Determining the Resistance within a 3-Dimensional Network

Christina H. Lamont

June 1989

1 Introduction

In this paper it is shown that the resistors in a cubic network can be determined by measurements at the boundary of the currents generated by imposed voltages. First we solve the Dirichlet problem for the continuity equation and give a discretized version of this solution. Next, we define a map of Dirichlet data to Neumann data and describe its uses within the context of this paper. Finally, we give an algorithm for using the boundary measurements to compute the resistances and supply numerical results.

2 The Dirichlet Problem

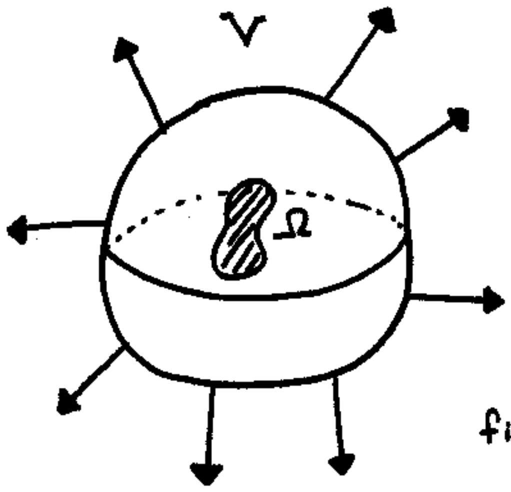

fig. 1

Assume we are given a 3-dimensional region, V, with conductivity $\gamma(x,y,z)$ and potential $u(x,y,z)$ throughout. If we assume, also, that there are no sources of current within V, then we know the total current flowing into V is zero. Thus, the Divergence Theorem implies for every region, Ω, within V

$$\begin{aligned} 0 &= \int\int_{\partial\Omega} \gamma \vec{grad u} \cdot \vec{u} dA \\ &= \int\int\int_{\Omega} div(\gamma grad u) dV \end{aligned}$$

Thus, since the integral must be zero for all such Ω, it must be true that $div(\gamma grad u) = 0$. Assume, now, that we are given only the potential, ϕ, along the boundary of V (call it ∂V) and the conductivity, $\gamma(x,y,z)$, throughout V. Our problem then becomes, find a potential function $u(x,y,z)$ defined throughout V such that $u = \phi$ on ∂V in some sense. That is, find u such that $u = \phi$ on ∂V and $L_\gamma u = 0$ where $L_\gamma u \equiv div(\gamma grad u)$.
Since

$$\begin{aligned} 0 &= div(\gamma grad u) \\ &= (\gamma u_x)_x + (\gamma u_y)_y + (\gamma u_z)_z \end{aligned}$$

and we are solving for u, we would like to find approximations to $(\gamma u_x)_x, (\gamma u_y)_y, (\gamma u_z)_z$ in terms of $u(x, y, z)$.

If we define

$$D_h(x) \equiv \frac{u(x+h,y,z) - u(x,y,z)}{h}$$

then

$$(\gamma u_x)_x \sim \frac{\gamma(x+\frac{h}{2},y,z)D_h(x) - \gamma(x-\frac{h}{2},y,z)D_h(x-h)}{h}$$

Note:

$$\lim_{h\to 0} \gamma(x+\frac{h}{2},y,z)\Big[\frac{u(x+h,y,z)-u(x,y,z)}{h^2}\Big] + \gamma(x-\frac{h}{2},y,z)\Big[\frac{u(x-h,y,z)-u(x,y,z)}{h^2}\Big]$$

$$\begin{aligned}
&= \lim_{h\to 0}\left[\frac{\gamma(x,y,z)+\frac{h}{2}\gamma_x(x,y,z)}{h}\right]\left[u_x(x,y,z)+\frac{h}{2}u_{xx}(x,y,z)\right] \\
&\quad + \lim_{h\to 0}\left[\frac{\gamma(x,y,z)-\frac{h}{2}\gamma_x(x,y,z)}{h}\right]\left[\frac{h}{2}u_{xx}(x,y,z)-u_x(x,y,z)\right] \\
&= \lim_{h\to 0}\left(\frac{\gamma(x,y,z)}{h}\right)(hu_{xx}(x,y,z)) + \frac{1}{h}(h\gamma_x(x,y,z))(u_x(x,y,z)) \\
&= \gamma\left(\frac{\partial^2}{\partial x^2}u\right) + \left(\frac{\partial}{\partial x}\gamma\right)\left(\frac{\partial}{\partial x}u\right) \\
&= \left(\gamma\frac{\partial u}{\partial x}\right)_x
\end{aligned}$$

Thus,

$$\frac{\gamma(x+\frac{h}{2},y,z)D_h(x) - \gamma(x-\frac{h}{2},y,z)D_h(x-h)}{h}$$

is a reasonable approximation to $(\gamma u_x)_x$.
Similarly,

$$(\gamma u_y)_y \sim \frac{\gamma(x,y+\frac{h}{2},z)D_h(y) - \gamma(x,y-\frac{h}{2},z)D_h(y-h)}{h}$$

$$(\gamma u_z)_z \sim \frac{\gamma(x,y,z+\frac{h}{2})D_h(z)-\gamma(x,y,z-\frac{h}{2})D_h(z-h)}{h}$$

where $D_h(y)$,$D_h(z)$ are defined in the same manner as $D_h(z)$.

If we let $h=1$, then

$$\begin{aligned}
0 &= div(\gamma gradu) \\
&= L_\gamma u \\
&= (\gamma u_x)_x + (\gamma u_y)_y + (\gamma u_z)_z \\
&\sim \gamma(x+\frac{1}{2})\,[u(x+1)-u(x)] + \gamma(x-\frac{1}{2})\,[u(x-1)-u(x)] \\
&\quad +\gamma(y+\frac{1}{2})\,[u(y+1)-u(y)] + \gamma(y-\frac{1}{2})\,[u(y-1)-u(y)] \qquad (1) \\
&\quad +\gamma(z+\frac{1}{2})\,[u(z+1)-u(z)] + \gamma(z-\frac{1}{2})\,[u(z-1)-u(z)]
\end{aligned}$$

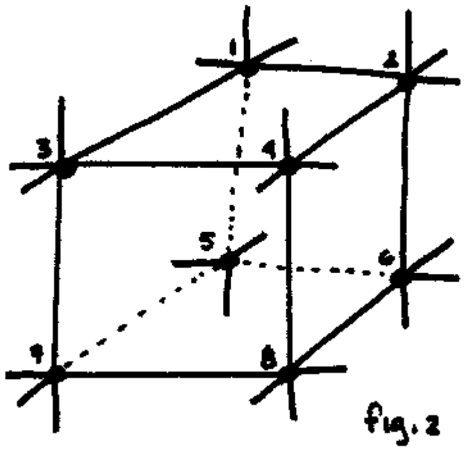

fig. 2

Let us now turn to the specific type of region with which we concern ourselves in this paper, the cubic lattice. Let there be N steps along each edge of the cube that the lattice network has $(N-1)^3$ interior points. In figure (2), for example, $N=3$. We will call any lattice point a node and the segment joining any two nodes an edge. Sometimes we refer to an edge as a resistor. If p is a node such that it is not an interior to the lattice, then p is said to be a boundary node.

Suppose, as with the general region, we are given a potential function $f(p)$, where p ranges over all boundary nodes, and conductivity, $\gamma(pq)$, for all edges, pq, in the network. Then we would like to find a potential function $u(p)$ over all nodes p (boundary and interior) such that $L_\gamma u = 0$ and $u(p) = f(p)$ whenever p is a boundary node. Notice, however, that equation (1) is just Kirchoff's Law applied at an interior node.

If p is any node, then define

$$N(p) \equiv \{q | q \text{ is a node connected to } p \text{ by a single edge }\}$$

Thus, equation (1) is equivalent to

$$0 = \sum_{q \in N(p)} \gamma(pq)(u(p) - u(q))$$

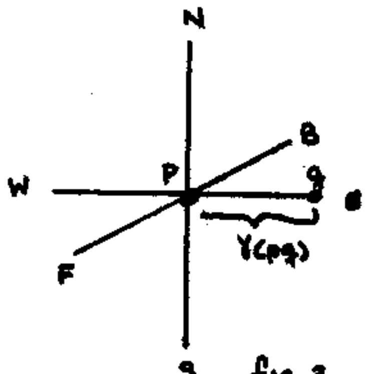

fig. 3

Notice, in addition, if we apply Kirchoff's Law at every interior node, then we get a matrix equation in $(N-1)^3$ variables and $(N-1)^3$ unknowns. The following proposition shows the function $u(p)$ is unique.

Definition Let Ω_0 be the set of all nodes in the network, $int\Omega_0$ be the set of all interior nodes, and $\partial\Omega_0$ be the set of all boundary nodes

Proposition Let $\phi : \partial\Omega_0 \to \Re$ be given. Then there exists a unique function $f : \Omega_0 \to \Re$ such that $L_\gamma f(p) = 0$ for all $p \in int\Omega_0$ and $f(p) = \phi(p)$ for all $p \in \partial\Omega_0$.

Therefore, if we are given potential along the boundary and conductivity throughout, then there is only one possible function u which solves our problem. As an example consider the cube of figure (2). Suppose $\gamma(pq) = 1$ for every edge within the network. Since $N = 3$ and there are always $(N-1)^3$ interior nodes, we must consider 8 equations; we need only find the potential at interior nodes (since the potential at the boundary nodes is given), thus, there are 8 unknown values. Below is the matrix equation, $Ax = b$, to be solved.

$$\begin{pmatrix} 6 & -1 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 6 & 0 & -1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 6 & -1 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 6 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 6 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & -1 & 6 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 & 0 & 6 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & -1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{pmatrix}$$

where

$$\begin{aligned} x_i &= u(p_i) \\ b_i &= \sum_{q' \in N(x_i)\ q' \in \partial\Omega_0} \gamma(p_i q') u(q') \end{aligned}$$

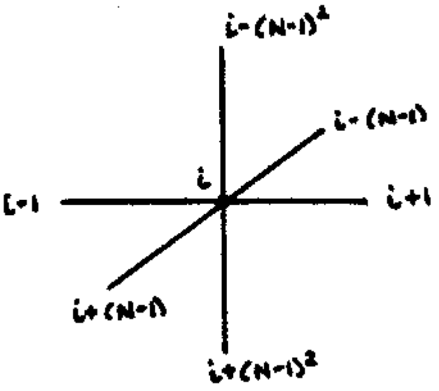

fig.4

In general, $b = (b_1, ..., b_{(N-1)^3})$ and $x = (x_1, ..., x_{(N-1)^3})$, where b_i, x_i are defined as above. Before we can give a general description of the matrix, A, we must establish a system of indexing. Assume

the upper left-hand node found furthest back in the network is always labelled node 1. Assume, also, the labelling of neighbors of a node i suggested in figure (4).
Note:

1. If $i \leq (N-1)^2$, then i does not have a northern neighbor.
2. If $i > (N-2)(N-1)^2$, then i does not have a southern neighbor.
3. If $j(N-1)^2 < i \leq j(N-1)^2 + (N-1)$, $j = 0, ..., (N-2)$, then i does not have a neighbor behind.
4. If $j(N-1)^2 \geq i > j(N-1)^2 - (N-1)$, $j = 1, ..., (N-1)$, then i does not have a neighbor in front.
5. If $i = j(N-1)$, $j = 1, ..., (N-1^2$, then i does not have an eastern neighbor.
6. If $i = j(N-1)+1$, $j = 0, ..., (N-1)^2$, then i does not have a western neighbor.

