DETERMINING THE SHAPE OF CRITICAL CIRCULAR PLANAR RESISTOR NETWORKS FROM BOUNDARY MEASUREMENTS

DEREK A. JERINA

ABSTRACT. The general topic of this paper is the recovery of the shape of a resistor network from boundary measurements. The particular focus is determining shape in the case where the graph is critical circular planar. An algorithm is presented which returns both the shape and the conductivities for a member of the graph's $Y - \Delta$ equivalence class. Additionally, applying the shape recovery algorithm to a non-critical graph will return the shape and conductivities for an electrically equivalent critical version of the graph.

1. INTRODUCTION

This article makes use of many concepts developed in [1]. Let a graph with boundary be a triple $\Gamma = (V, \partial V, E)$, where (V, E) is a finite graph with the set of nodes V and the set of edges E. Let ∂V be a nonempty subset of V called the set of boundary nodes. Let $I = V - \partial V$, representing the interior nodes of a graph with boundary Γ.

A resistor network (Γ, γ) is a graph with boundary analogous to an electric circuit. Similarly, each edge has a number $\gamma(pq)$ associated with it, representing the conductivity of the edge(resistor) connecting nodes p and q. For a resistor network (Γ, γ), the Kirchhoff matrix $K = K(\Gamma, \gamma)$ will be defined as in [1].

The forward problem for a resistor network: Given a graph Γ and a potential function for the voltage on ∂V, determine the current flow on ∂V. Given an ordering of ∂V and a potential function ϕ on ∂V, then there is a matrix Λ which satisfies the equation $\Lambda\phi = \psi$ where ψ is the current on ∂V induced on the network by ϕ. In [1] it was shown that if (Γ, γ) is a connected resistor network, then the network response matrix Λ is the Schur complement

$$\Lambda = K/K(I; I) \tag{1.1}$$

The network response matrix Λ can be obtained from boundary measurements on a resistor network. The network can be enclosed in a "black box" with access limited to the boundary nodes. To determine the entries in the jth column of Λ, set the voltage at boundary node j to one and the voltage at all other boundary nodes to zero. The resulting current vector corresponds to the jth column of Λ.

The standard inverse problem is: Given a graph with boundary Γ and a response matrix Λ for a resistor network, determine $\gamma(pq)$ for each edge in the network. The inverse problem considered here is more difficult: Given the response matrix Λ determine both Γ and $\gamma(pq)$. This involves finding V, E, and $\gamma(pq)$ from Λ which contains solely information from ∂V. We will begin by attempting to recover the "shape" (Γ) and conductivities ($\gamma(pq)$) for critical circular planar resistor networks.

2. Circular Planar and Critical Graphs

A graph Γ is defined as circular planar if and only if ∂V can be placed on a circle C in the plane so that the rest of Γ is in the interior of C. The ∂V of a circular planar graph will be labeled $v_1, \ldots, v_n$ in the (clockwise) circular order around C. A pair of sequences of boundary nodes $(P; Q) = (p_1, \ldots, p_k; q_1, \ldots, q_k)$ such that the entire sequence $(p_1, \ldots, p_k; q_k, \ldots, q_1)$ is in circular order will be called a circular pair. A circular pair $(P; Q) = (p_1, \ldots, p_k; q_1, \ldots, q_k)$ of ∂V is said to be connected through Γ if $\exists\ k$ disjoint paths $\alpha_1, \ldots, \alpha_k$ in Γ, so that α_i starts at p_i, ends at q_i, and passes through no other boundary nodes. We will call α a connection from P to Q. Let $\Lambda(A; B)$ correspond to the sub-matrix of Λ composed of the entries in Λ where rows A and columns B intersect. From [1], we know that $\det\Lambda(P; Q) = 0$, where $(P; Q)$ is a circular pair of ∂V, if and only if $P; Q)$ are not connected through Γ. For each circular planar graph Γ, let $\pi(\Gamma)$ be the set of all circular pairs $(P; Q)$ of ∂V which are connected through Γ. An edge can be removed from a graph by either deleting an edge or by contracting an edge to a single node. Removing an edge is said to break the connection from P to Q if $(P; Q)$ is in $\pi(\Gamma)$ before the edge is removed, but $(P; Q)$ is not in $\pi(\Gamma)$ after the edge is removed. A graph Γ is called critical if the removal of any edge breaks some connection in $\pi(\Gamma)$. From [1] we know that γ for a circular planar graph is recoverable from Λ and Γ if and only if Γ is critical. For this reason we begin by limiting our attempts to recover both Γ and γ from Λ for critical circular planar resistor networks.

3. $Y - \Delta$ Transformations

Suppose Γ is a circular planar graph and s is a trivalent interior node in Γ with incident edges sp, sq and sr, as in Figure 3A.

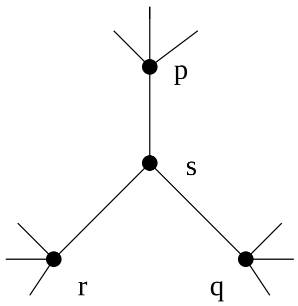

Figure 3A

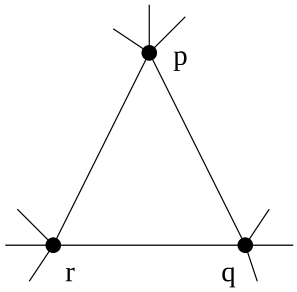

Figure 3B

A $Y-\Delta$ transformation removes the vertex s, the edges sp, sq and sr, and replaces them with three new edges pq, qr and rp as in Figure 3B. Similarly, suppose pqr is a triangle in Γ as in Figure 3B, then a $\Delta - Y$ transformation removes edges pq, qr and rp, replacing them with a new vertex s and three new edges sp, sq and sr, as in Figure 3A. Suppose Γ_1 and Γ_2 are two circular planar graphs. We say that Γ_1 and Γ_2 are $Y-\Delta$ equivalent if Γ_1 can be transformed to Γ_2 by a sequence of $Y-\Delta$ or $\Delta-Y$ transformations. We define the $Y-\Delta$ equivalence class of a graph Γ to be the set of all graphs $Y-\Delta$ equivalent to Γ. Suppose Γ_1 and Γ_2 are two circular planar graphs which are $Y-\Delta$ equivalent. From [1] we know the following:

(1) $\pi(\Gamma_1) = \pi(\Gamma_2)$

(2) Γ_1 is critical if and only if Γ_2 is critical.

(3) If γ_1 is a conductivity on Γ_1 then there is a γ_2 on Γ_2 such that $\Lambda(\Gamma_1, \gamma_1) = \Lambda(\Gamma_2, \gamma_2)$.

Hence, the response matrix can not differentiate between various members of a $Y-\Delta$ equivalence class. From Λ one can only hope to recover the $Y-\Delta$ equivalence class of Γ. For this reason, our inverse problem must be limited to: Given the response matrix Λ for a critical circular planar resistor network (Γ, γ) find a graph Γ' which is $Y-\Delta$ equivalent to Γ and a γ' on Γ' such that $\Lambda(\Gamma', \gamma') = \Lambda(\Gamma, \gamma)$.

4. Medial Graphs

Suppose Γ is a circular planar graph. We define the medial graph $M(\Gamma)$ and a geodesic as in [1]. Since Γ is circular planar, the boundary nodes $v_1, v_2, \ldots, v_n$ occur in clockwise circular order around a circle C and the rest of Γ is in the interior of C. For each edge e of Γ, let m_e be its midpoint. Now, place $2n$ points $t_1, t_2, \ldots, t_{2n}$ on C so that

$$t_1 < v_1 < t_2 < t_3 < v_2 < \ldots < t_{2n-1} < v_n < t_{2n} < t_1$$

in clockwise circular order around C. The vertices of $M(\Gamma)$ consist of the points m_e for $e \in E$ and the points t_i for $i = 1, 2, \ldots, 2n$.

The edges in $M(\Gamma)$ are defined as follows. Two vertices m_e and m_f are joined by an edge in $M(\Gamma)$ whenever e and f have a common vertex and are incident to the same face in Γ. In addition, there is one edge for each point t_j as follows. The point t_{2i} is joined by an edge to m_e where e is the edge in Γ of the form $e = v_i r$ which comes first after arc $v_i t_{2i}$ in clockwise order around v_i. The point t_{2i-1} is joined by an edge to m_f where f is the edge in Γ of the form $f = v_i s$ which comes first after arc $v_i t_{2i-1}$ in counter-clockwise order around v_i.

The vertices m_e of $M(\Gamma)$ are 4-valent, and the vertices of the form t_i are 1-valent. An edge uv of $M(\Gamma)$ has a direct extension vw if the edges uv and vw separate the two other edges incident to the vertex v in $M(\Gamma)$. A path $u_0 u_1 \ldots u_k$ in $M(\Gamma)$ is called a geodesic arc if each edge $u_{i-1}u_i$ has edge $u_i u_{i+1}$ as a direct extension. A geodesic arc $u_0 u_1 \ldots u_k$ is called a geodesic if either u_0 and u_k are points on the circle C or $u_k = u_0$ and $u_{k-1}u_k$ has $u_0 u_1$ as a direct extension. $M(\Gamma)$ is said to have a lens if two geodesics in $M(\Gamma)$ intersect each other more than once. $M(\Gamma)$ is said to be lensless if each geodesic in $M(\Gamma)$ begins and ends on C, has no self-intersection, and $M(\Gamma)$ has no lenses. From [1] we know that a circular planar graph is critical if and only if its medial graph is lensless.

A triangle in $M(\Gamma)$ is a triple $\{f, g, h\}$ of geodesics which intersect to form a triangle with no other intersections within the configuration, as in Figure 4A.

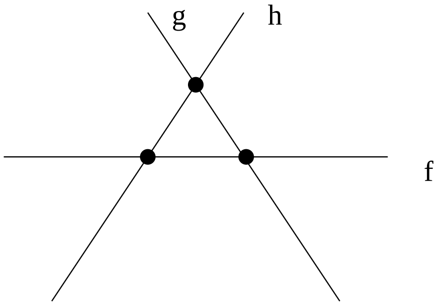

Figure 4A

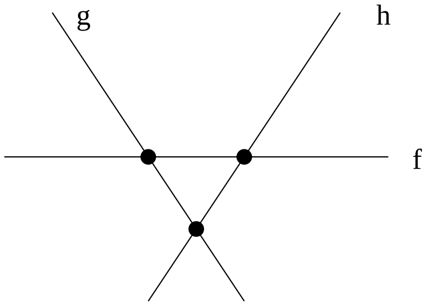

Figure 4B

Suppose $\{f, g, h\}$ form a triangle in $M(\Gamma)$ as in Figure 4A. A motion of $\{f, g, h\}$ consists of replacing this configuration with that of Figure 4B. Suppose Γ_1 and Γ_2 are two circular planr graphs. From [1] we know that Γ_1 and Γ_2 are $Y - \Delta$ equivalent if and only if their medial graphs are equivalent under motions.

5. Z-Sequence

Suppose Γ is a critical circular planar graph embeded in a circle C. Then $M(\Gamma)$ is lensless. In addition, $M(\Gamma)$ will have n geodesics each of which intersects C twice. The n geodesics intersect C in $2n$ distinct points. These

$2n$ points are labeled $t_1, \ldots, t_{2n}$, so that

$$t_1 < v_1 < t_2 < t_3 < v_2 < \ldots < t_{2n-1} < v_n < t_{2n} < t_1$$

in clockwise circular order around C. The geodesics are labelled as follows. Let g_1 be the geodesic which begins at t_1. The remaining geodesics are labelled $g_2, g_3, \ldots, g_n$ so that if $i < j$, then the first point of intersection of g_i with C occurs before the first point of intersection of g_j with C in clockwise circular order starting from t_1. For each $i = 1, 2, \ldots, 2n$, let z_i be the number associated with the geodesic which intersects C at t_i. In this way we obtain a sequence $z = z_1, z_2, \ldots, z_{2n}$, which we define as the z-sequence for $M(\Gamma)$. From [1] we know that two critical circular planar graphs are $Y - \Delta$ equivalent if and only if their z-sequences are the same. If $i < j$, and if the occurrences of i and j appear in z in the order

$$\ldots i \ldots j \ldots i \ldots j \ldots$$

we say that i and j interlace in z. Otherwise, we say that i and j do not interlace in z.

6. Finding B-B Edges and B-Spikes

Suppose we have a graph with boundary $\Gamma = (V, \partial V, E)$. We call an edge connecting two nodes $i, j \in \partial V$ a boundary to boundary edge between nodes i and j. Suppose there is some node $i \in \partial V$ with only one edge connected to it. We then say that there is a boundary spike at node i. Suppose there is a boundary spike at node i with the edge $e \in E$. If e connects node i to another node $j \in \partial V$ we say that there is a boundary to boundary spike at node i.

Lemma 6.1. *Suppose (Γ, γ) is a critical circular planar resistor network with a response matrix Λ. There is a boundary to boundary spike at node $i \in \partial V$ if and only if column i in Λ has only one strictly negative entry. In addition, if we let j equal the row in which column i in Λ has it's only strictly negative entry, then node i is a boundary to boundary spike connected to boundary node j.*

Proof. The proof is left to the reader. □

Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. We make use of recent results proven by David Ingerman to find boundary to boundary edges and boundary spikes by examining the ranks of sub-matrices of Λ. First, we examine the sub-matrix $M_{1,1}$ consisting of column one in Λ after dropping row one. Next, we drop row two and examine this sub-matrix $M_{1,2}$. Then we add column two except for rows one and two and examine this sub-matrix $M_{2,2}$. We continue in this manner so that each sub-matrix $M_{j,k}$ consists of either

(1) Columns $1 \ldots i$ in Λ except for rows $1 \ldots i$ labelled $M_{i,i}$

(2) Columns $1 \ldots i$ in Λ except for rows $1 \ldots i+1$ labelled $M_{i,i+1}$ as shown in Figure 6A.

