, i.e., \(2MN - M - N + 1\) equations in all.

It would seem that the number of equations exceeds the number of unknowns by one. We expect, however, from the continuous case that one compatibility condition has to be imposed on the data, analogous to (5.16), and that the matrix of system (5.3) have rank equal to the number of unknowns. This can be checked directly in the block form (5.18) to which we shall bring this matrix below; the compatibility condition turns out to be the one obtained when computing the integrals in (5.16) by the midpoint rule. A similar statement is true in the case in which \(u\) is prescribed on the horizontal sides of the rectangle, and \(v\) on the vertical sides; in that case the compatibility condition is the discrete analog of Stokes' curl theorem, involving \(e(x, y)\) rather than \(d(x, y)\).

We introduce the \(N\)-vectors \(U_i, D_i\) and the \((N - 1)\)-vectors \(V_i, E_i\) as in the previous case. Again, the first and last components of \(D_i, D_{i,0}\) and \(D_{i,N}\), are modified by the addition of \(\rho h^{-1}V_{i,0}\) and of \(-\rho h^{-1}V_{i,N}\), respectively. Also, \(D_1\) is modified by the addition of \(h^{-1}U_0\) and \(D_M\) by the addition of \(-h^{-1}U_M\).

After these changes of notation, system (5.3) becomes

\[
\begin{bmatrix}
  D_1 \\
  \vdots \\
  D_N \\
\end{bmatrix} = h
\begin{bmatrix}
  V_1 \\
  V_2 \\
  \vdots \\
  V_N \\
\end{bmatrix} + 
\begin{bmatrix}
  -E_1 \\
  \vdots \\
  -E_{N-1} \\
\end{bmatrix}
\]

(5.18)
where $T$ is the $N \times (N - 1)$ matrix defined by (5.7), and $I_{N-1}$, $I_N$ are identity matrices of the appropriate dimensions. Clearly, the sum of the $MN$ rows in the upper half of the matrix in (5.18) is zero. The corresponding compatibility condition that $\Sigma_{i,j} D_{ij} = 0$ is exactly the one we expected; we only need to remember that the $D_{ij}$ close to the boundary have been redefined to include the boundary data with appropriate coefficients.

The elimination of $V$ proceeds in a manner analogous to Case 1, by summing the blocks of the upper half of (5.18), in a discrete form of integration with respect to $x$. The result is

\begin{equation}
\begin{pmatrix}
T_{M-1} & 0 & I_N \\
T & T & \vdots \\
\vdots & \vdots & T & T & \vdots & \vdots \\
T & \cdots & T & T & 0 & \cdots & 0 \\
-\Sigma_{N-1} & I_{N-1} & T^* \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
-\Sigma_{N-1} & I_{N-1} & T^*
\end{pmatrix}
= h \begin{pmatrix}
V_1 \\
\vdots \\
V_{M-1} \\
V_M \\
U_1 \\
\vdots \\
U_{M-1} \\
-E_1 \\
\vdots \\
-E_{M-1}
\end{pmatrix} + \begin{pmatrix}
D_1 \\
D_1 + D_2 \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\Sigma_j M D_j
\end{pmatrix}.
\end{equation}

We rewrite this, with the obvious identifications, as

\begin{equation}
\begin{pmatrix}
P & I \\
T & 0 & \cdots & 0 \\
Q & R
\end{pmatrix}
= h \begin{pmatrix}
F \\
\vdots \\
\Sigma_j M D_j \\
-E
\end{pmatrix}.
\end{equation}

Notice that $P$ and $Q$ have block dimension $(M - 1) \times M$, $P$ has blocks of dimension $N \times (N - 1)$, and $Q$ has blocks of dimension $(N - 1) \times (N - 1)$.

From (5.20) we obtain

\begin{equation}
U = hF - PV,
\end{equation}

\begin{equation}
QV + RU = -hE.
\end{equation}
This allows us to eliminate $U$ and write

\[(5.22a) \quad (Q - RP)V = -h(E + RF),\]

\[(5.22b) \quad T^*(T \cdots T)V = hT^* \sum_i D_i;\]

here we introduced the block row missing in (5.21) as (5.22b). The elimination of the single redundant equation in (5.18) was done naturally by multiplication of (5.22b) with the $(N - 1) \times N$ matrix $T^*$.

System (5.22) becomes, after carrying out the matrix multiplications,

\[
\begin{bmatrix}
S + I & -I \\
S & S + I & -I \\
\vdots & \vdots & \ddots & \ddots \\
S & \cdots & S & S + I & -I \\
S & \cdots & S & S & S
\end{bmatrix}
\begin{bmatrix}
T^*D_1 + E_1 \\
T^*(D_1 + D_2) + E_2 \\
\vdots \\
T^* \sum_{i=1}^{M-1} D_i + E_{M-1} \\
T^* \sum_{i=1}^{M} D_i
\end{bmatrix}
= h
\]

with $S$ being the $(N - 1) \times (N - 1)$ matrix defined by (5.12). The matrix of this system is the same as that of (5.13), except for the lower right corner block; also the right-hand sides differ only slightly. Applying the block-tridiagonalization procedure used in Case 1, which corresponds to differencing in $x$, one obtains

\[
\begin{bmatrix}
S + I & -I \\
-1 & S + 2I & -I \\
\vdots & \vdots & \ddots & \ddots \\
-1 & S + 2I & -I \\
-1 & S + I
\end{bmatrix}
\begin{bmatrix}
T^*D_1 + E_1 \\
T^*D_2 + E_2 - E_1 \\
\vdots \\
T^*D_{M-1} + E_{M-1} - E_{M-2} \\
T^*D_M - E_{M-1}
\end{bmatrix}
= h
\]

We are now prepared to discuss the fast solution of (5.14) and of (5.24).

**Fast Sine Transform.** The fast solution of (5.14) and of (5.24) involves bringing the corresponding matrices to scalar tridiagonal form. This is done in two steps: the first and crucial step is to diagonalize $S$; the second is to bring the two diagonals which are identically $-1$ from the position of block subdiagonal and block superdiagonal to that of scalar sub and superdiagonal, i.e., immediately adjacent to the main diagonal.

We shall write the procedure for a slightly more general system, which includes (5.14) and (5.24) as special cases, to wit:
A FAST CAUCHY-RIEMANN SOLVER

\[
\begin{bmatrix}
S_1 + \alpha_1 I & \gamma_1 I \\
\beta_1 I & S_2 + \alpha_2 I & \gamma_2 I \\
& \ddots & \ddots & \ddots \\
& & \gamma_{M-1} I \\
& & \beta_{M-1} I & S_M + \alpha_M I
\end{bmatrix}
\]

(5.25) \[ W = B; \]

\[ W \text{ and } B \text{ are partitioned to conform with the blocks of the matrix,} \]

\[ W = (W_1^*, W_2^*, \ldots, W_M^*)^*, \quad B = (B_1^*, B_2^*, \ldots, B_M^*)^*, \quad \text{and } S_i = \delta_i S. \]

The eigenvalues \( \mu_k \) and eigenvectors \( \eta_k \) of \( S \),

(5.26a) \[ S \eta_k = \mu_k \eta_k, \quad 1 \leq k \leq N - 1, \]

are known:

(5.26b) \[ \mu_k = 2\rho^2 \left( 1 + \cos\left( \frac{\pi k}{N} \right) \right), \]

(5.26c) \[ \eta_{kl} = \left( \frac{2}{N} \right)^{1/2} \sin\left( \frac{\pi kl}{N} \right). \]

The matrix \( S \) differs from \( S \) of Section 3 inasmuch as its eigenvectors are generated by the sine function, rather than by the exponential, as in (3.12). Its diagonalization thus corresponds to the fast sine transform, in the same way in which \( S \) was connected to the FFT. The differences arise in the continuous problem because of the different boundary conditions.