This indexing scheme will be used throughout this paper.

Now we can give the general description of the matrix, A.
A is an $(N-1)^3 X (N-1)^3$ block tridiagonal matrix

$$A = \begin{bmatrix} B_1 & C_1 & 0 & \dots & 0 \\ C_1 & . & . & \dots & . \\ 0 & . & . & \dots & . \\ . & . & . & \dots & 0 \\ . & \dots & . & . & C_{N-2} \\ 0 & \dots & 0 & C_{N-2} & B_{N-1} \end{bmatrix}$$

where each B_i, C_i is an $(N-1)^2 X (N-1)^2$ matrix.

$$C_i = \begin{bmatrix} -\gamma_{i,1} & 0 & \dots & 0 \\ 0 & . & \dots & . \\ . & \dots & \dots & 0 \\ 0 & \dots & 0 & -\gamma_{i,(N-1)^2} \end{bmatrix} \qquad i = 1, ..., N-2$$

where $\gamma_{i,j}$ = conductivity between node $(i-1)(N-1)^2 + j$ and node $i(N-1)^2 + j$ (the southern interior neighbor).

1. $\gamma_{i,j}$ equals the conductivity between node $i(N-1)^2 + j$ and its northern interior neighbor.

2. If $i = 1$, then node j does not have a northern interior neighbor; if $i = (N-1)$, then node $(N-1)^2(N-2)+j$ does not have a southern interior neighbor.

B_i is a block tridiagonal matrix of the form:

$$B_i = \begin{bmatrix} D_{i,1} & E_{i,1} & 0 & \dots & 0 \\ E_{i,1} & . & . & \dots & . \\ 0 & . & . & \dots & . \\ . & . & . & \dots & 0 \\ . & \dots & . & . & E_{i,N-2} \\ 0 & \dots & 0 & E_{i,N-2} & D_{i,N-1} \end{bmatrix} \qquad i = 1, ..., N-1$$

where each $D_{i,j}, E_{i,j}$ is an $(N-1)X(N-1)$ matrix.

$$E_{i,j} = \begin{bmatrix} -\gamma_{i,j,1} & 0 & \dots & 0 \\ 0 & . & \dots & . \\ . & \dots & \dots & 0 \\ 0 & \dots & 0 & -\gamma_{i,j,(N-1)^2} \end{bmatrix} \qquad \begin{aligned} i &= 1, ..., N-1 \\ j &= 1, ..., N-2 \end{aligned}$$

where $\gamma_{i,j,k}$ = conductivity between node $(i-1)(N-1)^2 + (j-1)(N-1) + k$ and node $(i-1)(N-1)^2 + j(N-1) + k$ (the forward interior neighbor).
As with the matrix C_i, note:

1. $\gamma_{i,j,k}$ is also the conductivity between node $(i-1)(N-1)^2 + j(N-1) + k$ and its interior neighbor behind.

2. If $j = 1$, then node $(i-1)(N-1)^2 + k$ has no interior neighbor behind; if $j = N-2$, then node $(i-1)(N-1)^2 + (N-2)(N-1) + k$ has no forward interior neighbor.

Finally,

$$D_{i,j} = \begin{bmatrix} -\gamma_{i,j,1} & -\beta_{i,j,1} & 0 & \dots & 0 \\ -\beta_{i,j,1} & . & . & \dots & . \\ 0 & . & \dots & \dots & . \\ . & \dots & \dots & . & 0 \\ . & \dots & . & . & -\beta_{i,j,(N-1)^2} \\ 0 & \dots & 0 & -\beta_{i,j,(N-1)^2} & -\gamma_{i,j,(N-1)^2} \end{bmatrix} \qquad \begin{aligned} i &= 1, ..., N-1 \\ j &= 1, ..., N-1 \end{aligned}$$

where $\beta_{i,j,k}$ = conductivity between node $(i-1)(N-1)^2+(j-1)(N-1)+k$ and the node $(i-1)(N-1)^2+(j-1)(N-1)+k+1$ (the east interior neighbor) and

$$\gamma_{i,j,k} = \sum_{q \in N(p)} \gamma(pq)$$

where p is node $(i-1)(N-1)^2+(j-1)(N-1)+k$.

Note:

1. If $k=1$, then $(i-1)(N-1)^2+(j-1)(N-1)+1$ has no west interior neighbor
2. If $k=N-2$, then $(i-1)(N-1)^2+j(N-1)$ has no east interior neighbor.

Having solved the matrix equation Ax=b, we know the potential at every interior node. Now we need to find the current flowing in at every boundary node.

Let q' be a boundary node and let p be its interior neighbor. We have been given (as initial data), the potential at q' and the conductivity $\gamma(pq')$. In addition, we know, by definition of conductivity, that $\gamma V = I$. Therefore, if $C(q')$ is the current flowing in at q', then

$$C(q') = \gamma(pq')\left[u(p) - u(q')\right].$$

3 The Dirichlet to Neumann Map

Suppose we are given a conductivity function, γ, throughout a network. Let $i = 1,...,6(N-1)^2$ where $6(N-1)^2$ is the total number of boundary nodes. If we let $\{\Omega_{ij}\}_{j=1}^{6(N-1)^2}$ be a set of values across all boundary nodes where

$$\Omega_{ij} = \left\{ \begin{array}{ll} 0 & i \neq j \\ 1 & i = j \end{array} \right.$$

then for each i we can solve the Dirichlet problem as mentioned in the previous section and find a vector C_i containing the currents flowing in at each boundary node. Thus, we can form a $6(N-1)^2 X 6(N-1)^2$ matrix

$$\Lambda_\gamma = (C_1, ..., C_{6(N-1)^2})$$

There are two primary uses for Λ_γ

1. Given any vector v of potentials along the boundary, $\Lambda_\gamma v$ gives the corresponding current flowing in at each boundary node (for the given conductivity function, γ). Thus, Λ_γ maps Dirichlet data to Neumann data.

2. It is possible to define a potential function along the boundary which describes a specific pattern of potentials on the interior nodes. In the following paragraphs the desired pattern of potentials is described and it is explained how to find the necessary potential function along the boundary.

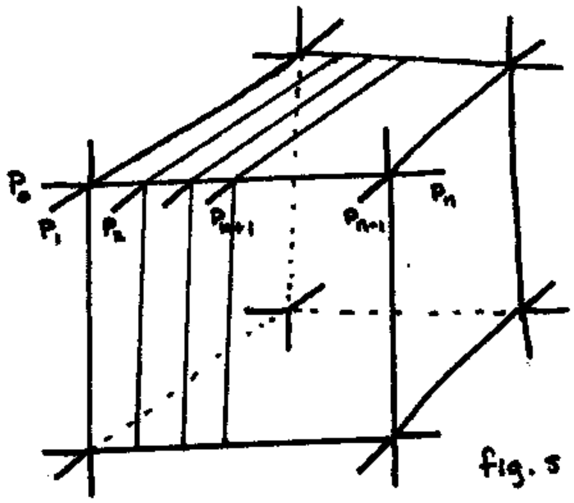

fig. 5

Lemma Let S be the set of all nodes in planes $P_0, P_1, \ldots, P_k$ and let γ be defined throughout the network. Assume f is a function defined on S such that $L_\gamma f(p) = 0$ for all nodes p interior to S. Then f can be defined on P_{k+1} so that $L_\gamma f(p) = 0$ for all nodes $p \in S$. The definition of f is uniquely determined on the interior nodes of P_{k+1} and can be given arbitrary values at the boundary nodes of P_{k+1}.

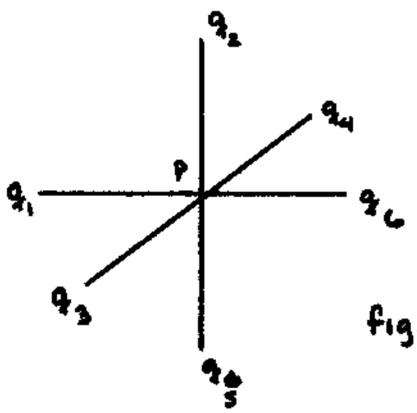

fig. 6

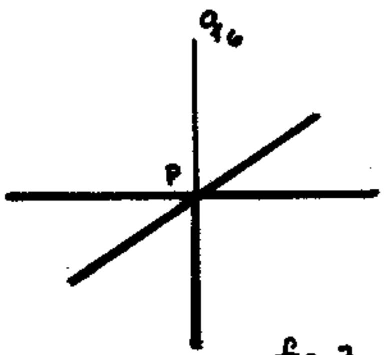

fig. 7

Consider figure (6). Basically, what the above lemma says is that if

$$\begin{aligned} q_1 &\in P_{k-1} \\ q_2, ..., q_5, p &\in P_k \\ q_6 &\in int P_{k+1}, \end{aligned}$$

then according to Kirchoff's Law there is only one choice for the value of q_6. If q_6 is a boundary node, as in figure (7), then

$$C(q_6) = \gamma(q_6 p)(f(q_6) - f(p)).$$

However, $C(q_6)$ is unknown and the lemma does not indicate any means, other than Kirchoff's Law, for determing $C(q_6)$. Thus, $f(q_6)$ is arbitrary. Note, however, that once the potential has been fixed at a boundary node and its adjacent neighbor, then the current flowing into that boundary node has been determined.

The next lemma follows immediately.

