NBSIR 80-2056

The Numerical Solution of A Nonseparable Elliptic Partial Differential Equation By Preconditioned Conjugate Gradients

Boeing Computer Services Company
Mail Stop 9C-01
P. O. Box 24346
Seattle, Washington 98124

U S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Applied Mathematics
Washington, DC 20234

April 1980

Final Report

Prepared for:
for Applied Mathematics
l Engineering Laboratory
l Bureau of Standards
gton, DC 20234

NBSIR 80-2056

THE NUMERICAL SOLUTION OF A NONSEPARABLE ELLIPTIC PARTIAL DIFFERENTIAL EQUATION BY PRECONDITIONED CONJUGATE GRADIENTS

John Gregg Lewis*
Ronald G. Rehm†

*Boeing Computer Services Company
Mail Stop 9C-01
P. O. Box 24346
Seattle, Washington 98124

†U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Applied Mathematics
Washington, DC 20234

April 1981

Final Report

Prepared for:
Center for Applied Mathematics
National Engineering Laboratory
National Bureau of Standards
Washington, DC 20234

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

JOURNAL OF RESEARCH of the National Bureau of Standards
Vol. 85, No. 5, September-October 1980

The Numerical Solution of a Nonseparable Elliptic Partial Differential Equation by Preconditioned Conjugate Gradients

John Gregg Lewis* and Ronald G. Rehm†

National Bureau of Standards, Washington, D.C. 20234

April 16, 1980

In this report the combination of an iterative technique, the conjugate gradient algorithm, with a fast direct method, cyclic reduction, is used to solve the linear algebraic equations resulting from discretization of a nonseparable elliptic partial differential equation. An expository discussion of the conjugate gradient and preconditioned conjugate gradient algorithms and of their use in the solution of partial differential equations is presented. New results extending the use of the preconditioned conjugate gradients technique to singular linear equations which arise from discretized elliptic equations with Neumann boundary conditions are also given. The algorithms are applied to solve a specific elliptic equation which arises in the study of buoyant convection produced by a room fire. A code was developed to implement the algorithms for this application. Numerical results obtained through testing and use of the code are discussed.
Key Words: Conjugate gradient algorithm; elliptic partial differential equations; iterative methods for linear algebraic equations; Neumann boundary conditions; sparse matrices.

The numerical solution of the linear algebraic equations which result from a discretization of an elliptic partial differential equation has been, and continues to be, the focus of much research in numerical analysis. Over the past 25 years there have been many advances, some toward improved iterative schemes (ADI, SOR, etc.), others toward fast direct methods (most notably those based on FFT's). See Rice [1] [1] for a brief survey of the impact of these advances. In this paper we discuss the combination of an iterative technique, the conjugate gradient algorithm, with a fast direct method, cyclic reduction, to extend the capabilities of the fast solver. For examples of the use of similar combinations of algorithms, see Concus & Golub [2], Concus, Golub and O'Leary [3], O'Leary [4] and O'Leary and Widlund [5].

The work described in this paper resulted from a study of buoyant convection carried out at the National Bureau of Standards, [6, 7, 8]. The specific elliptic partial differential equation for pressure arising in this work is used as a model problem in the present paper. The experience of the first author with the buoyant convection model motivated the writing of sections 1 and 2, which gives an exposition of the conjugate gradient and preconditioned conjugate gradient algorithms. These sections contain no new material; they were written so that this paper may be accessible to an audience unfamiliar with the development of the conjugate gradient algorithm and its use in the solution of partial differential equations. Section 3 has a discussion of the model problem, the pressure equation. Section 4 contains several new results extending the use of preconditioned conjugate gradient technology to the singular linear equations which result from Neumann boundary value problems. We conclude with some numerical examples from the buoyant convection problem. A listing of a FORTRAN program implementing the algorithm discussed here is presented in [15].

---

*Center for Applied Mathematics, National Engineering Laboratory; and Mathematical Sciences Department, Johns Hopkins University. Current Address: Boeing Computer Services, Co., Mail Stop 9C-01, P.O. Box 24346, Seattle, Washington 98124

†Center for Applied Mathematics, National Engineering Laboratory.

[1] Figures in brackets indicate literature references at the end of this paper.

1. An introduction to the conjugate gradient algorithm

The discretization of an elliptic differential operator by standard finite difference or finite element techniques produces, as the finite analog of the continuous operator, very large matrices with most elements zero. From an algebraic standpoint we can very often view the numerical solution of the differential equation as:

$$\text{Solve } Ax = b \tag{1}$$

where A is a k by k, (k large) real symmetric, positive definite and sparse matrix.

Fast direct methods with minimal storage requirements exist when A is the finite difference operator resulting from a separable elliptic operator on a rectangular region, and for some other common, but specific, cases. If the operator is nonseparable, or the region non-rectangular, the special direct methods may not apply. Moreover, the use of general direct methods such as factoring A into a Cholesky decomposition,

$$A = LL^T,$$

often impose unbearable storage requirements because many zero entries are filled in during the decomposition.

Iterative methods, such as SOR and Gauss-Seidel, solve a transformation of the problem. Instead of solving (1) directly, they "split" A as $A = M-N$, and solve the equivalent problem through an iteration of the form $Mx^{k+1} = Nx^k + b$; i.e., find a fixed point of the equation

$$Mx = Nx + b. \tag{2}$$

The efficiency of the solution method depends in part on the appropriatcness of the splitting and in part on acceleration of the iteration toward the fixed point.

The conjugate gradient algorithm can be motivated as the solution of another transformation of problem (1). Consider the inner product $\langle x,y\rangle \equiv x^TAy$. This induces a norm $\|x\|_A \equiv \sqrt{x^TAx}$ on R^k when A is positive definite.

Problem (1) is equivalent to: find x to minimize

$$E(x) = (x-x^*)^T A(x-x^*) = |x-x^*|_A^2 \tag{3}$$

where x^* is the solution of (1). The problems are equivalent because $E(x) \geq 0$ for all x, and $E(x) = 0 \Leftrightarrow x = x^*$. The immediate usefulness of the transformation is not obvious, especially since computing $E(x)$ appears to require the solution, x^*, of the problem we wish to solve. Note however, that we can evaluate whether a given x is or is not a solution by computing the residual $r = b - Ax$. Further, while we cannot evaluate $E(x)$ directly, we can evaluate its gradient vector:

$$\nabla E(x) = 2(Ax - Ax^*) = 2(Ax - b) = -2r.$$

Hence, r gives us the direction in which E decreases most rapidly (unless r is identically zero, which characterizes the solution).

These two characteristics of problem (3) lead directly to a simple algorithm for solving (3):

STEEPEST DESCENT: Given a guess x_i, not a solution,

$$\begin{cases} \text{set } r_i = b - Ax_i \\ \text{set } x_{i+1} = x_i + \alpha_i r_i \end{cases}$$

where α_i is a scalar chosen to make x_{i+1} minimize $E(x_{i+1})$ for all vectors of the form $x_i + \beta r_i$.

We can easily compute the gradient r_i. It is also easy to compute α_i so that $E(x_{i+1}) = E(x_i + \alpha_i r_i) \leq E(x_i + \beta r_i)$ for all β. The mimimum of $E(x_i + \beta r_i)$ is given by the solution of

$$\frac{d}{d\beta} E(x_i + \beta r_i) = 0$$

But

$$\frac{d}{d\beta} E(x_i + \beta r_i) = 2(\beta r_i^T A r_i - r_i^T r_i),$$

so

$$\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i} = \frac{\|r_i\|_2^2}{\|r_i\|_A^2} \tag{4}$$

is the optimal choice for α_i. Hence, we can carry out the iteration to minimize $E(x)$ without computing E. Note also that A enters the iteration only in forming the products Ax_i and Ar_i.

Unfortunately steepest descent is not a practical algorithm because the convergence can be very slow. We can obtain more rapid convergence by using not the steepest descent directions $\{r_i\}$, but instead a sequence of downhill or descent directions $\{p_i\}$ which satisfy additional properties. We characterize the term "descent direction" and simultaneously find the optimal distance to move in that direction, through the following simple result.

LEMMA: *If* $p^T r \neq 0$, *then*

$$\beta = \frac{p^T r}{p^T A p} \text{ minimizes } E(x + \beta p) \text{ for all } \beta. \tag{5}$$

PROOF: Simply differentiate $E(x + \beta p)$ with respect to β:

$$\frac{d}{d\beta} E(x + \beta p) = 2(p^T A(x - x^*) + \beta p^T A p)$$

$$= 2(-p^T r + \beta p^T A p).$$

It is a simple computation to show that $E(x + \beta p) < E(x)$. Note that β is positive if $p^T r > 0$, which characterizes p as a descent, not an ascent, direction.

The first observation which leads to improved convergence is that the choice in (5) for β, not only solves the one-dimensional minimization, it gives us the $p-$ component of x^* exactly. To see this, note that A positive definite implies that we can uniquely write

$$\begin{aligned} x + \beta p &= \alpha p + \delta w \\ x^* &= \alpha^* p + \delta^* z \end{aligned} \tag{6}$$

where $w^T A p = 0 = z^T A p$. This last condition is read: w, z are A-orthogonal or *A-conjugate* to p. Now substitute the decompositions (6) into E:

$$E(x + \beta p) = (\alpha - \alpha^*)^2 p^T A p + \text{ terms not involving } \alpha, \alpha^* \text{ or } p \tag{7}$$

The choice (5) for β minimizes the left side of (7), and clearly $\alpha = \alpha^*$ minimizes the right hand side of (7). Hence choosing $\beta = \frac{p^T r}{p^T A p}$ implies that

$$(x + \beta p) - x^* = \delta w - \delta^* z. \tag{8}$$

The error vector is A-conjugate to p, i.e., lies in the $k-1$ dimensional subspace of vectors t such that $t^TAp = 0$. If all succeeding descent directions are chosen from this subspace, we can preserve the exact solution of the problem in the direction p.

The second observation is that we can easily choose successive directions $p_1, p_2, p_3, \ldots$ such that the $\{p_i\}$ are all pairwise A-conjugate descent directions (hence are "A-conjugate gradients") and hence, such that the successive errors $x_1 - x^*, x_2 - x^*, x_3 - x^*$ are constrained to successively smaller dimensional subspaces. The set $\{p_i\}$, $i = 1, \ldots, k$ gives a basis for R^k and $x_k - x^*$ must be A-conjugate to all of the $\{p_i\}$, hence must be zero. Thus, the process must give the exact answer after at most k iterations. We now write out the conjugate gradient iteration directly:

CONJUGATE GRADIENTS: Given x_1, compute $r_1 = b - Ax_1$. Set the initial descent direction p_1 to be r_1. For $i = 2, 3, 4, \ldots,$ do.

Move to the minimum in direction p_i and evaluate the new residual

$$\begin{aligned} \alpha_i &= \frac{r_i^T p_i}{p_i^T A p_i} \\ x_{i+1} &= x_i + \alpha_i p_i \\ r_{i+1} &= b - Ax_{i+1} \\ &= r_i - \alpha_i A p_i \end{aligned} \tag{9}$$

Compute a new descent direction A-orthogonal to all of its predecessors

$$\begin{aligned} \beta_i &= -\frac{r_{i+1}^T A p_i}{p_i^T A p_i} \\ p_{i+1} &= r_{i+1} + \beta_i p_i \end{aligned} \tag{10}$$

The initial step (9) in the iteration is the step in the steepest descent direction p_1, as discussed above. Step (10), the choice of a new descent direction, is simply a single step of Gram-Schmidt orthogonalization, to remove from the steepest descent direction its component in the direction Ap_i.

Hence

$$p_{i+1}^T A p_i = 0 \tag{11}$$

Futher the choice of α_i implies directly that

$$r_{i+1}^T p_i = 0 \tag{12}$$

The following identities are derived easily from (9) and (10):

$$\begin{aligned} &r_{i+1}^T r_i = 0; \\ &r_i^T p_i = r_i^T r_i; \\ &r_i^T A p_i = p_i^T A p_i. \end{aligned} \tag{13}$$

The most important algebraic consequences of (9) and (10) yield only to a lengthy inductive argument (not given here—see Reid [12] or Hestenes and Stiefel [13] for example)

$$\left.\begin{aligned} r_{i+1}^T r_j &= 0 \\ p_{i+1}^T A p_j &= 0 \end{aligned}\right\} \text{ for } all\ j \leqslant i \tag{14}$$

The identities in (14) imply that all of the $\{p_i\}$ are A-conjugate, even though we perform an explicit orthogonalization only to one descent direction in (10).

In the conjugate gradient iteration observe that A enters only in forming a matrix-vector product, Ap_i; there are no transformations done which could destroy the sparseness of A. The dominant cost per step is usually that of forming the product Ap_i; in this case the conjugate gradient iteration is very little more expensive per iteration than the steepest descent algorithm and it offers the guarantee of convergence within k steps.

The practical reason for using the conjugate gradient algorithm is that we often obtain satisfactory accuracy after only a very few iterations. Certain theoretical bounds for the convergence of the algorithm depend on the condition number, $\varkappa$, of A, defined by:

$$\varkappa = \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \tag{15}$$

where λ_{max} (A) is the largest eigenvalue of A and λ_{min} (A) is the smallest eigenvalue of A (Note: A is positive definite, so $\varkappa$ is positive, and at least as great as one). The following bounds can be found in the literature (see Daniel [14], for example):

$$\|x_i - x^*\|^2 \leq 4 \frac{E(x_i)}{\lambda_{min}} \cdot \left(\frac{\sqrt{\varkappa}-1}{\sqrt{\varkappa}+1} \right)^{2(i-1)}$$

$$E(x_i) \leq 4 \left(\frac{1-\sqrt{\frac{1}{\varkappa}}}{1+\sqrt{\frac{1}{\varkappa}}} \right)^{2(i-1)} E(x_1)$$

Clearly, convergence is rapid when $\varkappa$ is close to one (and takes place in one step if $\varkappa$ equals one). For well-conditioned problems very few iterations will suffice to obtain highly accurate solutions.

2. Preconditioned Conjugate Gradients and Matrix-Splittings.

Our basic problem is to solve

$$Ax = b, \tag{16}$$

which is the large, sparse linear system which arises from the discretization of an elliptic partial differential equation. We can assume that the matrix A is symmetric and positive definite, so the method of conjugate gradients certainly can be used. However, the condition number, $\varkappa(A)$, is usually very large for these problems; the conjugate gradient algorithm converges very slowly when applied directly to A and is not competitive with other methods.

The preconditioned conjugate gradients method arises, like SOR, from matrix splittings. Consider

$$A = M - N$$

where M is another positive-definite, symmetric matrix, but one for which we can easily solve linear systems. There exists a symmetric positive-definite matrix $M^{-1/2}$ such that

$$M^{-1} = (M^{-1/2})(M^{-1/2})$$

or

$$[(M^{-1/2})(M^{-1/2})]^{-1} = M$$

(This is the symmetric matrix whose eigenvectors are the same as M's and whose corresponding eigenvalues are the reciprocals of the square roots of the eigenvalues of M. We shall not use this form, however; we need only the formal existence of $M^{-1/2}$.) We can solve the equations $Ax = b$ by computing

$$d = M^{-1/2}b; \tag{17}$$

and solving

$$Cz = d \tag{18}$$

where $C = M^{-1/2}AM^{-1/2}$, by the conjugate gradient algorithm (Note that C is a positive-definite, symmetric matrix), finally computing

$$x = M^{-1/2}z \tag{19}$$

This transformation from $Ax = b$ to $Cy = d$ will be an effective method if:
a) the condition number, $\kappa(C)$, is close to 1 so that the conjugate gradient algorithm converges rapidly, and
b) the multiplication $C \cdot y$ for any vector y can be computed efficiently and sparsely.
We shall examine condition (a) first. Note that

$$A = M - N$$

implies

$$C = M^{-1/2}AM^{-1/2} = I - (M^{-1/2}NM^{-1/2}) = I - R.$$

Hence

$$\kappa(C) = \frac{\lambda_{max}(I-R)}{\lambda_{min}(I-R)} \geqslant 1$$

This condition number, $\kappa(C)$, will be close to 1 if $\lambda_{max}(R)$ is small, which is true if all of the elements of R are small. (The condition number is 1 exactly when R is 0, or when $M = A$). We want, then, to choose M so that M represents A as well as possible (makes R as small as possible), and such that the choice for M allows for condition (b). We discuss a specific choice for M in section 3, the discussion of the model problem. Other examples are given in Concus, Golub, and O'Leary [3], O'Leary [4], and Meijerink and van der Vorst [9].

We now turn to condition (b); with a formal derivation of the actual algorithm used as "Preconditioned conjugate gradients," we show that condition (b) is met whenever M is such that the linear equation

$$Mz = w$$

can be solved efficiently and sparsely. Reconsider the conjugate gradient iteration to solve $Cz = d$, given an initial guess z_1:

$$(\mathrm{CG}_c)\text{: set } \bar{r}_1 = d - Cz_1;$$

$$\text{set } \bar{p}_1 = \bar{r}_1;$$

$$\text{for } i = 2, 3, 4 \ldots$$

$$\alpha_i = \frac{\bar{r}_i^T \bar{r}_i}{\bar{p}_i^T C \bar{p}_i}$$

$$z_{i+1} = z_i + \alpha_i \bar{p}_i$$

$$\bar{r}_{i+1} = \bar{r}_i - \alpha_i C\bar{p}_i$$

$$\beta_i = \frac{\bar{r}_{i+1}^T \bar{r}_{i+1}}{\bar{r}_i^T \bar{r}_i}$$

$$\bar{p}_{i+1} = \bar{r}_{i+1} + \beta_i \bar{p}_i$$

In the above, the residual vectors $\{\bar{r}_i\}$ and descent directions $\{\bar{p}_i\}$ have bars on them to denote that they pertain to the scaled problem $Cz = d$. Now view (CG_c) from the original space in which we solve $Ax = b$. Define

$$x_i = M^{-1/2} z_i,$$

$$r_i = b - Ax_i$$

Then

$$\bar{r}_i = d - Cz_i = M^{-1/2} r_i$$

Similarily, we can define formally

$$p_i \equiv M^{-1/2} \bar{p}_i$$

Then

$$\alpha_i = \frac{\bar{r}_i^T \bar{r}_i}{\bar{p}_i^T C \bar{p}_i} = \frac{r_i^T (M^{-1/2})^T (M^{-1/2}) r_i}{p_i^T (M^{1/2})^T C M^{1/2} p_i} = \frac{r_i^T (M^{-1} r_i)}{p_i^T A p_i}$$

Updating z_i corresponds to taking

$$x_{i+1} = x_i + \alpha_i M^{-1/2} \bar{p}_i = x_i + \alpha_i p_i$$

The corrected residual is

$$\begin{aligned} r_{i+1} &= M^{1/2} \bar{r}_{i+1} \\ &= M^{1/2} (\bar{r}_i - \alpha_i C \bar{p}_i) \\ &= M^{1/2} \bar{r}_i - \alpha_i A M^{-1/2} \bar{p}_i \\ &= r_i - \alpha_i A p_i. \end{aligned}$$

The Gram-Schmidt coefficient β_i can be computed as:

$$\beta_i = \frac{\bar{r}_{i+1}^T \bar{r}_{i+1}}{\bar{r}_i^T \bar{r}_i} = \frac{r_{i+1}^T (M^{-1} r_{i+1})}{r_i^T (M^{-1} r_i).}$$

The new direction becomes:

$$p_{i+1} = M^{-1/2} \bar{p}_{i+1} = M^{-1/2} (\bar{r}_{i+1} + \beta_i \bar{p}_i)$$

$$= M^{-1/2}(M^{-1/2} r_{i+1}) + \beta_i p_i$$

$$= M^{-1} r_{i+1} + \beta_i p_i.$$

The result of these manipulations is that we can now write a conjugate-gradient like iteration to solve the equations $Ax = b$, but whose convergence rate depends on

$$C = M^{-1/2} A M^{-1/2}, \text{ not } A.$$

PRECONDITIONED CONJUGATE-GRADIENTS (PCG): Given initial guess x_1: compute

$$v_1 = Ax_1$$

$$r_1 = b - Ax_1 = b - v_1$$

solve $Mu_1 = r_1$

and set $p_1 = u_1$

For $i = 2, 3, 4, \ldots$

compute $v_i = Ap_i$;

$$\text{set } \alpha_i = \frac{r_i^T u_i}{p_i^T v_i} = \frac{r_i^T M^{-1} r_i}{p_i^T A p_i}$$

$$\text{update } x_{i+1} = x_i + \alpha_i p_i$$
$$r_{i+1} = r_i - \alpha_i v_i \; (= r_i - \alpha_i A p_i)$$

solve $Mu_{i+1} = r_{i+1}$

$$\text{set } \beta_i = \frac{r_{i+1}^T u_{i+1}}{r_i^T u_i} \left(= \frac{r_{i+1}^T M^{-1} r_{i+1}}{r_i^T M^{-1} r_i}\right)$$

finally, $p_{i+1} = u_{i+1} + \beta_i p_i$

We introduced new vectors $\{u_i\}$ and $\{v_i\}$ in the algorithm above to emphasize the fact that the matrices A and M appear only one time during each iteration, and in very specific ways. The matrix A appears only in the formation of the product Ap_i (or Ax_i), an operation which can be done very efficiently for most representations of sparse matrices. The matrix M appears only implicitly; we must be able to solve the linear systems $Mu = r$ efficiently and in little storage, and this is the primary restriction on M. Note that the matrix square-roots, $M^{1/2}$, do not appear at all. The purpose of the matrix M is to *scale* or *pre-condition* the problem so that convergence takes place very quickly. The cost, of course, is that each iteration now requires the solution of a linear equation as well as the formation of a matrix-vector product.

