Recovering the Conductivity of a Resistor Network from the Dirichlet-to-Neumann Map when there is a limited set of possible Resistors

S H Shepard IV

14th August 1997

1 Introduction

Resistor networks are not usually to be found in computing systems or most machinery, but they are still interesting for their mathematical properties and other possible practical applications. This project, in particular, could be easily applied to situations where the conductivities of the objects being dealt with are known, and one wants to find an object whose conductivity differs from its background. Examples of this type of problem are mine/ore detection and medical imaging. In mine detection, the ground is assumed to have a different conductivity (very low) compared to the mine. If a ring of nodes are placed in the ground, then the ground becomes a resistor network whose conductivity is being recovered. The problem of medical imaging involves a way of taking internal pictures of the body with a less invasive method than an X-ray. A band of nodes are placed around a person and then a small current is directed through the subject's body. The currents and voltages at the nodes are known as are the conductivities of blood, bone, muscle, etc. thereby allowing an internal "picture" of the body to be found when the system is recovered. This project is far away from those dreams, but the methods developed here could eventually be applied to more complicated networks.

A resistor network consists of a set of nodes N and a corresponding set of edges E that connects pairs of nodes in N. The set of nodes is divided into two subsets: the boundary nodes ∂N and the interior nodes $int\ N$.

Two nodes are considered adjacent if there is an edge between the two of them. A particular type of resistor network is a rectangular resistor network which can be constructed on a lattice by choosing the nodes to be $p = (i, j)$ where $a \leq i \leq b$ and $c \leq j \leq d$ for $a < b$ and $c < d$ with the four corners $(a, c), (a, d), (b, c)$, and (b, d) removed. A node is a member of the boundary of this network if it is adjacent to only one other node. If a node is adjacent to four nodes, it is an element of the interior.

For each edge σ in E for a given resistor network, a function $\gamma : E \to R^+$, with $\gamma(\sigma)$ being called the *conductance* of σ, is defined. (Note that $1/\gamma(\sigma)$ is the *resistance* of σ.) The function γ is called the *conductivity* of the network. For any function $f : N \to R$, a function $L_\gamma f : int\ N \to R$ is defined to be

$$L_\gamma f(p) = \sum_{q \in \mathcal{N}(p)} \gamma(pq)(f(q) - f(p))$$

where $\mathcal{N}(p)$ is the set of edges adjacent to p. A function f is γ-*harmonic* if it satisfies $L_\gamma f(p) = 0$ for all $p \in int\ N$. A γ-*harmonic* function follows *Kirchhoff's Law*. A voltage $\phi(r)$ at each boundary node r of the network will determine a unique $f(p)$ at each interior node p such that f is γ-*harmonic*. This voltage function ϕ determines a current $I_\phi(r)$ at each boundary node r where $I_\phi(r) = \sum_{q \in \mathcal{N}(r)} \gamma(qr)(f(q) - f(r))$ with q being r's neighbors. Therefore, there is a relationship between the voltage ϕ and the current I_ϕ on the boundary with either function determining the voltage f at the interior nodes. The *Dirichlet to Neumann map* Λ_γ maps ϕ, the boundary value function, to the current function I_ϕ. The name is derived from the fact that I_ϕ is the solution to the Dirichlet problem with boundary values ϕ and ϕ is the solution of the Neumann problem with conditions I_ϕ. It will be shown that by taking measurements at the boundary, γ can be determined for a particular network.

The *Dirichlet to Neumann* map Λ_γ can be constructed from the Kirchhoff matrix of a network. The Kirchhoff matrix K is created in the following manner: entry j, j is the sum of the conductivities of all the edges that have node j as an end; entry i, j (where $i \neq j$) is the negative of the sum of all conductivities for the edges with endpoints i and j. Thus the Kirchhoff matrix is symmetrical. If the nodes are ordered such that the first bN are the boundary nodes, then it is quite simple to calculate Λ_γ. The Λ matrix is the Schur complement of the Kirchhoff matrix with respect to C, where C is the matrix that consists of the the last rows and columns of the Kirchhoff matrix which correspond to the interior nodes.

$$K = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$$

$$\Lambda = A - BC^{-1}B^T$$

Given a Λ matrix for a rectangular resistor network, the Curtis-Morrow algorithm can be used to recover the conductivity. The algorithm is based on the fact that by setting the voltages and the currents at the appropriate boundary nodes equal to 0, one can use the Dirichlet to Neumann map to find the voltages and currents for the rest of the boundary. Once the voltages and currents are known on the boundary, the inverse problem can be solved to find the interior voltages. As can be seen in figure 1, the Curtis-Morrow algorithm starts in one corner by setting the voltage at $P1$ equal to 1, the voltage at all other boundary nodes besides $P1$ and $Q1$ equal to 0, and the current at exactly one of these other boundary nodes equal to 0. This forces all the interior nodes below the first diagonal to be 0, and it allows one to solve for the voltage at $Q1$. Using Kirchhoff's law, the values of the conductors that connect $P1$ and $Q1$ to the interior can be found. This process is continued in the next step by setting $P2$ equal to 1, all the boundary nodes besides $P2, Q1$, and $Q2$ equal to 0, and then setting the current at two of these other nodes (not both in the same corner) equal to 0. Using the Dirichlet to Neumann map, $Q1$ and $Q2$ can be recovered and then since all the interior nodes below the second diagonal will have voltage equal to 0, the inverse problem needs to be solved just to find the value for the three interior nodes in the upper right hand corner. This process can be continued with $Q3$ and so on for as a network as necessary.