Lemma Let a network with conductivity function, γ, be given. Suppose $f \equiv 0$ on the nodes of planes $P_1, ..., P_l$. Suppose, also, we are given a boundary node $q \in P_{l+1}$ such that q is in the m_{th} row of the north face of the network and $f(q) = 1$. Then by letting $f(p') = 0$ for all boundary nodes $p' \neq q$ such that $f(p')$ is arbitrary, we can continue f so that $L_\gamma f(p) = 0$ for all interior nodes p in the network and $f(p) = 0$ for all interior nodes p_s in the network where

$$\begin{aligned} s &= i + j(N-1) + k(N-1)^2 \\ i &= k - l - 1, ..., N \\ j &= 0, ..., m - k + l + 1; \\ & \quad m + i - l - 1, ..., N - 2 \\ k &= l + 1, ..., N - 2 \end{aligned}$$

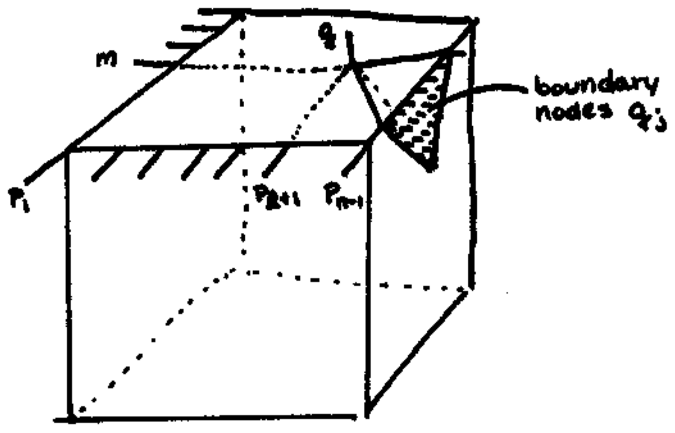

fig. 8

If f is continued as described in the previous lemma, then there are boundary nodes along the east face at which the potential is non-zero. We label such boundary nodes $q_1, ..., q_h$ and the associated potentials $\alpha_1, ..., \alpha_h$. In order to get these non-zero potentials, there must be some current flowing through the network. The nodes p_s, where s is as mentioned earlier, outline the region through which this current flows. Because of the shape it takes, we call this region the wedge, W. Notice that the α_j's, $j = 1, ..., h$ are uniquely determined by the continuation process.

In the problem to be considered in this paper, we will want to know the values α_j, but will not know the conductivity across all the resistors in the network. Thus, we need a reasonable method of finding the α_j's.

Note the following with regard to the previous lemma:

1. The current flowing in at each boundary node of the west face will be zero.

2. A potential function has been defined along the boundary.

Thus, if p is a boundary node of the west face and $C(p)$ is the amount of current flowing in at p, then

$$\begin{aligned} 0 &= C(p) \\ &= \Lambda_\gamma(p,q) + \alpha_1\Lambda_\gamma(p,q_1) + \ldots + \alpha_h\Lambda_\gamma(p,q_h) \end{aligned}$$

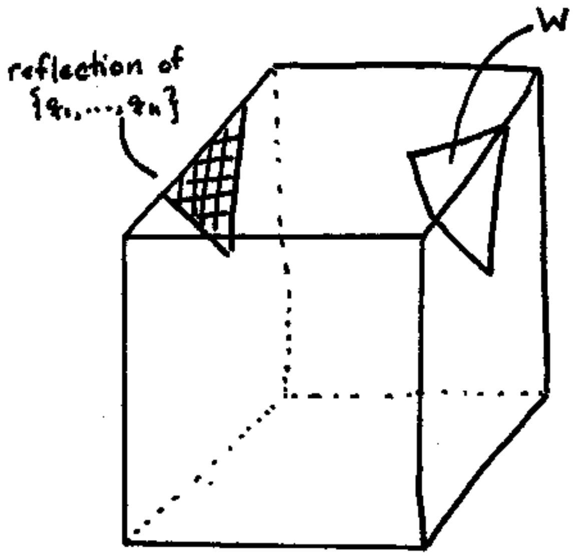

fig. 9

Assume $\{p_1, ..., p_h\}$, a collection of boundary nodes, is the reflection of $\{q_1, ..., q_h\}$ onto the west face. Calculations indicate that, in order to find reasonable approximations to $\alpha_1, ..., \alpha_h$, we only need to consider the h equations formed when $p \in \{p_1, ..., p_h\}$; we conjecture that these equations in fact determine a linearly independent solution. We will refer to this method of finding the α_j's as $A(W) = (\alpha_1, ..., \alpha_h)$.

4 Algorithm

We would like to construct an algorithm which finds the conductivity across every resistor in the network when we only know the corresponding Lambda matrix, Λ_γ. The use of the function $A(W) = (\alpha_1, ..., \alpha_h)$ is central to our approach since it restricts the flow of current to the wedge, W.

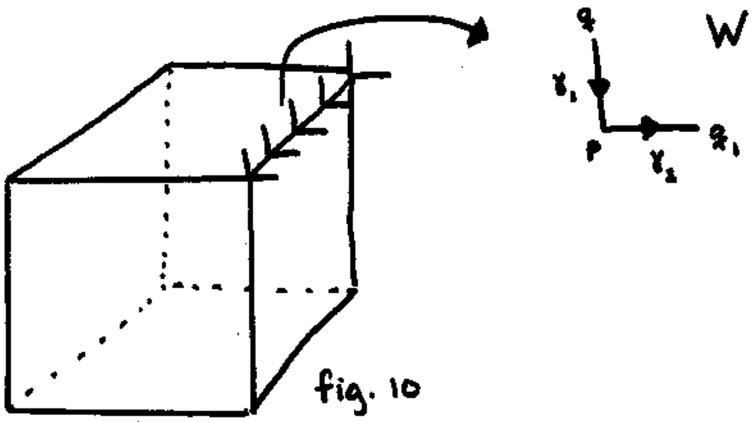

fig. 10

Consider the wedge, W, of figure (10). $A(W) = \alpha$ implies $f(q) = 1, f(q_1) = \alpha, f(p) = 0$, where f is a potential function. Since we know the potential at all boundary nodes, we can use Λ_γ to find $C(q), C(q_1)$, the current flowing in at q, q_1,respectively. Next, assume $\gamma_1 = \gamma(qp), \gamma_2 = \gamma(pq_1)$. Then, by definition of conductance

$$\gamma_1 = \frac{C(q)}{f(q) - f(p)} = C(q)$$

$$\gamma_2 = \frac{C(q_1)}{f(q_1) - f(p)} = \frac{C(q_1)}{\alpha}$$

Thus, it is relatively simple to find the conductances across any resistor within a wedge of the type shown in figure (10). Assume we have done this for all such wedges.

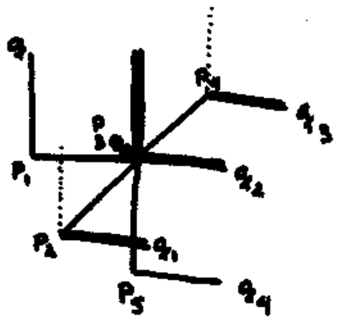

fig. 11

Consider, now, the wedge, W, of figure (11). As a result of our previous work, we know the conductivity across the resistors which are highlighted. Again we consider $A(W) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ which puts

$$f(q) = 1 \;\; f(q_1) = \alpha_1 \;\; f(q_2) = \alpha_2 \;\; f(q_3) = \alpha_3 \;\; f(q_4) = \alpha_4$$

and uses Λ_γ to find $C(q), ..., C(q_4)$. Let

$$\gamma_1 = \gamma(qp_1) \quad \gamma_2 = \gamma(p_1p_3) \quad \gamma_3 = \gamma(p_2p_3) \quad \gamma_4 = \gamma(p_3, p_4) \quad \gamma_5 = \gamma(p_3p_5) \quad \gamma_6 = \gamma(q_4p_5)$$

$$\gamma_1' = \gamma(q_1p_2) \quad \gamma_2' = \gamma(q_3p_4)$$

Note: γ_1', γ_2' are already known.
Thus,

$$\gamma_1 = C(q) \qquad \gamma_2 = -\frac{\gamma_1}{f(p_3)}$$

$$\gamma_3 = -\frac{\alpha_1\gamma_1'}{f(p_3)} \qquad \gamma_4 = -\frac{\alpha_3\gamma_2'}{f(p_3)}$$

$$\gamma_5 = -\frac{\gamma_6}{f(p_3)} \qquad \gamma_6 = \frac{C(q_4)}{\alpha_4}$$

The iteration continues in a similar manner, taking large wedges at each step. The process is generalized in what follows.

Let v be the iteration count (indicating the size of the wedge under consideration). v will run from 0 to $N-2$. We will also have a counter h which indicates at least how far from the end of an edge the boundary node, b, should be. $A(W)$ implies $f(b) = 1$ (see figure (12)). Initially, $h = 0$. Every time v is incremented, we check to see if v is odd; if it is, we increment h.

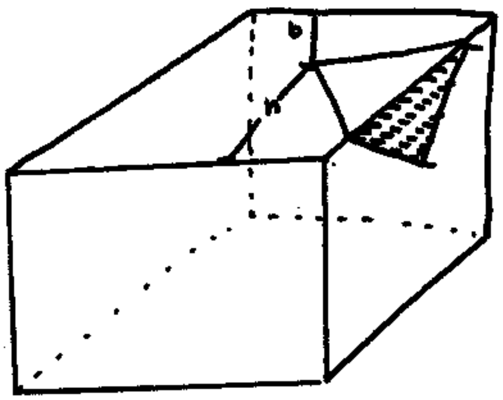

fig. 12

Let $i = h+1, ..., N-h-1$ and consider one of the twelve edges, E. i will index the boundary nodes b previously mentioned. At this point we implement $A(W)$ where q corresponds to the boundary node b. Thus, for this wedge, W_v, $A(W_v)$ defines a potential at every point of the boundary and we have forced current to flow only through W_v. Assume we have found the conductivity across all resistors in any wedge W_v where $v = k$. Our task is then to find the conductivity across all resistors in a wedge, W_{k+1}.