3. A model problem—the pressure equation.

The problem to be discussed here arises in a model of buoyant convection (Rehm and Baum [6]). Specifically, at each time step in the solution of a mixed hyperbolic-elliptic system, we are required to solve:

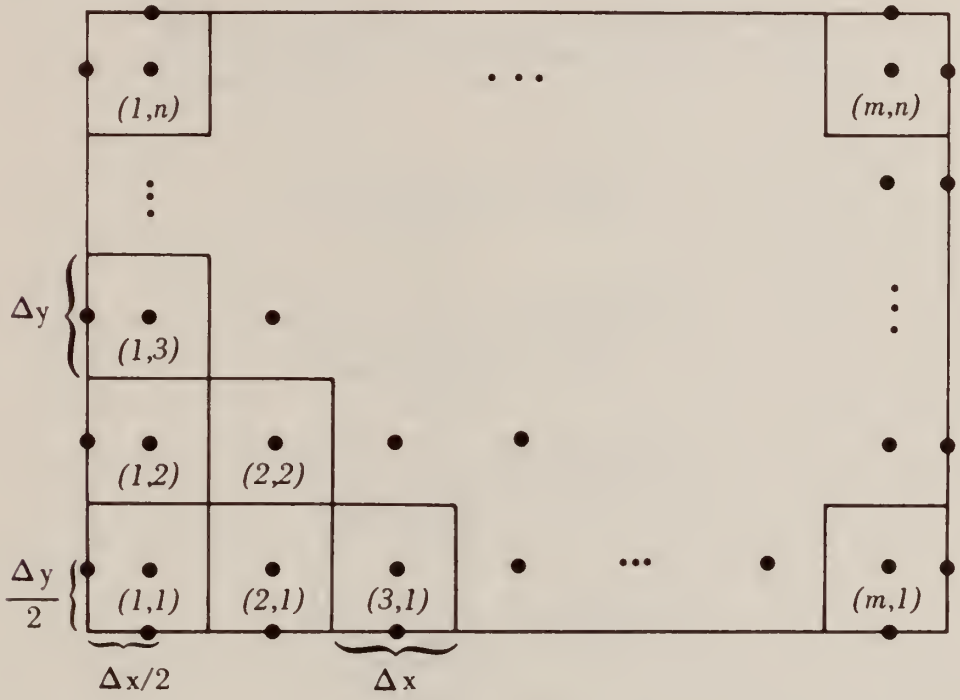
$$\nabla \cdot \left(\frac{1}{\varrho(x,y)} \nabla P(x,y)\right) = f(x,y) \tag{20}$$

on a rectangle R subject to the following condition on the exterior normal derivative

$$\frac{\partial P}{\partial \eta}(x,y) = \varrho(x,y) \cdot g(x,y) \text{ on the boundary } \partial R.$$

Here ϱ denotes gas density and P the unknown pressure. We assume that ϱ depends on both spatial variables. Equation (1) then describes a *nonseparable* elliptic equation. Since we must solve this equation repeatedly, speed is paramount. Were this a separable equation, the discrete linear system discussed below could be solved directly by the cyclic reduction routines of Sweet and Schwarztrauber [10]. Were the density and its first and second derivatives known analytically, the problem could be converted to an ordinary Poisson equation by the techniques of Concus and Golub [2]. Neither of these conditions can be met, which forces us to consider the discrete linear system in more detail.

In the context of the buoyant convection problem, we are given the density ϱ, and g and f, only on a discrete set of points; the solution P will be produced on the same grid of discrete points. A grid suitable for the hydrodynamic calculations is:



The functions ϱ and f are given on the interior points; g is given at the circles on the boundary. Note that the interior grid is offset by one half grid spacing from the boundary.

The derivatives in (20) are replaced by second-order accurate centered finite differences. For example,

$$\frac{\partial}{\partial x} \cdot \left(\frac{1}{\varrho(x,y)} \frac{\partial}{\partial x} P(x,y) \right)^{(1)}$$

becomes

$$\frac{1}{\Delta x} \left(\frac{1}{\varrho(x + \frac{\Delta x}{2}, y)} \left[\frac{1}{\Delta x} (P(x + \Delta x, y) - P(x,y)) \right] - \right.$$

$$\left. \frac{1}{\varrho(x - \frac{\Delta x}{2}, y)} \left[\frac{1}{\Delta x} (P(x,y) - P(x - \Delta x, y)) \right] \right)$$

Note that we require the reciprocal of the density at an intermediate point: an $O(\Delta x)$ approximation will suffice to give second order accuracy for the operator. For consistency with the hydrodynamic equations in the buoyant convection model, we use the reciprocal of the average density:

$$\frac{1}{\varrho(x + \frac{\Delta x}{2}, y)} \cong \frac{2}{[\varrho(x + \Delta x, y) + \varrho(x,y)]}$$

Define $\varrho_{ij} = \varrho((i-1/2)\,\Delta x, (j-1/2)\,\Delta y)$, and similarly for P_{ij}, f_{ij}, and g_{ij}. Then the general equation for an interior grid point i, j not adjacent to the boundary is

$$\begin{aligned}
&\Bigg\{\Bigg[\underbrace{\left(\frac{-2}{\varrho_{i-1,j} + \varrho_{ij}}\right)}_{d_1} + \underbrace{\left(\frac{-2}{\varrho_{ij} + \varrho_{i+1,j}}\right)}_{d_2}\Bigg]\left(\frac{1}{\Delta x}\right)^2 + \\
&\Bigg[\underbrace{\left(\frac{-2}{\varrho_{ij-1} + \varrho_{ij}}\right)}_{d_3} + \underbrace{\left(\frac{-2}{\varrho_{ij} + \varrho_{ij+1}}\right)}_{d_4}\Bigg]\left(\frac{1}{\Delta y}\right)^2\Bigg\} P_{ij} \\
&+\underbrace{\left(\frac{2}{\varrho_{ij} + \varrho_{i-1,j}}\right)}_{-d_1}\left(\frac{1}{\Delta x}\right)^2 P_{i-1,j} \\
&+\underbrace{\left(\frac{2}{\varrho_{ij} + \varrho_{i+1,j}}\right)}_{-d_2}\left(\frac{1}{\Delta x}\right)^2 P_{i+1,j} \\
&+\underbrace{\left(\frac{2}{\varrho_{ij-1} + \varrho_{ij}}\right)}_{-d_3}\left(\frac{1}{\Delta y}\right)^2 P_{i,j-1} \\
&+\underbrace{\left(\frac{2}{\varrho_{ij} + \varrho_{ij+1}}\right)}_{-d_4}\left(\frac{1}{\Delta y}\right)^2 P_{i,j+1} = f_{ij}
\end{aligned} \tag{21_{ij}}$$

The adjustment for points adjacent to the boundary uses the second order centered approximation to the boundary derivative: on boundary k (left = 1, right = 2, lower = 3, upper = 4), the terms d_k are evaluated using the first interior mesh point and an image point (one-half grid spacing outside the region). Formally the terms d_1 are evaluated at $i = 1/2, j$ at the left boundary for example. The discretized form of the Neuman boundary conditions become

$$\left(\frac{1}{\Delta x}\right)^2\left(-\frac{1}{2}d_1 p_{1,j} + \frac{1}{2}d_1 p_{0,j}\right) = g_{1/2,j}/\Delta x \equiv g_1(j) \tag{22_1}$$

$$\left(\frac{1}{\Delta x}\right)^2\left(-\frac{1}{2}d_2 p_{m,j} + \frac{1}{2}d_2 p_{m+1,j}\right) = g_{m+1/2,j}/\Delta x \equiv g_2(j) \tag{22_2}$$

$$\left(\frac{1}{\Delta y}\right)^2\left(-\frac{1}{2}d_3 p_{i,1} + \frac{1}{2}d_3 p_{i,0}\right) = g_{i,1/2}/\Delta y \equiv g_3(i) \tag{22_3}$$

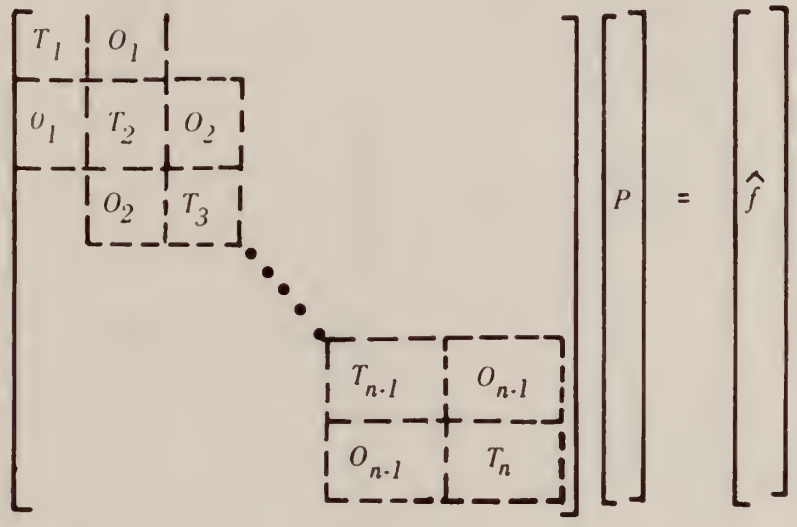
$$\left(\frac{1}{\Delta y}\right)^2\left(-\frac{1}{2}d_4 p_{i,n} + \frac{1}{2}d_4 p_{i,n+1}\right) = g_{i,n+1/2}/\Delta y \equiv g_4(i) \tag{22_4}$$

For a point adjacent to boundary k, subtract equation (22_k) from equation $(21_{i,j})$ which eliminates the d_k terms, and with them, all references to points i, j outside the grid. The right hand side is replaced by

$$f_{ij} - g_k$$

Corner points are treated by subtracting (22_k) and (22_l) from (21_{ij}), where the corner is adjacent to both its k-th and l-th boundary.

We arrange the grid points in the order
$\{(1,1), (2,1), \ldots, (m,1), (1,2), (2,2), \ldots, (m,2), \ldots, (1,n), \ldots, (m,n)\}$ and write the equation for each point in this order. The result is the matrix equation



where T_i is an $m \times m$ symmetric tridiagonal matrix, and O_i is an $m \times m$ diagonal matrix. We shall denote this matrix equation by

$$Ax = b \tag{23}$$

where b represents the right hand side, f, of the pressure equation, adjusted by the boundary data, g.

This is the linear algebraic system we need to solve. We make several observations:

1. A is symmetric
2. A is sparse. Only five diagonals contain non-zero elements.
3. The eigenvalues of A are all real and non-positive; A is a negative semi-definite matrix, generally of rank $mn - 1$. Hence $-A$ is a positive semi-definite matrix.

Although we could apply the method of conjugate gradients directly to $-A$, convergence would be very slow. O'Leary [4] and Meijerink and Van der Vorst [9] suggest several different approaches for creating scaling matrices M to use a preconditioned conjugate gradient scheme. One general approach is to consider that A originated from a differential equation and let M be the discrete operator from a separable differential equation which approximates the pressure equation.

A specific choice is suggested by Concus and Golub [2]: Let L be the discretization of the Poisson equation

$$\nabla^2 P = f$$

with Neumann boundary conditions (on the same staggered grid). Then

$$L = \begin{bmatrix} B_1 & c_1 I & & & \\ c_1 I & B_2 & c_1 I & & \\ & c_1 I & B_2 & & \\ & & & \ddots & \\ & & & B_2 & c_1 I \\ & & & c_1 I & B_1 \end{bmatrix} \tag{24}$$

where

$$B_1 = \begin{bmatrix} a_1 & c_2 & & & & & \\ c_2 & a_2 & c_2 & & & & \\ & c_2 & a_2 & c_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & c_2 & a_2 & c_2 & \\ & & & & c_2 & a_1 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} a_3 & c_2 & & & & & \\ c_2 & a_4 & c_2 & & & & \\ & c_2 & a_4 & c_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & c_2 & a_4 & c_2 & \\ & & & & c_2 & a_3 \end{bmatrix}$$

with $a_1 = -\frac{1}{(\Delta x)^2} - \frac{1}{(\Delta y)^2}$

$$a_2 = -\frac{2}{(\Delta x)^2} - \frac{1}{(\Delta y)^2}$$

$$a_3 = -\frac{1}{(\Delta x)^2} - \frac{2}{(\Delta y)^2}$$

$$a_4 = -\frac{2}{(\Delta x)^2} - \frac{2}{(\Delta y)^2}$$

$$c_1 = \frac{1}{(\Delta y)^2}; \; c_2 = \frac{1}{(\Delta x)^2}$$

We assume that we have available a good subroutine for solving the equation

$$Lu = v,$$

e.g., subroutine BLKTRI from the NCAR package of subroutines for elliptic partial differential equations [10].

Let $D^{1/2}$ be the diagonal matrix with entries

$$d_{kk} = \sqrt{a_{kk}/l_{kk}},$$

the square root of the ratio of corresponding main diagonal entries of A and L. Then

$$M = D^{1/2} L D^{1/2}$$

is a symmetric negative semi-definite sparse matrix whose main diagonal is identical to that of A. Hence, in the splitting

$$A = M - N,$$

the matrix N has zero main diagonal and non-zero entries on at most four off-diagonals.

In the preconditioned conjugate gradient iteration we must be able to solve

$$Mz = r$$

This is done without excessive storage demands by:

a) compute $w = D^{-1/2} r$ (D is a diagonal matrix)
b) solve $Lu = w$ by cyclic reduction (BLKTRI)
c) compute $z = D^{-1/2} u$.

For computational convenience, to avoid the repeated multiplications by the diagonal matrix $D^{1/2}$ or its inverse, we replace the original problem

$$Ax = b$$

by

$$\hat{A}\hat{x} = \hat{b}$$

where

$$\hat{A} = D^{-1/2} A D^{-1/2}$$

$$\hat{b} = D^{-1/2} b$$

solve $Ax = b$ by preconditioned conjugate gradients with L as the scaling matrix, and finally compute

$$x = D^{-1/2} \hat{x}$$

This is the approach actually used in the computations reported in section 5.

4. The Preconditioned Conjugate Gradient Algorithm for a Semi-definite Matrix.

In our discussion of the discrete operators A and L in the previous section we have ignored one characteristic of these operators which renders them unsuitable for direct use in a conjugate gradient or preconditioned conjugate gradient iteration. The conjugate gradient algorithm requires that A be a positive-definite matrix; otherwise the function E may have zeroes other than at the solution of the linear equation. The

algorithm becomes transparently a maximization algorithm for negative-definite matrices. (There is no need to change signs implicitly or explicitly to solve linear systems involving negative definite matrices, e.g., the discrete Laplacian with Dirichlet boundary conditions). However, our operators A and L are both negative semidefinite; they have negative eigenvalues, and also, one zero eigenvalue each.

In this section we will extend the theory of the conjugate gradient algorithm to allow for the special case of a semi-definite matrix with known nullspace. We shall then extend the preconditioned conjugate gradient algorithm to allow both A and M to be of this special type. The first part of the section will be elementary for readers well-versed in the conjugate gradient theory; the second part contains some new and unexpected results.

The problem is simple: A is a singular matrix. This is shown easily by noting that whenever a term d_k appears on the diagonal of A, the opposite term $-d_k$ appears on one of the off-diagonals. Hence, the sum of the coefficients in any row of A is exactly zero. But this is the same as saying

$$Ae = 0$$

where e is the vector $(1,1,1,\ldots 1)^T$. In general, $Ax = 0 <=> x = \alpha e$, where α is a real scalar.

The singularity of A is related to the Neumann boundary conditions for the pressure equation. If P is a solution to the differential equation so is $P + c$, where c is any constant. Similarly, if x is any solution to (3), so is $x + \alpha e$, where α is any scalar. But this is equivalent to adding the constant α to each point x_{ij} of the solution. The singularity of the operators also implies that not every system of equations has a solution. A system of equations is consistent if and only if it has a solution. The characterization of consistency for these operators is simple to express: The pressure equation is consistent $< = >$

$$\int\int_R f = \int_{\partial R} g \tag{25}$$

The linear equations $Ax = b$ are consistent <=>

$b^T e = 0$, that is, when b in Range (A). But

$$b^T e = 0 \ <=> \ \sum_{ij} b_{ij} = 0$$

$$<=> \ \sum_{ij} f_{ij} = \sum_j (g_1(j) + g_3(j)) + \sum_i (g_2(i) + g_4(i))$$

the discrete analog of the integral equality (25)

Further, even when the equation is consistent, the solution is not unique. There is a unique solution of shortest length in the usual L_2 norm. For the continuous case it is the solution with mean pressure zero, i.e.,

$$\int\int_R P = 0.$$

In the discrete case the analog is

$$x^T e = 0;$$

again, the mean (discrete) pressure is zero.

We shall now discuss the modifications to the conjugate gradient algorithm which would be necessary to obtain this unique solution to a consistent system of equations. For generality, assume that we want to solve

$$Ax = b \tag{26}$$

where A is symmetric positive semi-definite of dimension k with known nullspace the span of $\{n\}$. Hence, the rank of A is one less than its dimension. Further, assume $b^T n = 0$, so that (26) is a consistent system.

(note: if (26) is not consistent, $\hat{b} = b - \dfrac{b^T n}{n^T n}$ satisfies the consistency condition, and any solution of $Ax = \hat{b}$ is a least squares solution of (26)).

Let the vectors $\{n, q_2, q_3, q_4, \ldots, q_k\}$ form an orthonormal basis for R^k. Then Range (A) = span $\{q_2, q_3, \ldots, q_k\}$

Let Q be the orthogonal matrix ($Q^T = Q^{-1}$)

$$Q = [n \vdots q_2 \vdots q_3 \vdots \cdots \vdots q_k].$$

Then

$$Ax = b \iff (Q^T A\, Q)(Q^T x) = Q^T b \tag{27}$$

But, by the consistency condition $b^T n = 0$

$$Q^T b = \begin{bmatrix} 0 \\ d_1 \\ d_2 \\ \cdot \\ \cdot \\ \cdot \\ d_{k-1} \end{bmatrix}$$

Similarily

$$Q^T A\, Q = \begin{bmatrix} 0 & 0 \\ 0 & B \end{bmatrix} \begin{matrix} \} \\ \} \end{matrix} \begin{matrix} 1 \\ k-1 \end{matrix}$$

$$\underbrace{\qquad}_{k-1}$$

Hence, (27) really takes the form

$$\begin{bmatrix} 0 & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} w \\ z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_{k-1} \end{bmatrix} = \begin{bmatrix} 0 \\ d_1 \\ d_2 \\ \cdot \\ \cdot \\ \cdot \\ d_{k-1} \end{bmatrix} \tag{28}$$

Clearly w is arbitrary in (28). The equation holds whenever

$$Bz = d \tag{29}$$

a $(k-1)$ – dimensional problem. All solutions of (26) are given by

$$x = Q \begin{bmatrix} w \\ z \end{bmatrix}$$

where z solves (29). The unique minimal length solution is given by

$$x^* = Q \begin{bmatrix} 0 \\ z \end{bmatrix}$$

By the assumptions on A, the matrix B is positive-definite and the conjugate gradient iteration can be used to solve (29). The condition number

$$\varkappa(B) = \frac{\lambda_{k-1}(B)}{\lambda_1(B)} = \frac{\lambda_k(A)}{\lambda_2(A)},$$

(where we assume the eigenvalues are ordered algebraically) governs the convergence of the algorithm.