Let \( P \) be the matrix whose columns are \( \eta_1, \eta_2, \ldots, \eta_{N-1} \), and let \( M = \text{diag}(\mu_1, \ldots, \mu_{N-1}) \). Then

(5.27) \[ P^* SP = M, \quad P^* P = I_{N-1}. \]

We introduce \( \tilde{W}_i, \tilde{B}_i \) by

(5.28a) \[ \tilde{W}_i = P^* W_i, \]

(5.28b) \[ \tilde{B}_i = P^* B_i, \quad 1 \leq i \leq M, \]

and \( M_i = \delta_i M \). We premultiply (5.25) by a block-diagonal matrix with all the diagonal blocks equal to \( P^* \) and use (5.27), (5.28) to yield

\[
\begin{bmatrix}
M_1 + \alpha_1 I & \gamma_1 I \\
\beta_1 I & M_2 + \alpha_2 I & \gamma_2 I \\
& \ddots & \ddots & \ddots \\
& & \gamma_{M-1} I \\
& & \beta_{M-1} I & M_M + \alpha_M I
\end{bmatrix}
\]

(5.29) \[ \tilde{W} = \tilde{B}. \]

Reindexing \( \tilde{W} \) and \( \tilde{B} \) into \( \hat{W} \) and \( \hat{B} \) by

(5.30) \[ \hat{W}_{k,i} = \tilde{W}_{i,k}, \quad \hat{B}_{k,i} = \tilde{B}_{i,k}, \quad 1 \leq i \leq M, \quad 1 \leq k \leq N - 1, \]

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
the scalar tridiagonization of (5.25) is completed in the form

\[
(5.31) \quad \begin{bmatrix} C_1 & & \\ & C_2 & \\ & & \ddots & \downarrow \\ & & & C_N \end{bmatrix} \hat{W} = \hat{B};
\]

here each \( C_k \) is a nonsingular scalar tridiagonal \( M \times M \) matrix,

\[
(5.32a) \quad C_k = \begin{bmatrix} \delta_1 \mu_k + \alpha_1 & \gamma_1 & \\ \beta_1 & \delta_2 \mu_k + \alpha_2 & \gamma_2 & \vdots & \\ & \ddots & \ddots & \ddots & \gamma_{M-1} \\ & & \beta_{M-1} & \delta_M \mu_k + \alpha_M \end{bmatrix},
\]

\[ 1 \leq k \leq N - 1. \]

After performing and storing the \( LU \) decomposition of the \( C_k \)'s, the solution of (5.25) is now carried out by a forward fast sine transform (5.28b) of the data \( \hat{B} \) into \( \hat{W} \), elimination and substitution in each subsystem

\[
(5.32b) \quad C_k \hat{W}_k = \hat{B}_k, \quad 1 \leq k \leq N - 1,
\]

and a backward fast sine transform (5.28a) of the solution \( \hat{W} \) into \( W \); the reindexing of \( \hat{B} \) into \( B \) and of \( \hat{W} \) into \( W \) does not necessitate actual computer operations. Furthermore, in our application,

\[
\beta_i = \gamma_i = -1, \quad \delta_i = 1, \quad 1 \leq i \leq M,
\]

and all \( \alpha_i \) are 2, except for \( \alpha_M = 1 \) in (5.24).

The operation counts and storage requirements are, therefore, the same as in Section 3; in particular, the same remarks apply to the computation of the sine transform as to the real Fourier transform, i.e., the practical computer time required is essentially determined by twice the number of real data.

Numerical experiments similar to those in Tables I through VI of Section 4 were carried out for Cases 1 and 2, with the algorithm described above. The results were entirely analogous, confirming the fact that the algorithm is second-order accurate and the computational time it requires is essentially proportional to the number of mesh points used in the discretization.

It is remarkable that the numerical tests still indicate second-order accuracy for \( u \in C^p, v \in C^q \) with \( p > 1, q > 0 \), and lower-order accuracy for \( u \in C^p, v \in C^q \) with \( p < 1, q < 0 \). This is true in both Case 1 and Case 2 for jump discontinuities in \( u \) and \( v \) or their derivatives introduced along either \( x = \text{const} \) or \( y = \text{const} \). These numerical observations seem to indicate that the method's second-order accuracy even for solutions without the formally required differentiability is not restricted to the case of
periodic boundary conditions. Furthermore, the asymmetry with regard to the continuity requirements on \( u \) and \( v \) does not appear to depend on whether \( U \) or \( V \) are eliminated in the algorithm, or on the boundary conditions imposed.

6. Comparison with Existing Solvers. A fast direct solver for the inhomogeneous Cauchy-Riemann equations was published by Lomax and Martin [24]. Some applications and extensions are given in [25], [26]. We carried out a comparison between our solvers and those published previously. The comparison was made first for the test case of [24], [26], then for some of the test cases in our Tables III through VI and similar ones. These computations were carried out on a CDC 6600 computer with a FTN 4.6 compiler; single precision arithmetic was used throughout.

Thin Biconvex Airfoil. The example problem used in [24], [26] to illustrate the use of a Cauchy-Riemann solver in aerodynamics is that of steady, irrotational, subsonic, inviscid flow over a thin symmetrical parabolic-arc biconvex airfoil in the small-perturbation approximation. The linearized Prandtl-Glauert transformation (e.g., [25]) yields for this problem the formulation

\[
\begin{align*}
(6.1a) & \quad u_x + v_y = 0, \\
(6.1b) & \quad u_y - v_x = 0, \quad y > 0, -\infty < x < \infty,
\end{align*}
\]

with boundary conditions

\[
\begin{align*}
(6.2a) & \quad v(x, 0) = -4x, \quad -0.5 < x < 0.5, \\
& \quad = 0, \quad 0.5 < |x|,
\end{align*}
\]

and

\[
\begin{align*}
(6.2b) & \quad u, v \to 0, \quad x^2 + y^2 \to \infty.
\end{align*}
\]

The analytical solution to (6.1), (6.2) is well known and contains a logarithmic singularity at the leading and trailing edges of the airfoil, \( x = \pm 0.5, y = 0 \). It is most easily written as

\[
(6.3) \quad w = \frac{4}{\pi} \left\{ 1 - x \log \frac{z + 0.5}{z - 0.5} \right\},
\]

with \( w = u - iv \) and \( z = x + iy \). The most important quantity one wishes to compute is \( u \) on the airfoil \( \{(x, y): y = 0, -0.5 < x < 0.5\} \); from it one can obtain the lift. The exact \( u \) there, given by (6.3), is simply

\[
(6.4) \quad u(x, 0) = \frac{4}{\pi} \left\{ 1 - \log \left| \frac{x + 0.5}{x - 0.5} \right| \right\}.
\]

In [24] the numerical computation is carried out in a rectangle \( R_3 = \{(x, y): 0 < y < 2, -1 < x < 1\} \), which we also did. Computations were performed in this rectangle prescribing the exact, analytic solution as boundary data on the three sides \( \{x = \pm 1\}, \{y = 2\} \), as well as prescribing homogeneous boundary conditions there. Clearly, it is a matter of choice which function, \( u \) or \( v \), is prescribed on the sides of
the rectangle not containing the airfoil. Hence, we used both the solvers of Section 5, Case 1, as well as Case 2.

Among the solvers of [26], we chose for comparison purposes version D, option (d). Version D is the one suggested by the authors themselves for the application at hand, and its option (d) seemed, of all the versions and options offered, to be the only one which could be second-order accurate. This version corresponds to our Case 1 in the choice of boundary conditions.