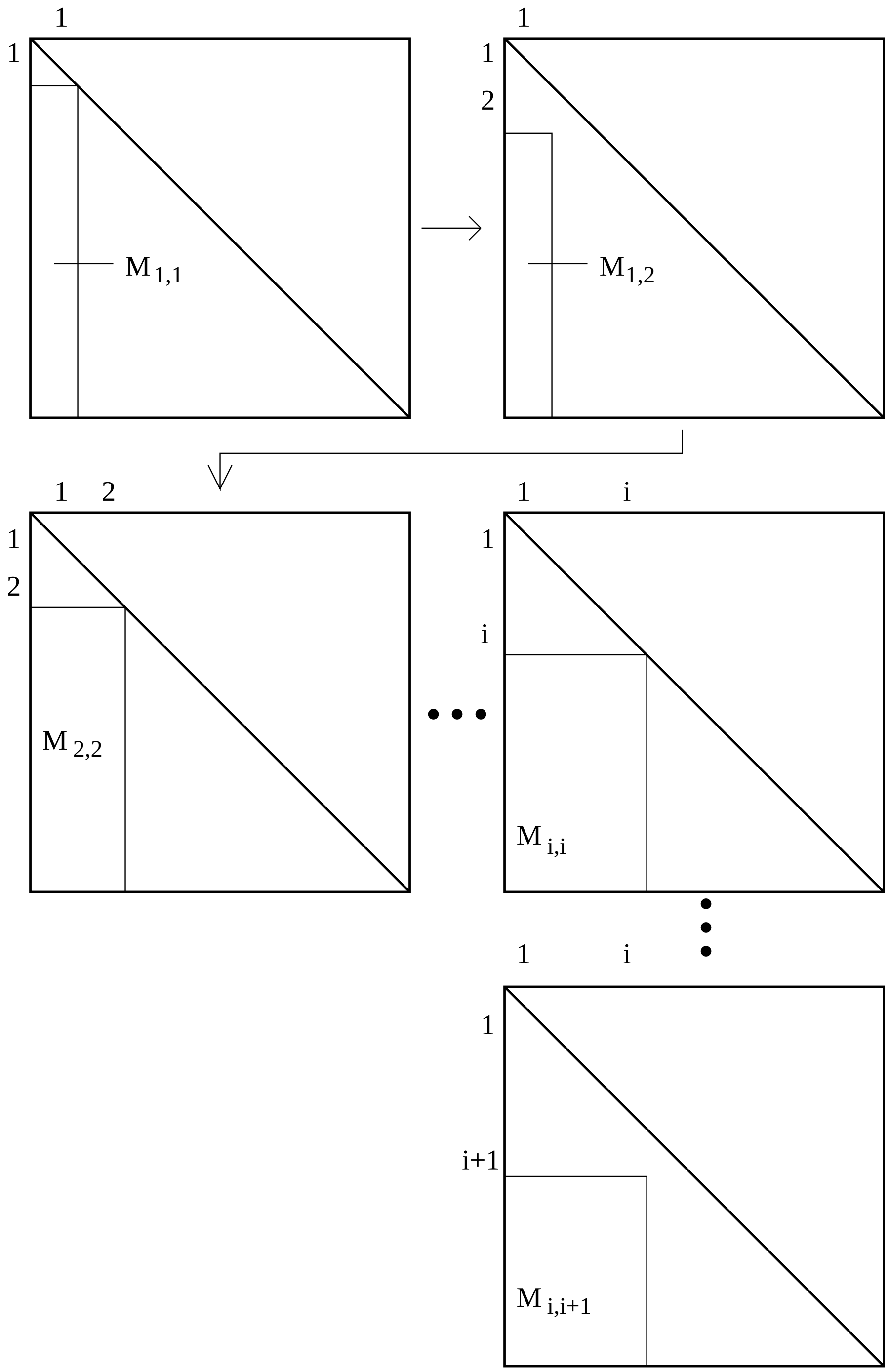

Figure 6A

In this way we generate a sequence of sub-matrices

$$M_{1,1}, M_{1,2}, M_{2,2}, \ldots, M_{i,i}, M_{i,i+1}, \ldots$$

which we call the M-sequence for Λ.

Lemma 6.2. *Suppose $M_{i,j}$ is an element of some M-sequence. If $M_{i,j}$ has maximum rank then all $M_{k,l}$ prior to $M_{i,j}$ must also have maximum rank.*

Proof. The proof is left to the reader. □

We make use of the following lemma proven by David Ingerman.

Lemma 6.3. *Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. Let us further suppose than Γ has no isolated nodes. Let $M_{i,j}$ be the first sub-matrix in the M-sequence associated with Λ which has rank less than i. If $i < j$ then Γ is $Y - \Delta$ equivalent to a graph $\Gamma^{'}$ with a boundary to boundary edge between nodes i and j. Otherwise, $i = j$ and Γ is $Y - \Delta$ equivalent to a graph $\Gamma^{'}$ with a boundary spike at node i.*

It is important to note that all sub-matrices in the M-sequence which occur before $M_{i,j}$, the first sub-matrix in the M-sequence which has rank less than i, will all have atleast as many rows as columns. This fact is used implicitly in the following section.

7. Breaking Boundary Edges and Spikes

In this section we show how break boundary to boundary edges and boundary spikes found using lemma 6.3.

Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. Let us further suppose that using lemma 6.2 we have found that Γ is $Y - \Delta$ equivalent to a graph with a boundary to boundary edge $e \in E$ between two nodes p and $q = p + 1$ in ∂V. In order to break this boundary to boundary edge we must find $\gamma(pq)$, the conductance of edge e. Let $\Lambda^{'}$ be the response matrix for the graph $\Gamma^{'}$ produced by removing e from Γ. Let $\xi = \gamma(pq)$. From [1] we know that $\Lambda^{'}$ will equal the following:

$$\Lambda^{'}_{p,p} = \Lambda_{p,p} - \xi$$
$$\Lambda^{'}_{q,q} = \Lambda_{q,q} - \xi$$
$$\Lambda^{'}_{p,q} = \Lambda_{p,q} + \xi$$
$$\Lambda^{'}_{q,p} = \Lambda_{p,q} + \xi$$
$$\Lambda^{'}_{i,j} = \Lambda_{i,j}, \text{ otherwise}$$

The only sub-matrix in the M-sequence of Λ which will differ from the M-sequence for $\Lambda^{'}$ will be $M_{p,p}$. Now, $M_{p,p}$ must no longer have maximum rank because otherwise, lemma 6.1 would indicate that there still is a boundary to boundary edge between p and q, contradicting the assumption of criticality. Thus, we can use linear algebra and gaussian elimination on $M_{p,p}$ to find the value of ξ which would cause $M_{p,p}$ to have rank less than maximum.

If node $q = p+1$ was a boundary to boundary spike in Γ, then node q will no longer have any edges incident to it in $\Gamma^{'}$ and will thus be an isolated node in $\Gamma^{'}$. If this is the case then we can remove node q from $\Gamma^{'}$, eliminating row q and column q from $\Lambda^{'}$, without affecting the ranks of any members of the M-sequence from $M_{1,1}$ to $M_{p,p}$.

If node p was a boundary to boundary spike in Γ, then node p will no longer have any edges incident to it in $\Gamma^{'}$ and will thus be an isolated node in $\Gamma^{'}$. If this is the case then we can remove node q from $\Gamma^{'}$, eliminating row q and column q from $\Lambda^{'}$, without affecting the ranks of any members of the M-sequence from $M_{1,1}$ to $M_{p-1,p-1}$.

Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. Let us further suppose that using lemma 6.3 we have found that Γ is $Y - \Delta$ equivalent to a graph with a boundary spike at node $i \in \partial V$. Node i is either a boundary to boundary spike or a boundary spike to some interior node. We will consider these two cases separately.

Let us consider the case in which node i is a boundary spike to some interior node j. In order to break this boundary spike we must find $\gamma(ij)$, the conductance of edge $e \in E$ between nodes i and j. Let $\Lambda^{'}$ be the response matrix for the graph $\Gamma^{'}$ produced by contracting edge e between nodes i and j in Γ. Let $\xi = \gamma(ij)$. We know from [1] that to contract edge e we can utilize the process of <u>adjoining a boundary spike</u> with conductance $-\xi$ to node i.

We will make use of a basic concept from Linear Algebra. Let W be a $m \times n$ matrix. Suppose

$$W = \left[\begin{array}{c|c} x & b \\ \hline a & M \end{array}\right] \tag{7.1}$$

where x is a single entry in W.

$$W/x = M - \frac{ba}{x} \tag{7.2}$$

$$rank(W) = 1 + rank(W/x) \tag{7.3}$$

The first step in the process of adjoining a boundary spike with conductance $-\xi$ to node i is to add a new boundary node j in circular order directly after node i with only one edge, which has conductance $-\xi$ and connects node j to node i. This will produce a new response matrix Λ_1 as follows:

$$\Lambda_1 = \begin{bmatrix} P & \tilde{b} & 0 & Q \\ a & m-\xi & \xi & \tilde{a} \\ 0 & \xi & -\xi & 0 \\ M & b & 0 & R \end{bmatrix} \tag{7.4}$$

where

$$P = \begin{bmatrix} \Lambda_{1,1} & \cdots & \Lambda_{1,i-1} \\ \vdots & \ddots & \vdots \\ \Lambda_{i-1,1} & \cdots & \Lambda_{i-1,i-1} \end{bmatrix}$$

$$Q = \begin{bmatrix} \Lambda_{1,i+1} & \cdots & \Lambda_{1,n} \\ \vdots & \ddots & \vdots \\ \Lambda_{i-1,i+1} & \cdots & \Lambda_{i-1,i+1} \end{bmatrix}$$

$$M = \begin{bmatrix} \Lambda_{i+1,1} & \cdots & \Lambda_{i+1,i-1} \\ \vdots & \ddots & \vdots \\ \Lambda_{n,1} & \cdots & \Lambda_{n,i-1} \end{bmatrix}$$

$$R = \begin{bmatrix} \Lambda_{i+1,i+1} & \cdots & \Lambda_{i+1,n} \\ \vdots & \ddots & \vdots \\ \Lambda_{n,i+1} & \cdots & \Lambda_{n,n} \end{bmatrix}$$

$$m = \Lambda_{i,i}$$

$$\begin{bmatrix} \tilde{b} \\ m \\ b \end{bmatrix} = \begin{bmatrix} \Lambda_{1,i} \\ \vdots \\ \Lambda_{n,i} \end{bmatrix}$$

$$\begin{bmatrix} a & m & \tilde{a} \end{bmatrix} = \begin{bmatrix} \Lambda_{i,1} & \cdots & \Lambda_{i,n} \end{bmatrix}$$

The next step in the process of adjoining a boundary spike with conductance $-\xi$ to node i is to take the Schur complement of Λ_1 with respect to the entry $\delta = m - \xi$. This will produce a new response matrix Λ_2 as follows:

$$\Lambda_2 = \begin{bmatrix} P - \frac{\tilde{b}a}{\delta} & -\frac{\xi\tilde{b}}{\delta} & Q - \frac{\tilde{b}\tilde{a}}{\delta} \\ -\frac{\xi a}{\delta} & -\xi - \frac{\xi^2}{\delta} & -\frac{\xi\tilde{a}}{\delta} \\ M - \frac{ba}{\delta} & -\frac{\xi b}{\delta} & R - \frac{b\tilde{a}}{\delta} \end{bmatrix} \tag{7.5}$$

Let $\Lambda^{'}$ be the response matrix for the graph $\Gamma^{'}$ produced by contracting the boundary spike at node i. We have just shown that $\Lambda^{'} = \Lambda_2$. Let $M_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ. Let $M^{'}_{a,b}$ be our notation for a sub-matrix in the M-sequence for $\Gamma^{'}$. By applying equations 7.1 - 7.3 we know the following:

$$M^{'}_{i-1,i-1} = \begin{bmatrix} -\frac{\xi a}{\delta} \\ M - \frac{ba}{\delta} \end{bmatrix} = \begin{bmatrix} a & \delta \\ 0 & \xi \\ M & b \end{bmatrix} / \delta \tag{7.6}$$

$$rank\left(M^{'}_{i-1,i-1}\right) = rank\left(\begin{bmatrix} a & \delta \\ 0 & \xi \\ M & b \end{bmatrix}\right) - 1 \tag{7.7}$$

We assumed that we used lemma 6.3 to find that Γ is $Y-\Delta$ equivalent to a graph with a boundary spike at node $i \in \partial V$. Therefore,

$$M_{i-1,i-1} = \begin{bmatrix} a \\ M \end{bmatrix}$$

must have maximum rank. Therefore,

$$\begin{bmatrix} a & \delta \\ 0 & \xi \\ M & b \end{bmatrix}$$

must also have maximum rank. Thus, $M'_{i-1,i-1}$ must also have maximum rank. By lemma 6.1 we know that $M'_{1,1} \ldots M'_{i-1,i-1}$ all have maximum rank. We know that

$$M_{i,i} = \begin{bmatrix} M & b \end{bmatrix}$$

has rank less than maximum. Therefore,

$$\begin{bmatrix} a & \delta & \xi \\ M & b & 0 \end{bmatrix}$$

must have rank less than maximum. Now, using the same methodology used to find the rank of $M'_{i-1,i-1}$, it follows that $M'_{i,i}$ must have rank less than maximum. If $M'_{i-1,i}$ had maximum rank, then by lemma 6.3 Γ' would be $Y-\Delta$ equivalent to a graph with a boundary spike at node i. Γ' can not be $Y-\Delta$ equivalent to a graph with a boundary spike at node i or the assumption that Γ was critical would be violated. Hence,

$$rank\left(M'_{i-1,i}\right) < maximum \tag{7.8}$$

We now have two useful results:

(1) $M'_{i-1,i}$ is the first sub-matrix in the M-sequence for Γ' with rank less than maximum.

(2) The value of the conductance $\xi > 0$ will be such that

$$rank\left(\begin{bmatrix} a & m-\xi \\ M & b \end{bmatrix}\right) < maximum$$

We now consider the case when Γ has a boundary to boundary spike at node i. Let node $j \in \partial V$ be one node which is connected to node i. Now, node j is either a boundary to boundary spike to node i or node j is connected to other nodes in Γ. We will consider these two cases separately.

Let us consider the case in which node j is a boundary to boundary spike to node i. Nodes i and j are disconnected from the rest of Γ since both i and j are boundary to boundary spikes to each other. Hence, columns i and j in Λ will be linearly independent from all other columns in Λ. Both rows i and j and columns i and j in Λ will have zero entries except for $\Lambda_{i,i}, \Lambda_{i,j}, \Lambda_{j,i}, \Lambda_{j,j}$. Now, if $M_{i-1,i}$ has maximum rank then $M_{i,i}$ must also have maximum rank. Similarly, if $M_{j-1,j}$ has maximum rank then $M_{j,j}$ must also have maximum rank. Thus, we would not have found a boundary spike at node i using lemma 6.3. Some sub-matrix other than $M_{i,i}$ in the M-sequence would have been the first sub-matrix with rank less than maximum.

Let us consider the case in which node j is not a boundary to boundary spike to node i. In order to break this boundary spike we must find $\gamma(ij)$, the conductance of edge $e \in E$ between nodes i and j. Let Λ' be the response matrix for the graph Γ' produced by contracting edge e between nodes i and j in Γ. Let $\xi = \gamma(ij)$. It follows from the fact that there is only one edge incident to node i that $\xi = \Lambda_{i,i}$. Thus, $m - \xi = 0$. We can not contract edge e by utilizing the process of adjoining a boundary spike with conductance $-\xi$ to node i because the Schur complement used in the process would not be defined. However, we can contract edge e, collapsing node i onto node j, by the using the process we call <u>collapsing a boundary to boundary spike</u> at node i to node j. This process is performed in three steps as follows:

(1) Remove edge e as a boundary to boundary edge between nodes i and j as described previously, producing a new response matrix Λ_1.

(2) Renumber the boundary nodes such that node j and node i are interchanged. This is done by switching row i with row j and column i with column j in Λ_1, producing a new response matrix Λ_2.

(3) Remove node j (previously node i before step 2) which has become isolated by removing row j and column j from Λ_2, producing a new response matrix Λ_3.