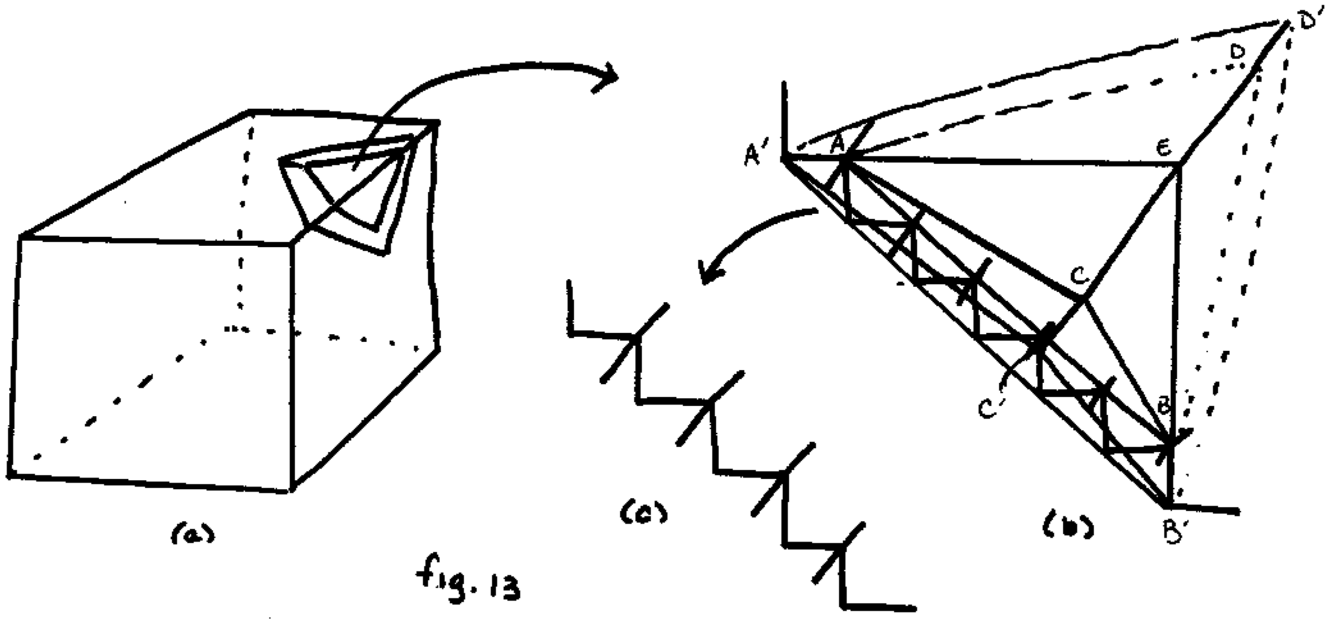

fig. 13

That is, assume we know the conductance for any wedge $ABCD$ and are trying to find the conductance for the wedge $A'B'C'D'$ which contains $ABCD$. Since we have also found the conductance for all other wedges considered at step $v = k$, we need only find the conductivity across those resistors shown in figure (13c).

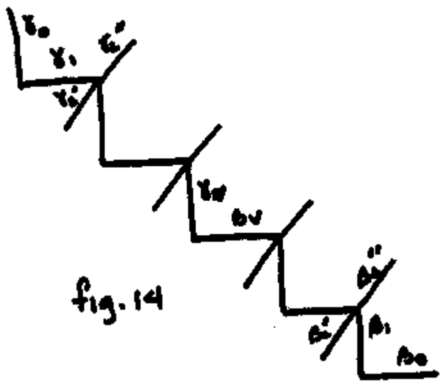

fig. 14

When finding the conductivity across these resistors, consider the system of labelling suggested in figure (14). Assume the γ's,β's are the conductances to be found across the indicated resistors. First we implement a procedure (yet to be described) to find $\gamma_0, ..., \gamma_v, \gamma_2', \gamma_2'', ..., \gamma_{v/2}', \gamma_{v/2}''$. Then we use a similar procedure to find the β's,β''s,β'''s.

γ_0 and γ_1 are trivial to determine; all we need to know is the current flowing in at the adjacent boundary nodes. As mentioned earlier, these values can be found through the use of Λ_γ.

Assume,now, that we are trying to find γ_j where j is even. At this step we must also find γ_j', γ_j''. However, the resistors across which these conductances flow may be buried deep within our network and we need to know something about the current and potential near these resistors. Thus, we define a recursive procedure to find the currents and potentials at certain nodes. Throughout this procedure we apply Kirchoff's Law at northern neighbors. Thus, the following assumption is helpful: if $f(p)$ was found while determining $\gamma_i, i < j$, then $f(p')$ is known (where p' is the northern neighbor of p) and so are the potentials at all other neighbors of p'.

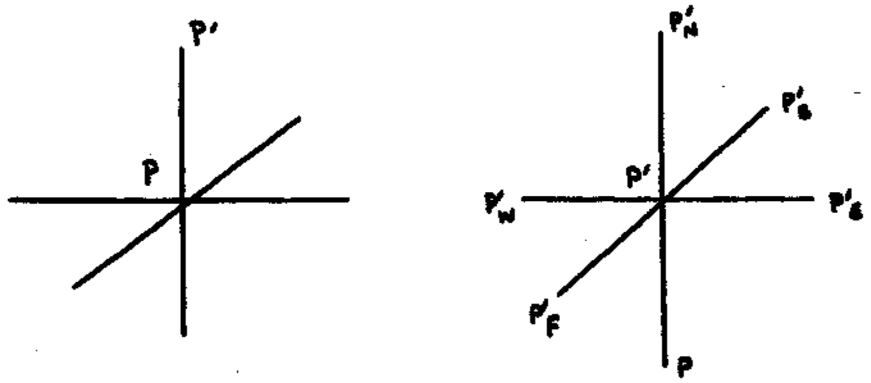

fig. 15

Consider now figure (16).

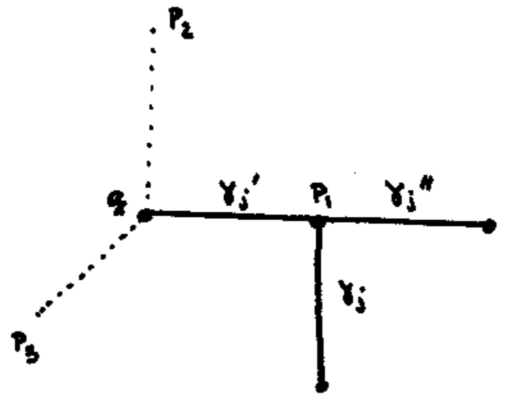

fig. 16

By our definition of the wedge, W_v, we know $f(q) = 0$. Assume $f(p_1), f(p_2)$ were determined for $i = j-1, j-2$, respectively. We would like to apply Kirchoff's Law at node q; however, $f(p_3)$ is yet to be determined. In order to find $f(p_3)$, we must consider the northern neighbor. Call it p'.

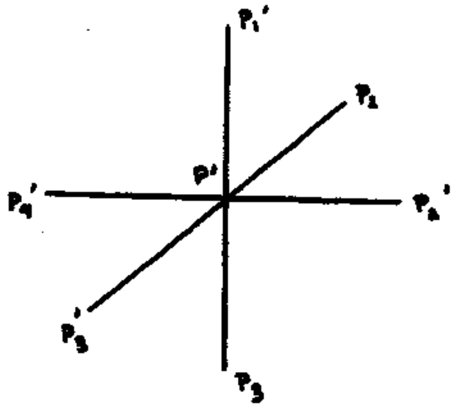

fig. 17

If p' is a boundary node it is trivial to determine $f(p)$. Otherwise, we must apply Kirchoff's Law at p'. By our assumption, we know $f(p_1'), f(p_2), f(p_2')$. However, we do not know $f(p')$. Thus, we must also apply Kirchoff's Law at p_1', where $f(p_1')$ is known. In addition, we don't know $f(p_3'), f(p_4')$ and we must apply Kirchoff's Law at these nodes as well. Since N, the number of steps along an edge, is finite, the northern neighbors will ultimately be boundary nodes. Therefore, we have defined a recursive procedure for finding $f(p_3)$ (see figure (16)). Thus, we can find γ_j'. Similarly, we could solve for γ_j''.

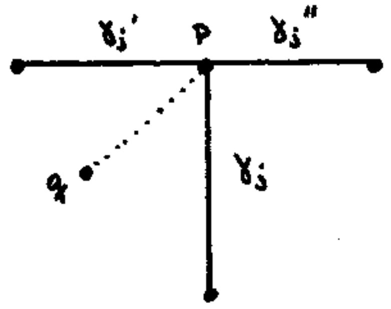

fig. 18

Likewise, we could use such a recursive to find $f(q)$, see figure (18). Applying Kirchoff's Law at node p allows us to solve for γ_j.

Assume, now, we wish to find γ_j when j is odd and γ_i is known for all $i < j$.

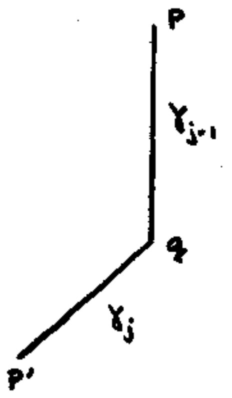

fig. 19

By definition of the wedge, W_v, we know $f(q) = 0$. Kirchoff's Law applied at q implies

$$0 = f(p')\gamma_j + f(p)\gamma_{j-1}$$

$$\gamma_j = -\gamma_{j-1}\frac{f(p)}{f(p')}$$

$f(p)$ was determined when we solved for γ_{j-1}. Thus, we need only to find $f(p')$. Note, if p is a boundary node, then finding $f(p')$ is trivial; otherwise, we must apply Kirchoff's Law at the northern neighbor of p'. Call it p_1.

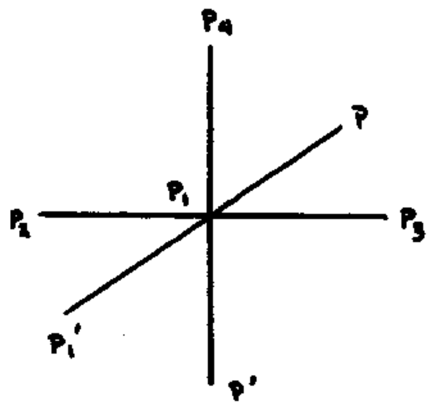

fig. 20

Assume the nodes are as labelled in figure (20).
Then $f(p_4)$ is known since $f(p)$ is known and $f(p_1)$,$f(p_2)$,$f(p_3)$ were determined for $i = j - 1$. Therefore, only $f(p_1')$ is unknown. As before, we apply a recursive procedure to find $f(p_1')$ and,thus, $f(p')$. Hence, we can determine γ_j when j is odd.

It can be shown that when j is odd and we have determined γ_{j-1}, γ_j, then we have also found the potentials at nodes necesary to meet the assumption made in finding γ_{j+1}. Thus, the process which has been described for finding the conductivity across the resistors of figure (13c) is legitimate.

Once we have determined all of the unknown conductances our wedge, we increment i. After $i = N - h - 1$ we move to the next edge. Finally, when we've considered all edges, we increment v. The algorithm is complete after $v = N - 2$.

5 Results

Two types of tests were performed. First,we considered various conductivity distributions within the cubic lattice, then we considered the matrix Λ whose entries were computed with decreasing accuracy. Note that γ refers to the conductivity.

5.1 Conductivity Distributions

For the cubic lattice with $N = 5$ and conductivity equal to 1 throughout, the conductivity was reconstructed with 100 % accuracy across all 240 resistors.