2 Rational Recovery of Conductivity

The first step in investigating this problem is to implement the Curtis-Morrow algorithm. The computer language of choice was Maple. Two programs needed to be designed in order to construct and test the algorithm: one program to create the Λ_γ matrix and another to use the algorithm to recover its conductivity. The first Maple program that was designed (Appendix 6.2 gamma.ms) automatically took advantage of Maple's rational arithmetic since the program did not designate the use of floating point calculations. The Λ matrix that was saved therefore had fractional entries, and when it

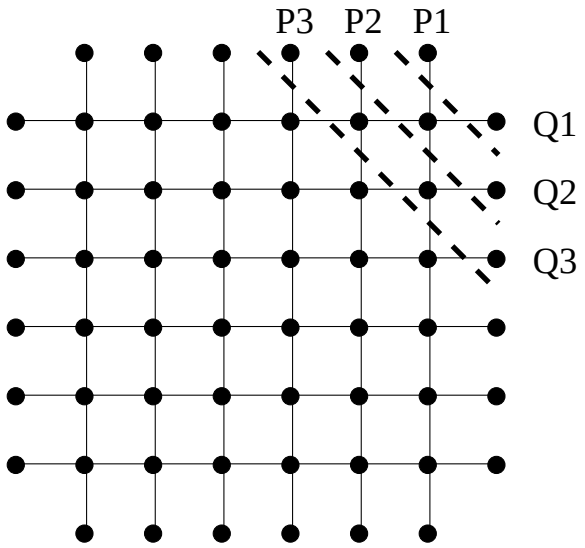

Figure 1: The Curtis-Morrow Algorithm

was recovered by the programmed algorithm, rational arithmetic was also used. The results of this situation were that the conductivities of the networks were recovered exactly. In theory, the algorithm should recover the conductivity exactly since with rational arithmetic all the operations should be performed without error. This program, however, was the first time in the REU program at Washington that rational arithmetic was used, and thus was the first time that the recoveries of networks as large as 20 by 20 or 23 by 23 were done exactly. The program was therefore a great step in the ability to recover a resistor network, but at first glance it made the original problem of recovering a resistor network when there are only a few possibilities for conductors obsolete. If a network can be covered exactly without taking advantage of this information, then knowing that there are only a couple of different resistor values is not necessary or helpful. However, there are three very important problems with using rational calculations.

2.1 Time

As the Maple help manual states, rational notation is anywhere from 50 to 100 times slower than floating point calculations. This fact made itself evident in the construction of the Λ matrix. The first version of the program

that constructed the Λ matrix made the time consuming demand of inverting a rather large matrix to take the Schur complement. This was changed in later versions to solving a system of linear equations which took considerably less time. Even with this advancement, the use of rational arithmetic was considered too time consuming, especially when compared to floating point calculations. Below is a table of how long the program took to run for different networks when using rational arithmetic.

Network	*seconds*	*minutes (approx)*
18 by 18	1656.5	27 1/2
20 by 20	3157.9	52 1/2
23 by 23	11359.3	189 2/3

The exponential growth of the amount of time needed shows that if the networks became much larger than the 20 by 20 or 23 by 23 used in these experiments, the time needed to create the Λ matrix would become unreasonable. If floating point numbers are used, the time required to calculate the Λ matrix from a network would be reduced.

2.2 Memory

The amount of memory that is needed to store the file that contains the Λ matrix is also a detriment. As the size of the resistor network (and hence the Λ matrix grows) so also does the size of the entries of the Λ matrix. Here is an example of a single entry from the Λ matrix for a 20 by 20 network which had the simplest pattern of having all its resistors as 1:

$$223837749837805528994075143401140847195137395678439 5\backslash$$
$$4168293815691141030913759181 1/32084303784525017\backslash$$
$$8551244431677833659732898412843132273457312117 5\backslash$$
$$16165542409086640$$

This fraction has an 81 digit numerator and an 81 digit denominator, and therefore is quite costly to store and manipulate. If there was a way to insure the same results while using floating point notation with fewer significant digits, then a considerable amount of energy could be conserved.

2.3 Error

The biggest problem that cannot be eliminated by using this method is the problem of error, or in other words, accuracy. In this artificial environment where we are able to use a program such as Maple to calculate a Λ matrix to almost as many digits as we like, the problem of error is not as important, but the hope for the program is that it can take any reasonable approximation of a Λ matrix and still be able to recovery the conductivities. Even the slightest error in the Λ matrix results in a matrix which can not be the Dirichlet to Neumann map for any network. The program as it was designed at this point could not recover the conductivity of a network unless the Λ matrix was exactly correct. The aim of this project is to find a program that could overcome this flaw, and to recover a network correctly with the minimum accuracy possible for the Λ matrix.

3 Floating Point Calculations

The program as it was originally designed was easily augmented to use floating point notation instead of rational arithmetic. Once this change was made, the difference in the amount of error that occurred was dramatic. The first experiments in recovery were done with Maple's default of 10 significant digits on a 10 by 10 rectangular network with all conductivities 1. The errors became so prevalent that many of the values of the conductors were found to be negative, and some values were found to be as large as several hundred when they were supposed to be 1. Therefore, there are two things to consider when using floating point arithmetic. The first is the number of significant digits called upon in the calculations. Answers will vary substantially depending on how many significant digits are used (as one can intuitively understand). One interesting occurrence was that more digits did not guarantee more accurate results. Accuracy in this case was quantified as the maximum error that the calculated result differed from the actual value of the conductor. An example of this phenomenon is given below for the case of a 10 by 10 rectangular network whose conductivity is all 1's. The recovery program used 30 significant digits to recover various Λ matrices of this network that were constructed with anywhere from 5 to 20 significant digits.