Now, $\Lambda' = \Lambda_3$. Let $M_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ. Let $M^1_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ_1. Let $M^2_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ_2. Let $M^3_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ_3. Let $M'_{a,b}$ be our notation for a sub-matrix in the M-sequence for Γ'.

Suppose that node j comes after node i. Λ_1 from Step (1) in collapsing a boundary to boundary spike at node i onto node j will be identical to Λ in columns $1 \dots i-1$. Thus, $M^1_{a,b} = M_{a,b}$ for $a \le i-1$ and $b \le i$. Hence, $M^1_{1,1} \dots M^1_{i-1,i}$ all have maximum rank. Λ_2 Step (2) will equal Λ_1 with rows i and j interchanged as well as columns i and j interchanged. After switching columns i and j $M^2_{1,1} \dots M^2_{i-1,i}$ will still equal $M^1_{1,1} \dots M^1_{i-1,i}$. Switching rows i and j can not affect the rank of $M^2_{1,1} \dots M^2_{i-1,i-1}$ because $M^2_{1,1} \dots M^2_{i-1,i-1}$ will be row equivalent to $M^1_{1,1} \dots M^1_{i-1,i-1}$. Thus, $M^2_{i-1,i-1}$ will have maximum rank because $M^1_{i-1,i-1}$ has maximum rank. Now, $M^2_{i-1,i}$ and $M^1_{i-1,i}$ will be row equivalent except that $M^2_{i-1,i}$ will have a row of zero entries from row i in Λ while $M^1_{i-1,i}$ will have a row of entries from row j in Λ. Using the fact that $M^1_{i,i}$ has rank less than maximum and applying equations 7.1 - 7.3, it can be shown that $M^3_{i-1,i}$ has rank less than maximum. Thus, $M'_{i-1,i}$ is the first sub-matrix in the M-sequence for Γ' with rank less than maximum as was the case when node i was a boundary spike to an interior node.

Suppose that node j comes before node i. As shown in figure 7A, nodes $j+1, \ldots, i-1$ can only be connected to each other and node j.

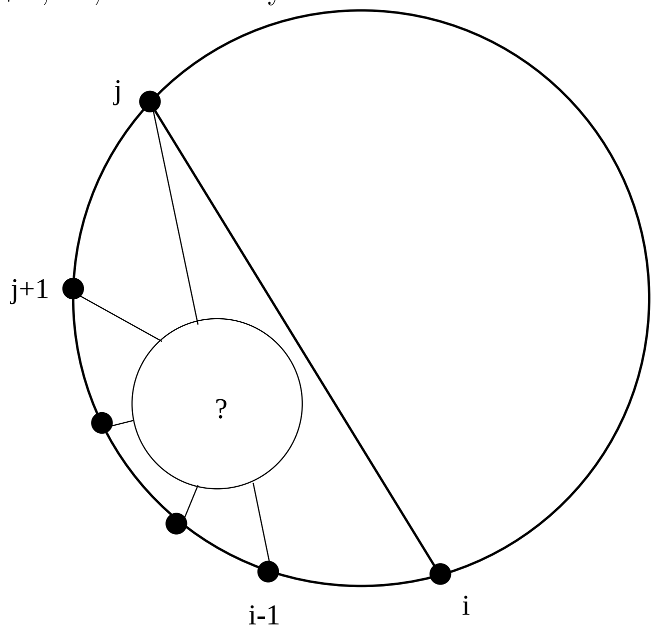

Figure 7A

Let $P = j, \ldots, i-1$ and $(P;Q)$ be a circular pair. We know that $\Lambda(P;Q) = 0$. It follows that $M_{i-1,i-1}$ would have to have rank less than maximum. Therefore, there can not be any nodes between j and i or we would not have found node i as a boundary spike using lemma 6.3. Thus, $j = i-1$. In this case, we use the process of collapsing a boundary to boundary spike at node i onto node $i-1$. However, we can not conclude which sub-matrix will be the first with rank less than maximumn in the new M-sequence for Γ' after node i has been collapsed onto node $i-1$.

8. Finding Geodesics

Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. Let $M_{i,j}$ be the first sub-matrix in the M-sequence associated with Λ which has rank less than i. Now, $M_{i,j}$ corresponds to either a boundary to boundary edge or a boundary spike. From [1] we have the following two lemmas:

Lemma 8.1. *Suppose Γ is a critical circular planar graph and pq is a boundary to boundary edge. Let Γ' be the graph obatined after the deletion of pq. Then Γ' is also a critical circular planar graph.*

Lemma 8.2. *Suppose Γ is a critical circular planar graph with a boundary spike rp where r is a boundary node of Γ. Let Γ' be the graph obatined after contracting rp to p. Then Γ' is also a critical circular planar graph.*

Thus, we can remove the boundary to boundary edge or a boundary spike corresponding to $M_{i,j}$ and the resulting graph $\Gamma^{'}$ will still be critical circular planar, but will have fewer edges and connections than Γ. In addition, if we return the boundary to boundary edge or a boundary spike to $\Gamma^{'}$, it would be $Y-\Delta$ equivalent to Γ.

Lemma 8.3. *Suppose we have a critical circular planar resistor network (Γ,γ) with n boundary nodes and a response matrix Λ. Let $M_{i,j}$ be the first sub-matrix in the M-sequence associated with Λ which has rank less than i. If $i<j$, then after removing the boundary to boundary edge between nodes i and $j=i+1$ in Γ producing a new graph $\Gamma^{'}$, $M_{i,i}$ will be the first sub-matrix in the M-sequence associated with $\Gamma^{'}$ which has rank less than maximal. This will guarantee that either $\Gamma^{'}$ is $Y-\Delta$ equivalent to a graph $\Gamma^{''}$ with boundary spike at node i, or node i will have become isolated in $\Gamma^{'}$.*

Proof. We have shown that breaking a boundary to boundary edge will cause node $i+1$ to become isolated in $\Gamma^{'}$ if node $i+1$ was a boundary to boundary spike Γ. However, the ranks of $M_{1,1}$ to $M_{i,i}$ will be unaffected by removing this isolated node. Similarly, node i could become isolated, but removing this isolated node would only leave the ranks of $M_{1,1}$ to $M_{i-1,i-1}$ unaffected.

If node i does not become isolated we have shown that $M_{i,i}$ will no longer have maximum rank but the ranks of $M_{1,1}\dots M_{i-1,i}$ will be unchanged and thus still maximal. Therefore, from lemma 6.1 we know that $\Gamma^{'}$ is $Y-\Delta$ equivalent to a graph $\Gamma^{''}$ with boundary spike at node i. However, if node i does become isolated removing it would affect the ranks of $M_{i-1,i}$ and subsequent sub-matrices in the M-sequence. Thus, we can not conclude that $\Gamma^{'}$ is $Y-\Delta$ equivalent to a graph $\Gamma^{''}$ with boundary spike at node i. □

Lemma 8.4. *Suppose we have a critical circular planar resistor network (Γ,γ) with n boundary nodes and a response matrix Λ. Let $M_{i,j}$ be the first sub-matrix in the M-sequence associated with Λ which has rank less than i. If $i=j$ and node i is not a boundary to boundary spike to node $i-1$, then after removing the boundary spike at node i in Γ producing a new graph $\Gamma^{'}$, $M_{i-1,i}$ will be the first sub-matrix in the M-sequence associated with $\Gamma^{'}$ which has rank less than maximal. Thus, there will be a boundary to boundary edge between nodes $i-1$ and i in $\Gamma^{'}$ or node i was a boundary to boundary spike to node $i-1$ in Γ.*

Proof. We have shown that if i is a boundary spike to an interior node then $M_{i-1,i}$ will be the first element of the M-sequence for $\Gamma^{'}$ with rank less than maximum. Thus there must be a boundary to boundary edge between nodes $i-1$ and i in $\Gamma^{'}$. We have shown that if i is a boundary to boundary spike to some node $j\in\partial V$ and j comes after i in clockwise circualr order, then $M_{i-1,i}$ will be the first element of the M-sequence for $\Gamma^{'}$ with rank less than maximum. Thus there must be a boundary to boundary edge between nodes $i-1$ and i in $\Gamma^{'}$. We have further shown that the only other possiblity is that node i is a boundary to boundary spike to node $i-1$, in which case we can not conlude which element of the M-sequence for $\Gamma^{'}$ will be the first with rank less than maximum. $\square$

Thus, given solely the Λ matrix for a critical circular planar resistor network (Γ, γ), we can show that Γ is $Y-\Delta$ equivalent to a graph $\Gamma^{'}$ which has a sequence of boundary spikes and boundary to boundary edges similar to one of the four forms shown in Figure 8A.

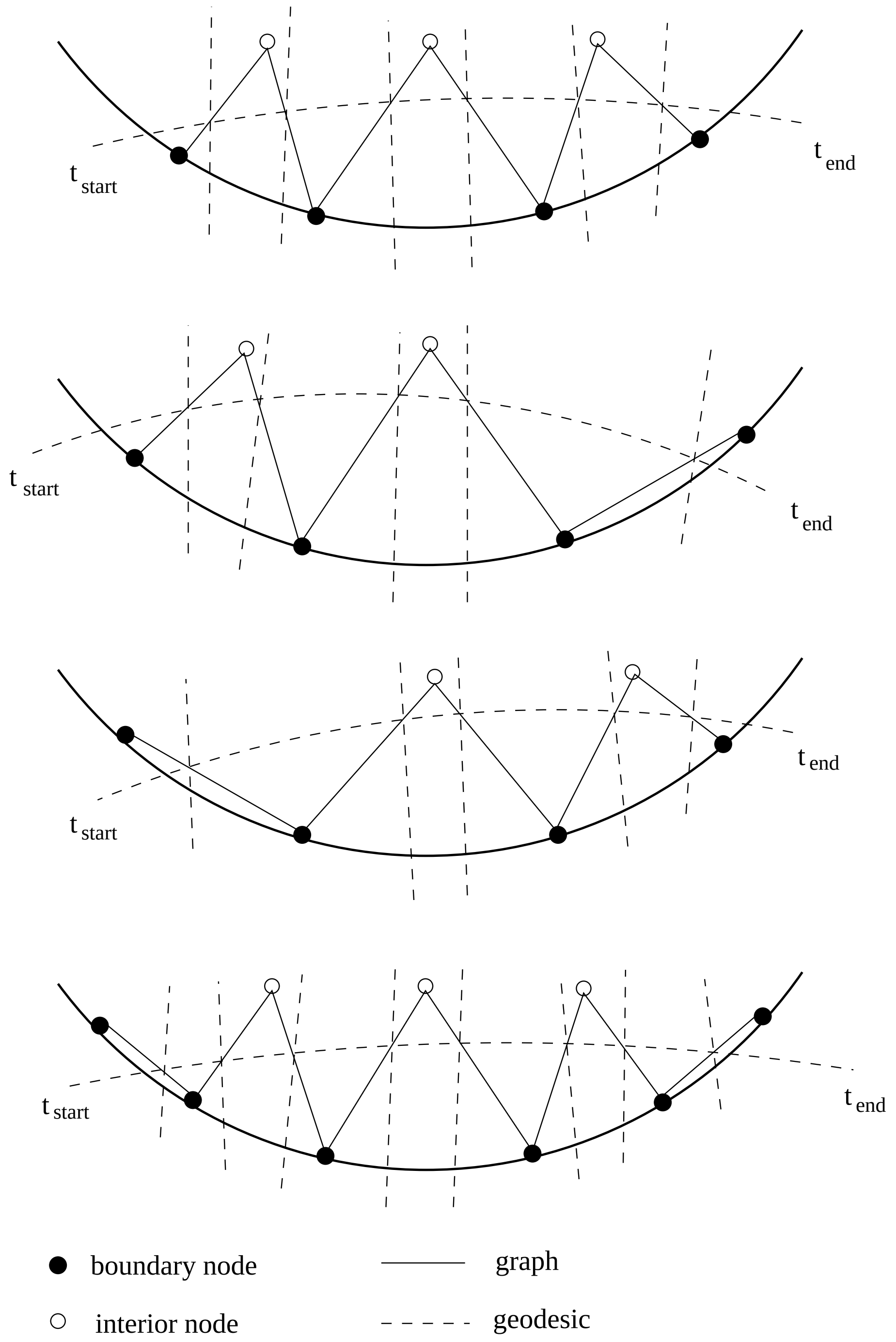

Figure 8A

Lemma 8.5. *Suppose Γ is a critical circular planar graph which is $Y-\Delta$ equivalent to a graph Γ' which has a sequence of nodes and edges in one* *of the four forms shown in Figure 8A. This uniquely determines the two endpoints of a geodesic G in $M(\Gamma)$. In addition G interlaces with any other geodesics in $M(\Gamma)$ which have an endpoint which lies between t_{start} and t_{end} as shown in Figure 8A.*

Proof. The proof is left to the reader. □

9. Recovery Algorithm

Given a critical circular planar resistor graph (Γ, γ) we can now find a graph Γ^1 which is $Y - \Delta$ equivalent to Γ and has a sequence of boundary spikes and boundary to boundary edges as shown in figure 8A. In addition, we located a geodesic in $M(\Gamma)$. Now we collapse boundary spikes and break boundary to boundary edges in Γ^1 as indicated by the sequence we found, producing a new graph Γ^2. We know the exact relationship between the z-sequence of Γ^2 and the z-sequence for Γ^1 because we know how a medial graph is affected by collapsing a boundary spike or breaking a boundary to boundary edge. Thus, we can continue this process, finding a graph Γ^3 which is $Y - \Delta$ equivalent to Γ^2 and again has a sequence of boundary spikes and boundary to boundary edges as shown in figure 8A. Now have found a geodesic in Γ^2. We can now collapse boundary spikes and break boundary to boundary edges, removing the sequence and producing a new graph Γ^4. Every time we find a geodesic and then remove it, we are guaranteed to have atleast one less boundary node in the new graph. Thus, we can continue this process until there is no longer anything left of the original graph Γ. By tracing the steps we took, we have found the z-sequence of Γ as well as a new graph $\Gamma^{'}$ with conductances $\gamma^{'}$ such that the resistor network $(\Gamma^{'}, \gamma^{'})$ is not only $Y - \Delta$ equivalent to the original resistor network (Γ, γ) but also has the same response matrix Λ.

10. Non-Critical Graphs

Suppose we have a circular planar resistor (Γ, γ) which is not critical. By removing all the superfluous edges in (Γ, γ) we would produce a critical circular planar resistor $(\Gamma^{'}, \gamma^{'})$. The response matrix $\Lambda^{'}$ for this network will be identically equal to the response matrix Λ for the original network if $(\Gamma^{'}, \gamma^{'})$ is produced in the following manner:

(1) Multiple edges in series $e_1, \ldots, e_n$ in Γ with conductances $\gamma_1, \ldots, \gamma_n$ are replaced by one edge in $\Gamma^{'}$ with conductance

$$\frac{1}{\frac{1}{\gamma_1} + \ldots + \frac{1}{\gamma_n}}$$

(2) Multiple edges in parallel $e_1, \ldots, e_n$ in Γ with conductances $\gamma_1, \ldots, \gamma_n$ are replaced by one edge in $\Gamma^{'}$ with conductance $\gamma_1 + \ldots + \gamma_n$ in $\Gamma^{'}$.