For the test case (6.1), (6.2) neither the norms of the solution over \( R_3 \), nor the order of accuracy of the solvers is particularly relevant, because of the singularities at the edges of the airfoil, \( x = \pm 0.5, y = 0 \). As a means of comparison we chose, therefore, the accuracy in computing \( u(x, 0) \). To compute \( U \) on \( y = 0 \), which is a \( V \)-line and not a \( U \)-line in the staggered mesh, [24] suggests the second-order accurate extrapolation formula

\[
U_{i,1/2} = (1/8) \{ 9U_{i,1} - U_{i,2} + 3(k/h)(V_{i,0} - V_{i+1,0}) - 3kE_{i,0} \};
\]

we used this formula in our program, as well as in theirs.

Because of the singularity at the edges, [24] did computations both with \( x = \pm 0.5 \) being \( V \)-lines, i.e., with computational mesh points coinciding with the edges, and with \( x = \pm 0.5 \) being \( U \)-lines, i.e., with the computational mesh straddling the edges. In Table VII we give results for the computation with both these meshes.

For each mesh, Table VII contains in successive columns the \( x \)-coordinates of the points along \( y = 0 \) at which \( U \) was calculated, the exact \( u \) there, the solution by the [26] solver, version D, option (d), and its error, the solution by our solver Case 1, its error, and finally the solution by our solver Case 2, with its error. We were only interested in comparing the different solvers, and did not wish to study the question of computing in a finite domain the solution to a half-plane problem; hence, only the comparison when using exact boundary data is given. The computations with homogeneous boundary data were carried out and gave slightly poorer results for all solvers.

For the mesh points coinciding with the edges, our Case 1 solver has a smaller error than the [26] solver at all points except four. The error, in particular, is smaller by a considerable factor over the airfoil \( |x| < 0.5 \), and it is smaller at the edges, where most of the error occurs. Our Case 2 solver has mostly smaller error than the Case 1 solver, but they are quite comparable.

For the computational mesh straddling the edges, our Case 1 solver seems to have mostly larger errors than [26] outside the airfoil, \( 0.5 < |x| \), but smaller errors inside. The Case 2 solver is still slightly better than the Case 1 solver, except at a few points.

As a conclusion, our solvers of Section 5, Case 1 and Case 2, are quite successful on the example problem of [24]. If anything, they have slightly smaller error than the [26] solver which appeared most promising. A better test would probably be to extract the singularities from the solution analytically (cf. the example in [13], for instance) and compute only the regular part of the solution. We felt, however, that [24] stressed the importance of computing the neighborhood of the singularities as accurately as possible, and made the comparison accordingly.
Numerical results for the $u$ component of velocity along $y = 0$ in the solution of the example problem (6.1), (6.2). The heading LM indicates results with the solver of [26], version D, option (d); headings GBuv and GBuu indicate results with the solvers of Section 5, Cases 1 and 2, respectively. The solutions were obtained on a $64 \times 64$ grid; every second point is given for reasons of space economy, except near the singularity at $x = \pm 0.5$, where every point is given; the "extra" points are marked by stars. The terms "coinciding mesh" and "straddling mesh" are explained in the text.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$u$ exact</th>
<th>LM error</th>
<th>GBuv error</th>
<th>GBuu error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-0.953125$</td>
<td>$-1.409-1$</td>
<td>$-1.141-1$</td>
<td>$1.962-4$</td>
<td>$-1.411-1$</td>
</tr>
<tr>
<td>$-0.890625$</td>
<td>$-1.666-1$</td>
<td>$-1.670-1$</td>
<td>$4.049-4$</td>
<td>$-1.670-1$</td>
</tr>
<tr>
<td>$-0.828125$</td>
<td>$-2.009-1$</td>
<td>$-2.016-1$</td>
<td>$6.647-4$</td>
<td>$-2.015-1$</td>
</tr>
<tr>
<td>$-0.765625$</td>
<td>$-2.486-1$</td>
<td>$-2.497-1$</td>
<td>$1.044-3$</td>
<td>$-2.496-1$</td>
</tr>
<tr>
<td>$-0.703125$</td>
<td>$-3.193-1$</td>
<td>$-3.210-1$</td>
<td>$1.703-3$</td>
<td>$-3.208-1$</td>
</tr>
<tr>
<td>$-0.640625$</td>
<td>$-4.341-4$</td>
<td>$-4.372-1$</td>
<td>$3.083-3$</td>
<td>$-4.370-1$</td>
</tr>
<tr>
<td>$-0.578125$</td>
<td>$-6.588-1$</td>
<td>$-6.640-1$</td>
<td>$5.230-3$</td>
<td>$-6.637-1$</td>
</tr>
</tbody>
</table>

**VIIA. Coinciding mesh**

**General Purpose Comparison.** Lomax and Martin developed their solvers [24], [26] with certain aerodynamical problems in mind [25], and we developed ours bearing in mind certain problems in geophysical fluid dynamics [15], [17]. On the other hand, the first Poisson solvers were also formulated for specific applications [4], [21], and only later developed into general purpose algorithms and into packages. Therefore, it seemed reasonable to test our solvers on solutions which had an appropriately general character (Tables III through VI, and discussion at the end of Section 5); these tests showed that our solvers are second-order accurate, given rather minimal continuity properties of the solution.
The intent of [24], [25], [26] was explicitly restricted to the solution of specific aerodynamic problems. It appeared worthwhile, however, to consider the more general applicability of their solvers.

We carried out a number of tests with version D, option (d) of [26] on solutions with a generic character. The results are given in Table VIII. This table is organized in the same way as Tables III and V of Section 4, and we refer to the description there.

For very simple test cases such as \( u, v \) constant or quadratic, the [26] solver is essentially exact to machine accuracy; the round-off error increases slightly with the number of grid points used. The difference between these results and those in Table III is due to the computer used: double-precision arithmetic on an IBM 360/95 is slightly more accurate than single precision on a CDC 6600.

The results with more severe test cases seem to indicate that the [26] solver is only first-order accurate in general. This is apparently due to the formulation of the algorithm at the boundaries. An indication that boundary inaccuracies are the cause of lower than second-order accuracy are those exceptional test cases which show high

<table>
<thead>
<tr>
<th>( x )</th>
<th>( u ) exact</th>
<th>( LM )</th>
<th>( LM ) error</th>
<th>( GB^{uv} )</th>
<th>( GB^{uv} ) error</th>
<th>( GB^{uu} )</th>
<th>( GB^{uu} ) error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.9375</td>
<td>-1.467-1</td>
<td>-1.467-1</td>
<td>-3.986-5</td>
<td>-1.467-1</td>
<td>-6.063-5</td>
<td>-1.467-1</td>
<td>-5.903-5</td>
</tr>
<tr>
<td>-0.8750</td>
<td>-1.763-1</td>
<td>-1.744-1</td>
<td>-1.054-4</td>
<td>-1.474-1</td>
<td>-1.741-1</td>
<td>-1.442-1</td>
<td></td>
</tr>
<tr>
<td>-0.8125</td>
<td>-2.114-1</td>
<td>-2.112-1</td>
<td>-2.127-4</td>
<td>-2.111-1</td>
<td>-2.765-4</td>
<td>-2.111-1</td>
<td>-2.717-4</td>
</tr>
<tr>
<td>-0.7500</td>
<td>-2.637-1</td>
<td>-2.633-1</td>
<td>-4.073-4</td>
<td>-2.632-1</td>
<td>-4.939-4</td>
<td>-2.632-1</td>
<td>-4.874-4</td>
</tr>
<tr>
<td>-0.6250</td>
<td>-4.753-1</td>
<td>-4.730-1</td>
<td>-2.236-3</td>
<td>-4.729-1</td>
<td>-2.237-3</td>
<td>-4.729-1</td>
<td>-2.262-3</td>
</tr>
<tr>
<td>-0.59375</td>
<td>-5.840-1</td>
<td>-5.793-1</td>
<td>-4.743-3</td>
<td>-5.791-1</td>
<td>-4.884-3</td>
<td>-5.791-1</td>
<td>-4.873-3</td>
</tr>
</tbody>
</table>

* The number in the * column is the number of grid points used.
Numerical results for general-purpose test cases using the solver of \[26\], version D, option (d). The smaller regions, such as \(R_4 = \{(x, y): 0 < x < \pi, 0 < y < \pi/2\}\) were used in some of the tests because of the difficulty of fitting computations with \(M = 256\) into the core memory of the CDC 6600; \(M\) is the actual number of grid points used in the \(x\)-direction in each test.