We now show that we can solve the consistent linear system (26) directly by the conjugate gradient algorithm, with convergence rate determined by $\varkappa(B)$. The proof is elementary: we show that the algorithm, when started with an initial vector x_1 and a right hand side b which lie in the same invariant subspace of A, solves the problem while remaining entirely in that subspace.

Suppose we have the equation

$$Ax = b$$

with $b^T n = 0$ and $x_1^T n = 0$. Then the conjugate gradient algorithm, when applied to A, produces vectors x_i, r_i and p_i exactly equal to

$$x_i = Q\begin{bmatrix} 0 \\ \hline z_i \end{bmatrix}$$

$$r_i = Q\begin{bmatrix} 0 \\ \hline \hat{r}_i \end{bmatrix}$$

and

$$p_i = Q\begin{bmatrix} 0 \\ \hline \hat{p}_i \end{bmatrix}$$

where z_i, $\hat{r}_i$ and $\hat{p}_i$ are the vectors produced by the conjugate gradient algorithm for

$$Bz = d$$

with initial vector $z_1 =$ the last $k-1$ entries of $Q^T x_1$. We note that

$$Q^T r_1 = Q^T(b - Ax_1) = Q^T b - Q^T A(QQ^T)x_1 = \begin{bmatrix} 0 \\ \hline d \end{bmatrix} - \begin{bmatrix} 0 \\ \hline Bz_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \hline \hat{r}_1 \end{bmatrix},$$

where it is essential that $b^T n = 0$. It follows that

$$Q^T p_1 = \begin{bmatrix} 0 \\ \hline \hat{p}_1 \end{bmatrix}.$$

So assume the required properties hold for x_i, r_i and p_i. We shall show the iteration preserves these properties for x_{i+1}, r_{i+1}, and p_{i+1}. The first step in the iteration is to compute

$$\alpha_i = \frac{r_i^T r_i}{p_i^T A p_i},$$

which by induction is

$$\alpha_i = \frac{\begin{bmatrix} 0 \\ \hline \hat{r}_i \end{bmatrix}^T Q^T Q \begin{bmatrix} 0 \\ \hline \hat{r}_i \end{bmatrix}}{\begin{bmatrix} 0 \\ \hline \hat{p}_i \end{bmatrix}^T Q^T A Q \begin{bmatrix} 0 \\ \hline \hat{p}_i \end{bmatrix}}$$

$$= \frac{\hat{r}_i^T \hat{r}_i}{\hat{p}_i^T B \hat{p}_i},$$

since $Q^TQ = I$ and

$$Q^TAQ = \left[\begin{array}{c|c} 0 & 0 \\ \hline 0 & B \end{array}\right]$$

Hence, $\hat{\alpha}_i = \alpha_i$, and it follows that

$$x_{i+1} = x_i + \alpha_i p_i = Q\left[\begin{array}{c} 0 \\ \hline z_i \end{array}\right] + \alpha_i Q\left[\begin{array}{c} 0 \\ \hline \hat{p}_i \end{array}\right] = Q\left[\begin{array}{c} 0 \\ \hline z_i + \hat{\alpha}_i p_i \end{array}\right] = Q\left[\begin{array}{c} 0 \\ \hline z_{i+1} \end{array}\right]$$

Then

$$\begin{aligned} r_{i+1} &= r_i - \alpha_i A p_i \\ &= Q\left[\begin{array}{c} 0 \\ \hline \hat{r}_i \end{array}\right] - \alpha_i A Q\left[\begin{array}{c} 0 \\ \hline \hat{p}_i \end{array}\right] \\ &= Q\left[\begin{array}{c} 0 \\ \hline \hat{r}_i \end{array}\right] - \alpha_i Q(Q^TAQ)\left[\begin{array}{c} 0 \\ \hline \hat{p}_i \end{array}\right] \\ &= Q\left[\begin{array}{c} 0 \\ \hline \hat{r}_i - \hat{\alpha}_i B p_i \end{array}\right] = Q\left[\begin{array}{c} 0 \\ \hline \hat{r}_{i+1} \end{array}\right] \end{aligned}$$

That $\hat{\beta}_i = \beta_i$ follows from exactly the same reasoning that showed equality for the numerators of α_i and $\hat{\alpha}_i$. It is equally easy to show that

$$p_{i+1} = r_{i+1} + \beta_i p_i \iff Q^T p_{i+1} = \left[\begin{array}{c} 0 \\ \hline \hat{r}_{i+1} + \hat{\beta}_i \hat{p}_i \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \hat{p}_{i+1} \end{array}\right].$$

Thus the conjugate gradient iteration will succeed in solving a consistent system.

We should note that the condition that $x_i^T n = 0$ is required only to assure that the solution also satisfies this property. If the minimal length solution is not desired, this condition can be ignored. The condition that $b^T n = 0$, that $Ax = b$ be consistent, is essential. If the system is not consistent, the initial residual r_i, and all successive residuals, will have the same component in the direction of the nullspace. Suppose that b (and r_1 and p_1) has a component $\gamma \cdot n$ of the nullspace. The recursion for $\{r_i\}$

$$r_{i+1} = r_i - \alpha_i A p_i$$

shows that r_{i+1} has a component $\gamma \cdot n$ of the nullspace for all i. This, of course, also follows from the property that r_{i+1} is a residual vector for Ax_i and b. The directions $\{p_i\}$, and the approximate solutions $\{x_i\}$, have components of n which increase with i (in fact, rapidly).

It suffices to examine the recursion for p_i

$$p_{i+1} = r_{i+1} + \beta_i p_i$$

where $\beta_i > 0$. If δ_i is the component of n in the vector p_i,

$$\delta_{i+1} = (\gamma + \beta_i \delta_i) = (\gamma + \beta_i (\gamma + \beta_{i-1} \cdot \delta_{i-1})) = \ldots$$

Thus the direction's component of the nullspace increases. At the same time, p_{i+1} is shorter than r_{i+1} in the A norm, so the relative component in the direction of the nullspace grows. In the limit, the directions may become nearly parallel (they are still A-orthogonal, but A does not induce a metric). We can see this also by looking at the limiting case when a iteration is started with an actual solution vector. The residual vector (which is not a descent direction) and the initial direction, are in the nullspace—the initial step α_1 is infinite.

The last example given above implies that the convergence rate for obtaining the least squares solution to

an inconsistent system no longer depends on $\kappa(B)$. The convergence obtained in practice may be much slower than the bound obtainable for a consistent system. In practice, we are interested only in minimizing the residual, which we can ensure by working with a consistent system. In actual finite precision computing, the residuals and directions may wander slightly into the nullspace, in which case we are in the same situation as if we started with a slightly inconsistent system. The effects will be negligible unless the rate of convergence is very slow. In any case, the effects may be suppressed by explictly reorthogonalizing the directions p_i to the nullspace.

In the preconditioned conjugate gradient algorithm we replace the system

$$Ax = b$$

by

$$Cz = d$$

where

$$C = VAV,$$

V some positive definite symmetric matrix. In the special case of a semi-definite A, a natural choice for a scaling matrix may also be semi-definite. This is the case for the two operators discussed in section 3. We have two possible cases to consider: V positive definite and V semi-definite.

The case of a positive definite V is little changed from the ordinary semi-definite conjugate gradient algorithm. We take $M = (V^2)^{-1}$, or more directly, $V = M^{-1/2}$. The rank of the matrix C is $k-1$ and its nullspace is

$$\text{span}\{M^{1/2}n_A\}$$

where n_A denotes a non-zero vector in the nullspace of A. The convergence condition for C required that $\bar{r}_i$ and $\bar{p}_i$ be orthogonal to $M^{1/2}n_A$.
The former condition will hold if $Ax = b$ is consistent since then

$$\bar{r}_i = M^{-1/2}(b - Ax_i)$$

and

$$(b - Ax_i)^T n_A = 0$$

The latter condition is equivalent to

$$p_i \perp Mn_A$$

since

$$p_i = M^{-1/2}\bar{p}_i$$

But clearly $p_i = M^{-1} r_1$ is orthogonal to $M n_A$, since $r_1^T n_A = 0$. By induction, if $p_i^T M n_A = 0$,

$$p_{i+1}^T M n_A = (r_{i+1}^T M^{-1}) M n_A + \beta_i (p_i^T M n_A) = r_{i+1}^T n_A = 0.$$

Thus, convergence is governed by the pseudo-condition number $\frac{\lambda_n(C)}{\lambda_2(C)}$, the residual vectors r_i remain those of a consistent system, and the coefficients α_i and β_i are determined by an iteration remaining in a subspace

of the range of C. However, the scaled directions p_i are allowed to venture into the nullspace of A, and hence, the solution vector x is no longer necessarily the minimal length solution. This may be repaired easily at the end of the iteration by computing

$$x^+ = x - \frac{x^T n_A}{n_A{}^T n_A} \; n_A .$$

The other natural case is one in which the scaling matrix M is also positive semi-definite, with the nullspace generated by a known vector n_M. The linear operator corresponding to M^{-1} in the definite case will be M^*, the pseudo-inverse of M. (For our purposes, we will need only a method for solving consistent systems $Mz = w$. Our knowledge of the nullspace of M then enables us to compute the minimal length least squares solution, M^*u, for any right hand side u). Formally, the matrix V appearing in the definition of C is $(M^*)^{1/2}$, so

$$C = (M^*)^{1/2} A (M^*)^{1/2} . \tag{6}$$

Several conditions which must be met by M are immediate. Since nullspace $(C) \supseteq$ nullspace (M), the rank of M must be at least the rank of A. For our specific problem rank (M) must be at least $k-1$. Otherwise C has rank at most $k-2$ and cannot possibly span the entire $k-1$ dimensional space of possible solutions to $Ax = b$. To guarantee that C has rank $k-1$, we must have rank $(M) \geq k-1$ and also that the nullspace of M is not orthogonal to the nullspace of A. If the latter condition fails, we know that n_A lies in the range of M (since it is orthogonal to the orthogonal complement of the range of M). Hence, by symmetry of M, n_A lies in the range of $(M^*)^{1/2}$. There exists a vector z such that $(M^*)^{1/2} z = n_A$ and $z \perp n_M$. We would have:

$$C n_M = (M^*)^{1/2} A ((M^*)^{1/2} n_M) = (M^*)^{1/2} A 0 = 0$$

$$Cz = (M^*)^{1/2} A ((M^*)^{1/2} z) = (M^*)^{1/2} A n_A = (M^*)^{1/2} 0 = 0$$

Thus, the dimension of the nullspace of C would be at least two.

Recall now the motivation for the preconditioned algorithm. The rate of convergence of the conjugate gradient algorithm is determined by the condition number, $\varkappa(A)$, which is large for most discrete elliptic operators. To accelerate the coverage of the conjugate gradient algorithm, we replaced A by the preconditioned operator $C = M^{-1/2} A M^{-1/2}$, where we presume that $\varkappa(C) \approx 1$. Our presumption can only be true if $\varkappa(A) \approx \varkappa(M)$, i.e., the preconditioning matrix must have roughly the same poor conditioning as has A. This requirement follows from the inequality

$$\varkappa(C) \geq \max \left\{ \frac{\varkappa(A)}{\varkappa(M^{-1})}, \frac{\varkappa(M^{-1})}{\varkappa(A)} \right\}$$

where we note that $\varkappa(M^{-1}) = \varkappa(M)$. A brief proof of this inequality is given below:

$$\varkappa(C) = \lambda_{max}(C) / \lambda_{min}(C).$$

However, the eigenvalues of C are the same as the eigenvalues of $M^{-1} A$. We can bound the eigenvalues of this latter product as:

$$\lambda_{max}(M^{-1} A) \geq \lambda_{max}(M^{-1}) \cdot \lambda_{min}(A),$$

$$\lambda_{min}(M^{-1} A) \leq \lambda_{min}(M^{-1}) \cdot \lambda_{max}(A).$$

It follows that

$$\kappa(C) \geq \frac{\lambda_{min}(A)}{\lambda_{max}(A)} \cdot \frac{\lambda_{max}(M)}{\lambda_{min}(M)}$$

$$= \kappa(M) / \kappa(A).$$

The other inequality above follows by a similar argument.

We can obtain suitable preconditioning matrices for most positive definite discrete elliptic operators only by using poorly conditioned matrices. The required poor conditioning of M may be even worse in the case where both A and M are semi-definite. The preconditioning breaks down if

$$n_A^T n_M = 0.$$

We shall now show that the preconditioning nearly breaks down, in the sense that M must be very ill-conditioned, if the nullspaces are almost orthogonal. Assume that n_A and n_M have length one, and that

$$n_A^T n_M = \delta,$$

which is much less than one. We can write A and M^+ in their respective eigendecompositions

$$A = U D_A U^T,$$

$$M^+ = V D_{M^+} V^T,$$

where the zero eigenvalue of A and of M^+ is given first. Then

$$C = M^{+1/2} A M^{+1/2} = S S^T,$$

where S is defined by

$$S = V D_{M^+}^{1/2} V^T U D_A^{1/2}.$$

For each of the matrices S, M^+, A, and X, define the condition number of the matrix as the condition number of the matrix restricted to the orthogonal complement of its nullspace. Then

$$\kappa(S) = \kappa[(D_{M^+}^{1/2})(X)(D_A^{1/2})].$$

In the above equation, the matrix X is the $(k-1)$ by $(k-1)$ matrix obtained by removing the first row and column from $V^T U$, that is,

$$V^T U = \left[\begin{array}{c|c} \delta & a^T \\ \hline b & X \end{array}\right]$$

We now note that the inequality

$$\kappa(BC) \geq \max\left\{\frac{\kappa(B)}{\kappa(C)}, \frac{\kappa(C)}{\kappa(B)}\right\}$$

holds for unsymmetric matrices B and C. (The proof follows from the triangle inequality for matrix norms,

$$\|B\| \leq \|BC\| \, \|C^{-1}\|.)$$

When we extend this inequality to products of three matrices we obtain the relatively weak result that

$$\varkappa(BCD) \geqslant \max \left\{ \frac{\varkappa(B)}{\varkappa(C)\varkappa(D)}, \frac{\varkappa(C)}{\varkappa(B)\varkappa(D)}, \frac{\varkappa(D)}{\varkappa(B)\varkappa(C)} \right\}$$

In the context of our preconditioned operator, only $\varkappa(X)$ is unknown. However, Alan Cline [11] has shown that when $\delta \ll 1$,

$$\varkappa(X) \cong 1 / \delta.$$

Thus, the condition number of the preconditioned operator C is bounded below by:

$$\varkappa(C) \geqslant (1 / \delta)^2 / (\varkappa(A)\varkappa(M)).$$

Since $\varkappa(A)$ is fixed, this implies that $\varkappa(M)$ must be large whenever δ is small.

V. Numerical Results

A code, written in FORTRAN, was developed to implement the preconditioned conjugate gradients scheme discussed in previous sections. Care was taken in the preparation of the code to make it portable and to introduce many comments for clarity. The code has been run successfully under a variety of conditions and has been compared with analytical results to determine its accuracy. In this section a description is given of some of the computations used to determine the performance of this code.

The model problem, the pressure equation, was discussed in section III. Special cases of this general non-separable elliptic equation were used to test the code for accuracy. All production runs of this code were performed when the code was imbedded in a larger linear or nonlinear fluid dynamics computation; timing studies were performed in such an environment.

Within the code, two tests are used to terminate the conjugate gradient iteration; these depend upon two specified parameters, the maximum number of iterations (less than or equal to 50) and a maximum relative residual, ε. (The relative residual is defined as the norm of the residual divided by the norm of the right hand side in the scaled problem discussed in sections 2 and 4.) In all successful computations to date, the iteration is terminated after a relatively small number of iterations by the relative residual norm satisfying the criterion that it be below ε.

To test the code under the simplest conditions, a Poisson equation on the unit square was discretized and solved with homogeneous Neumann conditions applied at the boundary. In continuous form this problem is

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \tilde{f}(x,y) \equiv \cos(x\pi\, x) \cos(l\pi\, y)$$

on $0 \leqslant x \leqslant 1$ and $0 \leqslant y \leqslant 1$. The boundary conditions are

$$\frac{\partial p}{\partial x} = 0 \text{ at } x = 0 \text{ and } x = 1,$$

$$\frac{\partial p}{\partial y} = 0 \text{ at } y = 0 \text{ and } y = 1.$$

The solution to this problem is

$$p(x,y) = -\frac{1}{(l\pi)^2 + (x\pi)^2} \cos(x\pi\, x) \cos(l\pi\, y)$$

When this problem is discretized to second order accuracy on the "staggered grid" (a grid displaced one half incremental unit δx in the x-direction and one half incremental unit δy in the y-direction) as discussed in section III, it can be written

$$\frac{1}{\delta x^2}(\bar{p}_{i+1,j} - 2\bar{p}_{ij} + \bar{p}_{i-1,j}) + \frac{1}{\delta y^2}(\bar{p}_{i,j+1} - 2\bar{p}_{ij} + \bar{p}_{i,j-1}) = \bar{f}_{ij}$$

Where

$$\bar{f}_{ij} = \cos[\frac{k\pi}{m}(i - \tfrac{1}{2})] \cos[\frac{l\pi}{n}(j - \tfrac{1}{2})]$$

for

$$1 \leqslant i \leqslant m \text{ and } 1 \leqslant j \leqslant n.$$

Here δx is the mesh spacing in the x-direction, $\delta x = 1/m$ (m is the number of mesh cells in the x-direction) and δy is the mesh spacing in the y-direction, $\delta y = 1/n$ (n is the number of cells in the y-direction). The boundary conditions are

$$\bar{p}_{0,j} = \bar{p}_{1,j} \text{ and } \bar{p}_{m+1,j} = \bar{p}_{m,j} \text{ for } 1 \leqslant j \leqslant n$$

$$\bar{p}_{i,0} = \bar{p}_{i,1} \text{ and } \bar{p}_{i,n+1} = \bar{p}_{i,n} \text{ for } 1 \leqslant i \leqslant m$$

The solution to this linear algebraic system is

$$\bar{p}_{ij} = -\frac{1}{\left(2m \sin\frac{k\pi}{2m}\right)^2 + \left(2n \sin\frac{l\pi}{2n}\right)^2} \cos[\frac{k\pi}{m}(i-\tfrac{1}{2})] \cos[\frac{l\pi}{n}(j-\tfrac{1}{2})]$$

$$\text{for } 0 \leqslant i \leqslant m+1, 0 \leqslant j \leqslant n+1$$

The solution to this simple discretized Poisson equation was computed using the code, and the solution compared with the analytical solution given above.

A small test problem, $m = n = 7$, was used to determine the accuracy obtainable when $\varepsilon = 10^{-6}$. Comparison with the exact solution demonstrated that the components of the solution vector obtained from the computation agreed to at least six significant figures with the exact solution. Generally the agreement was much better, being seven or eight significant figures for most values of i and j. (Note that the UNIVAC 1108, on which all computations were run, carries about eight significant figures.)

As a larger test problem, the equations with $m = n = 31$ were solved with $\varepsilon = 10^{-6}$. For this computation agreement was obtained to a few parts in the sixth significant figure.

A second test problem, a discretized approximation to a separable elliptic equation, was also solved analytically and using the code. Comparison of these results indicated that the accuracy was similar to that obtained in the first test problem.