Low conductivity was reconstructed in the center of a 5 by 5 by 5 cubic lattice. Sometimes the region of low conductivity was only partially closed. This means that the conductivity across one or more of the resistors at the center of the network was the same as the conductivity across a resistor in the background. The results are shown in Table 1.

Table1

γ values	γ values	accuracy range	comments
in the center	in the backgr.		
10^{-2}	1	99.9 % - 100 %	closed region
10^{-2}	1	97.3 % - 100 %	region open on 3 sides
10^{-3}	1	99.9 % - 100 %	closed region
10^{-3}	1	99.6 % - 100 %	region open on 1 side
10^{-3}	1	2 negative values, rest: 18.9 % - 100 %	region open on 2 sides
10^{-4}	1	88.5 % - 100 %	closed region
10^{-4}	1	56.4 % - 100 %	region open on 1 side
10^{-4}	1	58.9 % - 100 %	region open on 2 sides

High conductivity was reconstructed in the center of a 5 by 5 by 5 cubic lattice; the results are shown in Table 2.

Table2

γ values in the center	γ values in the backgr.	accuracy range	no. of neg. values	comments
9	1	99.4 % - 100 %	—	closed region
9	1	98.7 % - 100 %	—	region open on 3 sides
99	1	72.4 % - 100 %	—	closed region
99	1	93.4 % - 100 %	2	region open on 1 side
99	1	66.2 % - 100 %	2	region open on 2 sides
999	1	3.8 % - 100 %	—	closed region
999	1	3.5 % - 100 %	1	region open on 1 side

High conductivity, surrounded by a region of low conductivity, was reconstructed in the center of a 5 by 5 by 5 cubic lattice. The conductivity across resistors in the background was 1. The results are shown in Table 3.

Table3

γ values in center	γ values around center	accuracy range in center	accuracy range around center	accuracy range in background
10^1	10^{-1}	1 value negative 96.2 % - 99.2 %	1 value negative 98 % - 100 %	100 %
10^1	10^{-2}	1 value negative 0.02 % - 58.8 %	1 value negative 26 % - 100 %	100 %
10^2	10^{-1}	1 value negative 67.4 % - 81.7 %	1 value negative 72.2 % - 100 %	100 %
10^3	10^{-1}	1 value negative 34.7 % - 40.6 %	1 value negative 33.3 % - 100 %	100 %

5.2 Inaccurate Λ

In this test, the entries of Λ were rounded off to various numbers of decimal places. In each case, the conductivity across all resistors was 1. The results are shown in Table 5.

Table4

No. of Decimal Places in Λ	largest recovered grid	accuracy range
2	3 x 3 x 3	76 % - 100 %
3	3 x 3 x 3	90 % - 100 %
4	4 x 4 x 4	74.5 % - 100 %
5	4 x 4 x 4	98.4 % - 100 %
6	4 x 4 x 4	99.7 % - 100 %
7	4 x 4 x 4	99.8 % - 100 %
8	5 x 5 x 5	62.9 % - 100 %
9	5 x 5 x 5	97.2 % - 100 %
10	5 x 5 x 5	99.7 % - 100 %