<table>
<thead>
<tr>
<th>(u, v) constant (R_4)</th>
<th>(u = 1000.0)</th>
<th>(v = -100.0)</th>
<th>(u = x^2 + y^2)</th>
<th>(v = 100)</th>
<th>(u = y \sin(x))</th>
<th>(v = \sin(y))</th>
<th>(u = y^2 \sin(x))</th>
<th>(v = \sin(y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(|M|)</td>
<td>(|H_\mathcal{U}|)</td>
<td>(|t_\mathcal{V}|)</td>
<td>(|t_\mathcal{U}|)</td>
<td>(|t_\mathcal{V}|)</td>
<td>(||t_\mathcal{U}|)</td>
<td>(||t_\mathcal{V}|)</td>
<td>(||t_\mathcal{U}|)</td>
<td>(||t_\mathcal{V}|)</td>
</tr>
<tr>
<td>16</td>
<td>1.083-11</td>
<td>1.642-12</td>
<td>7.747-12</td>
<td>2.910-11</td>
<td>4.547-12</td>
<td>8.640-12</td>
<td>1.001-10</td>
<td>2.871-11</td>
</tr>
<tr>
<td>64</td>
<td>1.001-10</td>
<td>1.387-10</td>
<td>1.210-10</td>
<td>4.475-10</td>
<td>2.310-10</td>
<td>3.583-10</td>
<td>1.999-10</td>
<td>2.164-10</td>
</tr>
<tr>
<td>128</td>
<td>1.999-10</td>
<td>2.177-10</td>
<td>2.090-10</td>
<td>1.281-9</td>
<td>3.174-10</td>
<td>1.122-9</td>
<td>1.122-9</td>
<td>3.156-10</td>
</tr>
<tr>
<td>16</td>
<td>3.076-12</td>
<td>1.537-12</td>
<td>2.431-12</td>
<td>8.640-12</td>
<td>4.547-12</td>
<td>6.257-11</td>
<td>1.999-10</td>
<td>2.871-11</td>
</tr>
<tr>
<td>32</td>
<td>1.360-11</td>
<td>2.533-11</td>
<td>2.033-11</td>
<td>5.731-11</td>
<td>4.957-11</td>
<td>5.731-11</td>
<td>1.999-10</td>
<td>2.033-11</td>
</tr>
<tr>
<td>64</td>
<td>7.589-11</td>
<td>1.376-10</td>
<td>1.111-10</td>
<td>3.583-10</td>
<td>2.287-10</td>
<td>3.583-10</td>
<td>1.999-10</td>
<td>2.164-10</td>
</tr>
<tr>
<td>128</td>
<td>1.706-10</td>
<td>2.164-10</td>
<td>1.949-10</td>
<td>1.122-9</td>
<td>3.156-10</td>
<td>1.122-9</td>
<td>1.122-9</td>
<td>3.156-10</td>
</tr>
</tbody>
</table>

Accuracy, apparently because their solutions close to the boundary behave in a special way in the variable perpendicular to it. The series of tests with increasing powers of...
TABLE VIII (Continued)

<table>
<thead>
<tr>
<th>LM</th>
<th>M</th>
<th>L₂(U)</th>
<th>L₂(V)</th>
<th>L₂(U,V)</th>
<th>L₆(U)</th>
<th>L₆(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>u,v ∈ C²</td>
<td>u = x sin(y)</td>
<td>v = y sin(x)</td>
<td>16</td>
<td>1.938-1</td>
<td>6.645-2</td>
<td>1.448-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>1.087-1</td>
<td>3.787-2</td>
<td>8.140-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>5.845-2</td>
<td>2.017-2</td>
<td>4.372-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td>3.049-2</td>
<td>1.041-2</td>
<td>2.278-2</td>
</tr>
<tr>
<td></td>
<td>16/32</td>
<td>1.782</td>
<td>1.754</td>
<td>1.779</td>
<td>1.493</td>
<td>1.502</td>
</tr>
<tr>
<td></td>
<td>32/64</td>
<td>1.860</td>
<td>1.878</td>
<td>1.862</td>
<td>1.592</td>
<td>1.689</td>
</tr>
<tr>
<td></td>
<td>64/128</td>
<td>1.917</td>
<td>1.939</td>
<td>1.919</td>
<td>1.662</td>
<td>1.795</td>
</tr>
</tbody>
</table>

| u,v ∈ C² | u = sin(y) | v = y² sin(x) | 16 | 2.733 | 6.771-1 | 1.991 | 1.055+1 | 4.096 |
|       |       |        | 32 | 1.787 | 4.29-1 | 1.300 | 8.858 | 3.456 |
|       |       |        | 64 | 1.033 | 2.413-1 | 7.501-1 | 6.367 | 2.401 |
|       |       |        | 128 | 5.604-1 | 1.281-1 | 4.065-1 | 4.193 | 1.505 |
|       | 16/32 | 1.529 | 1.578 | 1.532 | 1.191 | 1.185 |
|       | 32/64 | 1.730 | 1.778 | 1.732 | 1.391 | 1.439 |
|       | 64/128 | 1.843 | 1.884 | 1.845 | 1.519 | 1.595 |

| u,v ∈ C² | u = x² sin(y) | v = y² sin(x) | 16 | 4.381-1 | 1.319-1 | 3.235-1 | 1.738 | 6.385-1 |
|       |       |        | 32 | 2.571-1 | 7.747-2 | 1.899-1 | 1.227 | 4.722-1 |
|       |       |        | 64 | 1.429-1 | 4.213-2 | 1.054-1 | 8.254-1 | 3.017-1 |
|       | 16/32 | 1.704 | 1.697 | 1.704 | 1.416 | 1.350 |
|       | 32/64 | 1.798 | 1.845 | 1.802 | 1.487 | 1.565 |

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
Subject to further testing, we are forced to conclude at this point that our algorithms, compared to previously published ones, are at least as good when applied to specific problems of interest, and that they are more suitable for development into general-purpose Cauchy-Riemann solvers.

7. Concluding Remarks. The inhomogeneous Cauchy-Riemann equations in a rectangle have been discretized by a finite-difference approximation. A number of different boundary conditions have been treated explicitly, leading to algorithms which have overall second-order accuracy. All boundary conditions with either $u$ or $v$ prescribed along a side of the rectangle can be treated by similar methods. A rigorous proof of the second-order accuracy of the algorithm was given for one combination of boundary conditions, and numerical experiments substantiate this result for all the boundary conditions tested.

The algorithms presented here have nearly minimal time and storage requirements and seem suitable for development into a general-purpose direct Cauchy-Riemann solver for arbitrary boundary conditions. This could be done for instance along the lines of the capacitance matrix methods of discrete potential theory (Widlund [33]); generalizations to nonrectangular domains can also be made by this approach and related ones. More experience with different applications should help in formulating a code which gives a reasonable compromise between efficiency and range of applicability.