The code was then imbedded in a linear finite difference computation obtained from a second-order discretization of a set of equations arising in fluid dynamics. These equations describe two-dimensional internal gravity waves in a stratified ambient fluid within a rectangular enclosure. The interest in this problem is discussed, the continuous and discrete equations are presented and exact analytical solution to the continuous and discrete problem are given by Baum and Rehm [8]. Additional linear computations using this code on a somewhat more general fluid-flow problem are described in Rehm and Baum [7]. In this latter paper the more general linear finite difference equations are given, and the computational procedure for solving them is presented. The manner in which the preconditioned conjugate gradients code is used in solving the elliptic pressure equation is discussed. In all of these linear computations, the equation for the pressure is separable; it is only in the general, nonlinear computations that the equation for the pressure is nonseparable.

For the linear computations reported in these papers, the number of iterations required to obtain a relative residual error less than $\varepsilon = 10^{-5}$ or 10^{-6} generally varied between 2 and 4. A large number of such computations have been run.

A representative one is a calculation for which $m = 15$, $n = 16$ and $\varepsilon = 10^{-6}$; in this computation the pressure-solver was called 200 times. Two iterations to convergence were taken ten of the first eleven times it

was called, each call using about 0.5 s of CPU time on the NBS Univac 1108. Thereafter, each subsequent call required only one iteration to convergence taking about 0.35 s of CPU. An indication of the rate of covergence of the algorithm is obtained by taking

$$\delta \equiv \frac{\log_{10}(\text{Initial relative residual}) - \log_{10}(\text{final relative residual})}{\text{number of iterations}}$$

For the computation reported above, this rate of convergence was about 4.5 when two iterations were taken and about 5.5 when one iteration was taken.

It should be noted that these computations determine a flow field which evolves with time. Except for the first time step, the pressure vector at the previous time step is used as the initial guess for the pressure vector each time the elliptic-solver is called. Hence the guess at each calling is quite good and convergence is rapid.

Computations of a set of nonlinear finite difference equations generalizing the linear ones described above have also been run. As noted previously, in this case the elliptic equation for the pressure is nonseparable. In a limited number of computations, the PCG code has been found to perform well in this case also. A representative computation, one for which $I = 31, J = 31$ and $\varepsilon = 10^{-5}$, was found to take between 2 and 5 iterations for convergence at each call. The CPU time taken was slightly over 2 s for 2 iterations and slightly under 5 s for 5 iterations (very roughly a second per iteration generally). In this calculation δ defined above was found to vary between about one and two.

In the Appendix to [15] a listing of the elliptic-solver, called FASTSL, is given. To use this code, subroutines from EISPACK, the Argonne Code Center package, and BLKTRI, a subroutine in the NCAR package developed by Schwarztrauber and Sweet [10] are required.

The authors wish to acknowledge the help of our colleagues, Mr. Martin Cordes of the Center for Applied Mathematics (C.A.M.) and Dr. Howard Baum of the Center for Fire Research (C.F.R.). This study would not have been completed without their help. We also wish to thank Drs. Dianne O'Leary and Francis Sullivan for reading and correcting the manuscript. One of us (J. L.) would like to express his appreciation to Drs. B. H. Colvin and F. C. Johnson for providing the opportunity to undertake this study in the C.A.M. during the summer of 1977.

VI. References

[1] Rice, John, Algorithm Progress in Solving Partial Differential Equations, SIGNUM Newsletter, Vol. **11**, No. 4, Dec. 1976.

[2] Concus, Paul and Golub, Gene H., Use of fast direct methods for the efficient numerical solution of non separable elliptic equations, SIAM J. Numer. Anal., **10**, 6 (1973).

[3] Concus, Paul; Golub, Gene H. and O'Leary, Dianne P., A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in *Sparse Matrix Computations*, J. Bunch and D. Rose, ed. (Academic Press, New York, 1975).

[4] O'Leary, Dianne P., Hybrid Conjugate Gradient Algorithms, Thesis, Computer Science Department, Stanford University, 1976, TR STANS-CS 548.

[5] O'Leary, Dianne P. and Widlund, Olof, Capacitance Matrix Methods for the Helmholtz Equation on General Three Dimensional Regions, Math of Comp. **33**, No. 147, 849-879 (July 1979).

[6] Rehm, Ronald G. and Baum, Howard R., The Equations of Motion for Thermally Driven, Buoyant Flows, J. Res. Nat. Bur. Std. (U.S.), **83**, No. 3, 297-308 (May-June 1978).

[7] Rehm, Ronald G.; Baum, Howard R.; Lewis, John G.; Cordes, Martin R.; A Linearized Finite—Difference Computation of Fluid Heating in an Enclosure, NBSIR 79-1754, May 3, 1979.

[8] Baum, Howard R. and Rehm, Ronald G., Finite Difference Solutions for Internal Waves in Enclosures, N.B.S. Internal Report, in preparation.

[9] Meijerink, J. A. and van der Vorst, H. A. An Iterative Solution for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix, Math. of Comp., **31**, 137, (Jan. 1977).

[10] Schwarztrauber, Paul and Sweet, Roland, Efficient FORTRAN Subprograms for the Solution of Elliptic Partial Differential Equations, NCAR Technical Note IA-109, July 1975.

[11] Cline, A., Private Communication.

[12] Reid, J. K., On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations, in *Large Sparse Sets of Linear Equations*, J. K. Reid ed., (Academic Press, New York, 1971), 231-254.

[13] Hestenes, M. R. and Stiefel, E., Methods of Conjugate Gradients for Solving Linear Systems, J. Res. Nat. Bur. Std. (U.S.), **49**, No. 6, 409-436 (Dec. 1952).

[14] Daniel, J. W., The cg method for linear and nonlinear operator equations, SIAM J. Numer. Anal. **4** (1967) 10-26.

[15] Lewis, J. G. and Rehm, R. G., The Numerical Solution of a Nonseparable Elliptic Partial Differential Equation by Preconditioned Conjugate Gradients, NBSIR80-2056.

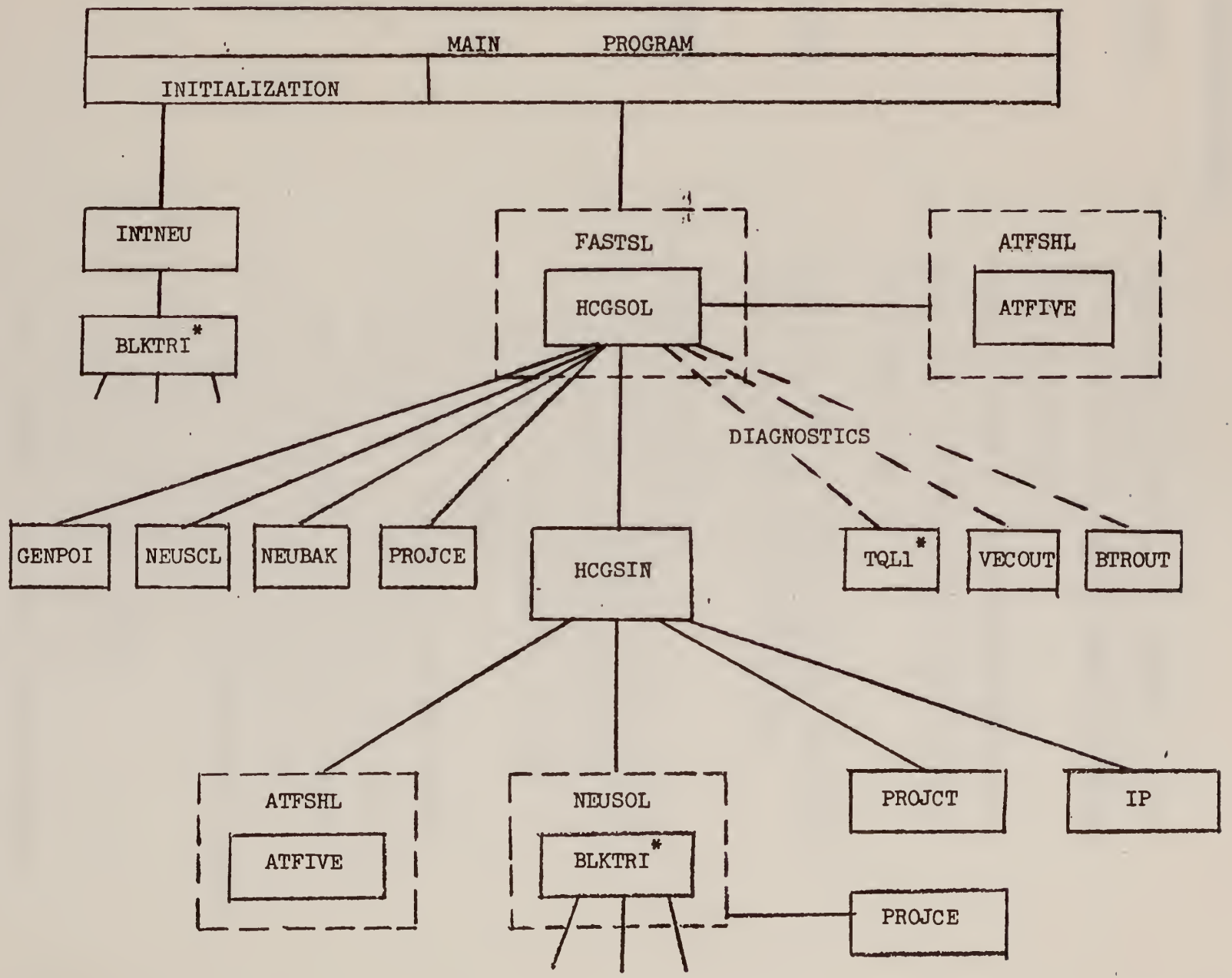
APPENDIX

DIAGRAM OF SUBROUTINE LINKAGES

BRIEF DESCRIPTION OF SUBROUTINES

LISTING OF ALL SUBROUTINES

DIAGRAM OF SUBROUTINE LINKAGES



* Subroutines from external libraries

Brief Description

INTNEU - Initializes the NCAR Routine BLKTRI to solve $\nabla^2 u = f$ on the rectangular grid (with Neumann Boundary Conditions)
Generates parameter vector PPARM which must be passed back to BLKTRI when we actually solve the pressure equation

FASTSL - Separates required working storage into pieces for the coordinating subroutine HCGSOL

HCGSOL - Coordinates the pieces

a) Calls GENPOI to generate the discrete five point operator for the pressure equation. The matrix's representation is stored in the vector 'APARM'

b) Calls NEUSCL to compute a diagonal scaling to make the discrete pressure equation have the same diagonal as the Poisson equation.

c) Calls HCGSIN with the diagonally scaled operator. HCGSIN uses preconditioned conjugate gradient algorithm with
A ≡ scaled pressure equation
P ≡ discrete Poisson equation

d) Calls NEUBAK to undo the diagonal scaling to get a solution of the original pressure equation

HCGSOL also coordinates diagnostic information

HCGSIN - Preconditioned congugate gradients repeated by calls to subroutine to compute

Ax for some x

and a subroutine to solve Py = x

for some x

The names of the subroutines are parameters from HCGSOL. Here they are

ATFSHL and NEUSOL

ATFSHL - Unpacks representation of scaled pressure equation in APARM to pass it to ATFIVE, which actually performs the matrix-vector product.

NEUSOL - Unpacks representation of POISSON equation in PPARM, to pass it to BLKTRI to actually use cyclic reduction to solve the discrete Poisson equation.

Other Subroutines

PROJCT, PROJCE - To compute the projection of an arbitrary vector on the orthogonal complement of some vector (PROJCT) or the vector of all one's (PROJCE)

IP - To compute the inner product of two vectors

BTROUT - Prints out a symmetric block matrix with five non-zero diagonals used only for diagnostic output.

VECOUT - Prints a vector in a grid-like form. Used only for diagnostic output.

Subroutines from external libraries

TQL1 - From EISPACK. Computes all eigenvalues of tri-diagonal matrix, (produced implicitly by conjugate gradients), used to estimate convergence rate of conjugate gradients