Thus, by applying our algorithm to a non-critical circular planar resistor network (Γ, γ) we will find the z-sequence for $(\Gamma^{'}, \gamma^{'})$ as described above, as well as a new graph $\Gamma^{''}$ with conductances $\gamma^{''}$ such that the resistor network

(Γ'', γ'') is not only $Y-\Delta$ equivalent to the resistor network (Γ', γ') but also has the same response matrix Λ as both (Γ, γ) and (Γ', γ').

We know that all members of a $Y - \Delta$ equivalence class have the same number of edges. Hence, our algorithm can be used to test the criticality of a graph. A graph will be critical if and only if it has the same number of edges as the graph which our algorithm produces.

11. Continuing Research

In our efforts to determine the shape of a graph from Λ we found an interesting result which we were unable to prove, but shall state as a conjecture.

Conjecture 11.1. *Suppose we have a critical circular planar resistor network (Γ, γ) with n boundary nodes and a response matrix Λ. Let t_{start} and t_{end} be two endpoints of geodesics (or possibly the two endpoints of one geodesic) in $M(\Gamma)$. Now t_{start} and t_{end} divide the boundary nodes of Γ into two halves. Let B be the set containing the boundary nodes contained in the half with the least number of boundary nodes. Let B' be the set containing the boundary nodes contained in the other half. Let $i \in B$ be the boundary node with no other boundary nodes between it and t_{start} on C, the circle which Γ is embedded in. Let $j \in B$ be the boundary node with no other boundary nodes between it and t_{end} on C. Let $k, l \in B'$ be the boundary nodes closest to i and j on C, respectively. If there is a geodesic with an endpoint between t_{start} and i then remove k from B'. If there is a geodesic with an endpoint between t_{end} and j then remove l from B'. Let M be a sub-matrix of Λ containing the intersection of the columns for the boundary nodes in B with the rows for the boundary nodes in B'. Let T be the set of endpoints t of geodesics in $M(\Gamma)$ such that all $t \in T$ are on the half of C corresponding to B between t_{start} and t_{end} inclusively. Let N be the number of geodesics with both ends contained in T. Now,*

$$rank(M) = maximum - N$$

If the conjecture is true, then it would be possible to recover the z-sequence for a critical circular planar resistor network solely from the ranks of sub-matrices of Λ, without recovering any conductivities or performing any operations on Λ.

12. Results and Examples

The algorithm we have described here to recover the shape of a graph from its response matrix Λ was coded in the C++ language in the unix environment. The program also implements a procedure for recovering the z-sequence based on conjecture 11.1. The program code and a README file explaining how to use the program appear in the Appendix. The program will work for graphs with up to about 70 to 80 edges. Graphs with more edges will result in roundoff error becoming large enough that the Λ matrix becomes corrupted. Roundoff error occurs from operations on the Λ matrix as well as from determining conductivities. However, the procedure based on conjecture 11.1 which recovers solely the z-sequence produces no roundoff error and can be applied to an arbitrarily large graph. In addition, this procedure, albeit unproven, has worked successfully on all sample data.

In the following example we ran the program on a five by five square lattice with conductances of one on each edge. The program has a built in procedure which will generate the Kirchoff matrices and Λ matrices for square lattices as explain in the README file contained in the appendix. We used this procedure for this example. A diagram of this graph is shown in Figure 12A.

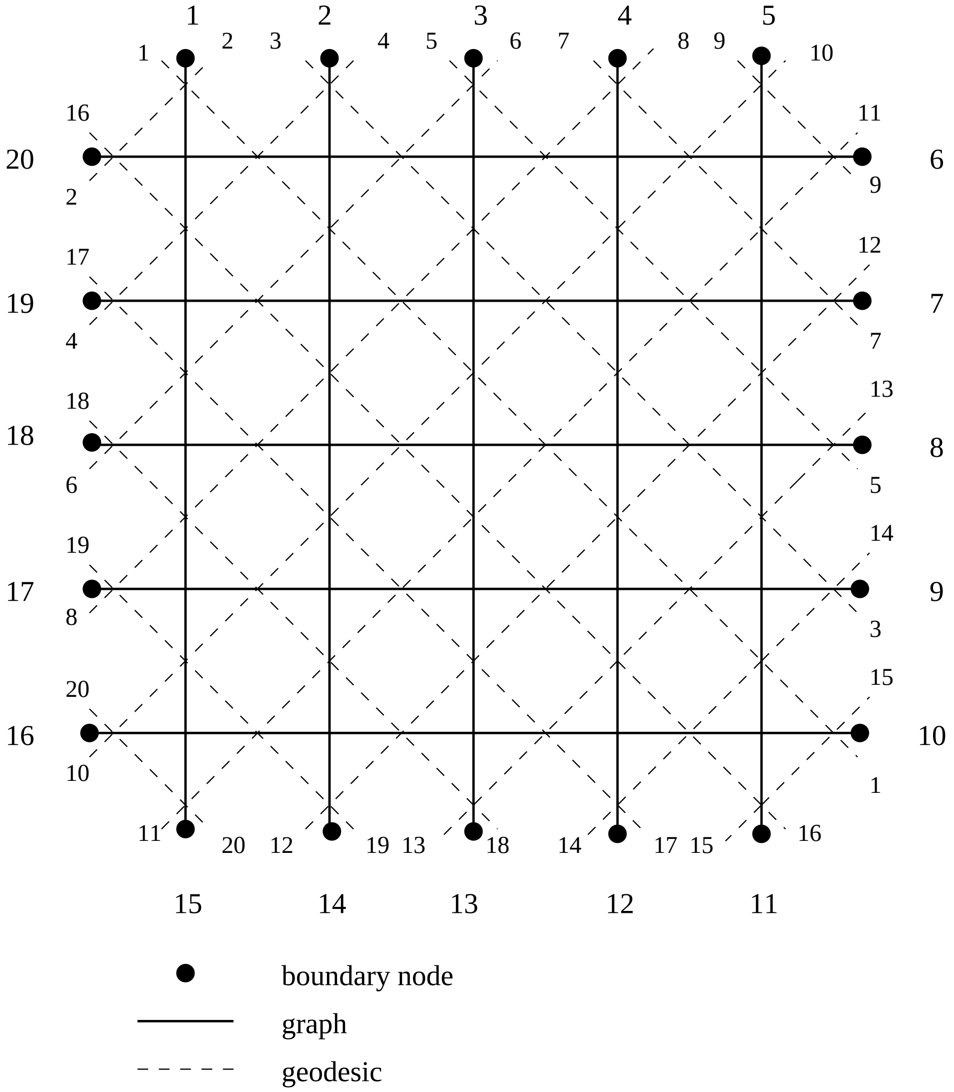

Figure 12A

```
escher% recover

Written by Derek A. Jerina
Based on paper by Derek A. Jerina, REU Summer 1996
Problems do have solutions you know...
0 1 0 5 1
0 0 1 0 0 1

tol 1e-06

N5 to N20 Spike cond 1
N4 to N20 BB cond 1
N5 to N21 Spike cond 1
N20 to N21 BB cond 1
N4 to N22 Spike cond 1
N3 to N22 BB cond 1
N5 to N23 Spike cond 1
N21 to N23 BB cond 1
N4 to N24 Spike cond 1
N22 to N24 BB cond 1
N3 to N25 Spike cond 1
N2 to N25 BB cond 1
N5 to N26 Spike cond 1
N23 to N26 BB cond 1
N4 to N27 Spike cond 1
N24 to N27 BB cond 1
N3 to N28 Spike cond 1
N25 to N28 BB cond 1
N2 to N29 Spike cond 1
N1 to N29 BB cond 1
N5 to N30 Spike cond 1
N26 to N30 BB cond 1
N4 to N31 Spike cond 1
N27 to N31 BB cond 1
N3 to N32 Spike cond 1
N28 to N32 BB cond 1
N2 to N33 Spike cond 1
N29 to N33 BB cond 1
N1 to N34 Spike cond 1
N0 to N34 BB cond 1
N10 to N30 BB cond 1
N5 to N35 Spike cond 1
N30 to N35 BB cond 1
N5 to N36 Spike cond 1
N35 to N36 BB cond 1
```

```
N35 to N31 BB cond 1
N5 to N37 Spike cond 1
N36 to N37 BB cond 1
N4 to N38 Spike cond 1
N31 to N38 BB cond 1
N5 to N39 Spike cond 1
N37 to N39 BB cond 1
N4 to N40 Spike cond 1
N38 to N40 BB cond 1
N38 to N32 BB cond 1
N39 to N15 BB cond 1
N4 to N41 Spike cond 1
N40 to N41 BB cond 1
N3 to N42 Spike cond 1
N32 to N42 BB cond 1
N41 to N16 BB cond 1
N3 to N43 Spike cond 1
N42 to N43 BB cond 1
N42 to N33 BB cond 1
N43 to N17 BB cond 1
N2 to N44 Spike cond 1
N33 to N44 BB cond 1
N44 to N18 BB cond 1
N44 to N34 BB cond 1
N34 to N19 BB cond 1

# of edges in derived graph: 60

# of edges in inputted kirchoff matrix: 60

Inputted kirchoff matrix was critical

Z-seq: 1 2 3 4 5 6 7 8 9 10 11 9 12 7 13 5 14 3 15 1 16 15 17
14 18 13 19 12 20 11 10 20 8 19 6 18 4 17 2 16


The greatest difference between any two entries of original
Lambda matrix and the derived Lambda matrix was 7.18331e-08%

Attempting to Obtain Z-seq without recovery.

Z-seq: 1 2 3 4 5 6 7 8 9 10 11 9 12 7 13 5 14 3 15 1 16 15 17
14 18 13 19 12 20 11 10 20 8 19 6 18 4 17 2 16
```

The exact shape of this graph was recovered because a square lattice is the only member of it's equivalence class. We did not output the Kirchoff matrix because of it's size. The program was able to determine that the graph was critical because we had the original Kirchoff matrix as well as the original Λ matrix.

The next example is of a cricular graph with two circles and three spikes, with conductances of one on each edge. The program has a built in procedure which will generate the Kirchoff matrices and Λ matrices for circular graphs as explained in the README file contained in the appendix. We used this procedure for this example. A diagram of this graph is shown in Figure 12B.

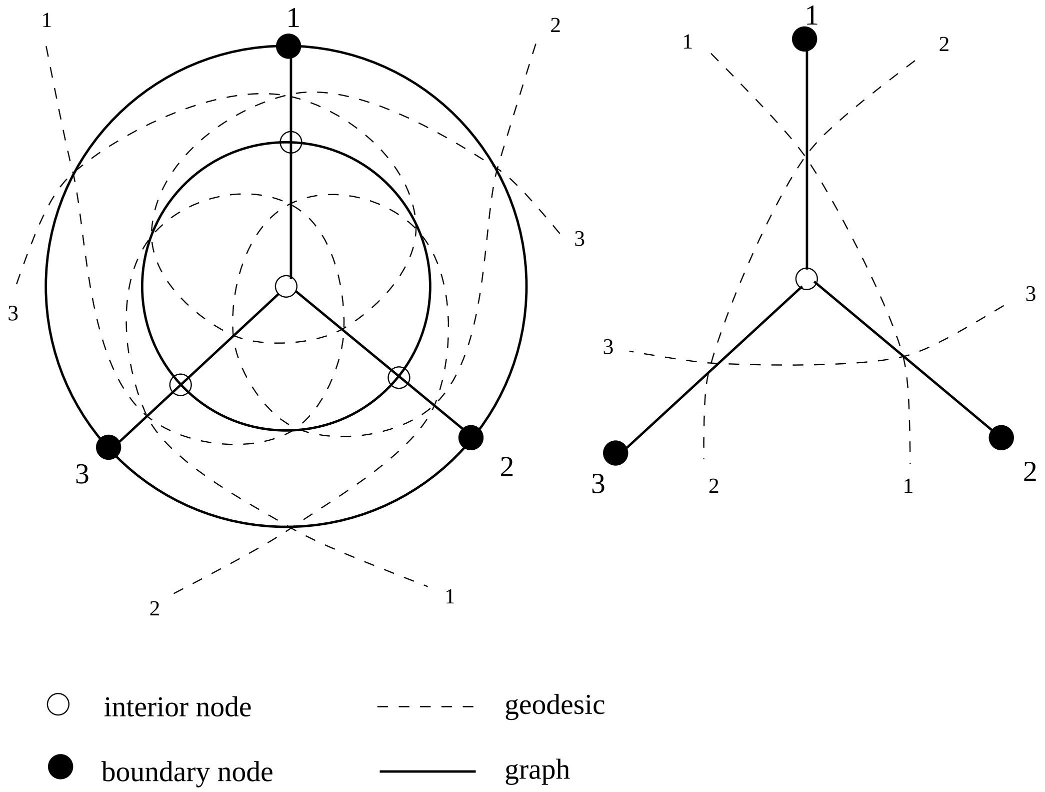

Figure 12B Figure 12C

```
escher% recover

Written by Derek A. Jerina
Based on paper by Derek A. Jerina, REU Summer 1996
Problems do have solutions you know...
0 0 1 3 2 1
1 1 1 1 1 1
```

```
Original Kmat
3       -1      -1      -1      0       0       0
-1      3       -1      0       -1      0       0
-1      -1      3       0       0       -1      0
-1      0       0       4       -1      -1      -1
0       -1      0       -1      4       -1      -1
0       0       -1      -1      -1      4       -1
0       0       0       -1      -1      -1      3

Original Lmat
2.53333         -1.26667        -1.26667
-1.26667        2.53333         -1.26667
-1.26667        -1.26667        2.53333

tol 1e-06

N1 to N3 Spike cond 3.8
N0 to N3 BB cond 3.8
N3 to N2 BB cond 3.8

# of edges in derived graph: 3

# of edges in inputted kirchoff matrix: 12

Inputted kirkchoff matrix was non-critical

Z-seq: 1 2 3 1 2 3


The derived Kmat
3.8     0       0       -3.8
0       3.8     0       -3.8
0       0       3.8     -3.8
-3.8    -3.8    -3.8    11.4

The derived Lmat
2.53333         -1.26667        -1.26667
-1.26667        2.53333         -1.26667
-1.26667        -1.26667        2.53333

The greatest difference between any two entries of original
Lambda matrix and the derived Lambda matrix was 1.75298e-14%

Attempting to Obtain Z-seq without recovery.
```

```
Z-seq: 1 2 3 1 2 3
```

The program was able to determine that the graph was not critical because we had the original Kirchoff matrix as well as the original Λ matrix. The actual graph recovered is shown in Figure 12C. While the derived graph may seem quite different from the original graph, the Λ matrix for the derived resistor network is identical to the Λ matrix for the original resistor network. The derived graph is essentially a lensless version of the original.

The next example is of a well-connected graph with nine boundary nodes and conductances of one on each edge. The program was run with data stored in an input file called “wellconn.dat”. This graph is shown in Figure 12D.