6 Computer Code

Given the conductivity throughout a cubic lattice, the following program creates a matrix Λ by solving the Dirichlet for various sets of boundary data as outlined in the section "The Dirichlet to Neumann Map". Λ is the matrix which is used as intial data in the program which reconstructs the conductivity.

```
      INTEGER N,MaxN,i,j,k,p,q,r,row,col,lev,IND,ii,jj,kk,temp
      PARAMETER (MaxN = 10)
      INTEGER IWORK((MaxN-1)**3)
      DOUBLE PRECISION Bnd(6,(MaxN-1)**2),C(6,(MaxN-1)**2)
      DOUBLE PRECISION gamma(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION U((MaxN-1)**3),WORK((MaxN-1)**3),g
      DOUBLE PRECISION A((MaxN-1)**3,2*(MaxN-1)**2+1)
      DOUBLE PRECISION Lambda(6*(MaxN-1)**2,6*(MaxN-1)**2)
C
      READ *,N
C
      DO 120 k=1,2*N-1
        DO 110 i=1,2*N-1
          READ *,(gamma(i,j,k), j=1,2*N-1)
  110   CONTINUE
  120 CONTINUE
C
      DO 780 p=1,6
      n1 = n0 - (N-1)**2
      n2 = n0 + (N-1)**2
      n3 = n0 + 1
      n4 = n0 - 1
C
      RETURN
      END
```

```
        DO 770 q=0,N-2
          DO 760 r=1,N-1
C
          Bnd(p,q*(N-1)+r) = 1d0
C
C For Ax=b, Create b
C
          DO 200 i=1,(N-1)**3
            U(i) = 0d0
            IF (i.LE.(N-1)**2) THEN
              g = gamma(2*((i-1)/(N-1)+1),2*mod(i-1,N-1)+2,1)
              U(i) = U(i) + g*Bnd(1,i)
            END IF
C
            IF (i.GT.(N-2)*(N-1)**2) THEN
              g = gamma(2*(mod(i-1,(N-1)**2)/(N-1))+2,
     2                   2*mod(i-1,N-1)+2,2*N-1)
              U(i) = U(i) + g*Bnd(3,i-(N-2)*(N-1)**2)
            END IF
C
            IF ((mod(i-1,(N-1)**2)+1).LE.(N-1)) THEN
              g = gamma(1,2*mod(i-1,N-1)+2,2*((i-1)/(N-1)**2)+2)
              U(i) = U(i)+g*Bnd(4,(i-1)/(N-1)+mod(i-1,(N-1)**2)+1)
            END IF
C
            IF ((mod(i-1,(N-1)**2)+1).GT.(N-1)*(N-2)) THEN
              g = gamma(2*N-1,2*mod(i-1,N-1)+2,2*((i-1)/(N-1)**2)+2)
              U(i) = U(i) + g*Bnd(2,(i-1-(N-1)*(N-2))/(N-1)
     1                         + mod(i-1-(N-1)*(N-2),(N-1)**2) + 1)
            END IF
C
            IF (mod(i,N-1).EQ.0) THEN
              g = gamma(2*((mod(i-1,(N-1)**2)+1)/(N-1)),2*N-1,
     2                  2*((i-1)/(N-1)**2)+2)
              U(i) = U(i) + g*Bnd(5,i/(N-1))
            END IF
C
            IF (mod(i-1,N-1).EQ.0) THEN
              g = gamma(2*(mod(i,(N-1)**2)/(N-1))+2,1,2*(i/(N-1)**2)+2)
```

```
C
      CALL DQRANK(A,45,n,n,0,KR,JPVT,QRAUX,WORK)
      CALL DQRLSS(A,45,n,n,KR,b,al,RSD,JPVT,QRAUX)
C
      RETURN
      END


      DOUBLE PRECISION FUNCTION Current(x,Lambda,alpha,nodes,cnt)
      INTEGER x,MaxN,cnt,i,nodes(0:45)
      PARAMETER (MaxN = 6)
      DOUBLE PRECISION alpha(45)
      DOUBLE PRECISION Lambda(6*(MaxN-1)**2,6*(MaxN-1)**2)
C
      Current = Lambda(x,nodes(0))
      DO 100 i=1,cnt
        Current = Current + alpha(i)*Lambda(x,nodes(i))
  100 CONTINUE
      RETURN
      END


      SUBROUTINE Nbhr(N,start,n0,n1,n2,n3,n4)
      INTEGER N,start,n0,n1,n2,n3,n4
C
      n0 = start - (N-1)**2
      n1 = n0 + N - 1
      n2 = n0 - N + 1
      n3 = n0 + 1
      n4 = n0 - 1
C
      RETURN
      END


      SUBROUTINE Nbhr2(N,start,n0,n1,n2,n3,n4)
      INTEGER N,start,n0,n1,n2,n3,n4
C
      n0 = start - (N-1)
```

```
              U(i) = U(i) + g*Bnd(6,i/(N-1) +1)
            END IF
  200     CONTINUE
C
C Fill The Matrix A of Ax=b
C
          DO 350 i=1,(N-1)**3
            row = 2*(mod(i-1,(N-1)**2)/(N-1)+1)
            col = 2*(mod(i-1,N-1)+1)
            lev = 2*((i-1)/(N-1)**2) + 1
C
            A(i,(N-1)**2+1) = gamma(row,col,lev)+gamma(row,col,lev+2)
     1              + gamma(row,col-1,lev+1) + gamma(row,col+1,lev+1)
     2              + gamma(row-1,col,lev+1) + gamma(row+1,col,lev+1)
C
            IF (i.GT.(N-1)**2) THEN
              A(i,1) = -gamma(row,col,lev)
            END IF
C
            IF (i.LE.(N-2)*(N-1)**2) THEN
              A(i,2*(N-1)**2+1) = -gamma(row,col,lev+2)
            END IF
C
            IF ((i.GT.(N-1)).AND.((mod(i-1,(N-1)**2)+1).GE.N)) THEN
              A(i,N**2-3*N+3) = -gamma(row-1,col,lev+1)
            END IF
C
            IF (((mod(i-1,(N-1)**2)+1).LE.(N-1)*(N-2)).AND.
     3                                (i.LE.(N-1)*N*(N-2))) THEN
              A(i,(N-1)**2+N) = -gamma(row+1,col,lev+1)
            END IF
C
            IF ((i.NE.1).AND.(mod(i-1,N-1).NE.0)) THEN
              A(i,(N-1)**2) = -gamma(row,col-1,lev+1)
            END IF
C
            IF ((mod(i,N-1).NE.0).AND.(i.LE.(N-1)**3-1)) THEN
               A(i,(N-1)**2+2) = -gamma(row,col+1,lev+1)
            END IF
          node1(cnt) = side + temp + i
          node2(cnt) = node1(cnt) + add
          cnt = cnt + 1
          DO 100 k=1,j
            IF ((temp+i+k).LE.(temp+N-1)) THEN
              node1(cnt) = side + temp + i + k
              node2(cnt) = node1(cnt) + add
              cnt = cnt + 1
            END IF
  100     CONTINUE
          DO 150 k=1,j
            IF ((temp+i-k).GE.(temp+1)) THEN
              node1(cnt) = side + temp + i -k
              node2(cnt) = node1(cnt) + add
              cnt = cnt + 1
            END IF
  150     CONTINUE
  200   CONTINUE
        RETURN
        END


        SUBROUTINE Alpha(Lambda,P,Q,n,al)
        INTEGER MaxN,i,j,x,y,z
        PARAMETER (MaxN = 6)
        INTEGER KR,JPVT(45),P(45),Q(0:45)
        DOUBLE PRECISION Lambda(6*(MaxN-1)**2,6*(MaxN-1)**2)
        DOUBLE PRECISION al(45),A(45,45),b(45)
        DOUBLE PRECISION QRAUX(45),WORK(45),RSD(45)
C
        x = Q(0)
        DO 100 i=1,n
          y = P(i)
          b(i) = -Lambda(y,x)
          DO 100 j=1,n
            z = Q(j)
            A(i,j) = Lambda(y,z)
  100     CONTINUE
  200   CONTINUE
  350     CONTINUE
C
C          PRINT *
C          PRINT *
C          PRINT *,'Bonudary Node #',(p-1)*(N-1)**2 + q*(N-1) + r,':'
C
C Solve The Matrix Equation Ax=b For x
C
          CALL DNBFS(A,(MaxN-1)**3,(N-1)**3,(N-1)**2,
     4                          (N-1)**2,U,1,IND,WORK,IWORK)
C
C Using x and Bnd, Find C
C
          DO 410 i=1,6
            DO 400 j=1,(N-1)**2
              C(i,j) = 0d0
              row = 2*((j-1)/(N-1)+1)
              col = 2*(mod(j-1,N-1)+1)
              IF (i.EQ.1) THEN
                C(i,j) = gamma(row,col,1)*(Bnd(i,j) - U(j))
              ELSE IF (i.EQ.2) THEN
                C(i,j) = gamma(2*N-1,col,row)*
     2              (Bnd(i,j)-U(j+(N-1)*(N-2)*(1+(j-1)/(N-1))))
              ELSE IF (i.EQ.3) THEN
            C(i,j) = gamma(row,col,2*N-1)*(Bnd(i,j)-U(j+(N-2)*(N-1)**2))
              ELSE IF (i.EQ.4) THEN
                C(i,j) = gamma(1,col,row)*
     3               (Bnd(i,j)-U(j+(N-1)*(N-2)*((j-1)/(N-1))))
              ELSE IF (i.EQ.5) THEN
                C(i,j) = gamma(col,2*N-1,row)*(Bnd(i,j)-U(j*(N-1)))
              ELSE IF (i.EQ.6) THEN
                C(i,j) = gamma(col,1,row)*(Bnd(i,j)-U((j-1)*(N-1)+1))
              END IF
  400       CONTINUE
  410     CONTINUE
C
C          DO 500 i=0,N-2
C            WRITE (*,910) (C(1,(N-1)*i+j), j=1,N-1)
C  500     CONTINUE
```

```
             alIn = alIn + 2
             t3 = (alIn + 1)/2
             In = In + 1
             In1(In) = min(t1,t3)
             In2(In) = min(t2,t3)
             alR = alC
             alL = alC + In1(In) + 1
 350       CONTINUE
         alC = alC - In1(In-1) - In2(In-1) - 1
         IF (mod(v,2).EQ.1) THEN
           IF (v.EQ.1) THEN
             IF ((N.NE.3).OR.(i.NE.N-h-1)) THEN
               gma(x+1,y1+1,z) = -gma(x,y1+2,z)*alph(alC+1)
               gma(x+1,y1+1,z) = gma(x+1,y1+1,z)/vt(st1)
             END IF
             IF ((i.EQ.(h+1)).AND.(N.NE.3)) THEN
         gma(x+1,y1-1,z) = -gma(x,y1-2,z)*alph(alC+In1(In-1)+1)/vt(st1)
             END IF
           ELSE
             gma(x+1,y1+1,z) = -(gma(x+1,y1+2,z-1)*vt(st1-(N-1)**2+1)
    2                + gma(x,y1+2,z)*vt(st1-N+2))/vt(st1)
             IF (i.EQ.(h+1)) THEN
               gma(x+1,y1-1,z) = -(gma(x+1,y1-2,z-1)*vt(st1-(N-1)**2-1)
    3                + gma(x,y1-2,z)*vt(st1-N))/vt(st1)
             END IF
           END IF
         END IF
 800   CONTINUE
       RETURN
       END


       SUBROUTINE Nodes1(N,node1,node2,cnt,side,add,v,i)
       INTEGER N,side,add,temp,j,k
       INTEGER node1(0:45),node2(45),cnt,v,i
C
       cnt = 1
       DO 200 j=0,v
         temp = (v-j)*(N-1)
```

```
C          PRINT *
C          DO 510 i=0,N-2
C            WRITE (*,910) (Bnd(1,(N-1)*i+j), j=1,N-1)
C  510     CONTINUE
C
C          DO 530 k=0,N-2
C            PRINT *
C            WRITE (*,910) (C(4,(N-1)*k+j), j=1,N-1)
C            WRITE (*,910) (Bnd(4,(N-1)*k+j),j=1,N-1)
C            DO 520 j=1,N-1
C          WRITE (*,930) C(6,(N-1)*k+j),Bnd(6,(N-1)*k+j),(U((N-1)*(j-1)+
C     5    k*(N-1)**2+i), i=1,N-1),Bnd(5,(N-1)*k+j),C(5,(N-1)*k+j)
C  520       CONTINUE
C            WRITE (*,910) (Bnd(2,k*(N-1)+j), j=1,N-1)
C            WRITE (*,910) (C(2,k*(N-1)+j), j=1,N-1)
C  530     CONTINUE
C
C          PRINT *
C          DO 540 i=0,N-2
C            WRITE (*,910) (Bnd(3,i*(N-1)+j), j=1,N-1)
C  540     CONTINUE
C          PRINT *
C          DO 550 i=0,N-2
C            WRITE (*,910) (C(3,(N-1)*i+j), j=1,N-1)
C  550     CONTINUE
C
         temp = (p-1)*(N-1)**2 + q*(N-1) + r
         DO 730 ii=1,6
           DO 720 jj=0,N-2
             DO 710 kk=1,N-1
         Lambda((ii-1)*(N-1)**2+jj*(N-1)+kk,temp)=C(ii,jj*(N-1)+kk)
  710        CONTINUE
  720      CONTINUE
  730    CONTINUE
C
         Bnd(p,q*(N-1)+r) = 0d0
         DO 750 i=1,(N-1)**3
           DO 740 j=1,2*(N-1)**2+1
             A(i,j) = 0d0
```

```
          z    = 2*(v-j+k)
          y1   = 2*i
          x    = 2*k - 1
          IF (k.EQ.1) THEN
            sst1    = (k-1)*(N-1)**2 + (v-j+k-1)*(N-1) + i
            C(sst1)  = Current(sst1+sft,Lambda,alph,Node,Cnt2)
            vt(st1) = -C(sst1)/gma(x,y1,z) + alph(alC)
          ELSE
            CALL Nbhr2(N,st1,n0,n1,n2,n3,n4)
            IF (k.EQ.2) THEN
              tc = alC - In1(In-1) - In2(In-1) - 1
              vt(st1) = -(vt(n0)-alph(tc))*gma(x-2,y1,z)
            ELSE
              vt(st1) = -(vt(n0)-vt(n0-(N-1)))*gma(x-2,y1,z)
            END IF
            IF (i.EQ.N-1) THEN
              vt(st1) = vt(st1) - vt(n0)*gma(x-1,y1+1,z)
            ELSE
            vt(st1) = vt(st1) - (vt(n0)-vt(n3))*gma(x-1,y1+1,z)
            END IF
            IF (i.EQ.1) THEN
              vt(st1) = vt(st1) - vt(n0)*gma(x-1,y1-1,z)
            ELSE
              vt(st1) = vt(st1) - (vt(n0)-vt(n4))*gma(x-1,y1-1,z)
            END IF
            vt(st1) = vt(st1) - (vt(n0)-vt(n1))*gma(x-1,y1,z+1)
     3                         - (vt(n0)-vt(n2))*gma(x-1,y1,z-1)
            vt(st1) = -vt(st1)/gma(x,y1,z) + vt(n0)
          END IF
325     CONTINUE
        gma(x+1,y1,z+1) = gma(x,y1,z+2)/vt(st1)
        IF (j.NE.1) THEN
          gma(x+1,y1,z+1) = -gma(x+1,y1,z+1)*vt(st1+(N-1)**2-N+1)
        ELSE
          tc = alC - In1(In-1) - In2(In-1) - 1
          gma(x+1,y1,z+1) = -gma(x+1,y1,z+1)*alph(tc)
        END IF
      END IF
      alC = alC + In1(In) + In2(In) + 1
```

```
  740        CONTINUE
  750      CONTINUE
C
  760      CONTINUE
  770    CONTINUE
  780 CONTINUE
C
      PRINT *,N
      PRINT *
      DO 893 i=1,6*(N-1)**2
            WRITE (*,940) (Lambda(i,k), k=1,6*(N-1)**2)
  893 CONTINUE
C
      STOP
C  910 FORMAT(' ',30X,10F15.6)
C  930 FORMAT(' ',14F15.6)
  940 FORMAT(' ',490F22.16)
      END
```