BLKTRI - NCAR routine. Used to solve $\nabla^2 P = f$ with Neumann boundary conditions.

```
@:ELT,L -PF1-.ATFIVE
ELT 8R1 S74Q1C 07/11/79 08:48:30 (1)
     1.           01            SUBROUTINE   ATFIVE   (M, N, D1, D2, DM, X, AX)
     2.           00      C
     3.           00      C     ========================================
     4.           00      C        MULTIPLY  X  BY  A  WHERE  A  IS THE REPRESENTATION
     5.           01      C        OF A SPARSE FIVE POINT OPERATOR
     6.           00      C     ========================================
     7.           00      C
     8.           00            INTEGER     M, N
     9.           00            REAL        X(1), AX(1), D1(1), D2(1), DM(1)
    10.           00      C
    11.           00      C
    12.           00      C
    13.           00      C
    14.           00            INTEGER     MP1, NMM1, NMMM, NM, I, NMMMP1
    15.           00      C
    16.           00            NM = N * M
    17.           00      C
    18.           00      C        ... 1ST BLOCK ...
    19.           00            AX(1) = D1(1) * X(1)  +  D2(1) * X(2)  +  DM(1) * X(M+1)
    20.           00            DO 100 I = 2, M
    21.           00               AX(I) = D1(I) * X(I)  +  D2(I) * X(I+1)  +  D2(I-1) * X(I-1)
    22.           00           1          +  DM(I) * X(I+M)
    23.           00         100 CONTINUE
    24.           00      C
    25.           00      C        ... INTERIOR BLOCKS ...
    26.           00      C
    27.           00            MP1 = M + 1
    28.           00            NMMM = NM - M
    29.           00            DO 200 I = MP1, NMMM
    30.           00               AX(I) = D1(I) * X(I)  +  DM(I-M) * X(I-M)  +  D2(I-1) * X(I-1)
    31.           00           1          +  DM(I) * X(I+M)  +  D2(I) * X(I+1)
    32.           00         200 CONTINUE
    33.           00      C
    34.           00            NMMMP1 = NMMM + 1
    35.           00            NMM1 = NM - 1
    36.           00      C
    37.           00      C        ... FINAL BLOCK ...
    38.           00      C
    39.           00            DO 300 I = NMMMP1, NMM1
    40.           00               AX(I) = D1(I) * X(I)  +  DM(I-M) * X(I-M)  +  D2(I-1) * X(I-1)
    41.           00           1          +  D2(I) * X(I+1)
    42.           00         300 CONTINUE
    43.           00      C
    44.           00            AX(NM) = D1(NM) * X(NM)  +  D2(NMM1) * X(NMM1)
    45.           00           1         +  DM(NMMM) * X(NMMM)
    46.           00      C
    47.           00      C
    48.           00            RETURN
    49.           00            END

END ELT.  ERRORS: NONE.  TIME:  0.209 SEC.  IMAGE COUNT: 49

@HDG,P  ******  ATFSHL  ******
```

```
@:ELT,L -PF1-.ATFSHL
ELT 8R1 S74Q1C 07/11/79 08:48:32 (3)
      1.           03            SUBROUTINE   ATFSHL   (APARM, X, AX)
      2.           00      C
      3.           00      C     ==================================================
      4.           00      C     SHELL ROUTINE FOR MATRIX-VECTOR MULTIPLY FOR DISCRETE
      5.           02      C     FIVE POINT OPERATOR ON A RECTANGULAR GRID.
      6.           00      C     ==================================================
      7.           00      C
      8.           03            REAL         APARM(1), X(1), AX(1)
      9.           00      C
     10.           03      C        VECTOR  APARM  CONTAINS PARAMETERS FOR MULTIPLY
     11.           00      C
     12.           00            INTEGER         M, N, NM, SUBD1, SUBD2, SUBD3
     13.           00      C
     14.           02      C        DECODE PARAMETERS AND PASS ON TO  'ATFIVE'
     15.           00      C
     16.           03            M = IFIX(APARM(1))
     17.           03            N = IFIX(APARM(2))
     18.           00            NM = N * M
     19.           00      C
     20.           00            SUBD1 = 3
     21.           00            SUBD2 = 3 + NM
     22.           00            SUBD3 = SUBD2 + NM
     23.           00      C
     24.           03            CALL ATFIVE (M, N, APARM(SUBD1), APARM(SUBD2), APARM(SUBD3),
     25.           00           1             X, AX)
     26.           00      C
     27.           00            RETURN
     28.           00            END

END ELT.  ERRORS: NONE. TIME:  0.159 SEC.  IMAGE COUNT: 28

@HDG,P  ******  BTROUT  ******
```

```
@:ELT,L -PF1-.BTROUT
ELT 8R1 S74Q1C 07/11/79 08:48:33 (4)
     1.          00            SUBROUTINE    BTROUT   (M, N, D1, D2, D3)
     2.          00      C
     3.          00      C     ------------------------------------------------------------
     4.          00      C        PRINT THE UPPER TRIANGLE OF A BLOCK TRIDIAGONAL MATRIX
     5.          00      C        RESULTING FROM A FINITE DIFFERENCE APPROXIMATION ON A
     6.          00      C        RECTANGULAR GRID USING A FIVE POINT OPERATOR.
     7.          00      C
     8.          02      C           M      : BLOCK SIZE
     9.          00      C           N      : NUMBER OF BLOCKS
    10.          00      C           D1     : MAIN DIAGONAL
    11.          00      C           D2     : 1ST SUPERDIAGONAL
    12.          00      C           D3     : MAIN DIAGONAL OF OFF-DIAGONAL BLOCKS
    13.          00      C     ------------------------------------------------------------
    14.          00      C
    15.          00            INTEGER     M, N
    16.          01            REAL        D1(1), D2(1), D3(1)
    17.          00      C                ACTUAL DIMENSION IS M * N
    18.          00      C
    19.          02            INTEGER             I, J, I1, I2
    20.          04      C
    21.          04            INTEGER     PRINTR
    22.          04            DATA        PRINTR / 6 /
    23.          01      C
    24.          04            WRITE (PRINTR, 60000)
    25.          02            DO 10 J = 1, N
    26.          04               I1 = (J-1) * M  +  1
    27.          04               I2 = J * M
    28.          04               WRITE (PRINTR, 60100) (D1(I), D2(I), D3(I), I = I1, I2)
    29.          04               WRITE (PRINTR, 60200)
    30.          00         10 CONTINUE
    31.          00      C
    32.          00            RETURN
    33.          00      C
    34.          00      C     ------------------------------------
    35.          00      C
    36.          00      60000 FORMAT ('0OUTPUT BY  R O W S  OF MATRIX'/)
    37.          00      60100 FORMAT (1P2E15.7, 10X, E15.7)
    38.          00      60200 FORMAT (' ')
    39.          00            END

END ELT.  ERRORS: NONE. TIME:  0.201 SEC. IMAGE COUNT: 39

@HDG,P ****** DIRBAK ******
```

```
@:ELT,L -PF1-.FASTSL
ELT 8R1 S74Q1C 07/11/79 08:48:36 (15)
     1.          11            SUBROUTINE   FASTSL   (M, N, MMAX, RHO, PRESSR, RHS, RESID,
     2.          12           1                      LEFTB, RIGHTB, LOWERB, UPPERB, PPARM,
     3.          12           2                      DELTAX, DELTAY, ITERS, RTOL,
     4.          12           3                      WORK, CNTL)
     5.          11      C
     6.          11             INTEGER     M, N, MMAX
     7.          12             REAL        RHO(MMAX,N), PRESSR(MMAX,N), RHS(MMAX,N),
     8.          12            1            RESID(MMAX,N)
     9.          12             REAL        LEFTB(N), RIGHTB(N), LOWERB(M), UPPERB(M), PPARM(1)
    10.          12             REAL        DELTAX, DELTAY, RTOL
    11.          12             INTEGER     ITERS
    12.          11             REAL        WORK(1)
    13.          11             INTEGER     CNTL
    14.          11      C
    15.          11      C     ------------------------------------------------------------------
    16.          11      C     SOLVE THE PRESSURE EQUATION WITH NEUMANN BOUNDARY CONDITIONS ON
    17.          11      C     A RECTANGLE, USING HYBRID CONJUGATE GRADIENTS WITH CYCLIC
    18.          11      C     REDUCTION AS THE FAST METHOD INSIDE THE ITERATION.
    19.          11      C     ------------------------------------------------------------------
    20.          11      C
    21.          11      C     PARAMETERS:
    22.          12      C          (STARRED PARAMETERS ARE ALTERED BY THE SUBROUTINE)
    23.          12      C
    24.          11      C         M       - THE NUMBER OF GRID POINTS IN THE HORIZONTAL DIRECTION
    25.          11      C         N       - THE NUMBER OF GRID POINTS IN THE VERTICAL DIRECTION
    26.          11      C         MMAX    - THE DECLARED ROW DIMENSION OF THE GRID ARRAYS
    27.          11      C                   (MUST BE AT LEAST M, MAY BE THE SAME AS MD.
    28.          11      C
    29.          11      C         RHO     - THE DENSITIES AT THE GRID POINTS
    30.          12      C        *PRESSR  - AN INITIAL GUESS AT THE SOLUTION PRESSURE.  THE FINAL
    31.          11      C                   SOLUTION IS RETURNED IN THIS ARRAY.
    32.          12      C        *RHS     - THE RIGHT HAND SIDE OF THE ELLIPTIC EQUATION
    33.          11      C                   (SHOULD NOT INCORPORATE THE BOUNDARY CONDITIONS.)
    34.          12      C                   THE VALUES IN THE ARRAY WILL BE DESTROYED BY THIS
    35.          12      C                   SUBROUTINE.
    36.          12      C        *RESID   - THE RESIDUAL 'VECTOR' FOR THE LINEAR DIFFERENCE
    37.          12      C                   EQUATIONS EVALUATED AT THE FINAL SOLUTION  'PRESSR'.
    38.          11      C                   (THIS ARRAY DOES NOT NEED TO BE INITIALIZED.)
    39.          11      C
    40.          11      C         LEFTB   - THE NORMAL DERIVATIVE OF  PRESSR, DIVIDED BY  RHO,
    41.          11      C                   ON THE LEFT BOUNDARY.
    42.          11      C         RIGHTB  - THE BOUNDARY DERIVATIVES ON THE RIGHT BOUNDARY
    43.          11      C         LOWERB  - THE BOUNDARY DERIVATIVES ON THE LOWER BOUNDARY
    44.          11      C         UPPERB  - THE BOUNDARY DERIVATIVES ON THE UPPER BOUNDARY
    45.          12      C
    46.          12      C         PPARM   - A VECTOR OF PARAMETERS TO THE CYCLIC REDUCTION
    47.          12      C                   FAST POISSON SOLVER.  THIS VECTOR IS INITIALIZED
    48.          12      C                   BY THE SUBROUTINE  'INTNEU', WHICH MUST BE CALLED
    49.          12      C                   (ONCE) BEFORE THIS FAST NON-SEPARABLE SOLVER IS
    50.          12      C                   CALLED.  AS LONG AS THE CALLING PROGRAM DOES NOT
    51.          12      C                   ALTER THE CONTENTS OF  PPARM, IT WILL NOT NEED
    52.          12      C                   TO BE REINITIALIZED.
    53.          12      C                   PPARM MUST BE OF LENGTH AT LEAST:
    54.          12      C                      3N + 9M + 2(N+2)(LOG(N)-1) + 12
    55.          12      C                   UNLESS
    56.          12      C                       N IS INAPPROPRIATE FOR THE FAST SOLVER
```

```
      12     C                    AND M IS APPROPRIATE
      12     C                  OR
      12     C                    BOTH ARE APPROPRIATE AND  N  IS THE LARGER,
      12     C                IN WHICH CASE PPARM MUST BE AT LEAST AS LONG AS:
      12     C                  3M + 9N + 2(M+2)(LOG(M)-1) + 12.
      11     C
      11     C       DELTAX - THE GRID SPACING IN THE HORIZONTAL DIRECTION
      11     C       DELTAY - THE GRID SPACING IN THE VERTICAL DIRECTION
      12     C      *ITERS  - ON INPUT, THE MAXIMUM NUMBER OF CG ITERATIONS ALLOWED
      11     C                ON OUTPUT, THE NUMBER OF ITERATIONS ACTUALLY USED.
      11     C      *RTOL   - ON INPUT, THE RELATIVE RESIDUAL DEMANDED OF CONJUGATE
      11     C                GRADIENTS (USING THE SCALED MATRIX).  ON OUTPUT, THE
      12     C                TWO NORM OF  RESID, RELATIVE TO THE NORM OF THE
      12     C                SOLUTION VECTOR.
      11     C
      12     C      *WORK   - A VECTOR OF WORKING STORAGE, OF LENGTH >= 6NM + 2
      11     C
      11     C      *CNTL   - CONTROL FLAG.  ON INPUT, CONTROLS THE AMOUNT OF PRINTING
      11     C                PRODUCED BY THE FAST SOLVER.
      11     C                VALID INPUT VALUES:
      11     C                    0 - SUPPRESS ALL OUTPUT
      11     C                    1 - SUMMARY OF CONJUGATE GRADIENT ITERATIONS
      12     C                    2 - ABOVE, PLUS SUMMARY STATISTICS ON INPUT AND
      12     C                        OUTPUT VECTORS.
      11     C                    3 - ABOVE, PLUS LISTING OF RHS, PRESSR AND RESID
      12     C                    4 - ABOVE, PLUS LISTING OF UNSCALED AND SCALED
      12     C                        MATRICES.
      11     C                VALID OUTPUT VALUES:
      11     C                    0 - SUCCESSFUL SOLUTION
      11     C                    1 - FAILURE TO MEET RESIDUAL TOLERANCE
      12     C                       (SOME FATAL ERRORS CAUSE IMMEDIATE TERMINATION)
      12     C
      12     C     ================================================================
      12     C
      12     C        LOCAL VARIABLES ...
      12     C
      12           INTEGER     NM, SAPARM, SALPHA, SBETA, SGAMMA, SNULL, SF, SX
      15           EXTERNAL    GENPOI, NEUBAK, NEUSCL, NEUSOL, ATFSHL
      12           LOGICAL     FLIP
      12     C
      12           INTEGER     PRINTR
      12           DATA        PRINTR / 6 /
      12     C
      12     C
      12     C        SPLIT THE WORK VECTOR INTO ARRAYS FOR CONJUGATE GRADIENTS
      12     C        AND CALL THE COORDINATING SUBROUTINE  HCGSOL.
      12     C
      12     C
      12           NM = N * M
      12           SAPARM = 1
      12           SALPHA = SAPARM + 2
      12           SBETA  = SALPHA + NM
      12           SGAMMA = SBETA  + NM
      12           SNULL  = SGAMMA + NM
      12           SF     = SNULL  + NM
      12           SX     = SF     + NM
      12     C
      12     C           DECIDE WHICH ONE-DIMENSIONAL ORDERING WILL BE MOST
```

```
C           APPROPRIATE FOR THE FAST SOLVER.
C
C           THE SECOND SUBSCRIPT MUST HAVE A LIMIT WHICH IS ONE
C           LESS THAN A POWER OF TWO (FOR SUBROUTINE BLKTRI).
C           SWITCH SUBSCRIPTS TO MEET THIS REQUIREMENT OR TO
C           PUT THE SMALLER LIMIT SECOND IF BOTH ARE APPROPRIATE.
C
C              THE INITIALIZING SUBROUTINE FOR THE SCALING MATRIX
C              MUST MAKE THE SAME DECISION ON THE ORDERING.
C
C
      FLIP = (  (N .NE. 2**(LOG2(N+1)) - 1)    .OR.
     1          ( (N. EQ. 2**(LOG2(N+1)) - 1) .AND.
     2            (M .EQ. 2**(LOG2(M+1)) - 1) .AND.
     3            (M .LT. N) )  )
C
C
C        CHECK DIMENSIONS OF SYSTEM -- MAKE SURE THAT THE
C        FAST SOLVER HAS BEEN INITIALIZED WITH THE SAME DIMENSIONS
C        AS ARE BEING USED HERE.
C        NB:  IF THE STRUCTURE OF THE PARAMETER VECTOR 'PPARM' IS
C           CHANGED (AS FOR A DIFFERENT FAST SOLVER), THIS CODE
C           MAY BE AFFECTED.
C
      IF (    (FLIP .AND.
     1           (IFIX(PPARM(2)) .NE. N) .AND. (IFIX(PPARM(3)) .NE. M) )
     2   .OR.
     3       (.NOT. FLIP    .AND.
     4          (IFIX(PPARM(2)) .NE. M) .AND. (IFIX(PPARM(3)) .NE. N) )
     5   .OR.
     6       (PPARM(1) .EQ. 0.0) )  GO TO 1000
C
C
      CALL HCGSOL (M, N, NM, MMAX, FLIP,
     1             CENPOI, NEUSCL, NEUBAK, ATFSHL, NEUSOL,
     2             RHO, RES, PRESSR, WORK(SF), WORK(SX), RESID,
     3             RHS, PRESSR, RESID, WORK(SNULL), WORK(SAPARM),
     4             WORK(SALPHA), WORK(SBETA), WORK(SGAMMA),
     5             LOWERB, UPPERB, LEFTB, RIGHTB, PPARM,
     6             DELTAX, DELTAY, ITERS, RTOL, CNTL)
C
C        (THE SECOND OCCURENCE OF PARAMETERS  RHS, PRESSR, AND RESID  IN
C         THE CALL ABOVE IS AS A WORKING ARRAY.  ALTERATIONS TO THE LOGIC
C         OF HCGSOL COULD REQUIRE ADDITIONAL TEMPORARY SPACE.  NOTE THAT
C         THESE ARRAYS ARE DESTROYED AS SOON AS THE ARRAYS WRK1, WRK2,
C         WRK3 OF HCGSOL ARE USED.)
C
      RETURN
C
C        ERROR HANDLING ...
C
 1000 WRITE (PRINTR, 61000) M, N
      STOP
C
61000 FORMAT ('0**********************************************************'/
     1        '0THE FAST POISSON SOLVER HAS NOT BEEN INITIALIZED'/
     2        ' OR HAS BEEN INITIALIZED WITH DIFFERENT DIMENSIONS.'/
     3        ' DIMENSIONS GIVEN TO  FASTSL:'/
```

```
  173.          12          4       '     M:', I6 /
  174.          12          5       '     N:', I6 /
  175.          12          6       ' **************************************************')
  176.          12            END
```

END ELT. ERRORS: NONE. TIME: 0.651 SEC. IMAGE COUNT: 176

@HDG,P ****** GENC3D ******

```
@:ELT,L -PF1-.GENPOI
ELT 8R1 S74Q1C 07/11/79 08:48:38 (41)
     1.          38            SUBROUTINE  GENPOI  (M, N, MMAX, FLIP, RHO, F, PRESSR,
     2.          38           1                    DELTAX, DELTAY,
     3.          38           2                    LOWERB, UPPERB, LEFTB, RIGHTB,
     4.          38           3                    ALPHA, BETA, GAMMA, FP, XP, DENS)
     5.          38      C
     6.          38      C        GENERATE THE MATRIX CORRESPONDING TO THE DISCRETE FIVE POINT
     7.          38      C        OPERATOR FOR THE NON-SEPARABLE GENERALIZED POISSON EQUATION
     8.          38      C
     9.          38      C           DIV . (1 / RHO(X,Y) * GRAD (P))  = F(X,Y)
    10.          38      C
    11.          38      C        WITH NEUMANN BOUNDARY CONDITIONS
    12.          38      C          NORMAL DERIVATIVE OF P = RHO(XMIN,Y) * LEFTB(Y) FOR  X = XMIN
    13.          38      C                                  RHO(XMAX,Y) * RIGHTB(Y) FOR X = XMAX
    14.          38      C                                = RHO(X,YMIN) * LOWERB(X) FOR Y = YMIN
    15.          38      C                                = RHO(X,YMAX) * UPPERB(X) FOR Y = YMAX
    16.          38      C
    17.          38      C        DOMAIN:  THE RECTANGLE  (XMIN<=X<=XMAX) * (YMIN<=Y<=YMAX)
    18.          38      C           THE FINITE DIFFERENCE GRID IS LOCATED AT:
    19.          38      C               [DELTAX/2, 3DELTAX/2, 5DELTAX/2, ... (2M-1)DELTAX/2]
    20.          38      C            *  [DELTAY/2, 3DELTAY/2, 5DELTAY/2, ... (2N-1)DELTAY/2]
    21.          38      C
    22.          38      C
    23.          38      C        THE INPUT TO THIS SUBROUTINE IS IN THE NATURAL TWO-DIMENSIONAL
    24.          38      C        REPRESENTATION OF THE FINITE DIFFERENCE GRID.  THE MATRIX WHICH
    25.          38      C        RESULTS IS IN A ONE-DIMENSIONAL FORM WHICH IS MORE CONVENIENT
    26.          38      C        FOR GENERAL SPARSE MATRIX TECHNIQUES.
    27.          38      C
    28.          38      C
    29.          38      C     INPUT PARAMETERS:
    30.          38      C        M       THE NUMBER OF MESH POINTS IN THE X DIRECTION
    31.          38      C        N       THE NUMBER OF MESH POINTS IN THE Y DIRECTION
    32.          38      C        MMAX    THE DECLARED ROW DIMENSION OF  RHO  AND  F
    33.          38      C        FLIP    LOGICAL: USE NATURAL ORDER OR TRANSPOSE THEREOF
    34.          38      C        RHO     MMAX BY N DENSITY MATRIX (GIVES DENSITY AT GRID POINTS)
    35.          38      C        F       MMAX BY N MATRIX HOLDING RIGHT HAND SIDE OF PDE
    36.          38      C        PRESSR  MMAX BY N MATRIX HOLDING GUESS AT SOLN OF PDE
    37.          38      C        DELTAX  SPACING BETWEEN HORIZONTAL GRID POINTS
    38.          38      C        DELTAY  SPACING BETWEEN VERTICAL GRID POINTS
    39.          38      C
    40.          38      C        BOUNDARY DATA IS  (1/RHO) * (D.P/D.NORMAL) ...
    41.          38      C        LOWERB  NORMAL DERIVATIVE BOUNDARY DATA FOR  Y = 0
    42.          38      C        UPPERB  BOUNDARY DATA FOR  Y = YMAX
    43.          38      C        LEFTB   BOUNDARY DATA FOR  X = 0
    44.          38      C        RIGHTB  BOUNDARY DATA FOR  X = XMAX
    45.          38      C
    46.          38      C
    47.          38      C     OUTPUT PARAMETERS
    48.          38      C
    49.          38      C        ALPHA   DIAGONAL OF SPARSE SYMMETRIC LINEAR OPERATOR
    50.          38      C        BETA    1ST OFF-DIAGONAL OF SPARSE MATRIX
    51.          38      C        GAMMA   MAIN DIAGONAL OF OFF-DIAGONAL BLOCK OF SPARSE MATRIX
    52.          38      C        FP      REVISED RIGHT HAND SIDE (WITH BOUNDARY CONDITIONS)
    53.          38      C        XP      ONE DIMENSIONAL FORM FOR INITIAL GUESS OF SOLN.
    54.          38      C
    55.          38      C
    56.          38      C           ALL OF THE ABOVE ARE SINGLE DIMENSION VECTORS, OF LENGTH
```

```
57.          38      C          M * N.  THE GRID POINTS ARE STORED IN EQUIVALENT ONE-
58.          38      C          DIMENSIONAL FASHION IN ONE OF TWO ORDERINGS.  THE NATURAL
59.          38      C          (FORTRAN) ORDER IS SCANNING THE GRID ACROSS ROWS FROM LEFT
60.          38      C          TO RIGHT, MOVING FROM BOTTOM ROW TO TOP ROW.  THIS ORDER
61.          38      C          WILL BE USED UNLESS IT IS MORE EFFICIENT (OR NECESSARY) TO
62.          38      C          USE THE TRANSPOSE OF THIS ORDER (UP COLUMNS, FROM LEFT TO
63.          38      C          RIGHT).  FLIP  IS  FALSE  UNLESS THE TRANSPOSED ORDER SHOULD
64.          38      C          BE USED.
65.          38      C
66.          38      C     WORKING STORAGE (TEMPORARY STORAGE)
67.          38      C
68.          38      C        DENS  A VECTOR OF LENGTH AT LEAST  M
69.          38      C
70.          38      C     ------------------------------------------------------------
71.          38      C     P A R A M E T E R   DECLARATIONS
72.          38      C
73.          38            INTEGER       M, N, MMAX
74.          38            LOGICAL       FLIP
75.          38            REAL          RHO(MMAX,N), F(MMAX,N), PRESSR(MMAX,N)
76.          38            REAL          DELTAX, DELTAY
77.          38            REAL          LOWERB(M), UPPERB(M), LEFTB(N), RIGHTB(N)
78.          38            REAL          ALPHA(1), BETA(1), GAMMA(1), FP(1), XP(1)
79.          38            REAL          DENS(MMAX)
80.          38      C
81.          38      C     ------------------------------------------------------------
82.          38      C
83.          38      C     L O C A L   VARIABLES
84.          38      C
85.          38            INTEGER       I, J, IJ
86.          38            REAL          ODX2, ODY2, RDENS, LDENS, DDENS
87.          38      C
88.          38      C     ------------------------------------------------------------
89.          38      C
90.          38      C        INITIALIZE GRID PARAMETERS
91.          38      C
92.          38            ODX2 = 1.0 / DELTAX**2
93.          38            ODY2 = 1.0 / DELTAY**2
94.          38      C
95.          38      C           THE DENSITY FUNCTION IS APPROXIMATED AT THE
96.          38      C        MIDPOINTS OF GRID LINES, SINCE WE NEED APPROXIMATIONS TO
97.          38      C        THE DERIVATIVE OF THE DENSITY FUNCTION.  THE VECTOR 'DENS'
98.          38      C        AND THE VARIABLE 'RDENS'  ARE USED TO PASS THE MIDPOINT
99.          38      C        CALCULATIONS TO THE NEXT ROW AND THE NEXT COLUMN (TO AVOID
100.         38      C        RECALCULATION).
101.         38      C
102.         38      C     ------------------------------------------------------------
103.         38      C
104.         38      C        STRUCTURE
105.         38      C           1) GENERATE VALUES FOR BOTTOM ROW
106.         38      C           2) GENERATE VALUES FOR INTERIOR ROWS
107.         38      C           3) GENERATE VALUES FOR TOP ROW
108.         38      C
109.         38      C           STRUCTURE FOR EACH ROW:
110.         38      C              A) GENERATE VALUE FOR LEFT COLUMN
111.         38      C              B) GENERATE VALUES FOR INTERIOR COLUMNS
112.         38      C              C) GENERATE VALUE FOR RIGHT COLUMN
113.         38      C
114.         38      C     ------------------------------------------------------------
```

```
115.          38      C
116.          38      C
117.          38            DO 200 J = 1, N
118.          38               DO 100 I = 1, M
119.          38      C
120.          38                  IF (.NOT. FLIP)  IJ = (J-1)*M + I
121.          38                  IF (FLIP)        IJ = (I-1)*N + J
122.          38      C
123.          38                  IF ( J .GT. 1 )  GO TO 10
124.          38      C              ... FIRST ROW ...
125.          38                     DDENS = 0.0
126.          41                     UDENS = 2.0 /( RHO(I,1) + RHO(I,2) )
127.          38                     DENS(I) = UDENS
128.          38                     FP(IJ) = F(I,J)  -  LOWERB(I) / DELTAY
129.          38                     GO TO 30
130.          38      C
131.          38        10           IF ( J .EQ. N )  GO TO 20
132.          38      C              ... INTERIOR ROWS ...
133.          38                     DDENS = DENS(I)
134.          41                     UDENS = 2.0 / ( RHO(I,J) + RHO(I,J+1) )
135.          38                     DENS(I) = UDENS
136.          38                     FP(IJ) = F(I,J)
137.          38                     GO TO 30
138.          38      C
139.          38      C              ... LAST ROW ...
140.          38        20           DDENS = DENS(I)
141.          38                     UDENS = 0.0
142.          38                     FP(IJ) = F(I,J)  -  UPPERB(I) / DELTAY
143.          38      C
144.          38        30           IF ( I .GT. 1 )  GO TO 40
145.          38      C              ... FIRST COLUMN ...
146.          38                     LDENS = 0.0
147.          41                     RDENS = 2.0 / ( RHO(1,J) + RHO(2,J) )
148.          38                     FP(IJ) = FP(IJ)  -  LEFTB(J) / DELTAX
149.          38                     GO TO 60
150.          38      C
151.          38        40           IF ( I .EQ. M ) GO TO 50
152.          38      C              ... INTERIOR COLUMNS ...
153.          38                     LDENS = RDENS
154.          41                     RDENS = 2.0 / ( RHO(I,J) + RHO(I+1,J) )
155.          38                     GO TO 60
156.          38      C
157.          38      C              ... LAST COLUMN ...
158.          38        50           LDENS = RDENS
159.          38                     RDENS = 0.0
160.          38                     FP(IJ) = FP(IJ)  -  RIGHTB(J) / DELTAX
161.          38      C
162.          38      C           CALCULATE FIVE POINT OPERATOR AT POINT (I,J)
163.          38      C
164.          38        60        ALPHA(IJ) = -ODX2 * (RDENS + LDENS)
165.          38          1                   -ODY2 * (DDENS + UDENS)
166.          40                  IF (FLIP) GO TO 80
167.          40                     BETA(IJ)  =  ODX2 * RDENS
168.          40                     GAMMA(IJ) =  ODY2 * UDENS
169.          40                     GO TO 90
170.          40      C
171.          40        80           BETA(IJ) =  ODY2 * UDENS
172.          40                     GAMMA(IJ) =  ODX2 * RDENS
```

```
   173.          40      C
   174.          40        90        XP(IJ)     =  PRESSR(I,J)
   175.          38      C
   176.          38       100     CONTINUE
   177.          38       200  CONTINUE
   178.          38      C
   179.          38           RETURN
   180.          38           END
```