Digits	*Maximum Error*
3	390.188493458717814626090491292
4	57.109436264511853363640061 3614
5	29.267022389373244660186335 8754
6	115.390147311184771163773406783
7	20.565349801640398321933763 9852
8	41.326075504082287274968498 8215
9	155.263009317946843818363150353
10	14.601742596932419597020470 2193

As the table shows, an increased number of significant digits did not guarantee a smaller maximum error, eg. 8 digits was not better than 7. While the definition used here to judge accuracy may not be the best, it is still an interesting situation that the maximum error did not improve.

The second fact that one should consider about this situation could lead the reader to not trust me about the first. While the table above was true for one of the executions of my recovery program, it would be hard to duplicate these exact results since the program does not always give the same output for the same set of data: a variation occurs during the recovery process. While I have not found any answers that satisfy my curiosity about this, a couple of reasons for this error have been conjectured. One explanation could be error that occurs during converting from decimal to binary and back to decimal. A second possibility involves Maple's software directly. No information could be gathered on how Maple implements changing it's significant digits. It is quite possible that Maple has a built in function that decides to round slightly differently each time it is called in order to not show a bias in its calculations. Without further knowledge of the Maple software, it is impossible to tell exactly how the program can use more significant digits than a normal computer CPU can handle. All the operations that are used in the program are simple operations except for the initial solving of a linear system. While it is the case that this initial solution does have some variance (at least on occasion), this variation can be removed and cannot be solely responsible for these erratic results. The deviation that occurs is almost strictly in the first digit, but usually not in magnitude. However, the case usually appears to be that the more significant digits that are used, the less error in the recovery, even with this variation. The following table will illustrate some examples of this variation in the maximum error of a 10 by 10 network for different significant digits in the recovery.

Significant Digits	*Maximum Error*
10	713.6573855
	77.12020260
	41.29722037
	7.46661109
20	$.2625554996\,10^{-9}$
	$.4308085707\,10^{-9}$
	$.3688829860\,10^{-9}$
15	.00008426889572
	.00005132124299
	.000034367381474
5	27.464
	274.00
	167.67

This variation is a bothersome problem, especially with the discussion of setting an epsilon. One approach to this problem is to run the program several times using some criteria to decide which results are the best. A variation on this approach could be to take a weighted average of the output if a reason can be devised to properly motivate it. A second approach would be to just trust the magnitude of the answers that are produced and use only that information. The rest of the project was done with this problem kept in the background in order to maintain the main thrust of the project.

4 Recovery with a Limited set of Resistors

4.1 Epsilon Method

The first step in using the fact that there are a known set of different possible values for the conductors is to find a way to modify the Curtis-Morrow algorithm to take advantage of this information. The first plan that I thought of was to use some sub-procedure to compare the resistor value calculated by the algorithm to members of the set of possible values. The question then became what was the best way to do this comparison. The natural answer was to create what I term an epsilon vector with an entry corresponding to each member of the set. If the calculated conductor was within epsilon of a corresponding member in the limited set, then the value for that conductor would be chosen as that member in the set. This method for determining

what value in the set a conductor is raises several questions. Is this the best method for determining which value is the true conductor? This was the method that occurred intuitively to me, but no proof has been devised to show it is the best one. Another method might implement a ratio test to determine what value the conductor should be. What are the best choices for epsilon? The values for epsilon were hard coded into the program with the usual starting values being 1 and 9 if the conductors being recovered had values 1 and 10 or 9 and 90 if the values were 1 and 100. The methods used here make sure that the epsilon values do not overlap, but that is not necessary as long as that scenario is considered properly. A bias toward one value may be programmed in, but that will be discussed further in the next section. The major problem with using the epsilon method is overcoming the problem of having a network where the correct resistors cannot be recovered by this method. This problem occurs when the calculated value of a smaller conductor is larger than the value calculated for a larger conductor. In this situation the epsilons can never be set in such a way that the correct recovery is made. The only way this problem could be overcome is with a different method.

4.2 Background vs. All

Another interesting consideration occurred during the experiments done with this program. There are two different approaches that can be taken when recovering the resistors with the epsilon method: only recovering a resistor when it is within epsilon of one of the possible values (All) or setting the value of the resistor to the most common conductivity when it is not within epsilon of any value in the set (Background). The reason this question becomes of importance is because often some conductors will not be recovered as any value because of the problem described in the previous paragraph with the epsilon method. When the program is designed to use All, the program fails when this incident occurs. The problem with the Background approach is that when the calculated value does not fall into any range, the value chosen for the conductor is not much better than a guess. Now, if the added information is given that one conductor value occurs most often, then this approach is not as haphazard. This same information could also be gathered by creating a loop that keeps track of how many times each conductor appears during the recovery, but while this method does not use any added information it is less reliable. From my experience, All should be

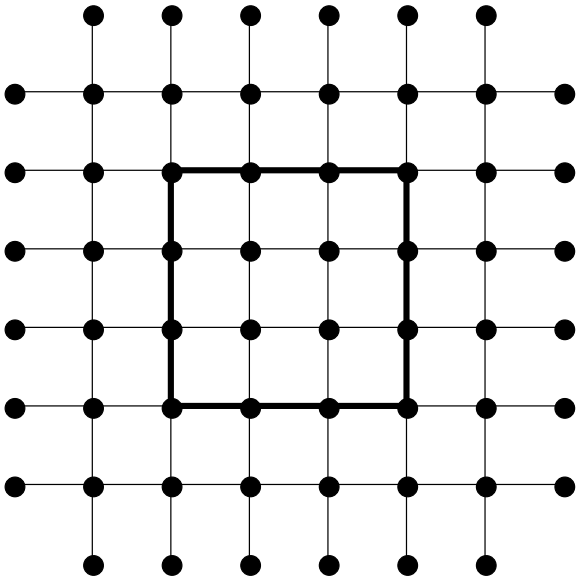

Figure 2: 6 by 6 network with square

used if possible when there is a variety of resistor values or when one resistor does not dominate. The Background approach is most helpful when there are only a few resistors different from the majority that are in the network.