The following is the main program for reconstructing the conductivity within a cubic lattice. It calls on four primary subroutines: NewL1,2,3; cube, move, PGamma. Cube is the subroutine which actually reconstructs the conductances. However, it only does so for the edge containing nodes 1 through $N-1$. Thus, the subroutines, NewL1,2,3 rearrange the matrix Λ appropriately so that subroutine cube can be applied to the other edges as well. The subroutine "move" moves information about the conductivity found on one egde to a matrix containing information about the conductivity found on another edge. Finally, the subroutine PGamma prints the known information about the conductivity.
The subroutine cube and its subroutines are also printed. A description of how they work can be found in the setion titled "Algorithm".

```
      INTEGER N,MaxN,i,j,temp,h,v,sft
      PARAMETER (MaxN = 6)
      DOUBLE PRECISION L1(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L2(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L3(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L4(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L5(6*(MaxN-1)**2,6*(MaxN-1)**2)
```

```
                    IF (k.LE.2) THEN
                      alL = alL + 1
                    END IF
                  END IF
 310            CONTINUE
              IF (k.LE.2) THEN
                alL = alL - In1(sub) - In2(sub-1) - 2
                alR = alR - In1(sub-1) - In2(sub-1) - 2
                sub = sub - 1
              END IF
 315          CONTINUE
 320        CONTINUE
C
            tempc1 = vt(st1)*gma(x+1,y1,z+1)
            tempc2 = vt(st2)*gma(x+1,y2,z+1)
            IF (j.EQ.2) THEN
              tempc1 = tempc1 + alph(3)*gma(x,y1,z+2)
              tempc2 = tempc2 + alph(4)*gma(x,y2,z+2)
            ELSE
              tempc1 = tempc1 + gma(x,y1,z+2)*vt(st1-N+1+(N-1)**2)
              tempc2 = tempc2 + gma(x,y2,z+2)*vt(st2-N+1+(N-1)**2)
            END IF
            gma(x+1,y1-1,z+2) = -tempc1/vt(st1+(N-1)**2-1)
            gma(x+1,y2+1,z+2) = -tempc2/vt(st2+(N-1)**2+1)
C
          t = (N-1)**2 - 1
          IF (j.EQ.2) THEN
            tempc1 = -(vt(st1+t)-alph(2))*gma(x,y1-2,z+2)
          ELSE
            tempc1 = -(vt(st1+t)-vt(st1+t-N+1))*gma(x,y1-2,z+2)
          END IF
      tempc1 = tempc1 - (vt(st1+t)-vt(st1-1))*gma(x+1,y1-2,z+1)
     3 - vt(st1+t)*(gma(x+1,y1-1,z+2) + gma(x+1,y1-3,z+2)
     4 + gma(x+1,y1-2,z+3))
          gma(x+2,y1-2,z+2) = tempc1/vt(st1+t)
C
          ELSE
            DO 325 k=1,1+(j-1)/2
              st1 = (k-1)*(N-1) + (v-j+k-1)*(N-1)**2 + i
      DOUBLE PRECISION L6(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L7(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L8(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L9(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L10(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L11(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION L12(6*(MaxN-1)**2,6*(MaxN-1)**2)
      DOUBLE PRECISION G1(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G2(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G3(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G4(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G5(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G6(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G7(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G8(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G9(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G10(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G11(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION G12(2*MaxN-1,2*MaxN-1,2*MaxN-1)
C
      READ *,N
C
      DO 100 i=1,6*(N-1)**2
        READ *,(L1(i,j), j=1,6*(N-1)**2)
  100 CONTINUE
C
      CALL NewL3(L2,L1,N)
      CALL NewL3(L3,L2,N)
      CALL NewL3(L4,L3,N)
      CALL NewL2(L9,L1,N)
      CALL NewL1(L10,L9,N)
      CALL NewL2(L5,L9,N)
      CALL NewL3(L8,L5,N)
      CALL NewL3(L7,L8,N)
      CALL NewL3(L6,L7,N)
      CALL NewL2(L12,L5,N)
      CALL NewL1(L11,L12,N)
C
      h = 0
```

```
                  END IF
                  vt(st1) = -vt(st1)/gma(x,y1,z) + vt(n0)
                END IF
              END IF
              IF (k.LE.2) THEN
                alR = alR + 1
              END IF
 305        CONTINUE
C
              check = check - N + 2
              DO 310 m=0,1
                st2 = check - l + i - m
                y2  = 2*(i-l-m+1)
                IF ((st2.GE.check).AND.((l.NE.1).OR.(m.EQ.1))) THEN
                  IF (k.EQ.1) THEN
                    sst2 = (v-j+k+l-1)*(N-1) + (k-1)*(N-1)**2 -N + 2
                    sst2 = sst2 + l + i - 2 - m
                    vt(st2) = alph(alL)-C(sst2)/gma(x,y2,z)
                  ELSE
                    CALL Nbhr2(N,st2,n0,n1,n2,n3,n4)
                    IF (k.EQ.2) THEN
                      vt(st2) = -(vt(n0)-alph(alL))*gma(x-2,y2,z)
                    ELSE
                 vt(st2) = gma(x-2,y2,z)*(vt(n0)-vt(st2-2*(N-1)))
                    END IF
              vt(st2) = vt(st2) - gma(x-1,y2,z+1)*(vt(n0)-vt(n1))
     2                          - gma(x-1,y2,z-1)*(vt(n0)-vt(n2))
                    IF (n3.LE.check+N-2) THEN
              vt(st2) = vt(st2) - gma(x-1,y2+1,z)*(vt(n0)-vt(n3))
                    ELSE
                      vt(st2) = vt(st2) - gma(x-1,y2+1,z)*vt(n0)
                    END IF
                    IF (n4.GE.check) THEN
             vt(st2) = vt(st2) - gma(x-1,y2-1,z)*(vt(n0)-vt(n4))
                    ELSE
                      vt(st2) = vt(st2) - gma(x-1,y2-1,z)*vt(st2)
                    END IF
                    vt(st2) = -vt(st2)/gma(x,y2,z) + vt(n0)
                  END IF
```

```
      sft = 3*(N-1)**2
      DO 850 v=0,N-2
        temp = mod(v,2)
        IF ((v.NE.(N-2)).AND.(temp.EQ.1)) THEN
          h = h + 1
        END IF
C
        CALL cube(L1,G1,N,sft,h,v)
        CALL move(4,G2,G1,N,v,h)
        CALL move(3,G3,G1,N,v,h)
        CALL move(2,G4,G1,N,v,h)
        CALL move(9,G12,G1,N,v,h)
        CALL move(12,G9,G1,N,v,h)
        CALL move(5,G5,G1,N,v,h)
C
        CALL cube(L2,G2,N,sft,h,v)
        CALL move(2,G1,G2,N,v,h)
        CALL move(3,G4,G2,N,v,h)
        CALL move(4,G3,G2,N,v,h)
        CALL move(5,G6,G2,N,v,h)
        CALL move(15,G10,G2,N,v,h)
        CALL move(16,G9,G2,N,v,h)
C
        CALL cube(L3,G3,N,sft,h,v)
        CALL move(3,G1,G3,N,v,h)
        CALL move(2,G2,G3,N,v,h)
        CALL move(4,G4,G3,N,v,h)
        CALL move(9,G10,G3,N,v,h)
        CALL move(5,G7,G3,N,v,h)
        CALL move(12,G11,G3,N,v,h)
C
        CALL cube(L4,G4,N,sft,h,v)
        CALL move(4,G1,G4,N,v,h)
        CALL move(2,G3,G4,N,v,h)
        CALL move(3,G2,G4,N,v,h)
        CALL move(5,G8,G4,N,v,h)
        CALL move(15,G12,G4,N,v,h)
        CALL move(16,G11,G4,N,v,h)
C
```

```
C
          DO 320 k=1,j/2
            IF (k.EQ.1) THEN
              sub = In
            ELSE IF (k.EQ.2) THEN
              sub = In - 1
              alR = alC - In1(In-1) - In2(In-1) - 1
              alL = alC - In2(In-1)
            END IF
            DO 315 l=1,j/2-k+1
              check = (v-j+k+l-2)*(N-1)**2 + k*(N-1)
              z     = 2*(v-j+k+l-1)
              x     = 2*k - 1
              DO 305 m=0,1
                st1 = check + l + i - N + m
                y1  = 2*(i+l+m-1)
                IF (st1.LE.check) THEN
                  IF (k.EQ.1) THEN
                    sst1 = (v-j+k+l-1)*(N-1) + (k-1)*(N-1)**2
                    sst1 = sst1 + l+ i -N +m
                    vt(st1) = alph(alR) - C(sst1)/gma(x,y1,z)
                  ELSE
                    CALL Nbhr2(N,st1,n0,n1,n2,n3,n4)
                    IF (k.EQ.2) THEN
                      vt(st1) = -(vt(n0)-alph(alR))*gma(x-2,y1,z)
                    ELSE
                  vt(st1) = -gma(x-2,y1,z)*(vt(n0)-vt(st1-2*(N-1)))
                    END IF
               vt(st1) =  vt(st1) - gma(x-1,y1,z+1)*(vt(n0)-vt(n1))
     1                            - gma(x-1,y1,z-1)*(vt(n0)-vt(n2))
                    IF (n3.LE.check) THEN
               vt(st1) = vt(st1) - gma(x-1,y1+1,z)*(vt(n0)-vt(n3))
                    ELSE
                      vt(st1) = vt(st1) - gma(x-1,y1+1,z)*vt(n0)
                    END IF
                    IF (n4.GE.check-N+2) THEN
               vt(st1) = vt(st1) - gma(x-1,y1-1,z)*(vt(n0)-vt(n4))
                    ELSE
                      vt(st1) = vt(st1) - gma(x-1,y1-1,z)*vt(n0)
```

```
      CALL cube(L5,G5,N,sft,h,v)
      CALL move(5,G1,G5,N,v,h)
      CALL move(9,G9,G5,N,v,h)
      CALL move(3,G7,G5,N,v,h)
      CALL move(2,G6,G5,N,v,h)
      CALL move(4,G8,G5,N,v,h)
      CALL move(12,G12,G5,N,v,h)
C
      CALL cube(L6,G6,N,sft,h,v)
      CALL move(6,G1,G6,N,v,h)
      CALL move(5,G2,G6,N,v,h)
      CALL move(4,G5,G6,N,v,h)
      CALL move(2,G7,G6,N,v,h)
      CALL move(3,G8,G6,N,v,h)
      CALL move(15,G9,G6,N,v,h)
      CALL move(16,G10,G6,N,v,h)
C
      CALL cube(L7,G7,N,sft,h,v)
      CALL move(7,G1,G7,N,v,h)
      CALL move(5,G3,G7,N,v,h)