It is well known [30], [31], [32] that fast solvers can be formulated for a single separable second-order elliptic equation with variable coefficients. Clearly, the

### Table VIII (Continued)

<table>
<thead>
<tr>
<th>$u, v$ harmonic</th>
<th>$u = \cos(x) \sinh(y)$</th>
<th>$v = \sin(x) \cosh(y)$</th>
<th>$n/8$</th>
<th>0</th>
<th>$\pi/4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$u, v \to \frac{16}{32}$</td>
<td>1.789</td>
<td>1.775</td>
<td>1.787</td>
<td>1.434</td>
<td>1.539</td>
</tr>
<tr>
<td>$u, v \to \frac{32}{64}$</td>
<td>1.876</td>
<td>1.868</td>
<td>1.878</td>
<td>1.551</td>
<td>1.690</td>
</tr>
<tr>
<td>$u, v \to \frac{64}{128}$</td>
<td>1.928</td>
<td>1.943</td>
<td>1.930</td>
<td>1.630</td>
<td>1.783</td>
</tr>
</tbody>
</table>

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
same generalizations can be carried out for a first-order system (2.1) in which $\partial/\partial x$ and $\partial/\partial y$ are replaced by $a(x)\partial/\partial x$ and by $b(y)\partial/\partial y$, and in which lower-order terms can also be introduced. A special case of such an extension appears in [26]. Efficient algorithms for generalizations of this nature can be based on appropriate modifications and combinations of matrix decomposition, cyclic reduction and Toeplitz factorization [30], [31], [32].

Linear problems with variable coefficients, as well as nonlinear problems, can also be handled by semidirect methods, i.e., by splitting the given operator to be inverted into one whose inverse is easily computed using a fast direct method, and another one which is small in some suitable sense [8], [19], [25], [32]. It is in this direction that we shall seek to extend the work presented here, in order to solve the nonlinear problem of [15], [17] and related ones. The straightforward iterative method of [17] will then be compared to semidirect methods.

**Appendix A. An Error Estimate.** We saw in Sections 4 and 5 that the proposed algorithm gives a second-order accurate solution to the inhomogeneous Cauchy-Riemann equations under the various boundary conditions considered. The second-order accuracy of the discrete solutions was obtained in our numerical experiments even for cases in which the solution to the continuous problem was merely $C^1$, more precisely, $u \in C^1$, $v \in C^0$. We shall give now a rigorous error estimate for the model problem of Section 2, making the stronger and more customary assumption that $u$, $v \in C^4$, i.e., that the solution to (2.1)–(2.3) has continuous derivatives up to fourth order.

The discrete operator $L_h$ we consider is that of (3.1a),

\[
L_h = \begin{bmatrix}
S + I & -I \\
-I & S + 2I & -I \\
& \ddots & \ddots & \ddots & \ddots \\
-2I & S + 2I & -I \\
-I & S + I \\
\end{bmatrix} + \alpha e e^*,
\]

where $\alpha > 0$ and $e$ is a vector of length $MN$ with all entries zero but one,

\[e_l = 0 \quad \text{for } l \neq M j_0 + i_0, \quad e_{M j_0 + i_0} = 1;\]

$L_h$ acts on the grid function $U$. We wish to show that

(A.2a) \[\|u - U\|_\infty = O(h^2 + k^2).\]

From such an estimate and from (3.1a) it will follow immediately that

(A.2b) \[\|v - V\|_\infty = O(h^2 + k^2)\]

as well; it suffices to observe that (3.1a) is a second-order accurate quadrature formula for

(A.3) \[v(x, y) = v^{(0)}(x) + \int_0^y \{d(x, \eta) - u(x, \eta)\} \, d\eta,\]

namely the midpoint rule. We only need to apply the result of Bramble and Hubbard [2] that (A.2a) implies similar estimates for the difference quotients of $u$. 

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
The matrix operator $L_h$ of (A.1) is of monotonic type (Collatz, [6, p. 42 ff.]) or of positive type with diagonal dominance (Forsythe and Wasow [11, p. 181]). To avoid confusion in the terminology, we shall simply state that $L_h$ satisfies the following

**Lemma 1 (Maximum Principle).** If $L_h W = H$, and $H \geq 0$ (in the sense that all the components of $H$ are nonnegative), then $W \geq 0$.

From this, one easily shows that

**Lemma 2 (Comparison Theorem).** If $|L_h W| \leq L_h \Phi$, then $|W| \leq \Phi$.

We notice that these results would still hold if, instead of $\alpha e e^*$, we included in $L_h$ a term $\alpha A$, with a single diagonal block equal to $I_M$, and all other blocks zero, $I_M$ being the $M \times M$ identity matrix.

These properties of $L_h$ suggest the familiar estimation procedure first used by Gershgorin [13]. Let $u$ be the solution of

\[(A.4a) \quad Lu = f,\]

where $L$ and $f$ are defined by the two equations

\[(A.4b) \quad \Delta u = u_{xx} + u_{yy} = d_x + e_y \quad \text{in} \ R,\]

\[(A.4c) \quad u_y = e + v_x \quad \text{on} \ \partial R \equiv \{(x, y): 0 < x < 2\pi, y = 0, \pi\},\]

with the additional requirements that $u$ be $2\pi$-periodic in $x$, and given at a point $(x_0, y_0) \in R$. For sufficiently smooth $d$, $e$ and $v^{(0)}$, $v^{(1)}$, (A.4) has a unique solution $u \in C^4$, subject to the familiar compatibility condition for the Neumann problem that

\[(A.4d) \quad \int_R (d_x + e_y) \, dx \, dy = -\oint_{\partial R} (e + v_x) \, dx.\]

We shall make the necessary smoothness and compatibility assumptions throughout this Appendix.

Let $U$ be the solution of

\[(A.5a) \quad L_h U = F,\]

where $L_h$ is defined by (A.1) and $F$ is defined as the right-hand side of (3.11a),

\[(A.5b) \quad F = k \begin{bmatrix} T^*D_1 - E_1 \\ T^*D_2 + E_1 - E_2 \\ \vdots \\ T^*D_{N-1} + E_{N-2} - E_{N-1} \\ T^*D_N + E_{N-1} \end{bmatrix} + \alpha e e^* U,\]

with $F_j$, $1 \leq j \leq N$, corresponding in obvious fashion to the partition of $L_h$. Clearly, for $2 \leq j \leq N - 1$, (A.5) is an approximation to (A.4b) with second-order local truncation error. This would further encourage us to seek an estimate (A.2a) by first showing that the truncation error satisfies

\[(A.6) \quad L_h(u - U) = (L_h - L)u + f - F = O(k^2(h^2 + k^2))\]

and then constructing a Gershgorin comparison function $\Phi$ which would allow us to
conclude based on Lemma 2 that (A.2) follows from (A.6). We remark at this point that, in order to make the notation of (A.6) transparent, we defined

\[(A.7a) \quad L = -k^2 \Delta \quad \text{in } \mathbb{R}, \quad L = -k \partial / \partial y \quad \text{on } \partial \mathbb{R}, \]

\[(A.7b) \quad f = -k^2 (d_x + e_y) \quad \text{in } \mathbb{R}, \quad f = -k (e + v_x) \quad \text{on } \partial \mathbb{R}; \]

furthermore, \( F \) is not merely \( f \) at the grid points.

A number of slight difficulties arise in carrying out the program above. First, (A.4) is essentially a Neumann problem, rather than a Dirichlet problem as in [13] and in most of the literature on elliptic systems. The work on boundary conditions of the second and third kind most relevant to the estimation which follows is that of Batschlet [1], who gives an \( O(h) \) estimate, and that of Bramble and Hubbard [3], who give an \( O(h^2 |\log h|) \) estimate, their assumptions being that \( u \in \mathcal{C}^4 \). The latter article also contains further references to estimates for the Neumann problem. The boundary condition approximations in these works are different from ours.

Second, (A.6) actually fails close to the boundary, i.e., for \( j = 1, N \), where the local truncation error is of first order only. The work of Bramble and Hubbard ([3] and references therein), combined with our numerical results, led us to expect that (A.2) could still be proved, with some additional effort; this turned out to be the case.