4.3 Experiments

A variety of experiments were performed using the programs listed in the appendix of this paper (or with slight modifications). The dimension of the network, the placement of the resistors, the number of digits used in the recovery and construction of the Λ matrix were a few of the aspects that were varied during these experiments. Some examples of the networks that were studied appear in these figures (with bolder lines being resistors different from the background): a 6 by 6 network with a 3 by 3 square of different resistors, a 14 by 14 network with an internal diamond of different resistors, and a 10 by 10 network with a solid 3 by 3 square of different resistors in the upper left-hand corner.

All these problems could be recovered for a large enough number of significant digits and the appropriate epsilons, but all of them also had significant digits for which a correct solution could not be found for the conductivity.

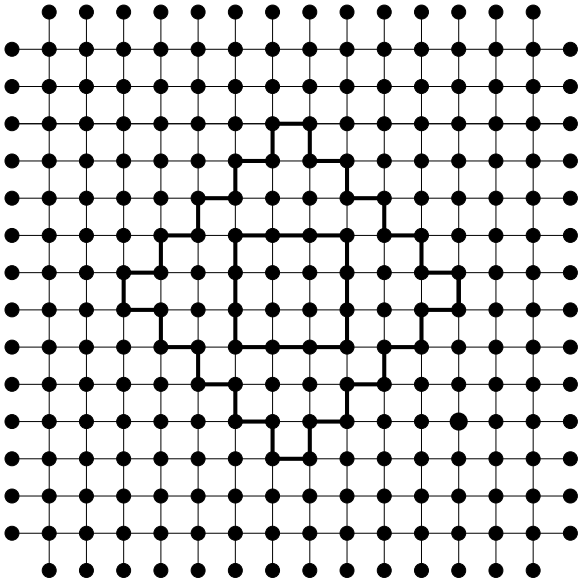

Figure 3: 14 by 14 network with diamond shape

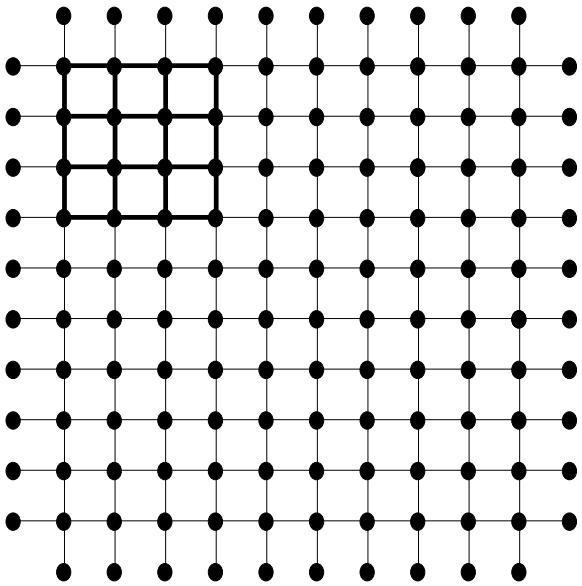

Figure 4: 10 by 10 network with left corner square

The problem that should be looked at which might give the most insight into this problem of finding how many significant digits are necessary for a correct recovery uses a network whose resistors are all the same. The example used for this project was a 10 by 10 network which had all its conductivity set as 1. It is therefore predetermined that the conductivity will be recovered correctly (especially if the Background approach is used), thus the program was modified to store the value that was originally calculated for each conductor before being compared to the set of conductors (Appendix 6.3 gammacal.ms). What is being learned from this experiment is how great the error can be from the calculated value to the true value of the conductor. This will demonstrate the minimum number of significant digits needed in the recovery to determine the values of the conductors correctly. The table in the beginning of section 3 illustrates some of the data that was collected, here is a table for larger values:

Digits	*Maximum Error*
11	7.4027146948
13	.0024201940211
14	.0006050122631
15	.00008805239372

The pattern of having one more digit of accuracy for one more significant digit continues in this experiment. This experiment shows that for a 10 by 10 network, if conductors with values of either 100 or 1 are being recovered, then at least 10 significant digits need to be used in the recovery to be able to recover the network correctly. If the values are 10 or 1, then that figure becomes 12 significant digits. This program, therefore, would not be practical in real world applications since the values of the Λ matrix would have to be correct to more significant digits than most tools could measure.

5 Conclusion

While the programs created for this project have improved on the ability to recover the conductivity of a rectangular resistor network, there are a lot more questions that are left unanswered in this field. The program given in this paper could be improved in many ways. The Curtis-Morrow algorithm is only used from two opposite corners. If the algorithm was applied to all four corners of the network, the accuracy would most likely improve. There may also be a suitable way to average the values the program produces that

would cut down on the error. The program could also calculate each step from both directions, but that modification has not yet been implemented. One major obstacle to solving this problem, which is a major problem in numerical analysis in general, is working with terms that differ substantially in magnitude. When finding the voltages on the boundary, the values can quickly grow to 10^{11}, 10^{12}, or greater while interior voltages exist that are as small as 10^{-9}. Unless a large number of significants are used which will also take up a large amount of memory, there is no way to overcome this problem. This project also did not fully look at the case when there are more than two possible resistor values. While the programs were designed for this case, not enough was discovered in the two resistors case to move on to more. This problem still has many directions from which it can be tackled, especially as new methods in computer science and numerical analysis are discovered.