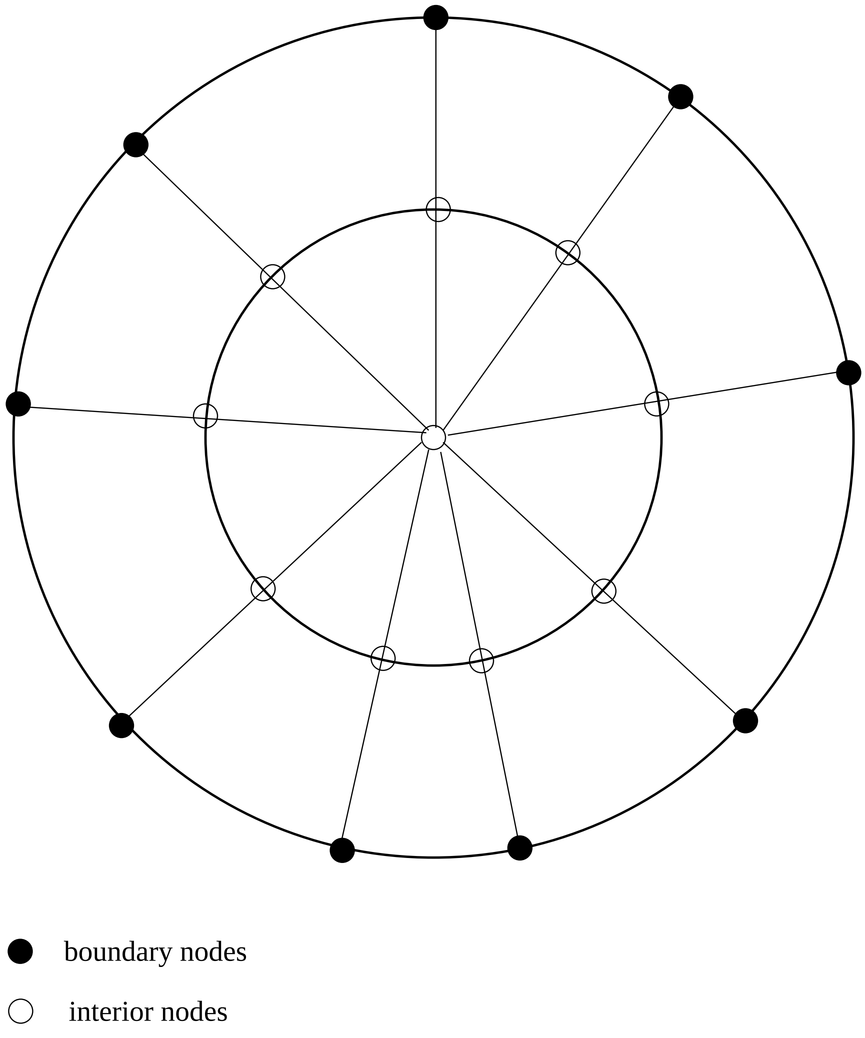

Figure 12D

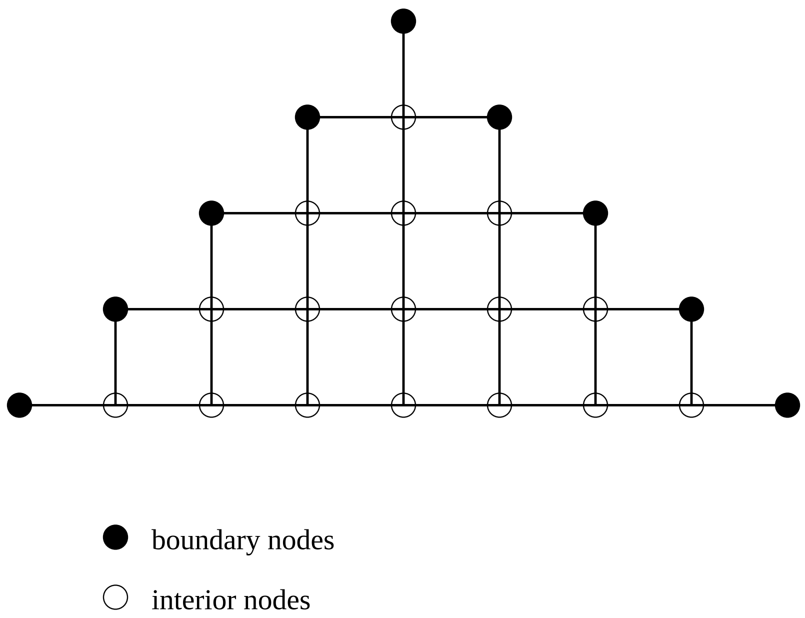

Figure 12E

```
escher% recover <wellconn.dat

Written by Derek A. Jerina
Based on paper by Derek A. Jerina, REU Summer 1996
Problems do have solutions you know...

tol 1e-06

N4 to N9 Spike cond 151
N3 to N9 BB cond 1.05594
N3 to N10 Spike cond 3.85315
N2 to N10 BB cond 1.88055
N2 to N11 Spike cond 1.88055
N1 to N11 BB cond 3.85315
N1 to N12 Spike cond 1.05594
N0 to N12 BB cond 151
N9 to N5 BB cond 1.05594
N3 to N13 Spike cond 0.760575
N10 to N13 BB cond 0.500472
N2 to N14 Spike cond 0.618944
N11 to N14 BB cond 1.10795
N1 to N15 Spike cond 0.0608849
```

```
N12 to N15 BB cond 7.39161
N3 to N16 Spike cond 3.85315
N13 to N16 BB cond 0.500472
N2 to N17 Spike cond 3.76671
N14 to N17 BB cond 9.54477
N1 to N18 Spike cond 0.409369
N15 to N18 BB cond 8.70655
N16 to N6 BB cond 1.88055
N2 to N19 Spike cond 0.618944
N17 to N19 BB cond 9.54477
N1 to N20 Spike cond 0.47541
N18 to N20 BB cond 9.03279
N2 to N21 Spike cond 1.88055
N19 to N21 BB cond 1.10795
N1 to N22 Spike cond 0.409369
N20 to N22 BB cond 9.03279
N21 to N7 BB cond 3.85315
N1 to N23 Spike cond 0.0608849
N22 to N23 BB cond 8.70655
N1 to N24 Spike cond 1.05594
N23 to N24 BB cond 7.39161
N24 to N8 BB cond 151

# of edges in derived graph: 36

# of edges in inputted kirchoff matrix: 36

Inputted kirchoff matrix was critical

Z-seq: 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9


The greatest difference between any two entries of original
Lambda matrix and the derived Lambda matrix was 4.16751e-11%

Attempting to Obtain Z-seq without recovery.

Z-seq: 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

The graph the program recovers turns out to be the standard graph as described in [1]. The derived graph is shown in Figure 12E. The reason for this is the way the recovery algorithm works. It finds a geodesic G and then removes it. Essentially, the algorithm is taking the inputted graph and performing motions such that all geodesics which interlace with G, intersect

G in the order which they occur in the z-sequence. These motions generate the "zig-zag" sequence along the boundary of the graph as shown in Figure 8A.

In our final example we ran the program on a ten by ten square lattice with conductances of one on each edge. We used the program's built in procedure to generate the network. This network is large enough that roundoff error prevents our algorithm from working. However, the procedure based on conjecture 11.1 successfully recovered the z-sequence.

```
escher% recover

Written by Derek A. Jerina
Based on paper by Derek A. Jerina, REU Summer 1996
Problems do have solutions you know...
0 1 0 10 1
0 0 0 0 0 1

Attempting to Obtain Z-seq without recovery.

Z-seq: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
19 22 17 23 15 24 13 25 11 26 9 27 7 28 5 29 3 30 1 31 30 32
29 33 28 34 27 35 26 36 25 37 24 38 23 39 22 40 21 20 40 18
39 16 38 14 37 12 36 10 35 8 34 6 33 4 32 2 31
```

The program and conjecture 11.1 were tested on several other resistor networks and both performed successfully every time.

13. Appendix

13.1. **README.**

```
This readme file explains how to use the recover program,
written by Derek A. Jerina.

The program may use user inputted data or generate a graph for the user.
If you use your own data (presumably in an data file) the format should
be as follows:

Line 1: "1 #ofB-nodes #ofInt-Nodes K-dat L-dat"
The "1" lets the program know that user inputted data will be inputted.
Next the number of boundary nodes and then the number of interior nodes.
Then use a "1" for K-dat if you will give the program the original
Kirchoff matrix and "0" if not.
Then use a "1" for L-dat if you will give the program the original
Lambda matrix and "0" if not.
Note: it wont matter what you give for #ofInt-Nodes if you are only giving
```

the Lambda matrix.

After Line 1, you must give the Kirchoff matrix if you inputted "1" for K-dat. Then you must enter the Lambda matrix if you inputted "1" for L-dat. The order is important.

Final Line: PrintOriginalK-mat PrintOriginalL-mat Do-Recovery
PrintDerivedK-mat PrintDerivedL-mat GetZ-seq-w/o-Recovery
Enter "1" if you what it done and "0" otherwise.

Now, the program can only tell you if the inputed graph was critical if you give the original K-mat. You can use "1" for Do-Recovery and the program will recover the z-seq and the K-mat and L-mat for a Y-Delta equivalent graph to the one you inputted. However, due to roundoff error the program will crash when the inputted graphs have around 80+ edges. But you can use a "1" for GetZ-seq-w/o-Recovery and still get just the z-sequence. However, the algorithm for getting the z-sequence has not be proven to be correct; it has however, worked for all inputted test files. The maximum number of boundary nodes is 150 due to constraints of the programming environment. If inputting a K-mat or recovering a graph with conductivities the total # of nodes must be less than or equal to 150.

To use a data file with the input simply type: "recover <filename"
There are a few sample data files already available to use or use as a template to make your own datafile. These files all have the extension ".dat"

If you wish you can have the program generate a graph on its own. The program can make two types of graphs: square lattice and circular graphs. To use this feature type "recover". Then type as follows:

Line 1: 0 Square_Lat Circle
The "0" lets the program know that you wont be inputting a graph of your own. Then use either "1 0" for a square lattice or "0 1" for a circular graph. If you use "1 0" then continue Line 1 with: Size Conductance
Size will be the number of boundary nodes per side. Conductance will be the conductance assigned to each edge in the graph. A size of "2" for example would produce a graph in the shape of a tic-tac-toe board.
For example with a Line 1: "0 1 0 2 1" the program would produce a graph in the shape of a tic-tac-toe board.

If you use "0 1" for a circular graph then continue
Line 1 with: Circles Spikes Conductance
Circles will correspond to the number of circles in the graph. You may

have none. Spikes will correspond to the number of radial spikes from the center of the graph. The spikes connected the innermost node to the outer most circle but don't extend beyond that. Conductance again is the conductance assigned to each edge. You can see examples of what these graphs would typically look like in my paper's results section. As an example Line 1: "0 0 1 5 0 1" would produce a graph with one interior node and 5 spikes to boundary nodes from that interior node.
Line 2 should be the same as the Last Line for user inputted graphs; it simple lets the program know what you what done with the given data and what output to print.

To direct output to a data file one can use "> filename". For example, "recover <inputfile >outputfile" will have the program read in input from the inputfile and write the output to outputfile.

13.2. Makefile.

```
#set up compiler and options
CC = g++
CFLAGS = -g

#set up C++ suffixes and relationship between .cc and .o files
.SUFFIXES: .cc

.cc.o:
$(CC) $(CFLAGS) -c $<

#
# program specific options
#

recover: tools.o matrix.o graph.o shape.o
$(CC) $(CFLAGS) -o recover shape.o graph.o matrix.o tools.o

tools.o: tools.h

matrix.o: matrix.h

graph.o: graph.h

shape.o: graph.h

clean:
/bin/rm *.o
```

13.3. tools.h.

```
/*
 * Author: Derek A. Jerina
 * Date:   July 25, 1996
 *
 */

#include <assert.h>
#include <iostream.h>

double abs(double val);
int half(int n);
int I(double r);
int size(int a[], int n);
void print_array(int a[], int n);
void arrange(int a[], int n);
void swap(int a[],int t1,int t2);
void reorder(int a[], int first, int size);
void rm_geo(int a[],int t,int n);
int t_left(int a[], int n);
int g_between(int a[], int t1, int t2);
void get_t(int& t_start, int& t_end, int bb_start, int start,
   int end, int bb_end, int L_size);
```

13.4. **tools.cc.**

```
/*
 * Author: Derek A. Jerina
 * Date:   July 25, 1996
 *
 * Contains various operations on matrices and various useful functions
 *
 */

#include "tools.h"

double abs(double val)
{
  if (val >= 0)
    {return(val);}
  else
    {return(-val);};
};

int half(int n)
{
  assert(n >= 0);
```

```
  int i = 0;
  while (i < (n*0.500))
    {++i;};
  return(i);
}

int I(double r)
{
  int j = 0;
  while (j < r) ++j;
  return(j);
}

int size(int a[], int n)
{
  int i,j;
  j = 0;
  for (i = 0; i < n; ++i)
    if (a[i] > 0)
      ++j;
  return(j);
}

void print_array(int a[], int n)
{
  int i;
  for (i = 0; i < n; ++i)
    cout << a[i] << " ";
  cout << endl;
}

void arrange(int a[], int n)
{
  int i,j,k,used;
  int result[n];
  used = 0;
  for (i = 0; i < n; ++i)
    {
      k = -1;
      for (j = i+1; j < n; ++j)
    if (a[i] == a[j])
      k = j;
      if (k > i)
{
  ++used;
```

```
  result[i] = used;
  result[k] = used;
};
    };
  for (i = 0; i < n; ++i)
    a[i] = result[i];
}

void swap(int a[],int t1,int t2)
{
  int temp;
  temp = a[t1];
  a[t1] = a[t2];
  a[t2] = a[temp];
}

void reorder(int a[], int first, int size)
{
  int i, j, result[size];
  for (i = 0; i < size; ++i)
    {
      j = i + first;
      if (j >= size)
j = j - size;
      result[i] = a[j];
    };
  for (i = 0; i < size; ++i)
    a[i] = result[i];
}

void rm_geo(int a[],int t,int n)
{
  int i,j,k;
  for (i=t; i < (n); ++i)
    a[i]=a[i+1];
}

int t_left(int a[], int n)
{
  int i,sum;
  sum = 0;
  for (i=0; i < n; ++i)
    if (a[i]<0)
      ++sum;
  return(sum);
```

```
}

int g_between(int a[], int t1, int t2)
{
  int i,j,sum;
  sum = 0;
  for (i=t1; i <= t2; ++i)
    {
      if (a[i]>0)
for (j = i+1; j < t2; ++j)
  if (a[j] == a[i])
    ++sum;
    };
  return(sum);
}

void get_t(int& t_start, int& t_end, int bb_start, int start,
   int end, int bb_end, int L_size)
{
  t_start = start*2 - bb_start;
  if (t_start<0)
    t_start += L_size*2;
  t_end = end*2 + 1 + bb_end;
  if (t_end >= L_size*2)
    t_end = t_end - L_size*2;
}
```

13.5. **matrix.h.**