END ELT. ERRORS: NONE. TIME: 0.652 SEC. IMAGE COUNT: 180

@HDG,P ****** GENTS3 ******

```
@:ELT,L -PF1-.HCGSIN
ELT 8R1 S74Q1C 07/11/79 08:48:43 (16)
      1.          07          SUBROUTINE   HCGSIN  (NM, MAXITR, RTOL, X, B, R, P, AP, NULLSP,
      2.          09         1                      ATIMES, PSOLVE, APARM, PPARM, LALPHA,
      3.          07         2                      LBETA, CNTL)
      4.          06     C
      5.          06          INTEGER     NM, MAXITR
      6.          06          REAL        RTOL
      7.          07          REAL        X(NM), B(NM), R(NM), P(NM), AP(NM), NULLSP(NM)
      8.          06          EXTERNAL    ATIMES, PSOLVE
      9.          09          REAL        APARM(1), PPARM(1)
     10.          07          REAL        LALPHA(MAXITR), LBETA(MAXITR)
     11.          07          INTEGER     CNTL
     12.          06     C
     13.          06     C    ------------------------------------------------------------
     14.          06     C
     15.          06     C    H Y B R I D    C O N J U G A T E   G R A D I E N T S
     16.          07     C         F O R    S I N G U L A R      S Y S T E M S
     17.          07     C         W I T H    K N O W N      N U L L S P A C E
     18.          07     C         O F      D I M E N S I O N     1
     19.          06     C
     20.          06     C       TO SOLVE THE LINEAR SYSTEM
     21.          06     C          A X  =  B
     22.          06     C
     23.          06     C    ------------------------------------------------------------
     24.          06     C
     25.          06     C    APPLY CONJUGATE GRADIENTS TO THE SCALED MATRIX
     26.          06     C       P**(-.5)  A  P**(-.5)
     27.          06     C    WITHOUT COMPUTING THE SQUARE ROOT OF  P  EXPLICITLY
     28.          06     C
     29.          06     C    P A R A M E T E R S:
     30.          06     C       INPUT:
     31.          06     C         NM - SQUARE DIMENSION OF SPARSE MATRICES  A  AND  P
     32.          06     C              ( N  TIMES  M  FOR FINITE DIFFERENCE SCHEMES ON
     33.          06     C              A RECTANGLE.)
     34.          06     C
     35.          06     C         MAXITR - MAXIMUM NUMBER OF ITERATIONS ALLOWED.
     36.          06     C                  (CHANGED TO ACTUAL NUMBER USED, ON OUTPUT.)
     37.          06     C
     38.          06     C         RTOL - RESIDUAL TOLERANCE DEFINING AN ACCEPTABLE
     39.          06     C                SOLUTION (TAKEN RELATIVE TO ||B||)
     40.          06     C
     41.          06     C         X    - INITIAL GUESS AT SOLUTION.  WILL BE DESTROYED BY
     42.          06     C                PROGRAM. WHICH RETURNS NEW APPROXIMATION IN  X .
     43.          06     C
     44.          06     C         B    - RIGHT HAND SIDE OF LINEAR SYSTEM
     45.          07     C
     46.          07     C         NULLSP  -  KNOWN NULLSPACE OF  MATRIX  A
     47.          06     C
     48.          06     C         ATIMES - A SUBROUTINE TO COMPUTE THE MATRIX-VECTOR PRODUCT
     49.          09     C                  AX.  CALLED AS 'ATIMES (APARM, X, AX)'
     50.          09     C                  'APARM'  IS A VECTOR OF PARAMETERS TO THE SUBROUTINE
     51.          06     C                  X  IS THE VECTOR TO BE MULTIPLIED BY  A
     52.          06     C                       THE VECTOR  X  MUST NOT BE DESTROYED
     53.          06     C                  AX  IS THE MATRIX-VECTOR PRODUCT
     54.          06     C
     55.          06     C         PSOLVE - A SUBROUTINE TO SOLVE THE LINEAR SYSTEM
     56.          06     C                       PY = C
```

```
                09      C                   CALLED AS  'CALL PSOLVE (PPARM, C, Y)':
                09      C                   PPARM IS A VECTOR OF PARAMETERS TO PSOLVE
                06      C                   C  IS THE RIGHT HAND SIDE (WHICH MUST BE PRESERVED)
                06      C                   Y  IS THE SOLUTION TO THE SYSTEM
                06      C
                09      C           APARM - A VECTOR OF PARAMETERS TO THE SUBROUTINE  ATIMES.
                06      C                   (USED TO OBVIATE THE NEED FOR USE OF COMMON)
                06      C
                09      C           PPARM - A VECTOR OF PARAMETERS TO THE SUBROUTINE  PSOLVE.
                06      C                   PASSED DIRECTLY BY THIS SUBROUTINE  TO  PSOLVE.
                07      C
                07      C           CNTL  - IF >= 1, PRINT SUMMARY OF ITERATIONS, OTHERWISE
                07      C                   SUPPRESS ALL OUTPUT
                07      C                  ON OUTPUT,  0 IS INDICATION OF SUCCESS.  1 INDICATES
                07      C                  FAILURE TO CONVERGE WITHIN ITERATION LIMIT.
                06      C
                06      C
                06      C       OUTPUT:
                06      C         X     - THE APPROXIMATE SOLUTION TO  AX = B.  (SEE ABOVE)
                06      C
                06      C         MAXITR - NUMBER OF ITERATIONS USED.
                06      C
                06      C         LALPHA, LBETA  -  THE COEFFICIENTS OF THE TRIDIAGONAL MATRIX
                06      C                           WHICH WOULD HAVE BEEN PRODUCED BY THE
                06      C                           LANCZOS ALGORITHM FOR FINDING EIGENVALUES.
                06      C                           USEFUL FOR ESTIMATING THE CONVERGENCE RATES
                06      C                           OF THE CONJUGATE GRADIENTS ALGORITHM, AND
                06      C                           A CHEAP BYPRODUCT OF THE LATTER ALGORITHM.
                06      C
                06      C       WORKING PARAMETERS:
                06      C            (VECTORS FOR WORKING STORAGE WITHIN SUBROUTINE)
                06      C         R   HOLDS APPROXIMATE (RECURSIVE) RESIDUAL FOR FINAL SOLUTION.
                06      C         P, AP  TEMPORARIES NEEDED BY ALGORITHM.
                06      C
                06      C       -----------------------------------------------------------------
                06      C
                06      C       LOCAL VARIABLES:
                06      C
                06              INTEGER     I, ITER
                06              REAL        PTAP, RTPIR, NORMR, NORMB, AK, BK, XRTPIR, IP,
               1                            RELRES, AKM1
                06              EXTERNAL    IP
                07              LOGICAL     PRINT
                10      C
                10              INTEGER     PRINTR
                10              DATA        PRINTR / 6 /
                07      C
                06      C
                06      C       -----------------------------------------------------------
                06      C
                06      C          COMPUTE INITIAL RESIDUAL AND INITIAL SEARCH DIRECTION
                06      C
                07      C          FORCE LINEAR SYSTEM TO BE CONSISTENT BY REMOVING ANY
                07      C          COMPONENT OF THE NULLSPACE (ORTHOGONAL COMPLEMENT OF RANGE)
                07      C          FROM  RIGHT HAND SIDE  B.  THEORETICALLY THE CONSISTENCY OF THE
                07      C          LINEAR SYSTEM THEN IMPLIES THAT THE VECTORS  X  AND  R, WHICH
                07      C          ARE GENERATED BY THE RECURSION ARE ALL ORTHOGONAL TO THE NULL-
                07      C          SPACE OF  A.  WE ENSURE THIS ORTHOGONALITY IN THE NUMERICAL
```

```
115.  07  C        PROBLEM BY INCORPORATING A "REORTHOGONALIZATION".  NOTE THAT
116.  07  C        THE VECTORS  P  ARE  NOT  NECESSARILY ORTHOGONAL TO THE
117.  07  C        NULLSPACE OF  A.
118.  07  C
119.  07            CALL PROJCT (NM, B, NULLSP)
120.  07  C
121.  09            CALL ATIMES (APARM, X, AP)
122.  06            NORMB = 0.0
123.  07            NORMR = 0.0
124.  06            DO 10 I = 1, NM
125.  06               R(I) = B(I) - AP(I)
126.  07               NORMR = NORMR  +  R(I)**2
127.  06               NORMB = NORMB  +  B(I) ** 2
128.  06         10 CONTINUE
129.  07            RELRES = SQRT(NORMR/NORMB)
130.  07  C
131.  07            CALL PROJCT (NM, R, NULLSP)
132.  06  C
133.  09            CALL PSOLVE (PPARM, R, P)
134.  06            RTPIR = IP (NM, P, R)
135.  06            ITER = 0
136.  06  C
137.  07            PRINT = (CNTL .GE. 1)
138.  10            IF  (PRINT)  WRITE (PRINTR, 6000)  ITER, RELRES
139.  06  C
140.  06  C
141.  06  C     ------------------------------------------------------------
142.  06  C        M A I N   I T E R A T I O N
143.  06  C     ------------------------------------------------------------
144.  06  C
145.  06       100    ITER = ITER + 1
146.  06  C
147.  06  C           COMPUTE NEW SOLUTION GUESS AND NEW RESIDUAL
148.  06  C
149.  09            CALL ATIMES (APARM, P, AP)
150.  06            PTAP = IP (NM, P, AP)
151.  06            AK   = RTPIR / PTAP
152.  06  C
153.  06            NORMR = 0.0
154.  06            DO 110 I = 1, NM
155.  06               X(I) = X(I)  +  AK * P(I)
156.  06               R(I) = R(I)  -  AK * AP(I)
157.  06               NORMR = NORMR  +  R(I) ** 2
158.  06       110    CONTINUE
159.  07  C
160.  07            CALL PROJCT (NM, X, NULLSP)
161.  07            CALL PROJCT (NM, R, NULLSP)
162.  06  C
163.  06  C
164.  06  C           COMPUTE LANCZOS DIAGONAL ELEMENT
165.  06  C
166.  06            IF ( ITER .GT. 1 )  GO TO 120
167.  06               ALPHA(1) = 1.0 / AK
168.  06               GO TO 130
169.  06       120     ALPHA(ITER) = 1.0 / AK   +   BK / AK1
170.  06  C
171.  06  C           CONVERGENCE TEST
172.  06  C
```

```
173.          06        130     RELRES = SQRT (NORMR / NORMB)
174.          06                IF  ( (RELRES  .LT.  RTOL)  .OR.
175.          06           1          (ITER .EQ. MAXITR) )  GO TO 200
176.          06     C
177.          06     C            COMPUTE NEW DIRECTION
178.          06     C
179.          06                AKM1 = AK
180.          09                CALL PSOLVE (PPARM, R, AP)
181.          06                XRTPIR = IP (NM, R, AP)
182.          06                BK = XRTPIR / RTPIR
183.          06                RTPIR = XRTPIR
184.          06     C
185.          06                DO 140 I = 1, NM
186.          06                   P(I) = BK * P(I)  +  AP(I)
187.          06        140     CONTINUE
188.          07     C
189.          06     C
190.          06     C            COMPUTE LANCZOS OFF-DIAGONAL
191.          06     C
192.          06                LBETA(ITER+1) = SQRT(BK) / AK
193.          10                IF  (PRINT)  WRITE (PRINTR, 6050) ITER, RELRES, AK, BK,
194.          07           1                                      LALPHA(ITER), LBETA(ITER+1)
195.          06     C
196.          06                GOTO 100
197.          06     C
198.          06     C     -------------------------------------------------------
199.          06     C
200.          10        200 IF  (PRINT)  WRITE (PRINTR, 6100) ITER, RELRES, AK, LALPHA(ITER),
201.          10           1                                  ITER, RELRES, RTOL
202.          07     C
203.          07              CNTL = 0
204.          07              IF (RELRES .GT. RTOL)  CNTL = 1
205.          06              MAXITR = ITER
206.          06              RETURN
207.          06     C
208.          06     C
209.          06         6000 FORMAT ('0CONJUGATE GRADIENTS ITERATION'/
210.          06           1        '0ITER.  RESIDUAL       AK         BK         ALPHA',
211.          07           2          '      BETA'/
212.          07           3          '0', I4, 1PE12.5)
213.          06     C
214.          06         6050 FORMAT (I5, 1P5E12.5)
215.          06     C
216.          07         6100 FORMAT (I5, 1P2E12.5, 12X, E12.5 /
217.          06           X        '0CONJUGATE GRADIENTS TERMINATED AFTER ', I2,
218.          07           1        ' ITERATIONS' /
219.          06           2        ' ESTIMATE OF RELATIVE RESIDUAL NORM', 1PE10.3 /
220.          06           3        ' REQUESTED RESIDUAL NORM BOUND     ', E10.3 )
221.          06              END

END ELT.  ERRORS: NONE.  TIME:  0.772 SEC.  IMAGE COUNT: 221

@EDC,P  ******  HCGSOL  ******
```

```
@:ELT,L -PF1-.HCGSOL
ELT 8R1 S74Q1C 07/11/79 08:48:46 (23)
      1.           19            SUBROUTINE   HCGSOL
      2.           19      C
      3.           20           1           (M, N, NM, MMAX, FLIP,
      4.           23           2            GENPOI, SCALE,   BKSCAL, ATIMES, PSOLVE,
      5.           22           3            RHO,    ELPDEF,   X0,     F,      X,       RESID,
      6.           20           4            WRK1,   WRK2,     WRK3,   NULLSP, APARM,   ALPHA,
      7.           20           5            BETA,   GAMMA,    LOWERB, UPPERB, LEFTB,   RIGHTB,
      8.           22           6            PPARM,  DELTAX,   DELTAY, MAXITR, RTOL,    CNTL)
      9.           19      C
     10.           19      C     ------------------------------------------------------------
     11.           19      C
     12.           19      C        SOLVE DISCRETE PRESSURE EQUATION
     13.           19      C           GENERATE DISCRETE GENERALIZED POISSON OPERATOR (GENPOI)
     14.           19      C           PERFORM DIAGONAL SCALING ON DISCRETE OPERATOR (SCALE)
     15.           19      C           SOLVE SCALED LINEAR SYSTEM BY CONJUGATE GRADIENTS (HCGSIN)
     16.           23      C              (USES FAST POISSON SOLVER OR OTHER SPLITTING (PSOLVE))
     17.           19      C           BACK-TRANSFORM TO SOLUTION OF ORIGINAL PROBLEM (BKSCAL)
     18.           19      C
     19.           19           INTEGER      M, N, NM, MMAX
     20.           20           LOGICAL      FLIP
     21.           19      C
     22.           22      C        SUBROUTINE PARAMETERS WHICH DESCRIBE THE TWO MATRICES
     23.           22      C        WHICH APPEAR IN THE HYBRID CONJUGATE GRADIENT ALGORITHM
     24.           22      C
     25.           23           EXTERNAL     GENPOI, SCALE, BKSCAL, ATIMES, PSOLVE
     26.           19      C
     27.           19      C
     28.           19      C        ARRAYS ARE ALLOCATED IN A "DYNAMIC" SENSE IN THE DRIVER
     29.           19      C
     30.           19      C     ARRAYS OF DIMENSION  MMAX  BY  N  (PASSED FROM HYPERBOLIC PDE SOLVER)
     31.           19      C
     32.           19           REAL      RHO(MMAX,N), ELPDEF(MMAX,N), X0(MMAX,N), RESID(MMAX,N)
     33.           19      C
     34.           19      C
     35.           19      C     ARRAYS OF DIMENSION  M * N
     36.           19      C
     37.           19           REAL      X(NM), F(NM), WRK1(NM), WRK2(NM), WRK3(NM), NULLSP(NM),
     38.           19          1          ALPHA(NM), BETA(NM), GAMMA(NM)
     39.           19      C
     40.           20      C        ... THE FOLLOWING ACTUAL PARAMETERS MAY OCCUPY THE SAME STORAGE
     41.           19      C              ELPDEF   AND  WRK1
     42.           19      C              X0       AND  WRK2
     43.           19      C              RESID    AND  WRK3
     44.           19      C
     45.           20      C     ARRAY OF PARAMETERS FOR FIVE-DIAGONAL MATRIX MULTIPLY ROUTINE
     46.           19      C        ACTUAL DIMENSION   3MN + 2
     47.           19      C
     48.           20           REAL      APARM(1)
     49.           19      C
     50.           19      C
     51.           19      C     ARRAYS OF DIMENSION  M
     52.           19      C
     53.           19           REAL      LOWERB(M), UPPERB(M)
     54.           19      C
     55.           19      C
     56.           19      C     ARRAYS OF DIMENSION  N
```

```
     19      C
     19            REAL     LEFTB(N), RIGHTB(N)
     19      C
     19      C
     19      C     ARRAY OF PARAMETERS FOR SOLVING SYSTEM WITH SPLITTING MATRIX
     20      C        ACTUAL DIMENSION:  3L + 9K + 2(L+2)*(LOG(L)-1) + 12
     20      C             WHERE (K,L) = (M,N)  UNLESS  N  IS NOT ONE LESS THAN
     20      C             A POWER OF TWO, OR  N  AND  M  ARE BOTH ONE LESS
     20      C             THAN A POWER OF TWO, AND  N  IS LARGER
     20      C             (IN WHICH CASE  (K,L) = (N,M).
     20      C             THE INITIALIZING SUBROUTINE  'INTNEU'  CHECKS THAT
     20      C             SUFFICIENT SPACE IS AVAILABLE IN  'PPARM'.
     19      C
     20            REAL     PPARM(1)
     19      C
     20      C         SCALARS
     19      C
     20            REAL        DELTAX, DELTAY, RTOL
     20            INTEGER     MAXITR, CNTL
     19      C
     19      C     ================================================================
     19      C
     19      C        LOCAL VARIABLES ...
     19      C
     20            INTEGER     LMXITR
     19            REAL        LALPHA(50), LBETA(50)
     20      C        DIMENSION   LMXITR
     20               DATA     LMXITR / 50 /
     20      C
     20      C
     20            INTEGER     IJ, J, I, K, L, ITER, ERROR
     19            INTEGER     TOTTIM, PRVTIM, STRTIM
     20            REAL        RNORM, FORMX, ELPTIM, CONSST, DXDYSQ, COND, CONRAT,
    20           1            SFACTR
     19      C
     20      C
     20            INTEGER     PRINTR
     20            DATA        PRINTR / 6 /
     19      C
     19      C
     19      C     ================================================================
     19      C
     19            CALL CPUSUP (STRTIM)
     19      C
     20            PRINT = CNTL
     20      C
     20            IF (PRINT .GE. 3) WRITE (PRINTR, 3000) M, N
     19      C
     20            MAXITR = MIN0(MAXITR, LMXITR)
     19      C
     20      C
     20      C
     19      C        GENERATE DISCRETIZATION OF PRESSURE EQUATION
     19      C
     20            CALL GENPOI (M, N, MMAX, FLIP, RHO, ELPDEF, X0, DELTAX, DELTAY,
    20           1             LOWERB, UPPERB, LEFTB, RIGHTB,
    20           2             ALPHA, BETA, GAMMA, F, X, WRK3)
     20      C
```

```
115.          20            IF (FLIP) GO TO 10
116.          20      C
117.          20      C        .. NATURAL ORDERING USED ..
118.          20      C
119.          20               APARM(1) = M
120.          20               APARM(2) = N
121.          20               GO TO 20
122.          20      C
123.          20      C        .. TRANSPOSE OF NATURAL ORDERING USED ..
124.          20      C
125.          20           10  APARM(1) = N
126.          20               APARM(2) = M
127.          20      C
128.          20          20 K = APARM(1)
129.          20             L = APARM(2)
130.          20      C
131.          20               IF (PRINT .GE. 4)  WRITE (PRINTR, 4000)
132.          20               IF (PRINT .GE. 4)  CALL BTROUT (K, L, ALPHA, BETA, GAMMA)
133.          19      C
134.          20      C        FORCE LINEAR DIFFERENCE EQUATIONS TO BE CONSISTENT ...
135.          20      C        (EQUIVALENT TO REQUIRING THE RIGHT HAND SIDE TO HAVE MEAN ZERO)
136.          20      C        FORCE INITIAL PRESSURE GUESS TO HAVE MEAN PRESSURE ZERO
137.          20      C        (HENCE, IS ORTHOGONAL TO THE NULL SPACE OF THE LINEAR OPERATOR)
138.          20      C
139.          19               CALL PROJCE (NM, F, CONSST)
140.          20               IF (PRINT .GE. 2)  WRITE (PRINTR, 2000) CONSST
141.          19      C
142.          19               CALL PROJCE (NM, X0, CONSST)
143.          20               IF (PRINT .GE. 2)  WRITE (PRINTR, 2010) CONSST
144.          19      C
145.          20      C        COMPUTE RESIDUAL OF CONTINUOUS SOLUTION (OPTIONAL)
146.          19      C
147.          20               IF (PRINT .LT. 2) GO TO 210
148.          22                  CALL ATIMES (APARM, X, WRK1)
149.          19                  RNORM = 0.0
150.          19                  NORMX = 0.0
151.          19                  DO 100 I = 1, NM
152.          19                     RNORM = RNORM  +  (F(I) - WRK1(I))**2
153.          19                     NORMX = NORMX  +  X(I)**2
154.          19                     WRK2(I) = 1.0
155.          19           100    CONTINUE
156.          19                  RNORM = SQRT(RNORM/NORMX)
157.          20                  WRITE (PRINTR, 2020) RNORM
158.          19      C
159.          20                  IF (PRINT .GT. 2)  WRITE (PRINTR, 3010)
160.          20                  IF (PRINT .GT. 2)  CALL VECOUT (K, L, F)
161.          19      C
162.          19                  RNORM = 0.0
163.          22                  CALL ATIMES (APARM, WRK2, WRK1)
164.          19                  DO 200 I = 1, NM
165.          19                     RNORM = RNORM + WRK1(I)**2
166.          20           200    CONTINUE
167.          19                  RNORM = SQRT(RNORM/NM)
168.          20                  WRITE (PRINTR, 2030) RNORM
169.          19      C
170.          19      C              SCALE DISCRETE OPERATOR
171.          19      C
172.          20           210 DXDYSQ = (DELTAX/DELTAY) ** 2
```

```
173.          20            SFACTR = DXDYSQ
174.          20            IF (FLIP) SFACTR = 1.0 / SFACTR
175.          20            CALL SCALE (K, L, ALPHA, BETA, GAMMA, SFACTR, NULLSP)
176.          19      C
177.          20            IF (PRINT .LE. 3) GO TO 220
178.          20               WRITE (PRINTR, 4010)
179.          20               CALL BTROUT (K, L, ALPHA, BETA, GAMMA)
180.          19      C
181.          19      C
182.          19      C        SCALE RIGHT HAND SIDE OF EQUATION
183.          19      C
184.          19        220 DO 300 I = 1, NN
185.          19               F(I) = F(I) / NULLSP(I)
186.          19               X(I)  = X(I) * NULLSP(I)
187.          19        300 CONTINUE
188.          19      C
189.          20      C        CHECK ACCURACY OF SCALING AND SCALED NULLSPACE (OPTIONAL)
190.          19      C
191.          19            IF (PRINT .LT. 2) GO TO 410
192.          22               CALL ATIMES (APARM, NULLSP, WRK1)
193.          19               RNORM = 0.0
194.          19               NORMX = 0.0
195.          19               DO 400 I = 1, NN
196.          19                  RNORM = RNORM + WRK1(I)**2
197.          19                  NORMX = NORMX + NULLSP(I)**2
198.          20          400     CONTINUE
199.          19               RNORM = SQRT(RNORM/NORMX)
200.          19               NORMX = SQRT(NORMX)
201.          20               WRITE (PRINTR, 2040) RNORM, NORMX
202.          19      C
203.          19      C        CALL HYBRID CONJUGATE GRADIENTS ALGORITHM
204.          19      C
205.          19        410 ITER = MAXITR
206.          19      C
207.          19            CALL CPUSUP (TOTTIM)
208.          19            ELPTIM = (TOTTIM - STRTIM) / 1000.
209.          19            PRVTIM = TOTTIM
210.          20            IF (PRINT .GT. 1)  WRITE (PRINTR, 2050) ELPTIM
211.          19      C
212.          19            CALL HCGSIN (NN, ITER, RTOL, X, F, WRK3, WRK1, WRK2, NULLSP,
213.          22           1             ATIMES, PSOLVE, APARM, PPARM, LALPHA, LBETA, CNTL)
214.          19      C
215.          20            IF (PRINT .LE. 1) GO TO 420
216.          19               CALL CPUSUP (TOTTIM)
217.          19               ELPTIM = (TOTTIM - PRVTIM) / 1000.
218.          20               WRITE (PRINTR, 2060) ELPTIM
219.          19               PRVTIM = TOTTIM
220.          19      C
221.          20      C           COMPUTE APPROXIMATE EIGENVALUES OF OPERATOR    (OPTIONAL)
222.          19      C
223.          19               CALL TQL1 (ITER, LALPHA, LBETA, ERROR)
224.          20               IF ( ERROR .NE. 0 ) WRITE (PRINTR, 2070) ERROR
225.          21               COND = LALPHA(ITER) / LALPHA(1)
226.          21               CONRAT = (COND - 1.0) / (COND + 1.0)
227.          20               WRITE (PRINTR, 2080) COND, CONRAT
228.          19      C
229.          19      C        SCALE THE PROBLEM BACK INTO THE ORIGINAL COORDINATES
230.          19      C
```

```
231.          20       420 CALL BKSCAL (K, L, ALPHA, BETA, GAMMA, NULLSP)
232.          19           DO 500 I = 1, NM
233.          19              X(I) = X(I) / NULLSP(I)
234.          19              F(I) = F(I) * NULLSP(I)
235.          19       500 CONTINUE
236.          19     C
237.          19     C        NORMALIZE SOLUTION TO MEAN PRESSURE ZERO (CONSISTENT)
238.          19     C
239.          19           CALL PROJCE (NM, X, CONSST)
240.          20           IF (PRINT .GE. 2)  WRITE (PRINTR, 2090) CONSST
241.          19     C
242.          19     C        COMPUTE RESIDUALS FOR FINAL SOLUTION
243.          19     C        AND REVERT TO TWO-DIMENSIONAL STORAGE
244.          19     C
245.          22           CALL ATIMES (APARM, X, WRK1)
246.          19           RNORM = 0.0
247.          19           NORMX = 0.0
248.          19           DO 600 I = 1, M
249.          19              DO 590 J = 1, N
250.          20                 IF (.NOT. FLIP)  IJ = (J-1)*M + I
251.          20                 IF (FLIP)        IJ = (I-1)*N + J
252.          19                 RESID(I,J) = F(IJ) - WRK1(IJ)
253.          20                 F(IJ) = RESID(I,J)
254.          19                 RNORM = RNORM  +  RESID(I,J) ** 2
255.          19                 X0(I,J) = X(IJ)
256.          19                 NORMX = NORMX  +  X(IJ)**2
257.          19       590    CONTINUE
258.          19       600 CONTINUE
259.          19     C
260.          19           IF (PRINT .LT. 2) GO TO 700
261.          19              RNORM = SQRT(RNORM/NORMX)
262.          20              WRITE (PRINTR, 1000) RNORM
263.          20           IF (PRINT .LT. 3)  GO TO 610
264.          20              WRITE (PRINTR, 3020)
265.          20                 CALL VECOUT (K, L, F)
266.          20                 WRITE (PRINTR, 3030)
267.          20                 CALL VECOUT (K, L, X0)
268.          19     C
269.          20       610    CALL CPUSUP (TOTTIM)
270.          19              ELPTIM = (TOTTIM-STRTIM) / 1000.
271.          20              WRITE (PRINTR, 2160) ELPTIM
272.          19     C
273.          20       700 RETURN
274.          19     C
275.          19     C
276.          19     C     ============================================================
277.          19     C
278.          20     C           ARRANGED BY PRINT LEVEL ... FIRST DIGIT OF FORMAT NUMBER
279.          20     C           INDICATES THE LEVEL AT WHICH IT BECOMES USED.
280.          20     C
281.          20      1000 FORMAT ('0RELATIVE NORM OF RESIDUAL VECTOR  A X - B :', 1PE10.2)
282.          20     C
283.          20      2000 FORMAT (' MEAN OF RIGHT HAND SIDE (REMOVED FOR CONSISTENCY): ',
284.          20          1         1PE12.3)
285.          20     C
286.          20      2010 FORMAT (' MEAN OF INITIAL PRESSURE APPROXIMATION: ',
287.          20          1         1PE12.3)
288.          20     C
```

```
289.          20      C
290.          20       2020 FORMAT (' RESIDUAL FOR UNSCALED CONTINUOUS PROB:', 1PE12.3)
291.          20      C
292.          20       2030 FORMAT (' NORM OF OPERATOR APPLIED TO "E"', 1PE12.3)
293.          20      C
294.          20       2040 FORMAT (' SCALED OPERATOR ON SCALED NULLSPACE', 1PE12.3/
295.          20          1         ' NORM OF SCALED NULLSPACE           ', E12.3)
296.          20      C
297.          20       2050 FORMAT ('0SET UP TIME (SECONDS):', F13.3)
298.          20      C
299.          20       2060 FORMAT ('0TIME IN CONJUGATE GRADIENTS(SECONDS):', F13.3)
300.          20      C
301.          20       2070 FORMAT (' TQL1 FAILURE ... SOME EIGENVALUES NOT COMPUTED'/
302.          20          1         ' ERROR CODE RETURNED :', I5 )
303.          20      C
304.          20       2080 FORMAT (' CONDITION NUMBER OF ITERATION MATRIX: ', 1PE12.3 /
305.          20          1         ' ASYMPTOTIC CONVERGENCE RATE: ', E12.3)
306.          20      C
307.          20       2090 FORMAT (' MEAN PRESSURE OF SOLUTION (REMOVED): ', 1PE12.3)
308.          20      C
309.          20       2100 FORMAT ('0TOTAL TIME FOR RUN:', F13.3)
310.          20      C
311.          20       3000 FORMAT ('0PRESSURE EQUATION TEST' /
312.          19          2         '0GRID SIZE:' /
313.          19          3         '       M :', I3 /
314.          19          4         '       N :', I3)
315.          19      C
316.          20       3010 FORMAT('0RIGHT HAND SIDE:')
317.          20      C
318.          20       3020 FORMAT ('0R E S I D U A L    V E C T O R')
319.          20      C
320.          20       3030 FORMAT ('0' / '0F I N A L      S O L U T I O N')
321.          20      C
322.          20       4000 FORMAT ('0'/'0UNSCALED DISCRETE OPERATOR ... '/)
323.          19      C
324.          20       4010 FORMAT ('0'/'0SCALED DISCRETE OPERATOR ... '/)
325.          19      C
326.          19            END
```