6 Appendix

6.1 sqrgen.ms

This program creates the Lambda matrix for a square rectangular network. Input for N the size of one side of the network and at the end of the program the file name under which you want to store the matrix. The file will store the size of the network, the conductivities, and the Lambda matrix. The Digits command sets the number of significant digits. If one would rather use rational arithmetic, then change all occurrences of a decimal with an integer. The numbering of the nodes of this network starts with 1 in the upper right side corner and then continues clockwise around the boundary. The interior picks up this numbering in the upper left corner and then continues right and then down. This program will be for a 3 by 3 network of all 1's with a cross of 10's in the middle, but it can be easily edited for any rectangular network by appending the set 'conduct' and the vector of edges 'edge[i]' (an edge is denoted by a coordinate [i, j] where i and j are the two nodes the edge goes between- order does not matter).

```
with(linalg):
Digits:=10:
N:=3:
edges:={}:
TN:=N*(N+4):
```

```
for i from 1 by 1 to N do
 edges:=edges union {[i, (i+4)*N]};
 edges:=edges union { [N+i, TN+1-i]};
 edges:=edges union {[2*N +i, TN - i*N +1]}
od:
for j from 4*N+1 by 1 to TN do
 edges:=edges union {[j-N, j]}
od:
Nd:=N*(N-1):
for i from 1 to N do
 for j from 2 to N do
  edges:=edges union {[(3+i)*N+j-1, (3+i)*N+j]}
 od:
od:
k:=matrix(TN, TN, 0):
conduct:={1.0, 10.0}:
edge[10.0]:={[14, 17], [16, 17], [17, 18], [17, 20]}:
edge[1.0]:=edges minus edge[10.0]:
for i in conduct do
  for p in edge[i] do
    k[p[1], p[1]]:=k[p[1], p[1]] + i;
    k[p[2], p[2]]:=k[p[2], p[2]] + i;
    k[p[1], p[2]]:= -i;
    k[p[2], p[1]]:= -i
  od:
od:
c:=submatrix(k, 4*N+1..TN, 4*N+1..TN):
b:=submatrix(k, 1..4*N, 4*N+1..TN):
a:=submatrix(k, 1..4*N, 1..4*N):
x:=linsolve(c, transpose(b)):
Lambda:=add (a, -multiply(b, x)):
save(N, conduct, Lambda, `lamb2a.m`):
```

6.2 gamma.ms

This program uses the Curtis-Morrow algorithm to recover conductivity from the Lambda matrix of a rectangular network. The file ‘lamb.m‘ needs to be