```
/*
 * Author: Derek A. Jerina
 * Date:   July 20, 1996
 *
 * Includes the header file for matrices and various operations on them.
 *
 */

#include <iostream.h>
#include <iomanip.h>
#include <assert.h>
#include "tools.h"

#define max_nodes 150

class matrix{
 private:
```

```
    int c_size, r_size;
    double p[max_nodes][max_nodes];
 public:
    matrix(int c, int r);
    init(int val);
    clean(double tol);
    wash(int n);
    get_matrix();
    void print_matrix(int precision);
    r_swap(int r1, int r2);
    c_swap(int c1, int c2);
    r_mult(int row, double val);
    c_mult(int col, double val);
    add_mult_r(int receive, double val, int send);
    add_mult_c(int receive, double val, int send);
    put(int row, int col,double val);
    matrix add(const matrix left, const matrix right);
    matrix subtract(const matrix left, const matrix right);
    matrix mult(const matrix left, const matrix right);
    matrix inv_mat(matrix mat);
    matrix kirk_to_lamda(matrix K, int b_n, int i_n);
    int rows() {return (c_size);}
    int cols() {return (r_size);}
    double operator()(int i, int j) {return (p[i][j]); }
    void operator=(const matrix m);
    int rank(matrix m,double tol);
    void get_medial(matrix L, int& bb_start, int& start, int& end,
    int& bb_end, double tol);
    form(matrix L, int bb_start, int start, int end, int bb_end);
    double get_cond(matrix L, int bb_start, int start, int end,
    int bb_end, double tol);
    matrix cyclic_reorder(matrix m, int first);
    break_edge(matrix& L, matrix& K, matrix C_V, int n1, int n2,
       double C, double tol);
    break_spike(matrix& L, matrix& K, matrix& C_V, int C_G[], int n,
int geo[], int& z_num, double C, double tol);
    rm_row(int row);
    rm_col(int col);
    rm_unconn_n(matrix& L, matrix& C_V, double tol, int& z_num,
int C_G[], int geo[]);
    recover(matrix& L, matrix& K, int& i_nodes, double frac, int o_edges);
    square_lat(int n, double C);
    circle(int spikes, int circles, double C);
    double min();
    get_geodesic(matrix L, int geo[], int& t_start, int& t_end,
```

```
 int level, double tol);
    get_geodesic2(matrix L, int geo[], int& t_start, int& t_end,
 int level, double tol);
    find_zseq(matrix L, double tol);
    int count_edges();
    double maxdiff(matrix LO, matrix LD);
  };
```

13.6. **matrix.cc.**

```
/*
 * Author: Derek A. Jerina
 * Date:   July 20, 1996
 *
 * Includes the implementation for matrices and various operations on them.
 *
 */

#include "matrix.h"
#include <iomanip.h>

matrix:: matrix(int c, int r)
{
  c_size = c;
  r_size = r;
}

matrix:: init(int val)
{
  for (int i = 0; i < c_size; ++i)
    for (int j = 0; j < r_size; ++j)
      p[i][j] = val;
}

matrix:: clean(double tol)
{
  int i,j;
  for (i = 0; i < c_size; ++i)
    for (j = 0; j < r_size; ++j)
      if (abs(p[i][j]) < tol)
p[i][j] = 0;
}

matrix:: wash(int n)
{
  int i,j;
```

```
  for (i = 0; i < c_size; ++i)
    {
      p[i][n] = 0;
      p[n][i] = 0;
    };
}

matrix:: get_matrix()
{
  int i,j;
  for (i = 0; i < c_size; ++i)
    for (j = 0; j < r_size; ++j)
      cin >> p[i][j];
}

void matrix:: print_matrix(int precision)
{
 setprecision(precision);
 for (int i = 0; i < c_size; ++i)
    {
     for (int j = 0; j < r_size; ++j)
       {
 cout << p[i][j] << '\t';
/*
 if (i==j) {cout << "*" << ' ';}
 else if (p[i][j] != 0) {cout << '-' << ' ';}
 else cout << "0" << " ";
*/
       };
     cout << endl;
    };
}

matrix:: r_swap(int r1, int r2)
{
  double temp;
  for (int i = 0; i < r_size; ++i)
    {
      temp = p[r1][i];
      p[r1][i] = p[r2][i];
      p[r2][i] = temp;
    }
}

matrix:: c_swap(int c1, int c2)
```

```
{
  double temp;
  for (int i = 0; i < c_size; ++i)
    {
      temp = p[i][c1];
      p[i][c1] = p[i][c2];
      p[i][c2] = temp;
    }
}

matrix:: r_mult(int row, double val)
{
  for (int i = 0; i < r_size; ++i)
    p[row][i] = p[row][i] * val;
}

matrix:: c_mult(int col, double val)
{
  for (int i = 0; i < c_size; ++i)
    p[i][col] = p[i][col] * val;
}

matrix:: add_mult_r(int recieve, double val, int send)
{
  for (int i = 0; i < r_size; ++i)
    p[recieve][i] = p[recieve][i] + (val * p[send][i]);
}

matrix:: add_mult_c(int recieve, double val, int send)
{
  for (int i = 0; i < c_size; ++i)
    p[i][recieve] = p[i][recieve] + (val * p[i][send]);
}

matrix:: put(int row,int col,double val)
{
  p[row][col] = val;
}

matrix matrix:: add(const matrix left, const matrix right)
{
 assert(left.r_size == right.r_size);
 assert(left.c_size == right.c_size);
 int i,j;
 matrix result(left.c_size,left.r_size);
```

```
 for (i = 0; i < result.c_size; ++i)
    for (j = 0; j < result.r_size; ++j)
       result.p[i][j] = left.p[i][j] + right.p[i][j];
 return(result);
}

matrix matrix:: subtract(const matrix left, const matrix right)
{
 assert(left.r_size == right.r_size);
 assert(left.c_size == right.c_size);
 int i,j;
 matrix result(left.c_size,left.r_size);
 for (i = 0; i < result.c_size; ++i)
    for (j = 0; j < result.r_size; ++j)
       result.p[i][j] = left.p[i][j] - right.p[i][j];
 return(result);
}

matrix matrix:: mult(const matrix left, const matrix right)
{
 assert(left.r_size == right.c_size);
 matrix result(left.c_size,right.r_size);
 int i,j,k;
 double sum;
 for (i = 0; i < result.c_size; ++i)
    for (j = 0; j < result.r_size; ++j)
      {
       sum = 0;
       for (k = 0; k < left.r_size; ++k)
          sum += left.p[i][k] * right.p[k][j];
       result.p[i][j] = sum;
      };
 return(result);
}

matrix matrix:: inv_mat(matrix mat)
{
  matrix Inv(mat.c_size,mat.r_size);
  Inv.init(0);
  int i,j;
  for (i = 0; i < Inv.r_size; ++i)
    Inv.put(i,i,1);
  for (i = 0; i < Inv.r_size; ++i)
    {
      for (j = Inv.r_size - 1; j > i; --j)
```

```
{
  if (
//  Can't use fabs for some reason
//       fabs(mat(j,i)) > abs(mat(j-1,i))

(
  (mat.p[j-1][i]>0) && (
(mat.p[j][i] > mat.p[j-1][i]) ||
(-mat.p[j][i] > mat.p[j-1][i])
     )
)     ||
( (mat.p[j-1][i]<0) && (  (mat.p[j][i] < mat.p[j-1][i]) ||
(-mat.p[j][i] < mat.p[j-1][i])
     )
)
      )
    {
      Inv.r_swap(j,j-1);
      mat.r_swap(j,j-1);
    }
}
      Inv.r_mult(i,(1/mat(i,i)));
      mat.r_mult(i,(1/mat(i,i)));
      for (j = i+1; j < Inv.r_size; ++j)
{
  Inv.add_mult_r(j,-mat(j,i),i);
  mat.add_mult_r(j,-mat(j,i),i);
// Should decrease roundoff error
  mat.p[j][i]=0;
}
    }
  for (i = Inv.r_size -1; i > 0; --i)
    {
      for (j = i -1; j >= 0; --j)
{
  Inv.add_mult_r(j,-mat(j,i),i);
  mat.add_mult_r(j,-mat(j,i),i);
// Should decrease roundoff error
  mat.p[j][i]=0;
}
    }
  return (Inv);
}

matrix matrix:: kirk_to_lamda(matrix K, int b_n, int i_n)
```

```
{
  assert((K.c_size == K.r_size) && (K.r_size == (b_n+i_n)));
  matrix result(b_n,b_n);
  result.init(0);
  int i,j;
  if (i_n == 0)
    {
      result = K;
      return(result);
    };
  matrix A(b_n,b_n);
  for (i=0; i<b_n; ++i)
    for (j=0; j<b_n; ++j)
      A.put(i,j, K(i,j) );
  matrix B(b_n,i_n);
  matrix BT(i_n,b_n);
  for (i = 0; i < b_n; ++i)
    for (j = b_n; j < K.r_size; ++j)
      {
B.put(i,j-(b_n), K(i,j));
BT.put(j-(b_n),i, K(j,i));
      };
  matrix C(i_n,i_n);
  for (i = b_n; i < K.r_size; ++i)
    for (j = b_n; j < K.c_size; ++j)
      C.put(i-(b_n),j-(b_n), K.p[i][j]);
  result = result.subtract(A,result.mult(result.mult(B,result.inv_mat(C)),BT));
  return result;
}

void matrix::operator=(const matrix m)
{
  int i,j;
  c_size = m.c_size;
  r_size = m.r_size;
  for (i = 0; i < c_size; ++i)
    for (j = 0; j < r_size; ++j)
      p[i][j] = m.p[i][j];
}

int matrix:: rank(matrix m,double tol)
{
  if ( (m.r_size == 0) || (m.c_size == 0) )
    return(0);
  int i,j,k,row;
```

```
  k = 0;
  row = -1;
  for (i = 0; i < m.r_size; ++i)
    {
      if (row < (m.c_size-1))
{do
   {
     ++row;
     for (j = m.r_size-1; j > i; --j)
       if (abs(m.p[row][j]) > abs(m.p[row][j-1]))
 {m.c_swap(j,j-1);};
   }
 while ( (abs(m.p[row][i]) < tol) && (row < (m.c_size-1)) );
       };
//  Must take care of when there are more cols than rows
      if ( (abs(m.p[row][i]) >= tol) && (row < m.c_size) )
{
  ++k;
  m.c_mult(i,1/m.p[row][i]);
  for (j = i+1; j < m.r_size; ++j)
    m.add_mult_c(j,-m.p[row][j],i);
};
    };
  return(k);
}

void matrix:: get_medial(matrix L, int& bb_start, int& start, int& end,
 int& bb_end, double tol)
{
  assert((L.c_size == L.r_size) && (L.r_size != 0));
  int i,j,k,row,col;
// -1 used as a flag until initialized
  bb_start = -1;
  start = -1;
  end = -1;
  bb_end = -1;
  matrix D(0,0);
  for (i = 0; i < half(L.c_size); ++i)
    {
      D.c_size = L.c_size-(i+1);
      D.r_size = i+1;
      for (j = i+1; j < L.c_size; ++j)
for (k = 0; k < i+1; ++k)
  D.p[j-(i+1)][k] = L.p[j][k];
```

```
//For debugging purpposes only.
//D.print_matrix(5);
//cout << "rank " << D.rank(D,tol);

      if (D.rank(D,tol) < D.r_size)
{
// Note end = position in L matrix. So node 1 would be 0.
  end = i;
  bb_end = 0;
  break;
}
// Perhaps I should use --D.c_size
      D.c_size = L.c_size-(i+2);
      D.r_size = i+1;
      for (j = i+2; j < L.c_size; ++j)
for (k = 0; k < i+1; ++k)
  D.p[j-(i+2)][k] = L.p[j][k];
      if (D.rank(D,tol) < D.r_size)
{
// Note end = position in L matrix. So node 1 would be 0.
  end = i;
  bb_end = 1;
  break;
}
    }
  for (i = end; i > (end - half(L.r_size)); --i)
    {
      D.c_size = L.c_size - (end - i + 1) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
    D.p[j][k] = L.p[row][col];
  };
      if (D.rank(D,tol) < D.r_size)
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
```

```
    start = start + L.r_size;
  bb_start = 0;
  break;
}
// Perhaps I should use --D.c_size
      D.c_size = L.c_size - (end - i + 2) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
    D.p[j][k] = L.p[row][col];
  };
      if (D.rank(D,tol) < D.r_size)
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
    start = start + L.r_size;
  bb_start = 1;
  break;
}
    }
  assert( (bb_start != -1) && (start != -1) && (end != -1) && (bb_end != -1) );
}

matrix:: form(matrix L, int bb_start, int start, int end, int bb_end)
{
  assert( bb_start || (start != end) || bb_end );
  int i,j,k,row,col;
  i = start-1;
  j = 0;
  do
    {
      ++i;
      ++j;
      if (j >= L.r_size) j = j - L.r_size;
    }
  while (i!=end);
  r_size = j;
```

```
  c_size = L.c_size - r_size - bb_start + (1-bb_end);
  for (i=0; i < c_size; ++i)
    for (j=0; j < r_size; ++j)
      {
row = i + end + bb_end;
if (row >= L.c_size) row = row - L.c_size;
col = j + start;
if (col >= L.r_size) col = col - L.r_size;
p[i][j] = L.p[row][col];
      }
}

double matrix:: get_cond(matrix L, int bb_start, int start,
 int end, int bb_end, double tol)
{
  assert( bb_start || (start != end) || bb_end );
  if ( (start == end) && (bb_end==0) )
    return(L.p[start][start]);
  matrix m(0,0);
  m.form(L,bb_start,start,end,bb_end);
  assert( (m.r_size != 0) && (m.c_size != 0) );
  int i,j,k,row;
  k = 0;
  row = m.c_size;
  for (i = 0; i < (m.r_size-1); ++i)
    {
      if (row > 0)
{
  do
    {
      --row;
      for (j = m.r_size-2; j > i; --j)
if (abs(m.p[row][j]) > abs(m.p[row][j-1]))
  {m.c_swap(j,j-1);};
    }
  while ( (abs(m.p[row][i]) < tol) && (row > 0) );
};
//  Must take care of when there are more cols than rows
      if ( (abs(m.p[row][i]) >= tol) && (row >= 0) )
{
  ++k;
  m.c_mult(i,1/m.p[row][i]);
  for (j = i+1; j < m.r_size; ++j)
    m.add_mult_c(j,-m.p[row][j],i);
};
```