Our method to obtain (A.2) is actually a modification of that in [1], which exploits the fact that the boundary condition we use is second-order accurate, while that in [1] is only first-order. We shall see below that the failure of (A.6) for \( j = 1, N \) stems from \( L_h \), there being a linear combination of the discrete analog to (A.4b) and of that to (A.4c). Hence the truncation error, as defined by (A.6), is of first order there, although both (A.4b) and (A.4c) are separately approximated to second order. In spite of this formally first-order truncation error, the fact that both the equation and the boundary condition are approximated by second-order discrete analogs yields an overall \( O(h^2 + k^2) \) error estimate (A.2).

After these observations, we proceed with the business at hand. To start, we derive (A.6) for \( 2 \leq j \leq N - 1 \); let the discretization error \( W \) be defined as

\[(A.8a) \quad W = u - U, \]

considered as a mesh function. Then

\[(L_h W)_j = -W_{j-1} + (S + 2I)W_j - W_{j+1} \]

\[= -u_{j-1} + (S + 2I)u_j - u_{j+1} + k^2 (\Delta u)_j \]

\[= -k^2 (d_x + e_y) - k(T^* D_j + E_{j-1} - E_j) \]

\[= O(k^2 (h^2 + k^2)), \quad 2 \leq j \leq N - 1. \]

In (A.8b), and in the sequel, \( u_j \), \( (\Delta u)_j \), and similar terms are interpreted as vectors of grid values, in the same way as \( U_j \). The factor \( k^2 \) is convenient in order to keep the coefficients of \( L_h \) as \( O(1) \). The terms \( O(h^2 k^2) \) appear due to differentiation in the \( x \) direction, those \( O(k^4) \) due to differentiation in the \( y \) direction. The constants implied
in writing $O(h^p k^q)$ depend on the derivatives of order $p + q$ for the functions involved; they will not be written down explicitly, since those derivatives are known to be bounded from our assumptions.

Here, as in the derivation of (A.2b) from (A.3) and as in the sequel, it is important to remember the staggering of Figure 1, which essentially guarantees that all differences are centered. Notice also that, for $e_i 
eq 0$, (A.8b) will only be modified by the term $\alpha e_i e_i^* u_i$ in $L_h u$, and by the term $\alpha e_i e_i^* U_i$ in $F$; the condition we chose to eliminate the indeterminacy in the Neumann problem is exactly that these two terms cancel, and hence the estimate is not affected. If instead of $e e^*$ we have $A$, the same observation holds as after Lemma 2; i.e., prescribing the average of $U_i$, rather than one of its components, $U_i$, does not affect the estimate either.

To analyze the situation for $j = 1$, it is convenient to rewrite (A.5) for $j = 1$ as a linear combination of two equations, involving an auxiliary vector $U_0$; an entirely similar procedure can be carried out for $j = N$, introducing $U_{N+1}$, but we shall omit writing out the latter analysis and merely draw the conclusions we need from it. The two equations for $j = 1$ are

(A.9a) $-U_0 + (S + 2I)U_1 - U_2 = k(T^* D_1 + E_0 - E_1)$,

(A.9b) $U_0 - U_1 = -kE_0 + T^* V_0$;

here we revert to the original definition of $D_1$, which had been replaced by $D_1 + k^{-1} V_0$ in writing Eq. (3.3). Equation (A.9a) is now simply the discrete analog of (A.4b) for $j = 1$, or $y = k/2$, with $E_0$ defined on $y = 0$ in the usual manner. Equation (A.9b) is the discrete analog of (A.4c) written on $y = 0$. If the auxiliary vector $U_0$ is thought of as given on $y = -k/2$, then both (A.9a) and (A.9b) are formally second-order accurate.

With this motivation in mind, it is easy to obtain

\[(L_h W)_{1} = (S + I)W_1 - W_2 = (S + I)u_1 - u_2 + k^2 \Delta u|_{y=k/2} + k(\partial/\partial y)u|_{y=0}\]

(A.10a) $-k^2(d_x + e_y)|_{y=k/2} - k(T^* D_1 + E_0 - E_1)$

$\equiv [(L_h - L)u]_1 + [f - F]_1$.

We have from (A.10a) that

$[(L_h - L)u]_1 = O(k^2 (h^2 + k))$, \quad $[f - F]_1 = O(k^2 (h^2 + k^2)) + O(kh^2)$.

We assume throughout that $k/h = O(1)$ and drop the fourth-order terms; thus finally

(A.10b) $(L_h W)_1 = O(k (h^2 + k^2))$.

For $j = N$ we obtain in the same way
At this point we have to exhibit a comparison function $\phi$ which shall lead us from (A.8) and (A.10) to (A.2). Let $\psi$ be the solution of

(A.11a) \[ L\psi = k^2 a \quad \text{in } R, \]

(A.11b) \[ L\psi = \begin{cases} -kab & \text{on } y = 0, \\ k\beta b & \text{on } y = \pi. \end{cases} \]

$L$ is defined by (A.7), and $\psi$ is $2\pi$-periodic in $x$ and fixed at one point; this yields a unique and smooth $\psi$, which is bounded independently of $x$, $y$, $h$ and $k$. Notice that $\psi$ satisfies inhomogeneous boundary conditions as in [1]. The constants $a$, $b$, $\alpha$ and $\beta$ are positive and chosen so as to satisfy the compatibility condition

(A.11c) \[ \int_R \Delta \psi \, dx \, dy = \int_0^{2\pi} \{ \psi_y(x, \pi) - \psi_y(x, 0) \} \, dx, \]

that is, $\pi a = (\alpha + \beta)b$.

We want to find $\eta$ independent of $x$, $y$, but not of $k$, $h$, so that

(A.12a) \[ \phi = m\psi \]

and

(A.12b) \[ |L_n W| \leq L_n \Phi, \]

where $W$ is given by (A.8a) and $\Phi$ is the mesh function corresponding to $\phi$. Lemma 2 will then provide the desired estimate on $W$ in terms of $m$.

We consider first the "interior" mesh domain of $U$-points $R_h = \{(x_i, y_j): 1 \leq i \leq M, 2 \leq j \leq N - 1\}$. There

(A.13a) \[ L_n \phi = L\phi + (L_n - L)\phi = mk^2 a + mO(k^2(h^2 + k^2)), \]

as in (A.8b). For $h$, $k$ sufficiently small, with $h/k = O(1)$,

\[ L_n \phi \geq mk^2 a/2. \]

It suffices, therefore, given some sufficiently large constant $C_R$, to have set

(A.14a) \[ m = C_R h^2 \]

for (A.12b) to hold in $R_h$; $C_R$ will depend on certain bounds for the derivatives of $u$.

Consider now the "boundary" mesh domain of $U$-points $\Gamma_h = \{(x_i, y_j): 1 \leq i \leq M, j = 1, N\}$. Here, cf. (A.9, 10), we have

(A.13b) \[ (L_n \phi)_1 = (S + l)\Phi_1 - \Phi_2 = k^2 \Delta \phi|_{y=k/2} + k\phi_y|_{y=0} + O(k^2(h^2 + k)) \]

\[ = -mk^2 a + mkab + O(k^3); \]

a similar equation holds for $(L_n \Phi)_N$. Thus, for $h$, $k$ sufficiently small, we have in $\Gamma_h$

\[ L_n \phi \geq mkab/2; \]

remembering again (A.10), it suffices here too that
(A.14b) \[ m = C_T h^2, \]
for (A.12b) to hold. The constant \( C_T \) depends only on the derivatives of \( u \), and on the domain \( R \).