included with this file storing N = the size of the network, conduct = set of possible conductors, and Lambda = the lambda matrix. The algorithm will automatically use rational arithmetic if that is how the matrix was constructed. If not, the Digits command determines how many significant digits are used in the recovery. The two matrices that are produced contain the values for the horizontal and vertical conductors with the coordinate system starting in the top right corner of the network and continuing left and down.

```
with(linalg):
read(‘lamb.m‘):
Digits:=40:
bN:=4*N:
gammav:=matrix(N+1, N, 0):
gammah:=matrix(N, N+1, 0):
mu:=matrix(N, N, 0):
# This will find the top right corner
j:=4*N:  k:=1:  i:=3*N:
gammav[1, 1]:= Lambda[j, j] - Lambda[i, j]*Lambda[j,k]/Lambda[i, k]:
gammah[1, 1]:= Lambda[k, k] - Lambda[i,k]*Lambda[k,j]/Lambda[i,j]:
# This part will do the steps of the network
for i from 2 to N do
  c:=4*N+1-i:
  A1:=submatrix(Lambda, 2*N+1..2*N+i, 1..i):
  d1:=-subvector(Lambda, 2*N+1..2*N+i, c):
  alpha1:=vector(bN, 0):
  alpha:=linsolve(A1, d1):
  for j from 1 to i do
    alpha1[j]:= alpha[j]
  od:
  alpha1[c]:=1:
  gammav[1, i]:= dotprod(row(Lambda, c), alpha1):
  gammah[i, 1]:= dotprod(row(Lambda, i), alpha1)/alpha1[i]:
#  This solves the Dirichlet problem to find the interior voltages
  for j from 1 to i-1 do
    mu[j, 1]:= alpha1[j] - dotprod(row(Lambda, j), alpha1)/gammah[j, 1]
  od:
  if i>2 then
    for h from 2 to i-1 do
```

```
      mu[1, h]:= -dotprod(row(Lambda, 4*N+1-h), alpha1)/gammav[1, h]
    od
  fi:
  if i>3 then
    for k from 2 to i-2 do
      mu[2, k]:= ((gammav[1, k]+gammav[2,k]+gammah[1, k]+gammah[1, k+1])*
mu[1, k]-gammah[1,k]*mu[1,k-1] -gammah[1, k+1]*mu[1, k+1])/gammav[2,k]
    od
  fi:
  if i>4 then
    for p from 3 to i-2 do
      for q from 2 to i-p do
        mu[p,q]:=((gammav[p,q]+gammav[p-1,q]+gammah[p-1,q]+gammah[p-1
,q+1])*mu[p-1,q]-gammav[p-1,q]*mu[p-2,q]-gammah[p-1,q]*mu[p-1,q-1]-
gammah[p-1,q+1]*mu[p-1,q+1])/gammav[p,q]
      od
    od
  fi:
# Now, we will find new gamma's
  gammah[1,i]:=-gammav[1,i]/mu[1,i-1]:
  gammav[i,1]:=-alpha1[i]*gammah[i,1]/mu[i-1,1]:
  if i>2 then
    a:= 2:
    b:= i-1:
    gammav[a, b]:=-(gammah[a-1,b+1]+gammav[a-1,b]+gammah[a-1,b])+ga
mmah[a-1,b]*mu[a-1,b-1]/mu[a-1,b]:
    gammah[a, b]:=-gammav[a,b]*mu[a-1,b]/mu[a,b-1]:
    if i>3 then
      for count from 1 to i-3 do
        a:=a+1:
        b:=b-1:
        gammav[a, b]:=-(gammah[a-1,b+1]+gammav[a-1,b]+gammah[a-1,b])+(
gammah[a-1,b]*mu[a-1,b-1]+ gammav[a-1,b]*mu[a-2,b])/mu[a-1,b]:
        gammah[a,b]:=-gammav[a,b]*mu[a-1,b]/mu[a,b-1]
      od
    fi
  fi
od:
```

```
# This will find the bottom left corner
j:=2*N:  k:=2*N+1:  i:=N:
gammav[N+1, N]:= Lambda[j, j] - Lambda[i, j]*Lambda[j,k]/Lambda[i, k]:
gammah[N, N+1]:= Lambda[k, k] - Lambda[i,k]*Lambda[k,j]/Lambda[i,j]:
for i from 2 to N do
  c:=2*N+1-i:
  mu:=matrix(N, N, 0):
  A1:=submatrix(Lambda, 1..i, 2*N+1..2*N+i):
  d1:=-subvector(Lambda, 1..i, c):
  alpha1:=vector(bN, 0):
  alpha:=linsolve(A1, d1):
  for j from 1 to i do
    alpha1[2*N+j]:= alpha[j]
  od:
  alpha1[c]:=1:
  gammav[N+1, N+1-i]:= dotprod(row(Lambda, c), alpha1):
  gammah[N+1-i, N+1]:= dotprod(row(Lambda, 2*N+i), alpha1)/alpha1[2*N+i]:
#  This part solves the Dirichlet problem to find the interior voltages
  for j from 1 to i-1 do
    mu[N+1-j, N]:= alpha1[2*N+j] - dotprod(row(Lambda, 2*N+j), alpha1)/gam
mah[N+1-j, N+1]
  od:
  if i>2 then
    for h from 2 to i-1 do
      mu[N, N+1-h]:= -dotprod(row(Lambda, 2*N+1-h), alpha1)/gammav[N+1,
N+1-h]
    od
  fi:
  if i>3 then
    for k from 2 to i-2 do
      mu[N-1, N+1-k]:= ((gammav[N+1, N+1-k]+gammav[N, N+1-k]+gammah[N
, N+1-k]+gammah[N, N+2-k])*mu[N, N+1-k]-gammah[N, N+2-k]*mu[N,N+2-
k] -gammah[N, N+1-k]*mu[N, N-k])/gammav[N,N+1-k]
    od
  fi:
  if i>4 then
    for p from 3 to i-2 do
      for q from 2 to i-p do
```

```
        mu[N+1-p,N+1-q]:=((gammav[N+3-p,N+1-q]+gammav[N+2-p,N+1-q]+g
ammah[N+2-p, N+1-q]+gammah[N+2-p, N+2-q])*mu[N+2-p, N+1-q]-gamm
av[N+3-p,N+1-q]*mu[N+3-p,N+1-q]-gammah[N+2-p, N+2-q]*mu[N+2-p, N+
2-q]-gammah[N+2-p, N+1-q]*mu[N+2-p, N-q])/gammav[N+2-p, N+1-q]
      od
    od
  fi:
# Now, we will find new gamma's
  gammah[N,N+2-i]:=-gammav[N+1,N+1-i]/mu[N,N+2-i]:
  gammav[N+2-i,N]:=-alpha1[2*N+i]*gammah[N+1-i,N+1]/mu[N+2-i,N]:
  if i>2 then
    a:= N:
    b:= N+2-i:
    gammav[a, b]:=-(gammah[a,b]+gammav[a+1,b]+gammah[a,b+1])+gamma
h[a,b+1]*mu[a,b+1]/mu[a,b]:
    gammah[a-1, b+1]:=-gammav[a,b]*mu[a,b]/mu[a-1,b+1]:
    if i>3 then
      for count from 1 to i-3 do
        a:=a-1:
        b:=b+1:
        gammav[a, b]:=-(gammah[a,b]+gammav[a+1,b]+gammah[a,b+1])+(gam
mah[a,b+1]*mu[a,b+1]+ gammav[a+1,b]*mu[a+1,b])/mu[a,b]:
        gammah[a-1,b+1]:=-gammav[a,b]*mu[a,b]/mu[a-1,b+1]
      od
    fi
  fi
od:
#print(gammah, gammav):
Digits:=4:
h:=evalm(gammah*1.000):
v:=evalm(gammav*1.000):
print(h):
print(v):
```