```
     };
  assert(k==(m.r_size-1));
  if (bb_end)
    {return(-m.p[0][m.r_size-1]);}
  else return(m.p[0][m.r_size-1]);
}

matrix matrix:: cyclic_reorder(matrix m, int first)
{
  matrix result(m.c_size,m.r_size);
  int i,j,row,col;
  for (i = 0; i < m.c_size; ++i)
    for (j = 0; j < m.r_size; ++j)
      {
row = i + first;
col = j + first;
if (row >= m.c_size)
  row = row - m.c_size;
if (col >= m.r_size)
  col = col - m.r_size;
result.p[i][j] = m.p[row][col];
      };
  return(result);
}

matrix:: break_edge(matrix& L, matrix& K, matrix C_V, int n1, int n2,
    double C, double tol)
{
  cout << "N" << C_V.p[n1][0] << " to N" << C_V.p[n2][0] << " BB cond ";
  cout << C << endl;
  assert(C > 0);
  K.p[I(C_V.p[n1][0])][I(C_V.p[n1][0])] += C;
  K.p[I(C_V.p[n2][0])][I(C_V.p[n2][0])] += C;
  K.p[I(C_V.p[n1][0])][I(C_V.p[n2][0])] += (-C);
  K.p[I(C_V.p[n2][0])][I(C_V.p[n1][0])] += (-C);
  L.p[n1][n1] += (-C);
  L.p[n2][n2] += (-C);
  L.p[n1][n2] += C;
  L.p[n2][n1] += C;
  L.clean(tol);
// If either node should become unconnected then it is washed to avoid
// possible roundoff error
  int i,j,k;
  j = 0;
  k = 0;
```

```
  for (i = 0; i < L.c_size; ++i)
    {
      if ( (abs(L.p[i][n1]) > tol) && (i!=n1) && (i!=n2) )
++j;
      if ( (abs(L.p[i][n2]) > tol) && (i!=n1) && (i!=n2) )
++k;
    };
  if (!j) L.wash(n1);
  if (!k) L.wash(n2);
}

matrix:: break_spike(matrix& L, matrix& K, matrix& C_V, int C_G[], int n,
     int geo[], int& z_num, double C, double tol)
{
  matrix D(0,0);
// First deal with the possibility of having to collapse one b_node
// onto another b_node.
  int i,j,k;
  double delta;
  j = 0;
  for (i = 0; i < L.c_size; ++i)
     if ( (abs(L.p[i][n]) > tol) && (i!=n) )
       {
 ++j;
 k = i;
       }
  assert(j > 0);
  if (j==1)
    {
      D.break_edge(L,K,C_V,n,k,C,tol);
// Since n was connected to just 1 node it should have all zeros.
// Thus it's washed.
      L.wash(n);
      L.r_swap(n,k);
      L.c_swap(n,k);
      C_V.r_swap(n,k);
// Now we deal with the medial lines and adjusting Lambda and C_V, etc.
      ++z_num;
      geo[C_G[n*2-1]]=z_num;
      geo[C_G[k*2+1]]=z_num;
      swap(C_G,n*2-1,k*2);
      rm_geo(C_G,k*2+1,L.c_size*2); // Need to do end before start
      rm_geo(C_G,k*2,L.c_size*2);
      L.rm_row(k);
      L.rm_col(k);
```

```
      C_V.rm_row(k);
    }
  else
    {
      matrix L2(0,0);
      L2 = L;
      L2 = D.cyclic_reorder(L2,n);
// Take Schur Complement with respect to [0][0]
      double E = -C;
      delta = L2.p[0][0] + E;
      matrix L3(0,0);
      L3 = L2;
      matrix A(1,L3.r_size-1);
      matrix B(L3.c_size-1,1);
      for (i = 0; i < A.r_size; ++i)
{
  A.p[0][i] = L3.p[0][i+1];
  B.p[i][0] = L3.p[i+1][0];
};
      matrix C2(L3.c_size-1,L3.r_size-1);
      for (i = 0; i < C2.c_size; ++i)
for (j = 0; j < C2.r_size; ++j)
  C2.p[i][j] = L3.p[i+1][j+1];
      matrix A2(0,0);
      A2 = A;
      A2.r_mult(0,(1/delta));
      C2 = D.subtract(C2,D.mult(B,A2));
      A.r_mult(0,(E/delta));
      B.c_mult(0,(E/delta));
      L3.p[0][0] = E - E*E/delta;
      for (i = 0; i < A.r_size; ++i)
{
  L3.p[0][i+1] = A.p[0][i];
  L3.p[i+1][0] = B.p[i][0];
};
      for (i = 0; i < C2.c_size; ++i)
for (j = 0; j < C2.r_size; ++j)
  L3.p[i+1][j+1] = C2.p[i][j];
      L2 = L3;
      L2 = D.cyclic_reorder(L2,(L2.c_size-n));
      L = L2;
// Adjust K and C_V matrices
      K.c_size++;
      K.r_size++;
      for (i = 0; i < K.c_size; ++i)
```

```
{
  K.p[i][K.r_size-1] = 0;
  K.p[K.c_size-1][i] = 0;
};
      K.p[K.c_size-1][K.r_size-1] = C;
      K.p[I(C_V.p[n][0])][I(C_V.p[n][0])] += C;
      K.p[K.c_size-1][I(C_V.p[n][0])] = -C;
      K.p[I(C_V.p[n][0])][K.r_size-1] = -C;
      C_V.p[n][0] = K.c_size - 1;
      cout << "N" << n << " to N" << C_V.p[n][0] << " Spike cond " << C << endl;
    };
  L.clean(tol);
}

matrix:: rm_row(int row)
{
  int i,j = 0;
  for (i = row; i < (c_size-1); ++i)
    for (j = 0; j < r_size; ++j)
      p[i][j] = p[i+1][j];
  --c_size;
}

matrix:: rm_col(int col)
{
  int i,j;
  for (i = col; i < (r_size-1); ++i)
    for (j = 0; j < c_size; ++j)
      p[j][i] = p[j][i+1];
  --r_size;
}

matrix:: rm_unconn_n(matrix& L, matrix& C_V, double tol, int& z_num,
     int C_G[], int geo[])
{
  int i,j,k = 0;
  int t_start,t_end;
  for (i = 0; i < L.c_size; ++i)
    {
      k = 0;
// For efficiency matrix is cleaned of roundoff error as unconnected
// nodes are removed.  Saves having an extra call to clean function
      for (j = 0; j < L.r_size; ++j)
if (abs(L.p[i][j]) >= tol)
  {++k;}
```

```
else L.p[i][j] = 0;
      if (k == 0)
{
  t_start = 2*i;
  t_end = 2*i+1;
  ++z_num;
  geo[C_G[t_start]] = z_num;
  geo[C_G[t_end]] = z_num;
  rm_geo(C_G,t_end,L.c_size*2); // Need to do end before start
  rm_geo(C_G,t_start,L.c_size*2);
  L.rm_row(i);
  L.rm_col(i);
  C_V.rm_row(i);
          --i;
}
    };
}

matrix:: recover(matrix& L, matrix& K, int& i_nodes, double frac, int o_edges)
{
// C_V is a correspondence vector relating the b-node in the lamda matrix
// as it gets modified to the node it corresponds to in the kirchoff
// matrix being constructed.
  int geo[L.c_size*2]; // geodesics
  int C_G[L.c_size*2]; // correspondence vector for geodesics
  int* z_seq;
  int t_start,t_end,z_num,temp;
  z_num = 0;
  double tol = frac;
  cout << "tol " << tol << endl << endl;
  int i,b_nodes,edges;
  b_nodes = L.c_size;
  matrix C_V(b_nodes,1);
  matrix D(0,0);
  double cond;
// Initialize the correspondence vectors
  for (i = 0; i < b_nodes; ++i)
    C_V.put(i,0,i);
  for (i = 0; i < (L.c_size*2); ++i)
    {
      C_G[i] = i;
      geo[i] = -1; // A nice flag to know if something went wrong
    }
  K = L;
  K.init(0);
```

```
  int bb_start, start, end, bb_end = -1;
  D.rm_unconn_n(L,C_V,tol,z_num,C_G,geo);
  edges = 0;
  while (L.c_size)
    {
      D.get_medial(L,bb_start,start,end,bb_end,tol);
      t_start = start*2 - bb_start;
      if (t_start<0)
t_start += L.c_size;
      t_end = end*2 + 1 + bb_end;
      if (t_end >= L.c_size*2)
t_end = t_end - L.c_size;
      ++z_num;
      geo[C_G[t_start]] = z_num;
      geo[C_G[t_end]] = z_num;
      while ( bb_start || bb_end || (start != end) )
{
  t_start = start*2 - bb_start;
  if (t_start<0)
    t_start += L.c_size;
  t_end = end*2 + 1 + bb_end;
  if (t_end >= L.c_size*2)
    t_end = t_end - L.c_size;
  if (bb_end)
    {
      cond = D.get_cond(L,bb_start,start,end,bb_end,tol);
      i = end + 1;
      if (i >= L.r_size)
i = i - L.r_size;
      D.break_edge(L,K,C_V,end,i,cond,tol);
      bb_end = 0;
      temp = t_end-1;
      if (temp < 0)
temp+=L.c_size;
      swap(C_G,t_end,temp);
    }
  else if (start != end)
    {
      cond = D.get_cond(L,bb_start,start,end,bb_end,tol);
      D.break_spike(L,K,C_V,C_G,end,geo,z_num,cond,tol);
      --end;
      if (end<0)
end += L.r_size;
      bb_end = 1;
      temp = t_end-1;
```

```
      if (temp < 0)
temp+=L.c_size;
      swap(C_G,t_end,temp);
    }
  else if (bb_start)
    {
      cond = D.get_cond(L,bb_start,start,end,bb_end,tol);
      i = start - 1;
      if (i < 0)
i += L.r_size;
      D.break_edge(L,K,C_V,start,i,cond,tol);
      bb_start = 0;
      temp = t_start+1;
      if (temp >= L.c_size*2)
temp = temp - L.c_size;
      swap(C_G,t_start,temp);
    };
  ++edges;
};
// The medial line has been removed in such a way that the node start
// must be an isolated node now.  Thus, start is removed.
      t_start = start*2;
      if (t_start<0)
t_start += L.c_size;
      t_end = start*2 + 1;
      if (t_end >= L.c_size*2)
t_end = t_end - L.c_size;
      rm_geo(C_G,t_end,L.c_size*2); // Need to do end before start
      rm_geo(C_G,t_start,L.c_size*2);
      L.rm_row(start);
      L.rm_col(start);
      C_V.rm_row(start);
      D.rm_unconn_n(L,C_V,tol,z_num,C_G,geo);
    };
  i_nodes = K.c_size - b_nodes;
  L = D.kirk_to_lamda(K,b_nodes,i_nodes);
  cout << "\n# of edges in derived graph: " << edges << endl;
  if (o_edges > 0)
    {
      cout << "\n# of edges in inputted kirchoff matrix: " << o_edges << endl;
      if (o_edges == edges)
{cout << "\nInputted kirchoff matrix was critical" << endl;}
      else cout << "\nInputted kirkchoff matrix was non-critical" << endl;
    };
// Adjust the numbering of geodesics for z-seq
```

```
  arrange(geo,b_nodes*2);
  cout << "\nZ-seq: ";
  print_array(geo,L.c_size*2);
  cout << endl;
}

matrix:: square_lat(int n, double C)
{
  int i,j,k;
  r_size = n*(n+4);
  c_size = r_size;
  init(0);
// Set the diagonal
  for (i = 0; i < r_size; ++i)
    {
      if (i < (4*n))
{p[i][i] = C;}
      else p[i][i] = (4*C);
    };
// Set connections
  for (i = 0; i < (n+4); ++i)
    {
      for (j = 0; j < n; ++j)
{
  switch (i)
    {
    case 0:
      {p[i*n+j][(i+4)*n+j] = -C;
       p[(i+4)*n+j][i*n+j] = -C;
       break;}
    case 1:
      {p[i*n+j][(j+5)*n-1] = -C;
       p[(j+5)*n-1][i*n+j] = -C;
       break;}
    case 2:
      {p[i*n+j][(n+4)*n-1-j] = -C;
       p[(n+4)*n-1-j][i*n+j] = -C;
       break;}
    case 3:
      {p[i*n+j][(n+3-j)*n] = -C;
       p[(n+3-j)*n][i*n+j] = -C;
       break;}
    default:
      {if (i < (n+3))
 {p[i*n+j][(i+1)*n+j] = -C;
```

```
  p[(i+1)*n+j][i*n+j] = -C;};
       if (j < (n-1))
 {p[i*n+j][i*n+j+1] = -C;
  p[i*n+j+1][i*n+j] = -C;};}
    }
}
    }
}

matrix:: circle(int spikes, int circles, double C)
{
  int i,j,k;
  c_size = spikes*circles + 1;
  if (circles==0)
    c_size += spikes;
  r_size = c_size;
  init(0);
// Set the diagonal
  for (i = 0; i < spikes; ++i)
    if (circles>0)
      {p[i][i] = 3*C;}
    else p[i][i] = C;
  for (i = spikes; i < (c_size-1); ++i)
    p[i][i] = 4*C;
  p[c_size-1][r_size-1] = spikes*C;
// Set connections
  for (i = 0; i < (circles-1); ++i)
    for (j = 0; j < spikes; ++j)
      {
// Connect circle to next most inner circle
p[i*spikes+j][(i+1)*spikes+j] = -C;
p[(i+1)*spikes+j][i*spikes+j] = -C;
// Connect points on a circle to each other
if (j<(spikes-1))
  {p[i*spikes+j][i*spikes+j+1] = -C;
   p[i*spikes+j+1][i*spikes+j] = -C;}
else
  {p[i*spikes+j][i*spikes] = -C;
   p[i*spikes][i*spikes+j] = -C;};
      };
// Deal with the inner most circle
  for (j = 0; j < spikes; ++j)
      {
// Connect points on innermost circle to each other
if (circles)
```

```
  if (j<(spikes-1))
    {p[(circles-1)*spikes+j][(circles-1)*spikes+j+1] = -C;
     p[(circles-1)*spikes+j+1][(circles-1)*spikes+j] = -C;}
  else
    {p[(circles-1)*spikes+j][(circles-1)*spikes] = -C;
     p[(circles-1)*spikes][(circles-1)*spikes+j] = -C;};
// Connect circle to the most inner node
if (circles)
  {p[(circles-1)*spikes+j][r_size-1] = -C;
   p[r_size-1][(circles-1)*spikes+j] = -C;}
else
  {p[j][r_size-1] = -C;
   p[r_size-1][j] = -C;};
      };
}

double matrix:: min()
{
  int i,j;
  double small;
  small = p[0][0];
  for (i = 0; i < c_size; ++i)
    for (j = 0; j < r_size; ++j)
      if ((abs(p[i][j])<abs(small)) && (p[i][j] != 0))
small = p[i][j];
  return(abs(small));
}

matrix:: get_geodesic(matrix L, int geo[], int& t_start, int& t_end,
      int level, double tol)
{
  int i,j,k,row,col,bb_start,start,end,bb_end;
  assert((L.c_size == L.r_size) && (L.r_size != 0));
// -1 used as a flag until initialized
  bb_start = 0;
  start = 0;
  end = 0;
  bb_end = 0;
  int found = 0;
  matrix D(0,0);
  for (i = 0; i < half(L.c_size); ++i)
    {
      D.c_size = L.c_size-(i+1);
      D.r_size = i+1;
      for (j = i+1; j < L.c_size; ++j)
```

```
for (k = 0; k < i+1; ++k)
  D.p[j-(i+1)][k] = L.p[j][k];