END ELT. ERRORS: NONE. TIME: 1.128 SEC. IMAGE COUNT: 326

GHDG,P ****** INTDIR ******

```
@:ELT,L -PF1-.INTNEU
ELT SR1 S74Q1C 07/11/79 08:48:54 (21)
     1.          18            SUBROUTINE  INTNEU  (PPARM, LPPARM, M, N, DXDYSQ)
     2.          18      C
     3.          18      C        INITIALIZE PARAMETER ARRAY FOR SOLVING THE DISCRETE LAPLATIAN
     4.          18      C        ON A RECTANGLE WITH NEUMANN BOUNDARY CONDITIONS, USING
     5.          18      C        THE NCAR ROUTINE 'BLKTRI' BY PAUL SCHWARZTRAUBER.
     6.          18      C
     7.          18      C
     8.          18      C        ============================================================
     9.          18      C        THIS ROUTINE MUST BE CALLED ONCE, BEFORE THE ELLIPTIC
    10.          18      C        PDE SOLVER  'HCGSOL'  IS USED.  IT DOES NOT NEED TO BE
    11.          18      C        REINITIALIZED AGAIN.
    12.          18      C        ============================================================
    13.          18      C
    14.          18      C
    15.          18            INTEGER      LPPARM, M, N
    16.          18            REAL         PPARM(LPPARM)
    17.          18            REAL         DXDYSQ
    18.          18      C
    19.          18      C        PPARM IS AN ARRAY CONTAINING ALL THE PARAMETERS NEEDED BY
    20.          18      C        THE SUBROUTINE  NEUSOL.  THE INDIVIDUAL ELEMENTS ARE NAMED
    21.          18      C        AND COMMENTED IN THE CODE SETTING UP THE ARRAY VALUES.
    22.          18      C        PPARM  MUST BE A VECTOR OF LENGTH AT LEAST
    23.          18      C              3N + 9M + 2(N+2)(LOG(N)-1) + 12
    24.          18      C           OR
    25.          18      C              3M + 9N + 2(M+2)(LOG(M)-1) + 12
    26.          18      C           WHERE THE LOGARITHM IS BASE 2, AND  N (M  MUST BE ONE LESS
    27.          18      C           THAN A POWER OF  2.
    28.          18      C
    29.          18      C        LPPARM  IS THE DECLARED LENGTH OF  PPARM  (THE INTEGER
    30.          18      C                CONSTANT IN THE REAL OR DIMENSION STATEMENT.)  THIS
    31.          18      C                SUBROUTINE WILL CALCULATE THE ACTUAL STORAGE NEEDED
    32.          18      C                AND WILL STOP IF THERE IS NOT ENOUGH.
    33.          18      C
    34.          18      C        M   IS THE NUMBER OF GRID POINTS IN THE  X  DIRECTION,
    35.          18      C        N   IS THE NUMBER OF GRID POINTS IN THE  Y  DIRECTION.
    36.          18      C
    37.          18      C        DXDYSQ  IS THE SQUARE OF THE RATIO  DELTA X / DELTA Y.
    38.          18      C
    39.          18      C
    40.          18            INTEGER      I, SUBAL, SUBAK, SUBBL, SUBBK, SUBCL, SUBCK, SUBWRK,
    41.          19           1             INIT, NEED, K, L
    42.          18            LOGICAL      FLIP
    43.          21            REAL         SFACTR
    44.          19      C
    45.          19            INTEGER      PRINTR
    46.          19            DATA         PRINTR / 6 /
    47.          18      C
    48.          18      C     ------------------------------------------------------------
    49.          18      C
    50.          18      C        DETERMINE WHICH ORDER IS MORE APPROPRIATE FOR THE FAST SOLVER.
    51.          18      C        AT LEAST ONE OF THE LIMITS  M  AND  N  MUST BE ONE LESS THAN A
    52.          18      C        POWER OF TWO.  L  WILL BE THE SMALLER OF  M  AND  N  WHICH
    53.          18      C        MEETS THIS CONDITION.
    54.          18      C
    55.          18            FLIP = (  (N .NE. 2**(LOG2(N+1)) - 1)    .OR.
    56.          18           1          ( (N. EQ. 2**(LOG2(N+1)) - 1) .AND.
```

```
     2            (M .EQ. 2**(LOG2(M+1)) - 1) .AND.
     3     .      (M .LT. N) )  )
C
      IF (FLIP) GO TO 10
C
C        .. NATURAL ORDERING USED ..
C
         K = M
         L = N
         GO TO 20
C
C        ..TRANSPOSED ORDERING USED ..
C
   10    K = N
         L = M
C
C        .. CHECK STORAGE ALLOCATION ..
C
      NEED = 3*L + 9*K + 2*(L+2)*(LOG2(L+1)-1) + 12
   20 IF (LPPARM .LT. NEED) GO TO 1000
C
C
C        INITIALIZATION FLAG:
      PPARM(1) = 0.0
C
C        X DIMENSION
      PPARM(2) = K
C
C        Y DIMENSION
      PPARM(3) = L
C
C        LOCATION OF  AK
      SUBAK = 11
      PPARM(4) = SUBAK
C
C        LOCATION OF  AL
      SUBAL = SUBAK + K
      PPARM(5) = SUBAL
C
C        LOCATION OF  BK
      SUBBK = SUBAL + L
      PPARM(6) = SUBBK
C
C        LOCATION OF  BL
      SUBBL = SUBBK + K
      PPARM(7) = SUBBL
C
C        LOCATION OF  CK
      SUBCK = SUBBL + L
      PPARM(8) = SUBCK
C
C        LOCATION OF  CL
      SUBCL = SUBCK + K
      PPARM(9) = SUBCL
C
C        LOCATION OF  WORKING STORAGE
      PPARM(10) = SUBCL + L
C
```

```
115.          18      C
116.          18      C         INITIALIZE CONSTANTS FOR DIAGONALS
117.          18      C
118.          21            SFACTR = DXDYSQ
119.          21            IF  (FLIP)  SFACTR = 1.0 / SFACTR
120.          18      C
121.          21            PPARM(SUBAK) = 0
122.          18            PPARM(SUBAL) = 0
123.          21            PPARM(SUBBK) = -1.0
124.          21            PPARM(SUBBL) = -SFACTR
125.          21            PPARM(SUBCK) = 1.0
126.          21            PPARM(SUBCL) = SFACTR
127.          18      C
128.          18            DO 100 I = 3, K
129.          18               SUBAK = SUBAK + 1
130.          18               SUBBK = SUBBK + 1
131.          18               SUBCK = SUBCK + 1
132.          21               PPARM(SUBAK) = 1.0
133.          21               PPARM(SUBBK) = -2.0
134.          21               PPARM(SUBCK) = 1.0
135.          18        100 CONTINUE
136.          18      C
137.          21            PPARM(SUBAK+1) = 1.0
138.          21            PPARM(SUBBK+1) = -1.0
139.          18            PPARM(SUBCK+1) = 0.0
140.          18      C
141.          18            DO 200 I = 3, L
142.          18               SUBAL = SUBAL + 1
143.          18               SUBBL = SUBBL + 1
144.          18               SUBCL = SUBCL + 1
145.          21               PPARM(SUBAL) = SFACTR
146.          21               PPARM(SUBBL) = -2.0 * SFACTR
147.          21               PPARM(SUBCL) = SFACTR
148.          18        200 CONTINUE
149.          18      C
150.          21            PPARM(SUBAL+1) = SFACTR
151.          21            PPARM(SUBBL+1) = -SFACTR
152.          18            PPARM(SUBCL+1) = 0.0
153.          18      C
154.          18      C                   INITIALIZE FAST POISSON SOLVER
155.          18      C
156.          18            SUBAK = PPARM(4)
157.          18            SUBAL = PPARM(5)
158.          18            SUBBK = PPARM(6)
159.          18            SUBBL = PPARM(7)
160.          18            SUBCK = PPARM(8)
161.          18            SUBCL = PPARM(9)
162.          18            SUBWRK = PPARM(10)
163.          18            INIT = 0
164.          18      C
165.          18            CALL BLKTRI (INIT, 1, L, PPARM(SUBAL), PPARM(SUBBL),
166.          18           1                         PPARM(SUBCL),
167.          18           2                    1, K, PPARM(SUBAK), PPARM(SUBBK),
168.          18           3                          PPARM(SUBCK), K,
169.          18           4              PINVX, ERROR, PPARM(SUBWRK))
170.          18      C
171.          18            IF (ERROR .NE. 0) GO TO 2000
172.          18      C
```

```
  173.          19            PPARM(1) = 1.0
  174.          18            RETURN
  175.          18       C
  176.          18       C
  177.          19        1000 WRITE (PRINTR, 6060) NEED, LPPARM
  178.          18             STOP
  179.          18        6000 FORMAT ('0**********************************'/
  180.          18            1          ' NOT ENOUGH PARAMETER STORAGE FOR BLKTRI'/
  181.          18            2          ' PPARM MUST BE AT LEAST', I4, ' LONG'/
  182.          18            3          ' LPPARM WAS ONLY', I10 /
  183.          18            4          ' **************************************')
  184.          18       C
  185.          18       C
  186.          19        2000 WRITE (PRINTR, 6100) ERROR
  187.          18             STOP
  188.          18       C
  189.          18        6100 FORMAT ('0**********************************************'/
  190.          18            1          ' FATAL ERROR IN INITIALIZING POISSON SOLVER'/
  191.          18            2          '0ERROR RETURN FLAG:  ', I3 /
  192.          18            3          '0CHECK WRITEUP OF NCAR ROUTINE  BLKTRI' /
  193.          18            4          ' FOR INTERPRETATION OF FLAG' /
  194.          18            5          ' **********************************************')
  195.          18             END
```