In fact, one can write down \( \psi = \psi(x, y; a, b, \alpha, \beta) \) explicitly and optimize the parameters \( a, b, \alpha, \beta \) subject to (A.11c). This yields

(A.15a) \[ |W| \leq \frac{\pi}{2} h^2 \max \left\{ \frac{\pi}{2} C_R, C_T \right\}, \]
where

(A.15b) \[ C_R = \frac{1}{24} \sup_{0 < x < 2\pi, 0 < y < \pi} \left| 2(u_{xxxx} + u_{yyyy}) - (d_{xxx} + e_{yyy}) \right|, \]
and

(A.15c) \[ C_T = \frac{1}{24} \sup_{0 < x < 2\pi, y = 0, \pi} \left| u_{yyyy} + v_{xxxx} \right|. \]

The bound \( e(u) \) in Table V of Section 4 was computed using (A.15).

Having derived bound (A.15) completes the proof that under our assumptions
\[ \|u - U\|_\infty + \|v - V\|_\infty = O(h^2). \]

It might be of interest to notice that the algorithm proposed and the error estimate just given apply not only to the inhomogeneous Cauchy-Riemann equations (2.1), (2.2); they apply directly to the Neumann problem (A.4), with \( d_x + e_y, e + v_x \) arbitrarily prescribed as well. In other words, this is also a second-order algorithm for the Poisson equation with Neumann boundary conditions in a rectangle (compare Schumann and Sweet [29]).

Appendix B. The Basic Algorithm. At the end of the paper we present a FORTRAN listing of the program used to compute the results in Tables I through VI. The program implements the algorithm described in Section 3, tested in Section 4, and analyzed in Appendix A. It solves the inhomogeneous Cauchy-Riemann equations for \( u \) and \( v \) in a rectangle, with \( v \) prescribed on the upper and lower side of the rectangle, and with periodicity in the \( x \)-direction. The programs for other combinations of boundary conditions, as discussed in Section 5, are very similar.

The program was tested for compatibility with ANSI FORTRAN (U.S.A. Standard X3.9—1966) by a special diagnostic option of the CDC FTN 4.6 compiler; it was further cleaned up to conform to common usage by a program called TIDY, implemented at and available from the Courant Institute. It was run on a CDC 6600 with an FTN 4.6 compiler, and on an IBM 360/95 and an Amdahl 470V/6 with a FORTRAN IV, level H compiler. It is felt that these procedures and the numerical tests reported on in the text are some reasonable steps in the direction of validation and portability; further steps in this direction taken by potential users would be of the greatest interest to the authors.
The main part of the program is subroutine FASTCAR. For user convenience, the driver subroutine, as well as the FFT subroutine and the tridiagonal solver we used are included. Clearly, these subroutines can be changed to others which the user for one reason or another finds are better adapted to his needs or preferences; e.g., cyclic reduction can be used instead of the FFT, or Toeplitz factorization instead of the LU factorization we use for the scalar tridiagonal systems, etc. The use of a more general FFT algorithm would remove the restriction of \( M \) being a power of two.

The driver program contains some diagnostics on error norms, which would have to be modified for problems whose solution is not known in closed form. FORMAT statements are grouped together and can thus easily be modified. Execution speed can be increased if \( MN \) additional storage locations are available, by avoiding the shifting required for the reindexing process within the same array. Other trade-offs between storage and execution time are also possible.

The implementation follows rather closely the algorithm description in the last subsection of Section 3. With the help of the COMMENTS provided in the program, we hope that it is fairly readable.

Questions and comments by users are warmly invited.
A FAST CAUCHY-RIEMANN SOLVER

1. SETTING UP R.H.S. OF THE INHOMOGENEOUS CAUCHY-RIEMANN EQUATIONS.
   - E IS THE R.H.S. OF THE VORTICITY EQUATION
   - D IS THE R.H.S. OF THE DIVERGENCE EQUATION
   
2. E(K) = DUDY(X1, Y1) - DVDX(X1, Y1)
   - D(K) = DUDX(X2, Y2) + DVDY(X2, Y2)

3. NEXT STORE THE BOUNDARY VALUES V(TOP) IN VN(I).
4. NEXT STORE THE BOUNDARY VALUES V(BOTTOM) IN VO(I).

5. DO 20 I = 1, M
   - XZ'XJV
   - V0(I) = VEX(XJU(I), YU)
   - VN(I) = VEX(XJV(I), YL)
   20 CONTINUE

6. NEXT STORE THE FIXED U VALUE IN SOL.
   - IROW IS THE ROW WHERE FIXED VALUE U IS ASSIGNED.
   - SOL = 0.0
   - IROW = N
   - Y1 = G*(2*(IROW-1)+1) + YL
   - DO 30 J = 1, M
     - X1 = XJU(J)
     - SOL + SDL + UEX(X1, Y1)
   30 CONTINUE

7. CALL FASTCR(SOL, H, G, PI, IROW, MPOW, ISAM, ICDE, N)

8. NEXT FINDING NORMS.
   - XN IS THE L2 NORM OF U.
   - YN IS THE L2 NORM OF V.
   - XYN IS THE L2 NORM OF (U+V).
   - XM IS THE MAX NORM OF U.
   - YM IS THE MAX NORM OF V.

9. YM = 0.0
   - YN = 0.0
   - DO 40 J = 1, M
     - XZ'XJV
     - DO 40 I = 1, N
       - Y2 = YK(I)
       - K = (1-1)*M + J
       - DKKK = ABS(E(K)) - UEX(X2, Y2))
       - YN + DKKK
       - IF (DKKK .GT. YM) YM = DKKK
   40 CONTINUE

10. XN = 0.0
    - XM = 0.0
    - DO 50 J = 1, M
      - X1'XJU(J)
      - DO 50 I = 1, N
        - YL = YK(I)
        - K = (1-1)*M + J
        - EKKK = ABS(E(K)) - UEX(X1, Y1))
        - XM + EKKK
        - IF (EKKK .GT. XM) XM = EKKK
    50 CONTINUE

11.WRITE (6,120)
12. WRITE (6,110) XN, YN, XYN, XM, YM

13. W(1,M) = XN
    - W(2,M) = YN
    - W(3,M) = XYN
    - W(4,M) = XM
    - W(5,M) = YM
SUBROUTINE FASTCR (SOL,DELTAX,DELTAY,pI,IRUW,MPGW,iSAM,ICODU,N)
COMMON /HAND/ E(o 292),D(B292)
COMMON /DRY/ X(6*),Y(6*1
COMMON /WET/ A(126,2)

SOLVES INHUMOGENOUS CAUCHY-RIEMANN EQUATIONS IN A RECTANGLE
U + V  = D(X,Y) *
X   Y WITH 3.C.  V(x,0) , V(X,YMAX)    ♦
U - V  = E(X,Y) *
Y   X AND A FIXED VALUE OF U ASSIGNED  ♦

THE R.H.S. E(X,Y) , D(X,Y) ARE STORED IN *
E(M*J+1) = E( DELTAX*(2*l-l) , JElTAY*(2*J+2) ) ♦
FOR  J = 0...N-2 ,  I » 1...M *
D(M*J+I) = D( DELTAX*2*(I-1) DELTAY*(2*J+2) ) ♦
FOR  J = 0...N-l ,  I • 1...M *

THE BOUNDARY CONDITIONS ARE STORED IN *
A(I,1) = VlDELTAX*I,0) ,A(I,2) = V(utLTAX*I,D.tLTAY»N)      *
FOR    I • 1...M *

SOLUTIONS  U  AND  V ARE  RETURNED IN *
E(M*J+I) = U( DELTAX*(2*I-1) , OtLTAY*(2♦J+l) ) ♦
FOR  J = 0...N-2 ,  I » 1...M *
D(M*J + I) = V( DELTAX*2*(I-1) , C)ELTAY*<2*J+2) ) ♦
FOR  J = 0...N-2 ,   1 • 1..,M *