6.3 gammacal.ms

This program is similar to gamma.ms, but instead uses a procedure to take advantage of the limited set of possible conductors. This program was created

for a diamond pattern with a background of 1's and the rest of the conductors as 100. It can easily be modified for other cases. It also stores the calculated value of the conductor at each step.

```
with(linalg):
read(‘lamb.m‘):
Digits:=20:
bN:=4*N:
gammav:=matrix(N+1, N, 0):
calv:=matrix(N+1, N, 0):
gammah:=matrix(N, N+1, 0):
calh:=matrix(N, N+1, 0):
mu:=matrix(N, N, 0):
ct:=0:
print(conduct):
epsilon[1.00]:=11.0:
epsilon[100.00]:=88.0:
getcon:=proc(cal) local num, ans, cond; global epsilon, conduct, ct;
ans:=1:  cond:= conduct:
#print(cal):
for num in cond do
    if abs(cal-num)<epsilon[num] then
     ans:=num;
    fi
  od;
if ans=1 then ct:=ct+1 fi;
ans end:
# This will find the top right corner
j:=4*N:  k:=1:  i:=3*N:
calv[1,1]:=Lambda[j, j] - Lambda[i, j]*Lambda[j,k]/Lambda[i, k]:
gammav[1,1]:=getcon(calv[1,1]):
calh[1,1]:=Lambda[k, k] - Lambda[i,k]*Lambda[k,j]/Lambda[i,j]:
gammah[1,1]:=getcon(calh[1,1]):
# This part will do the steps of the network
for i from 2 to N do
  c:=4*N+1-i:
  A1:=submatrix(Lambda, 3*N+1-i..3*N, 1..i):
  d1:=-subvector(Lambda, 3*N+1-i..3*N, c):
```

```
  alpha1:=vector(bN, 0):
  alpha:=linsolve(A1, d1):
  for j from 1 to i do
    alpha1[j]:= alpha[j]
  od:
  alpha1[c]:=1:
  calv[1,i]:=dotprod(row(Lambda, c), alpha1):
  gammav[1, i]:=getcon(calv[1,i]):
  calh[i,1]:=dotprod(row(Lambda, i), alpha1)/alpha1[i]:
  gammah[i, 1]:=getcon(calh[i,1]):
#  This is the unslick method where I find all the mu's
  for j from 1 to i-1 do
    mu[j, 1]:= alpha1[j] - dotprod(row(Lambda, j), alpha1)/gammah[j, 1]
  od:
  if i>2 then
    for h from 2 to i-1 do
      mu[1, h]:= -dotprod(row(Lambda, 4*N+1-h), alpha1)/gammav[1, h]
    od
  fi:
  if i>3 then
    for k from 2 to i-2 do
      mu[2, k]:= ((gammav[1, k]+gammav[2,k]+gammah[1, k]+gammah[1, k+1])*
mu[1, k]-gammah[1,k]*mu[1,k-1] -gammah[1, k+1]*mu[1, k+1])/gammav[2,k]
    od
  fi:
  if i>4 then
    for p from 3 to i-2 do
      for q from 2 to i-p do
        mu[p,q]:=((gammav[p,q]+gammav[p-1,q]+gammah[p-1,q]+gammah[p-1
,q+1])*mu[p-1,q]-gammav[p-1,q]*mu[p-2,q]-gammah[p-1,q]*mu[p-1,q-1]-
gammah[p-1,q+1]*mu[p-1,q+1])/gammav[p,q]
      od
    od
  fi:
# Now, we will find new gamma's
  calh[1,i]:=-gammav[1,i]/mu[1,i-1]:
  gammah[1,i]:=getcon(calh[1,i]):
  calv[i,1]:=-alpha1[i]*gammah[i,1]/mu[i-1,1]:
```

```
  gammav[i,1]:=getcon(calv[i,1]):
   if i>2 then
    a:= 2:
    b:= i-1:
    calv[a,b]:=-(gammah[a-1,b+1]+gammav[a-1,b]+gammah[a-1,b])+gammah
[a-1,b]*mu[a-1,b-1]/mu[a-1,b]:
    gammav[a, b]:=getcon(calv[a,b]):
    calh[a,b]:=-gammav[a,b]*mu[a-1,b]/mu[a,b-1]:
    gammah[a, b]:=getcon(calh[a,b]):
    if i>3 then
      for count from 1 to i-3 do
        a:=a+1:
        b:=b-1:
        calv[a,b]:=-(gammah[a-1,b+1]+gammav[a-1,b]+gammah[a-1,b])+(gam
mah[a-1,b]*mu[a-1,b-1]+ gammav[a-1,b]*mu[a-2,b])/mu[a-1,b]:
        gammav[a,b]:=getcon(calv[a,b]):
        calh[a,b]:=-gammav[a,b]*mu[a-1,b]/mu[a,b-1]:
        gammah[a,b]:=getcon(calh[a,b]):
      od
    fi
  fi
od:
# This will find the bottom left corner
j:=2*N:  k:=2*N+1:  i:=N:
calv[N+1, N]:=Lambda[j, j]-Lambda[i,j]*Lambda[j,k]/Lambda[i,k]:
gammav[N+1,N]:=getcon(calv[N+1,N]):
calh[N,N+1]:=Lambda[k, k]-Lambda[i,k]*Lambda[k,j]/Lambda[i,j]:
gammah[N,N+1]:=getcon(calh[N,N+1]):
for i from 2 to N do
  c:=2*N+1-i:
  A1:=submatrix(Lambda, N+1-i..N, 2*N+1..2*N+i):
  d1:=-subvector(Lambda, N+1-i..N, c):
  alpha1:=vector(bN, 0):
  alpha:=linsolve(A1, d1):