//For debugging purposses only.
//D.print_matrix(5);
//cout << "rank " << D.rank(D,tol);

      end = i;
      bb_end = 0;
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
// cout << t_start << " " << t_end << " ";
// cout << g_between(geo,t_start,t_end) << endl;
      if ( (level == g_between(geo,t_start,t_end)+1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note end = position in L matrix. So node 1 would be 0.
  end = i;
  bb_end = 0;
  found = 1;
  break;
};
// Just 'cuz you don't want to go past half way around graph
//      if ( ((2*half(L.c_size)-L.c_size)+i) < half(L.c_size) )
// {
// Perhaps I should use --D.c_size
  D.c_size = L.c_size-(i+2);
  D.r_size = i+1;
  for (j = i+2; j < L.c_size; ++j)
    for (k = 0; k < i+1; ++k)
      D.p[j-(i+2)][k] = L.p[j][k];
  bb_end = 1;
  get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
  if ( (level == g_between(geo,t_start,t_end)+1)
      && (D.rank(D,tol) == D.r_size-level) )
    {
// Note end = position in L matrix. So node 1 would be 0.
      end = i;
      bb_end = 1;
      found = 1;
      break;
    }
// }
    }
// Make sure you caught the end of a geodesic also
  int happy = 0;
```

```
  for (i = end;( (i > -2) && (found==1) ); --i)
    {

      D.c_size = L.c_size - (end - i + 1) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
    D.p[j][k] = L.p[row][col];
  };
      start = i;
      bb_start = 0;
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
// cout << t_start << " " << t_end << " "
// cout << g_between(geo,t_start,t_end) << endl;

      if ( (level == g_between(geo,t_start,t_end) +1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
    start = start + L.r_size;
  bb_start = 0;
  ++happy;
  break;
}
// Perhaps I should use --D.c_size
      D.c_size = L.c_size - (end - i + 2) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
```

```
    D.p[j][k] = L.p[row][col];
  };
      bb_start = 1;
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
      if ( (level == g_between(geo,t_start,t_end)+1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
    start = start + L.r_size;
  bb_start = 1;
  ++happy;
  break;
}
    }
// cout << "found " << found << " happy " << happy << endl;

  if (happy>0)
    {get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);}
  else
    {t_start = -1;
     t_end = -1;};
}

matrix:: get_geodesic2(matrix L, int geo[], int& t_start, int& t_end,
      int level, double tol)
{
  int i,j,k,row,col,bb_start,start,end,bb_end;
  assert((L.c_size == L.r_size) && (L.r_size != 0));
// -1 used as a flag until initialized
  bb_start = 1;
  start = 0;
  end = 0;
  bb_end = 0;
  int found = 0;
  matrix D(0,0);
  for (i = 0; i < half(L.c_size); ++i)
    {
      D.c_size = L.c_size-(i+1)-1;
      D.r_size = i+1;
      for (j = i+1; j < L.c_size; ++j)
for (k = 0; k < i+1; ++k)
  D.p[j-(i+1)][k] = L.p[j][k];
```

```
//For debugging purposses only.
//D.print_matrix(5);
//cout << "rank " << D.rank(D,tol);

      end = i;
      bb_end = 0;
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
// cout << t_start << " " << t_end << " ";
// cout << g_between(geo,t_start,t_end) << endl;
      if ( (level == g_between(geo,t_start,t_end)+1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note end = position in L matrix. So node 1 would be 0.
  end = i;
  bb_end = 0;
  found = 1;
  break;
};
// Just 'cuz you don't want to go past half way around graph
//      if ( ((2*half(L.c_size)-L.c_size)+i) < half(L.c_size) )
// {
// Perhaps I should use --D.c_size
  D.c_size = L.c_size-(i+2)-1;
  D.r_size = i+1;
  for (j = i+2; j < L.c_size; ++j)
    for (k = 0; k < i+1; ++k)
      D.p[j-(i+2)][k] = L.p[j][k];
  bb_end = 1;
  get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
  if ( (level == g_between(geo,t_start,t_end)+1)
      && (D.rank(D,tol) == D.r_size-level) )
    {
// Note end = position in L matrix. So node 1 would be 0.
      end = i;
      bb_end = 1;
      found = 1;
      break;
    }
// }
    }
// Make sure you caught the end of a geodesic also
  int happy = 0;
  for (i = end;( (i > -2) && (found==1) ); --i)
    {
```

```
      D.c_size = L.c_size - (end - i + 1) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
    D.p[j][k] = L.p[row][col];
  };
      start = i;
      bb_start = 0;
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
// cout << t_start << " " << t_end << " "
// cout << g_between(geo,t_start,t_end) << endl;

      if ( (level == g_between(geo,t_start,t_end) +1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
    start = start + L.r_size;
  bb_start = 0;
  ++happy;
  break;
}
// Perhaps I should use --D.c_size
      D.c_size = L.c_size - (end - i + 2) - bb_end;
      D.r_size = end - i + 1;
      for (j = 0; j < D.c_size; ++j)
for (k = 0; k < D.r_size; ++k)
  {
    row = j + end + 1 + bb_end;
    if (row >= L.c_size)
      row = row - L.c_size;
    col = end - k;
    if (col < 0)
      col = col + L.r_size;
    D.p[j][k] = L.p[row][col];
  };
      bb_start = 1;
```

```
      get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);
      if ( (level == g_between(geo,t_start,t_end)+1)
    && (D.rank(D,tol) == D.r_size-level) )
{
// Note start = position in L matrix. So node 1 would be 0.
  start = i;
  if (start < 0)
    start = start + L.r_size;
  bb_start = 1;
  ++happy;
  break;
}
    }
// cout << "found " << found << " happy " << happy << endl;

  if (happy>0)
    {get_t(t_start,t_end,bb_start,start,end,bb_end,L.c_size);}
  else
    {t_start = -1;
     t_end = -1;};
}


matrix:: find_zseq(matrix L, double tol)
{
  assert(L.c_size == L.r_size) ;
  matrix D(0,0);
  int geo[L.c_size*2]; // geodesics
  int i,j,k,level,t_start,t_end,edges,bb_start,start,end,bb_end,z_num;
  edges = 0;
  z_num = 0;
  level = 0;
  for (i = 0; i<(L.c_size*2); ++i)
    geo[i] = -1;
  while (t_left(geo,L.c_size*2)>2)
    {
      ++level;
      for (i = 0; i < L.c_size; ++i)
{
  L = D.cyclic_reorder(L,1);

  reorder(geo,2,L.c_size*2);
// print_array(geo,L.c_size*2);
// cout << "level: " << level << " znum: " << z_num << endl;
  D.get_geodesic(L,geo,t_start,t_end,level,tol);
```

```
  if ( (t_start!=-1) && (t_end!=-1) )
    {
//       assert( ((geo[t_end] == -1) && (geo[t_start] == -1)) ||
//      ((geo[t_end] != -1) && (geo[t_start] != -1)) );
      if ((geo[t_end] == -1) && (geo[t_start] == -1) )
 // If you found a new geodesic
{
// Record geodesic
  ++z_num;
  geo[t_start] = z_num;
  geo[t_end] = z_num;
// I think you know the # of edges created by each step
  j = t_end - t_start;
  if (j < 0)
    j += L.c_size*2;
  edges = edges + i - level;
};
    };

  D.get_geodesic2(L,geo,t_start,t_end,level,tol);
  if ( (t_start!=-1) && (t_end!=-1) )
    {
//       assert( ((geo[t_end] == -1) && (geo[t_start] == -1)) ||
//      ((geo[t_end] != -1) && (geo[t_start] != -1)) );
      if ((geo[t_end] == -1) && (geo[t_start] == -1) )
 // If you found a new geodesic
{
// Record geodesic
  ++z_num;
  geo[t_start] = z_num;
  geo[t_end] = z_num;
// I think you know the # of edges created by each step
  j = t_end - t_start;
  if (j < 0)
    j += L.c_size*2;
  edges = edges + i - level;
};
    };


};
    };
// Adjust the numbering of geodesics for z-seq
  arrange(geo,L.c_size*2);
  cout << "\nZ-seq: ";
```

```
  print_array(geo,L.c_size*2);
  cout << endl;
}

int matrix:: count_edges()
{
  int i,j,sum;
  sum = 0;
  for (i = 0; i < c_size; ++i)
    for (j = i; j < r_size; ++j) // only looks at lower triangle
      if (p[i][j] < 0)
++sum;
  return(sum);
}

double matrix:: maxdiff(matrix LO, matrix LD)
{
  int i,j,k;
  double diff,temp;
  diff = 0;
  assert( (LO.c_size == LD.c_size) && (LO.r_size == LD.r_size) );
  for (i=0; i < LO.c_size; ++i)
    for (j = 0; j < LO.r_size; ++j)
      {
temp = abs( (LO.p[i][j] - LD.p[i][j])/LO.p[i][j] );
if ( temp > diff ) diff = temp;
      };
  return(diff);
}
```

13.7. **graph.h.**

```
/*
 * Author: Derek A. Jerina
 * Date: July 21, 1996
 *
 * Header file for graph class.
 *
 */

#include "matrix.h"

class graph{
 public:
  int b_nodes,int_nodes,K_dat,L_dat;
  matrix K_mat,L_mat;
```

```
  graph (int b_n,int i_n, int K_d, int L_d,const matrix K,const matrix L);
};
```

13.8. **graph.cc.**

```
/*
 * Author: Derek A. Jerina
 * Date: July 21, 1996
 *
 * Implementation for graph class
 *
 */

#include "graph.h"

graph:: graph(int b_n, int i_n, int K_d, int L_d,
      const matrix K, const matrix L):
              K_mat(K), L_mat(L)
{
  b_nodes = b_n;
  int_nodes = i_n;
  K_dat = K_d;
  L_dat = L_d;
};
```

13.9. **shape.cc.**

```
/*
 * Author: Derek A. Jerina
 * Date: July 20, 1996
 *
 * This program is designed to calculate the shape and conductivities
 * of a critical circular planar resistor network from its Lamba matrix.
 *
 */

#include <iostream.h>
#include <assert.h>
#include "graph.h"

#define frac 0.000001

void get_info(int& User_dat, int& b_nodes, int& i_n, int& K_dat,
      int& L_dat, int& SQ_Lat, int& C_Lat, double& C)
{
// Data must have format of B-nodes, int-nodes, is there a Kirchoff matrix
// is there a Lamda matrix (1 means true and 0 means false)
```

```
// If one inputs both a Kirkoff and a Lamba matrix the Kirchoff should
// come first.  In a Kirchoff matrix it is assumed that the first
// rows and cols are filled by all the data for b-nodes before
// any int-node data comes.

  cin >> User_dat;
  if (User_dat)
    {cin >> b_nodes >> i_n >> K_dat >> L_dat;}
  else
// If you use computer generated K_mat uses b_n for vertices per
// side in square matrix and i_n is # of circles
    {
      cin >> SQ_Lat >> C_Lat;
      if (SQ_Lat)
{
  cin >> b_nodes >> C;
}
      else
{
  if (C_Lat)
   cin >> b_nodes >> i_n >> C;
};
    };
}

main()
{
  cout << "\nWritten by Derek A. Jerina";
  cout << "\nBased on paper by Derek A. Jerina, REU Summer 1996";
  cout << "\nProblems do have solutions you know...\n";

  matrix D(max_nodes,max_nodes);
  int User_dat=0, b_nodes=0, i_nodes=0, K_dat=0, L_dat=0;
  int SQ_Lat=0, C_Lat=0, edges = 0;
  double C=0;
  get_info(User_dat,b_nodes,i_nodes,K_dat,L_dat,SQ_Lat,C_Lat,C);

// Kirk will be good for either since square_lat and circle will
// set the size of the kirchoff matrix
  matrix Kirk(b_nodes+i_nodes,b_nodes+i_nodes);
// SQ_Lat has b_nodes on each of 4 sides
  if (SQ_Lat)
    {
      Kirk.square_lat(b_nodes,C);
      i_nodes = b_nodes*b_nodes;
```

```
      b_nodes = b_nodes*4;
    };
  if (C_Lat)
    {
      Kirk.circle(b_nodes,i_nodes,C);
      i_nodes = Kirk.rows()-b_nodes;
    };
  matrix Lamda(b_nodes,b_nodes);
  graph original(b_nodes, i_nodes, K_dat, L_dat, Kirk, Lamda);
//  if (SQ_Lat || C_Lat)
//    original.K_mat = Kirk;

  if (User_dat)
    {
// Must either input a kirkoff matrix or a lamda matrix or both with
// kirkoff first.
      assert(K_dat || L_dat);

      if (K_dat)
{ original.K_mat.get_matrix();
}
      else { original.K_mat.init(0);
   };
      if (L_dat)
{ original.L_mat.get_matrix();
}
      else
{ original.L_mat = D.kirk_to_lamda(original.K_mat,b_nodes,i_nodes);
};
    };
  if (SQ_Lat)
    {
      original.K_mat = Kirk;
      original.L_mat = D.kirk_to_lamda(original.K_mat,b_nodes,i_nodes);
      K_dat = 1;
      original.K_dat = 1;
    };
  if (C_Lat)
    {
      original.K_mat = Kirk;
      original.L_mat = D.kirk_to_lamda(original.K_mat,b_nodes,i_nodes);
      K_dat = 1;
      original.K_dat = 1;
    };
```

```
// Need some data to work with
  assert(User_dat || SQ_Lat || C_Lat);

  int temp;
  cin >> temp;
  if (temp && K_dat)
    {
      cout << "\nOriginal Kmat\n";
      original.K_mat.print_matrix(5);
    };
  cin >> temp;
  if (temp)
    {
      cout << "\nOriginal Lmat\n";
      original.L_mat.print_matrix(5);
    };

  cin >> temp;
  if (temp)
    {
      cout << endl;
      matrix K2(0,0);
      matrix L2(0,0);
      graph derived(b_nodes, 0, 0, 1, K2, L2);
      derived.L_mat = original.L_mat;
      derived.K_mat = original.L_mat;
      derived.K_mat.init(0);
      edges = -1; // Flag in case no original k_dat
      if (original.K_dat)
edges = original.K_mat.count_edges();
      D.recover(derived.L_mat,derived.K_mat,derived.int_nodes,frac,edges);
      int temp2;
      derived.K_dat = derived.L_dat = 1;
      cin >> temp2;
      if (temp2)
{
  cout << "\nThe derived Kmat\n";
  derived.K_mat.print_matrix(5);
};
      cin >> temp2;
      if (temp2)
{
  cout << "\nThe derived Lmat\n";
  derived.L_mat.print_matrix(5);
};
```

```
      cout << "\nThe greatest difference between any two entries of original";
      cout << "\nLambda matrix and the derived Lambda matrix was ";
      cout << D.maxdiff(original.L_mat,derived.L_mat)*100 << "%\n";
    } else cin >> temp >> temp;
  cin >> temp;
  if (temp)
    {
      cout << endl << "Attempting to Obtain Z-seq without recovery." << endl;
      D.find_zseq(original.L_mat,frac);
    };
}
```

References

[1] E. Curtis, D. Ingerman, J. Morrow, *Circular planar graphs and resistor networks*, submitted.

E-mail address: daj2@acpub.duke.edu