      CALL move(12,G10,G7,N,v,h)
      CALL move(9,G11,G7,N,v,h)
      CALL move(3,G5,G7,N,v,h)
      CALL move(4,G6,G7,N,v,h)
      CALL move(2,G8,G7,N,v,h)
C
      CALL cube(L8,G8,N,sft,h,v)
      CALL move(8,G1,G8,N,v,h)
      CALL move(5,G4,G8,N,v,h)
      CALL move(2,G5,G8,N,v,h)
      CALL move(4,G7,G8,N,v,h)
      CALL move(3,G6,G8,N,v,h)
      CALL move(15,G11,G8,N,v,h)
      CALL move(16,G12,G8,N,v,h)
C
      CALL cube(L9,G9,N,sft,h,v)
      CALL move(9,G1,G9,N,v,h)
      CALL move(3,G10,G9,N,v,h)
      CALL move(12,G5,G9,N,v,h)
```

```
  225         CONTINUE
              gma(x+1,y1,z+1) = -gma(x+2,y1,z)/vt(st1)
              IF (j.NE.1) THEN
                gma(x+1,y1,z+1) = gma(x+1,y1,z+1)*vt(st1-(N-1)**2+N-1)
              END IF
            END IF
  250     CONTINUE
C
          gma(1,2*i,2*(v+1))=Current(sft+v*(N-1)+i,
     1                                 Lambda,alph,Node,Cnt2)
          gma(1,2*i,2*(v+1)) = gma(1,2*i,2*(v+1))/alph(1)
C
          In  = 1
          In1(1) = 1
          In2(1) = 1
          In1(0) = 0
          In2(0) = 0
          t1 = N - i - 1
          t2 = i - 1
          alC = 2
          alIn = 1
          DO 350 j=1,v
            IF (mod(j,2).EQ.0) THEN
              DO 304 k=1,j/2
                check = (v-j+k)*(N-1)
                DO 301 m=0,1
                  st1 = check - N + k + i + m
                  IF (st1.LE.check) THEN
                    C(st1) = Current(st1+sft,Lambda,alph,Node,Cnt2)
                  END IF
  301           CONTINUE
                check = check - N + 2
                DO 302 m=0,1
                  st2 = check + k + i - 2 - m
                  IF ((st2.GE.check).AND.((m.EQ.1).OR.(k.NE.1))) THEN
                    C(st2) = Current(st2+sft,Lambda,alph,Node,Cnt2)
                  END IF
  302           CONTINUE
  304         CONTINUE
```

```
         CALL move(5,G12,G9,N,v,h)
         CALL move(13,G6,G9,N,v,h)
         CALL move(14,G2,G9,N,v,h)
C
         CALL cube(L10,G10,N,sft,h,v)
         CALL move(10,G1,G10,N,v,h)
         CALL move(12,G3,G10,N,v,h)
         CALL move(3,G9,G10,N,v,h)
         CALL move(5,G11,G10,N,v,h)
         CALL move(9,G7,G10,N,v,h)
         CALL move(13,G2,G10,N,v,h)
         CALL move(14,G6,G10,N,v,h)
C
         CALL cube(L11,G11,N,sft,h,v)
         CALL move(11,G1,G11,N,v,h)
         CALL move(9,G3,G11,N,v,h)
         CALL move(5,G10,G11,N,v,h)
         CALL move(12,G7,G11,N,v,h)
         CALL move(3,G12,G11,N,v,h)
         CALL move(13,G8,G11,N,v,h)
         CALL move(14,G4,G11,N,v,h)
C
         CALL cube(L12,G12,N,sft,h,v)
         CALL move(12,G1,G12,N,v,h)
         CALL move(5,G9,G12,N,v,h)
         CALL move(3,G11,G12,N,v,h)
         CALL move(9,G5,G12,N,v,h)
         CALL move(13,G4,G12,N,v,h)
         CALL move(14,G8,G12,N,v,h)
c        CALL PGamma(G1,N)
  850 CONTINUE
      CALL PGamma(G1,N)
C
      STOP
      END
```

```
          tempc1 = -vt(st1+N-2)*gma(x+2,y1-2,z)
        ELSE
    tempc1 = -(vt(st1+N-2)-vt(st1+N-2-(N-1)**2))*gma(x+2,y1-2,z)
        END IF
    tempc1 = tempc1 - (vt(st1+N-2)-vt(st1-1))*gma(x+1,y1-2,z+1)
   3 - vt(st1+N-2)*(gma(x+2,y1-1,z+1) + gma(x+2,y1-3,z+1)
   4 + gma(x+3,y1-2,z+1))
        gma(x+2,y1-2,z+2) = tempc1/vt(st1+N-2)
C
      ELSE
        DO 225 k=1,1+(j-1)/2
          st1 = (k-1)*(N-1)**2 + (v-j+k-1)*(N-1) + i
          x   = 2*(v-j+k)
          y1  = 2*i
          z   = 2*k - 1
          IF (k.EQ.1) THEN
            C(st1)  = Current(st1,Lambda,alph,Node,Cnt1)
            vt(st1) = -C(st1)/gma(x,y1,z)
          ELSE
            CALL Nbhr(N,st1,n0,n1,n2,n3,n4)
            IF (k.EQ.2) THEN
              vt(st1) = -vt(n0)*gma(x,y1,z-2)
            ELSE
              vt(st1) = -(vt(n0)-vt(n0-(N-1)**2))*gma(x,y1,z-2)
            END IF
            IF (i.EQ.N-1) THEN
              vt(st1) = vt(st1) - vt(n0)*gma(x,y1+1,z-1)
            ELSE
              vt(st1) = vt(st1) - (vt(n0)-vt(n3))*gma(x,y1+1,z-1)
            END IF
            IF (i.EQ.1) THEN
              vt(st1) = vt(st1) - vt(n0)*gma(x,y1-1,z-1)
            ELSE
              vt(st1) = vt(st1) - (vt(n0)-vt(n4))*gma(x,y1-1,z-1)
            END IF
            vt(st1) = vt(st1) - (vt(n0)-vt(n1))*gma(x+1,y1,z-1)
   3                          - (vt(n0)-vt(n2))*gma(x-1,y1,z-1)
            vt(st1) = -vt(st1)/gma(x,y1,z) + vt(n0)
          END IF
      SUBROUTINE cube(Lambda,gma,N,sft,h,v)
      INTEGER N,MaxN,h,v,i,k,l,m,check,sft,Cnt1,Cnt2,t
      INTEGER Node(0:45),beta(45),Cnt,alIn,alC,alR,alL,tc
      INTEGER x,y1,y2,z,st1,st2,n0,n1,n2,n3,n4,sst1,sst2
      PARAMETER (MaxN = 6)
      INTEGER In,In1(0:MaxN-2),In2(0:MaxN-2),t1,t2,t3,sub
      DOUBLE PRECISION Current,tempc1,tempc2,C((MaxN-1)**2)
      DOUBLE PRECISION gma(2*MaxN-1,2*MaxN-1,2*MaxN-1)
      DOUBLE PRECISION vt((MaxN-1)**3),alph(45)
      DOUBLE PRECISION Lambda(6*(MaxN-1)**2,6*(MaxN-1)**2)
C
      DO 800 i=h+1,N-h-1
        Node(0) = v*(N-1) + i
        CALL Nodes1(N,Node,beta,Cnt,3*(N-1)**2,-2*(N-1)**2,v,i)
        Cnt1 = Cnt - 1
        Cnt2 = Cnt - 1
C
        CALL Alpha(Lambda,beta,Node,Cnt1,alph)
C
        gma(2*(v+1),2*i,1)=Current(v*(N-1)+i,Lambda,alph,Node,Cnt1)
C
        DO 250 j=1,v
          IF (mod(j,2).EQ.0) THEN
            DO 204 k=1,j/2
              check = (v-j+k)*(N-1)
              DO 201 m=0,1
                st1 = check - N + k + i + m
                IF (st1.LE.check) THEN
                  C(st1) = Current(st1,Lambda,alph,Node,Cnt1)
                END IF
 201          CONTINUE
              check = check - N + 2
              DO 202 m=0,1
                st2 = check + k + i - 2 - m
                IF ((st2.GE.check).AND.((m.EQ.1).OR.(k.NE.1))) THEN
                  C(st2) = Current(st2,Lambda,alph,Node,Cnt1)
                END IF
 202          CONTINUE
 204        CONTINUE
```

```
                  y2  = 2*(i-1-m+1)
                  IF ((st2.GE.check).AND.((1.NE.1).OR.(m.EQ.1))) THEN
                    IF (k.EQ.1) THEN
                      vt(st2) = -C(st2)/gma(x,y2,z)
                    ELSE
                      CALL Nbhr(N,st2,n0,n1,n2,n3,n4)
                      IF (k.EQ.2) THEN
                        vt(st2) = -vt(n0)*gma(x,y2,z-2)
                      ELSE
                 vt(st2) = -gma(x,y2,z-2)*(vt(n0)-vt(st2-2*(N-1)**2))
                      END IF
                  vt(st2) = vt(st2) - gma(x+1,y2,z-1)*(vt(n0)-vt(n1))
     2                              - gma(x-1,y2,z-1)*(vt(n0)-vt(n2))
                      IF (n3.LE.check+N-2) THEN
                 vt(st2) = vt(st2) - gma(x,y2+1,z-1)*(vt(n0)-vt(n3))
                      ELSE
                        vt(st2) = vt(st2) - gma(x,y2+1,z-1)*vt(n0)
                      END IF
                      IF (n4.GE.check) THEN
                 vt(st2) = vt(st2) - gma(x,y2-1,z-1)*(vt(n0)-vt(n4))
                      ELSE
                        vt(st2) = vt(st2) - gma(x,y2-1,z-1)*vt(st2)
                      END IF
                      vt(st2) = -vt(st2)/gma(x,y2,z) + vt(n0)
                    END IF
                  END IF
 210            CONTINUE
 215          CONTINUE
 220        CONTINUE
            tempc1 = vt(st1)*gma(x+1,y1,z+1)
            tempc2 = vt(st2)*gma(x+1,y2,z+1)
            IF (j.NE.2) THEN
              tempc1 = tempc1 + gma(x+2,y1,z)*vt(st1+N-1-(N-1)**2)
              tempc2 = tempc2 + gma(x+2,y2,z)*vt(st2+N-1-(N-1)**2)
            END IF
            gma(x+2,y1-1,z+1) = -tempc1/vt(st1+N-2)
            gma(x+2,y2+1,z+1) = -tempc2/vt(st2+N)
C
            IF (j.EQ.2) THEN
```

```
C
          DO 220 k=1,j/2
            DO 215 l=1,j/2-k+1
              check = (v-j+k+l-1)*(N-1) + (k-1)*(N-1)**2
              x     = 2*(v-j+k+l-1)
              z     = 2*k - 1
              DO 205 m=0,1
                st1 = check + l + i - N + m
                y1  = 2*(i+l+m-1)
                IF (st1.LE.check) THEN
                  IF (k.EQ.1) THEN
                    vt(st1) = -C(st1)/gma(x,y1,z)
                  ELSE
                    CALL Nbhr(N,st1,n0,n1,n2,n3,n4)
                    IF (k.EQ.2) THEN
                      vt(st1) = -vt(n0)*gma(x,y1,z-2)
                    ELSE
               vt(st1) = -gma(x,y1,z-2)*(vt(n0)-vt(st1-2*(N-1)**2))
                    END IF
               vt(st1) =  vt(st1) - gma(x+1,y1,z-1)*(vt(n0)-vt(n1))
     1                            - gma(x-1,y1,z-1)*(vt(n0)-vt(n2))
                    IF (n3.LE.check) THEN
                vt(st1) = vt(st1) - gma(x,y1+1,z-1)*(vt(n0)-vt(n3))
                    ELSE
                      vt(st1) = vt(st1) - gma(x,y1+1,z-1)*vt(n0)
                    END IF
                    IF (n4.GE.check-N+2) THEN
                vt(st1) = vt(st1) - gma(x,y1-1,z-1)*(vt(n0)-vt(n4))
                    ELSE
                      vt(st1) = vt(st1) - gma(x,y1-1,z-1)*vt(n0)
                    END IF
                    vt(st1) = -vt(st1)/gma(x,y1,z) + vt(n0)
                  END IF
                END IF
  205         CONTINUE
C
              check = check - N + 2
              DO 210 m=0,1
                st2 = check + l + i - 2 - m
```