  for j from 1 to i do
    alpha1[2*N+j]:= alpha[j]
  od:
  alpha1[c]:=1:
```

```
  calv[N+1, N+1-i]:=dotprod(row(Lambda, c), alpha1):
  gammav[N+1, N+1-i]:=getcon(calv[N+1,N+1-i]):
  calh[N+1-i,N+1]:=dotprod(row(Lambda, 2*N+i), alpha1)/alpha1[2*N+i]:
  gammah[N+1-i, N+1]:=getcon(calh[N+1-i,N+1]):
#  This is the unslick method where I find all the mu's
  for j from 1 to i-1 do
    mu[N+1-j, N]:= alpha1[2*N+j] - dotprod(row(Lambda, 2*N+j), alpha1)/gam
mah[N+1-j, N+1]
  od:
  if i>2 then
    for h from 2 to i-1 do
      mu[N, N+1-h]:= -dotprod(row(Lambda, 2*N+1-h), alpha1)/gammav[N+1,
N+1-h]
    od
  fi:
  if i>3 then
    for k from 2 to i-2 do
      mu[N-1, N+1-k]:= ((gammav[N+1, N+1-k]+gammav[N, N+1-k]+gammah[N
, N+1-k]+gammah[N, N+2-k])*mu[N, N+1-k]-gammah[N, N+2-k]*mu[N,N+2-
k] -gammah[N, N+1-k]*mu[N, N-k])/gammav[N,N+1-k]
    od
  fi:
  if i>4 then
    for p from 3 to i-2 do
      for q from 2 to i-p do
        mu[N+1-p,N+1-q]:=((gammav[N+3-p,N+1-q]+gammav[N+2-p,N+1-q]+g
ammah[N+2-p, N+1-q]+gammah[N+2-p, N+2-q])*mu[N+2-p, N+1-q]-gamm
av[N+3-p,N+1-q]*mu[N+3-p,N+1-q]-gammah[N+2-p, N+2-q]*mu[N+2-p, N+
2-q]-gammah[N+2-p, N+1-q]*mu[N+2-p, N-q])/gammav[N+2-p, N+1-q]
      od
    od
  fi:
# Now, we will find new gamma's
  calh[N,N+2-i]:=-gammav[N+1,N+1-i]/mu[N,N+2-i]:
  gammah[N,N+2-i]:=getcon(calh[N,N+2-i]):
  calv[N+2-i,N]:=-alpha1[2*N+i]*gammah[N+1-i,N+1]/mu[N+2-i,N]:
  gammav[N+2-i,N]:=getcon(calv[N+2-i,N]):
  if i>2 then
```

```
    a:= N:
    b:= N+2-i:
    calv[a,b]:=-(gammah[a,b]+gammav[a+1,b]+gammah[a,b+1])+gammah[a,b
+1]*mu[a,b+1]/mu[a,b]:
    gammav[a, b]:=getcon(calv[a,b]):
    calh[a-1,b+1]:=-gammav[a,b]*mu[a,b]/mu[a-1,b+1]:
    gammah[a-1, b+1]:=getcon(calh[a-1,b+1]):
    if i>3 then
      for count from 1 to i-3 do
        a:=a-1:
        b:=b+1:
        calv[a,b]:=-(gammah[a,b]+gammav[a+1,b]+gammah[a,b+1])+(gammah[
a,b+1]*mu[a,b+1]+ gammav[a+1,b]*mu[a+1,b])/mu[a,b]:
        gammav[a, b]:=getcon(calv[a,b]):
        calh[a-1,b+1]:=-gammav[a,b]*mu[a,b]/mu[a-1,b+1]:
        gammah[a-1,b+1]:=getcon(calh[a-1,b+1]):
      od
    fi
  fi
od:
print(gammah, gammav):
checkv:=matrix(N+1, N, 1.00):
checkh:=matrix(N, N+1, 1.00):
checkh[3,8]:=100.00:  checkh[4,7]:=100.00:  checkh[4,9]:=100.00:
checkh[5, 6]:=100.00:  checkh[5,10]:=100.00:  checkh[6, 5]:=100.00:
checkh[6, 11]:=100.00:  checkh[7, 4]:=100.00:  checkh[7, 12]:=100.00:
checkh[8, 4]:=100.00:  checkh[8,12]:=100.00:  checkh[9, 5]:=100.00:
checkh[9, 11]:=100.00:  checkh[10, 6]:=100.00:  checkh[10,10]:=100.00:
checkh[11, 9]:=100.00:  checkh[11,7]:=100.00:  checkh[12, 8]:=100.00:
checkh[6,7]:=100.00:  checkh[6,8]:=100.00:  checkh[6,9]:=100.00:
checkh[9,7]:=100.00:  checkh[9,8]:=100.00:  checkh[9,9]:=100.00:
checkv[8,3]:=100.00:  checkv[7,4]:=100.00:  checkv[9,4]:=100.00:
checkv[6,5]:=100.00:  checkv[10,5]:=100.00:  checkv[5,6]:=100.00:
checkv[11,6]:=100.00:  checkv[4,7]:=100.00:  checkv[12,7]:=100.00:
checkv[4,8]:=100.00:  checkv[12,8]:=100.00: checkv[5,9]:=100.00:
checkv[11,9]:=100.00:  checkv[6,10]:=100.00:  checkv[10,10]:=100.00:
checkv[9,11]:=100.00:  checkv[7,11]:=100.00:  checkv[8, 12]:=100.00:
checkv[7,6]:=100.00:  checkv[8,6]:=100.00:  checkv[9,6]:=100.00:
```

```
checkv[7,9]:=100.00:  checkv[8,9]:=100.00:  checkv[9,9]:=100.00:
errorh:=add(gammah, -checkh);
errorv:=add(gammav, -checkv);
ct;
Digits:=5:
h:=evalm(1.00*calh);
v:=evalm(1.00*calv);
```

References

1. E.B Curtis and J.A. Morrow, *Determining the resistors in a network*, SIAM J. of Applied Math., 50 (1990), pp. 918-930.

2. Bruce W. Char et al., *Maple V Library Reference Manual*, Springer-Verlag, 1992.