N • NUMBER OF MESH POINTS IN Y *
2**MP0W  = NUilBER OF MtSH POINTS IN X *

I************************************************************
C M=2**MP0W
M=2**MP0W
M=2**MP0W
M=2**MP0W

A FAST CAUCHY-RIEMANN SOLVER

MHALFP=M8Y2+1
MPLUS2=M+2
IL=ILN
IL=IL-M
ALAM=DELTAY/DELTAX
ALSO=ALAM*ALAM

C SETTING UP THE R.H.S. OF THE LINEAR SYSTEM

G2=2.0*DELTAY
D0 10 K=1,ILM
E(K)=E(K)*G2
D0 20 I=1,M
KP=I+ILM
D(KP)=D(KP)*G2-A(I,2)
O(I)=O(I)+A(I,1)
20 A(I,1)=G,0
IP=1
IS=2
DO 40 J=1,ILM,M
KM=J-1
DO 30 I=1,M1
KI=KM+I
A(K,I)*E(K1)
30 E(K1)=ALAM*(D(K1+1)-D(K1))+E(K1)-A(I,IP)
KMMP=K1+1
A(KMMP)=CKMMP
E(KMMP)=ALAM*(D(KM+1)-D(KMMP))+E(KMMP)-A(M,IP)
IEX=IP
IP=IS
40 IS=IEX
DO 50 I=1,M
KP=I+ILM
DO 30 K=1,M
KL=K-1
DO 40 J=1,M
A(I,JS)=E(K1)
40 E(K1)=ALAM*(D(K1+1)-D(K1))+E(K1)-A(I,IP)
E(IL)=ALAM*(D(ILM+1)-D(IL))+E(IL)-A(F1,IP)

C PERFORMING THE FAST FOURIER TRANSFORM BLOCK BY BLOCK

SIG=1.0
DO 80 K=1,N
JF=(K-1)*M
DO 60 I=1,M
A(1,2)=E(J)
60 A(1,2)=0.0
CALL FFT2 (SIG,MP0W,PI)
DO 70 I=1,M
IQF=JF+I*2
K1(K1)+1.0
70 E(IQF)=A(I,1)
80 Y(K1)=XMHALFP,1

C SOLVING EACH SUB-SYSTEM

DO 120 J=2,MBY2
IF (ICODE.EQ.3) GO TO 100
X1=-2.0*ALSO*(COS(2.0*PI*FLOAT(J)-1.0)*M)-1.0
A(1,1)+X1+1.0
A(N,1)=X1+1.0
DO 130 K=2,N
100 A(K,K)+1.0
I=(K-1)*M+2*J
IQF=K+N
A(K,2)=E(I)
110 A(IQF,2)=E(I)
CALL CONSOL (N,NEXT,ICODE)
DO 120 K=1,N
I=(K-1)*M+2*J
E(I)=A(K,2)
IQF=K+N
120 E(I)=A(IQF,2)
IF (ICODE.EQ.3) GO TO 140
X1=-2.0*ALSO*(COS(2.0*PI*FLOAT(J)-1.0)*M)-1.0
A(1,1)+X1+1.0
A(N,1)=X1+1.0
DO 130 K=2,N
130 A(K,1)=X1
140 DD 150 J=1,N
A(J,2)=X(J)
IQF=J+N

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
MICHAEL GHIL AND RAMESH BALGOVIND

CALL CONSOL (N,NEXT,IC0DL)
DO 160 J=1,N
IQF=N+J
X(J)=A(J,1)
160 Y(J)=A(I0F,2)
IF (ICODE.EQ.3) GO TO 1B0
X(N+1)=1.0
DO 170 K=2,N
170 X(K)=2.0
DO 180 K=1,N
I=(K-1)*M+1
A(K,2)=E(I)
IQF=N+K
A(IF,2)=E(1+1)
A(IR0W,1)=A(IR0W,1)-1.0
A(IR0W,2)=A(IR0W,2)-SOL
CALL CONSOL (N,NEXT,IC0DL)
DO 200 K=1,N
I=(K-1)*M+1
E(I)=A(K,2)
IQF=N+K
200 E(I+1)=A(I0F,2)

FAST FOURIER TRANSFORM OF SOLUTION U

SOLUTION V WILL BE IN ARRAY D AND U WILL BE IN E

DO 230 K1=1,N
IF (ISAM.EQ.0) GO TO 300
IF (ISAM.EQ.-1) GO TO 260
DO 250 K=1,N1
KM=M*(N-KD+1)
KJ=KMP+J
A(KJ+1)=ALAM*(E(KJ)-E(KM))+A(KJ)
DO 240 J=1, M
J*KMMP+J
D(J)=D(J)+A(I1)
IF (K11.LT.1) GO TO 250
K11=J-M
D(I1)=D(I1)+D(K11)
250 CONTINUE
DO TO 300
260 DO 270 I=1,M
J=I+ILM
270 A(I1)=D(J)
I1=1
IP=2
DO 290 KD=1,N1
II=E11+IS
IS=IP
IP=IP+1
KMMP=KM-M
D(J1)=D(J1)+A(I1)
IF (K11.LT.1) GO TO 250
K11=J-M
D(I1)=D(I1)+D(K11)
250 CONTINUE
DO TO 300
A FAST CAUCHY-RIEMANN SOLVER

K11=J-M
A(I,IS)=O(K11)
D(K11)=A(I,IP)
IF (KD.GT.1) D(K11)=D(J)+D(K11)
290 CONTINUE
300 CONTINUE
RETURN
END

SUBROUTINE FFT2 (SIG,M,PI)
COMMON /WET/ X(126),Y(126)

FAST FOURIER TRANSFORM
N=2*M
NV2=N/2
NM1=N-1
J = 1
DO 30 1=1,NM1
IF (I.GE.J) GO TO 10
TX=X(I)
TY=Y(I)
X(I)=X(J)
Y(I)=Y(J)
X(J)=TX
Y(J)=TY
10 K=NV2
20 IF (K.GE.J) GO TO 30
J-J-K
K=K/2
GO TO 20
30 J*J+K
DO 50 L=1,M
LE=2**L
LE1=LE/2
UX=1.0
UY=0.0
ANG=PI/FLOAT(LE1)
WX=1.0*W
WY=SIG*SIN(ANG)
DO 50 J=1,LE1
DO 50 I=J+1,N,LE
IP=I+LE1
TX*X(IP)*UX-Y(IP)*UY
TY=X(IP)*UY+Y(IP)*UX
X(I)=X(IP)*TX
Y(I)=Y(IP)*TY
50 UX=UX
RETURN
END

SUBROUTINE CONSOL (N,NEXT,ICODE)
COMMON /KEEP/ (Z(8256))
COMMON /WET/ X(126),Y(126)

SOLVES A SPECIAL TRI-DIAGONAL LINEAR SYSTEM BY LU
FACTORIZATION.
TWO R.H.S. ARE GIVEN AND STORED IN Y, STARTING AT Y(1)
AND AT Y(NEXT+1) RESPECTIVELY.
The TWO CORRESPONDING VECTOR UNKNOWNS ARE RETURNED IN X,
STARTING AT X(1) AND AT X(NEXT+1) RESPECTIVELY.
IF ICODE = 3 THEN Z ARRAY MUST BE KEPT RESERVED FOR
USE BY CONSOL.

IF (ICODE.EQ.3) GO TO 20
Z(NEXT+1)=X(1)
DO 10 K=2,N
NEXTPL=NEXT+K
K1=K-1
X(K1)=1.0/Z(NEXTPL-1)
10 Z(NEXTPL)=X(K)-X(K1)
20 DO 30 K=2,N
KPL=K+H
KNEXT=K+K-1
20 Y(K)=Y(K)-Y(K-1)/Z(KNEXT)
Courant Institute of Mathematical Sciences
New York University
New York, New York 10012