END ELT. ERRORS: NONE. TIME: 0.703 SEC. IMAGE COUNT: 195

@HDC,P ****** IP ******

```
@:ELT,L -PF1-.IP
ELT 881 S74Q1C 07/11/79 08:48:56 (F)
     1.          00            REAL  FUNCTION   IP   (N, X, Y)
     2.          00      C
     3.          00      C           COMPUTE THE INNERPRODUCT OF TWO REAL VECTORS
     4.          00      C
     5.          00            INTEGER     N, I
     6.          00            REAL        X(N), Y(N)
     7.          00      C
     8.          00            IP = 0.0
     9.          00            DO 10 I = 1, N
    10.          00               IP = IP  +  X(I) * Y(I)
    11.          00         10 CONTINUE
    12.          00      C
    13.          00            RETURN
    14.          00            END

END ELT.  ERRORS: NONE. TIME:  0.109 SEC. IMAGE COUNT: 14

@HDG,P  ******  LISTPDE  ******
```

```
@:ELT,L -PF1-.NEUBAK
ELT 8R1 S74Q1C 07/11/79 08:49:06 (3)
```

```
      1.           03          SUBROUTINE  NEUBAK  (M, N, ALPHA, BETA, GAMMA, NULLSP)
      2.           00      C
      3.           00      C     ------------------------------------------------------
      4.           02      C     REVERSE THE SCALING CARRIED OUT BY 'NEUSCL'.  THERE WE
      5.           02      C     COMPUTED A SCALING TO REDUCE THE MAIN DIAGONAL OF A SPARSE
      6.           02      C     MATRIX TO BE THAT OF THE DISCRETE LAPLATIAN-NEUMANN PROBLEM.
      7.           02      C     WE WANT NOW:
      8.           02      C           A = D**.5  B  D**.5
      9.           03      C     WHERE THE VECTOR  NULLSP  HOLDS THE VALUES OF D**.5
     10.           00      C
     11.           00      C     ------------------------------------------------------
     12.           00      C
     13.           00      C     INTEGER     M, N
     14.           03          REAL        ALPHA(1), BETA(1), GAMMA(1), NULLSP(1)
     15.           00      C
     16.           00      C        THE VECTOR PARAMETERS ARE ONE-DIMENSIONAL ORDERINGS OF THE
     17.           00      C        TWO DIMENSIONAL GRID PROBLEM.  THE ACTUAL DIMENSION OF
     18.           00      C        THE VECTORS IS  'N' * 'M'.
     19.           00      C
     20.           00      C
     21.           02          INTEGER     K, XI, YJ
     22.           00          REAL        SCALE
     23.           00      C
     24.           00      C     ------------------------------------------------------------
     25.           00      C
     26.           00          K = 1
     27.           00          DO 200 YJ = 1, N
     28.           00             DO 100 XI = 1, M
     29.           03                SCALE = NULLSP(K)
     30.           02                ALPHA(K) = ALPHA(K) * (SCALE**2)
     31.           00                BETA(K)  = BETA(K)  * SCALE
     32.           00                GAMMA(K) = GAMMA(K) * SCALE
     33.           00                IF ( YJ .GT. 1)  GAMMA(K-M) = GAMMA(K-M) * SCALE
     34.           00                IF ( XI .GT. 1 )  BETA(K-1) = BETA(K-1) * SCALE
     35.           00                K = K + 1
     36.           00       100   CONTINUE
     37.           00       200 CONTINUE
     38.           00      C
     39.           00          RETURN
     40.           00          END
```

```
END ELT.  ERRORS: NONE.  TIME:  0.214 SEC.  IMAGE COUNT: 40

@HDG,P  ******  NEUSCL  ******
```

```
G:ELT,L -PF1-.NEUSCL
ELT 8R1 S74Q1C 07/11/79 08:49:07 (7)
      1.          07          SUBROUTINE  NEUSCL  (M, N, ALPHA, BETA, GAMMA, DXDYSQ, NULLSP)
      2.          04      C
      3.          04      C     ------------------------------------------------------
      4.          04      C     MULTIPLY THE SPARSE SYMMETRIC MATRIX WITH FIVE NON-ZERO
      5.          04      C     DIAGONALS (ALPHA, BETA, GAMMA)  BY A SCALING MATRIX  D
      6.          04      C     CHOSEN SO THE MAIN DIAGONAL REDUCES TO THAT OF THE DISCRETE
      7.          04      C     LAPLATIAN WITH NEUMANN BOUNDARY CONDITIONS.  IF  A  IS THE
      8.          04      C     ORIGINAL MATRIX, WE COMPUTE
      9.          04      C
     10.          04      C        B = D**(-.5)  A  D**(-.5)
     11.          04      C
     12.          04      C     WE RETURN THE DIAGONALS OF  B  IN THE PLACE OF THE
     13.          07      C     DIAGONALS OF  A.  THE VECTOR  NULLSP  RETURNS THE VALUES
     14.          07      C     OF THE NON-ZERO ELEMENTS OF  D**.5, WHICH GIVES THE NULLSPACE OF
     15.          07      C     THE SCALED OPERATOR (ASSUMING THE ORIGINAL OPERATOR WAS DERIVED
     16.          07      C     FROM A FIVE-POINT FORMULA, WITH NULLSPACE (1,1,1,...,1)')
     17.          04      C
     18.          04      C     ------------------------------------------------------
     19.          04      C
     20.          04      C     INTEGER     M, N
     21.          07          REAL        ALPHA(1), BETA(1), GAMMA(1), NULLSP(1), DXDYSQ
     22.          04      C
     23.          04      C        THE VECTOR PARAMETERS ARE ONE-DIMENSIONAL ORDERINGS OF THE
     24.          04      C        TWO DIMENSIONAL GRID PROBLEM.  THE ACTUAL DIMENSION OF
     25.          04      C        THE VECTORS IS  'N' * 'M'.
     26.          04      C
     27.          04      C
     28.          04          INTEGER     K, XI, YJ
     29.          04          REAL        FACTOR, SCALE
     30.          04      C
     31.          04      C     ------------------------------------------------------------
     32.          04      C
     33.          04      C        TWO DIMENSIONAL ORDERING USED IMPLICITLY TO FIND AND ADJUST
     34.          04      C        THE ELEMENTS CORRESPONDING TO BOUNDARY POINTS IN THE GRID
     35.          04      C
     36.          04          K = 1
     37.          04      C
     38.          04          DO 200 YJ = 1, N
     39.          04             DO 100 XI = 1, M
     40.          05                FACTOR = -2.0 * (1.0 + DXDYSQ)
     41.          04                IF ((XI .EQ. 1) .OR. (XI .EQ. M ))  FACTOR = FACTOR + 1.0
     42.          06                IF ((YJ .EQ. 1) .OR.  (YJ .EQ. N))  FACTOR = FACTOR + DXDYSQ
     43.          04      C
     44.          04                SCALE = SQRT (FACTOR / ALPHA(K) )
     45.          07                NULLSP(K) = 1.0 /  SCALE
     46.          04                ALPHA(K) = FACTOR
     47.          04                BETA(K)  = BETA(K)  * SCALE
     48.          04                GAMMA(K) = GAMMA(K) * SCALE
     49.          04                IF ( YJ .GT. 1)  GAMMA(K-M) = GAMMA(K-M) * SCALE
     50.          04                IF ( XI .GT. 1 )  BETA(K-1) = BETA(K-1) * SCALE
     51.          04                K = K + 1
     52.          04           100     CONTINUE
     53.          04           200 CONTINUE
     54.          04      C
     55.          04          RETURN
     56.          04          END
```

END ELT. ERRORS: NONE. TIME: 0.269 SEC. IMAGE COUNT: 56

@HDG,P ****** NEUSOL ******

```
@:ELT,L -PF1-.NEUSOL
ELT SR1 S74Q1C 07/11/79 08:49:10 (32)
      1.          31            SUBROUTINE  NEUSOL  (PPARM, X, PINVX)
      2.          28      C
      3.          28      C     ------------------------------------------------
      4.          28      C
      5.          28      C        USE SCHWARZTRAUBER ROUTINE  BLKTRI  TO SOLVE POISSON'S
      6.          28      C        EQUATION ON A RECTANGLE WITH NEUMANN BOUNDARY CONDITIONS
      7.          28      C
      8.          30      C        SOLVE    P Y  =  X
      9.          29      C        WHERE  P  IS THE DISCRETE NEUMANN LAPLATIAN
     10.          29      C
     11.          28      C     ------------------------------------------------------
     12.          28      C
     13.          31            REAL        PPARM(1), X(1), PINVX(1)
     14.          28      C
     15.          31      C        PPARM IS A LIST OF PARAMETERS FOR THIS ROUTINE.
     16.          28      C        THIS ROUTINE UNPACKS THEM TO PASS THEM TO  BLKTRI
     17.          28      C
     18.          28      C
     19.          28            INTEGER     INIT, N, M, SUBAM, SUBAN, SUBBM, SUBBN, SUBCM,
     20.          28           1           SUBCN, SUBWRK, NM, I, ERROR
     21.          28      C
     22.          29            REAL        ADUMMY
     23.          32      C
     24.          32            INTEGER     PRINTR
     25.          32            DATA        PRINTR / 6 /
     26.          29      C
     27.          28      C        UNPACKING ...
     28.          28      C
     29.          31            INIT = IFIX(PPARM(1))
     30.          31            M    = IFIX (PPARM(2))
     31.          31            N    = IFIX (PPARM(3))
     32.          28      C
     33.          31            SUBAM = IFIX (PPARM(4))
     34.          31            SUBAN = IFIX (PPARM(5))
     35.          31            SUBBM = IFIX (PPARM(6))
     36.          31            SUBBN = IFIX (PPARM(7))
     37.          31            SUBCM = IFIX (PPARM(8))
     38.          31            SUBCN = IFIX (PPARM(9))
     39.          31            SUBWRK = IFIX (PPARM(10))
     40.          29      C
     41.          28      C
     42.          28      C        ... COPY RIGHT HAND SIDE VECTOR  X  TO PROTECT IT ...
     43.          28      C
     44.          28            NM = N * M
     45.          28            DO 10 I = 1, NM
     46.          28               PINVX(I) = X(I)
     47.          28           10 CONTINUE
     48.          29      C
     49.          29      C        FORCE LINEAR SYSTEM TO BE CONSISTENT
     50.          29      C
     51.          29            CALL PROJCE (NM, PINVX, ADUMMY)
     52.          28      C
     53.          29      C
     54.          29      C        INVOKE NCAR ROUTINE ...
     55.          29      C
     56.          29            IF (INIT .NE. 1) GO TO 200
```

```
   57.          31           CALL BLKTRI (INIT, 1, N, PPARM(SUBAN), PPARM(SUBBN),
   58.          31          1                         PPARM(SUBCN),
   59.          31          2                    1, M, PPARM(SUBAM), PPARM(SUBBM),
   60.          31          3                         PPARM(SUBCM), M,
   61.          31          4               PINVX, ERROR, PPARM(SUBWRK))
   62.          29     C
   63.          29           CALL PROJCE (NM, PINVX, ADUMMY)
   64.          29     C
   65.          28     C
   66.          29           IF ( ERROR .EQ. 0 )  RETURN
   67.          28     C
   68.          28     C
   69.          32       100     WRITE (PRINTR, 6100) ERROR
   70.          29               STOP
   71.          29     C
   72.          29     C
   73.          32       200     WRITE (PRINTR, 6200)
   74.          29               STOP
   75.          28     C
   76.          28     C
   77.          28      6100 FORMAT ('0IMPROPER CALLING SEQUENCE TO ''BLKTRI''' /
   78.          28          1        '0EXECUTION BEING TERMINATED BY ''NEUSOL''' /
   79.          28          2        '0ERROR FLAG IS', I3)
   80.          29     C
   81.          29      6200 FORMAT ('0ERROR IN USE OF "BLKTRI"'/
   82.          29          1        ' FAST SOLVER NOT INITIALIZED')
   83.          28           END

END ELT.  ERRORS: NONE. TIME:  0.331 SEC.  IMAGE COUNT: 83

@HDG,P  ******  PROJCE  ******
```

```
@:ELT,L -PF1-.PROJCE
ELT 3R1 S74Q1C 07/11/79 08:49:10 (7)
      1.          04        SUBROUTINE   PROJCE   (N, X, ORTHOG)
      2.          03    C
      3.          03    C     PROJECT THE  N  VECTOR  X  ONTO THE ORTHOGONAL
      4.          04    C     COMPLEMENT OF THE VECTOR  E.
      5.          03    C
      6.          03    C     ================================================
      7.          03    C
      8.          03        INTEGER      N
      9.          04        REAL         X(N), ORTHOG
     10.          03    C
     11.          03        INTEGER      I
     12.          07        REAL         IPXE
     13.          03    C
     14.          04        IPXE  = 0.0
     15.          03    C
     16.          03        DO 100 I = 1, N
     17.          04           IPXE  = IPXE  +  X(I)
     18.          03      100 CONTINUE
     19.          03    C
     20.          04        ORTHOG = IPXE / N
     21.          03    C
     22.          03        DO 200 I = 1, N
     23.          04           X(I) = X(I) - ORTHOG
     24.          03      200 CONTINUE
     25.          03    C
     26.          03        RETURN
     27.          03        END

END ELT.  ERRORS: NONE. TIME:  0.151 SEC. IMAGE COUNT: 27

@MDG,P  ******  PROJCT  ******
```

```
@:ELT,L -PF1-.PROJCT
ELT 8R1 S74Q1C 07/11/79 08:49:12 (8)
     1.          04          SUBROUTINE   PROJCT   (N, X, P)
     2.          04     C
     3.          04     C    PROJECT THE  N  VECTOR  X  ONTO THE ORTHOGONAL
     4.          04     C    COMPLEMENT OF THE VECTOR  P.
     5.          04     C
     6.          04     C    ============================================
     7.          04     C
     8.          04          INTEGER     N
     9.          04          REAL        X(N), P(N)
    10.          04     C
    11.          04          INTEGER     I
    12.          08          REAL        NORMP, IPXP, ORTHOG
    13.          04     C
    14.          04          NORMP = 0.0
    15.          04          IPXP  = 0.0
    16.          04     C
    17.          04          DO 100 I = 1, N
    18.          04             NORMP = NORMP  +  P(I)**2
    19.          04             IPXP  = IPXP  +  X(I) * P(I)
    20.          04        100 CONTINUE
    21.          04     C
    22.          05          ORTHOG = IPXP / NORMP
    23.          04     C
    24.          04          DO 200 I = 1, N
    25.          04             X(I) = X(I) - ORTHOG*P(I)
    26.          04        200 CONTINUE
    27.          04     C
    28.          04          RETURN
    29.          04          END

END ELT.  ERRORS: NONE. TIME:  0.157 SEC. IMAGE COUNT: 29

@HDG,P  ******  VECOUT  ******
```

```
@:ELT.L -PF1-.VECOUT
ELT 8R1 S74Q1C 07/11/79 08:49:13 (7)
     1.          03            SUBROUTINE  VECOUT   (M, N, VEC)
     2.          03      C
     3.          03      C        OUTPUT A BLOCK VECTOR IN SEMI-READABLE FORM
     4.          03      C
     5.          04      C           M    : X DIMENSION GRID POINTS
     6.          03      C           N    : Y DIMENSION GRID POINTS
     7.          03      C
     8.          03            INTEGER         M, N
     9.          03            REAL            VEC(M,N)
    10.          03      C
    11.          03      C
    12.          04            INTEGER         MAXWID, NPAGES, P, START, FINISH
    13.          04            DATA            MAXWID  /  8  /
    14.          07      C
    15.          07            INTEGER     PRINTR
    16.          07            DATA        PRINTR / 6 /
    17.          03      C
    18.          03      C     ================================================
    19.          03      C
    20.          04            NPAGES = M / MAXWID
    21.          04            IF (NPAGES * MAXWID  .LT.  M)   NPAGES = NPAGES + 1
    22.          03      C
    23.          03            DO 100 P = 1,NPAGES
    24.          03                START = (P-1) * MAXWID  +  1
    25.          04                FINISH = MIN1(P*MAXWID, M)
    26.          03                DO 50 J = 1, N
    27.          07                    WRITE (PRINTR, 6000) (VEC(I,J), I = START, FINISH)
    28.          03           50   CONTINUE
    29.          07            WRITE (PRINTR, 6100)
    30.          03          100 CONTINUE
    31.          03      C
    32.          04            RETURN
    33.          04      C
    34.          04      C     ============================
    35.          04      C
    36.          06       6000 FORMAT (' ', 1P8E9.2)
    37.          04      C
    38.          05       6100 FORMAT (' ')
    39.          04      C
    40.          04            END

END ELT.  ERRORS: NONE. TIME:  0.181 SEC. IMAGE COUNT: 40

@HDG,N

@OFF
```

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 80-2056 (NBS)	2. Performing Organ. Report No.	3. Publication Date April 16, 1980

4. TITLE AND SUBTITLE

The Numerical Solution of a Nonseparable Elliptic Partial Differential Equation by Preconditioned Conjugate Gradients

5. AUTHOR(S)

John Gregg Lewis and Ronald G. Rehm

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Center for Applied Mathematics
National Engineering Laboratory
National Bureau of Standards

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

In this report the combination of an iterative technique, the conjugate gradient algorithm, with a fast direct method, cyclic reduction, is used to solve the linear algebraic equations resulting from discretization of a nonseparable elliptic partial differential equation. An expository discussion of the conjugate gradient and preconditioned conjugate gradient algorithms and of their use in the solution of partial differential equations is presented. New results extending the use of the preconditioned conjugate gradients technique to singular linear equations which arise from discretized elliptic equations with Neumann boundary conditions are also given. The algorithms are applied to solve a specific elliptic equation which arises in the study of buoyant convection produced by a room fire. A code was developed to implement the algorithms for this application. Numerical results obtained through testing and use of the code are discussed.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

conjugate gradient algorithm; elliptic partial differential equations; iterative methods for linear algebraic equations; Neumann boundary conditions; space matrices.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

66

15. Price

$8.00