ON NUMERICAL RECOVERY METHODS FOR THE INVERSE PROBLEM

GEORGE TUCKER, SAM WHITTLE, AND TING-YOU WANG

Abstract. In this paper, we present the approaches we took to recover the conductances of an electrical network, concentrating on the method of non-linear least squares optimization. The methods include steepest descent with symbolic differentiation and numerical differentiation, Newton's method, and Levenberg-Marquardt. Using these algorithms, we were able to recover large networks. In order to better reflect real life situations, we decided to add noise to our measurements and include some knowledge of the resistors. This led us to add a regularization term and try a method of rounding between iterations. In the end, without any a priori knowledge, we recovered a 50x50 square lattice network with constant conductance to within 0.01% error.

Contents

1. Introduction

An electrical network is a graph with nodes arbitrarily partitioned into boundary and interior sets and with a non-negative conductance function γ defined on each edge. Such networks are introduced and discussed in [1].

Definition 1.1 (Kirchhoff Matrix). Suppose an electrical network contains n boundary nodes and m interior nodes. Let the boundary nodes be numbered 1 to n and

Date: August 18, 2006.

the interior nodes be numbered $n+1$ to $n+m$. Then the Kirchhoff matrix K for the network is the $(m+n)$ by $(m+n)$ matrix defined:

$$K_{ij} = \begin{cases} -\gamma_{ij}, & \text{if } i \neq j \\ \sum_{k \neq i} \gamma_{ik}, & i = j \end{cases}$$

where γ_{ij} equals 0 if there is no edge between nodes i and j and γ_{ij} equals the conductance function evaluated on the edge between nodes i and j otherwise. K is symmetric and can be expressed in block form as $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$.

Definition 1.2 (Response Matrix)**.** The response matrix of a network Λ is the square matrix that maps boundary voltages to boundary currents. If K is a Kirchhoff matrix in block form $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$, then Λ is the Schur complement of C in K:

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

The inverse problem for electrical networks is to determine the conductance function γ by measuring voltages and currents at the boundary nodes. We attempted to numerically solve the inverse problem by minimizing the sum of square errors between predicted currents and measured currents. The algorithm takes as inputs voltages, the resulting currents, an adjacency matrix, and an initial guess for the conductivity function. Then, we iteratively minimized the following quantities with respect to γ

$$F(\gamma) = \text{vec}(\tilde{\Lambda}_\gamma - \Lambda)$$

$$f(\gamma) = \frac{1}{2}||F(\gamma)||^2 = \frac{1}{2}F(\gamma)^T F(\gamma). \tag{1}$$

where $\tilde{\Lambda}_\gamma$ is the response matrix evaluated with γ as the conductance function. We tried several methods of optimization and imposed additional constraints to deal with the ill-posedness of the problem.

2. Arbitrary Conductances

2.1. Steepest Descent using Symbolic Derivatives.

```
initialize tau (step size)
symbolically compute gradient of f
while iterationsDone < maxIterations
  grad = evaluate symbolic derivative at x
  x = x - tau*grad
end
```

Our first algorithm symbolically computed the gradient by assuming every entry in K was a variable, which allowed us to symbolically compute the functional (Eq. 1) and then to symbolically differentiate the functional. We thought this technique would be more exact because the whole computation was accomplished in a single step. Also we thought we could save time by computing the gradient once symbolically, instead of numerically every iteration. With the symbolically computed gradient, we used a gradient descent algorithm with a fixed time step. The conductivity function was modified at each iteration as follows: $\gamma_{k+1} = \gamma_k -$

$\alpha\nabla f(\gamma_k)$, where α is a fixed constant. As we increased the network size, this method quickly became infeasible. Even though the gradient was computed only once, evaluating the gradient, which had to be done every iteration, was extremely slow.

2.2. **Steepest Descent using Numerical Derivatives.**

```
initialize tau (step size)
while iterationsDone < maxIterations
  foreach j = 1 to n
    e_j = [0, ... 1, ... 0], 1 in jth position, 0 elsewhere
    f_j = ( f( x + epsilon*e_j ) - f( x ) ) / epsilon
  end
  x = x - tau*grad
end
```

To increase the speed of our algorithm, we switched to numerical computations of the gradient by forward differences.

$$\partial_x F(\gamma) \approx \frac{F(\gamma + hx) - F(\gamma)}{h} \qquad \text{for small } h$$

This algorithm was much faster, and we were able to tackle larger networks, but we were still using a constant time step. Instead of writing an algorithm to vary the time step, we leveraged built-in Matlab minimization algorithms. At first we used a subspace trust-region method based on an interior-reflective Newton method, which allowed us to specify a lower bound on γ. During each iteration, computing $\tilde{\Lambda}$ required inverting a principle proper submatrix of K. Instead of inverting the matrix, it is more numerically stable and quicker to solve a linear system of equations. Because γ was always positive and K has row sums equal to zero, any principle proper submatrix of K is positive definite, which allowed us to use Cholesky factorization to solve the system of equations even quicker. We were able to recover 5x5 square lattice networks, but as the size of the network increased, the algorithm converged much slower and the solutions were less accurate (refer to Fig. 2 on page 7 for a picture of a square lattice network). We switched to an unconstrained Levenberg-Marquardt algorithm. Because γ was unconstrained, during intermediary steps in the algorithm, some of the γ values became negative, so we were unable to use the Cholesky factorization. Although the negative values are troubling physically, they do not seem to have an adverse effect on the numerical calculations. After trying several other algorithms, we settled on the Levenberg-Marquardt method (Section 5).

3. Finite Set of Known Conductances

In an attempt to simplify the problem, we restricted the conductances in the network to be from a small set of known conductances. For example, suppose you have a network with 12 edges but you know the conductance for each edge is either 1 or 10. This scenario is perhaps more likely to occur in practice, due to a limited supply of resistor types. With only a finite set of conductances, it seemed feasible to try other techniques for minimization, taking advantage of this restriction.

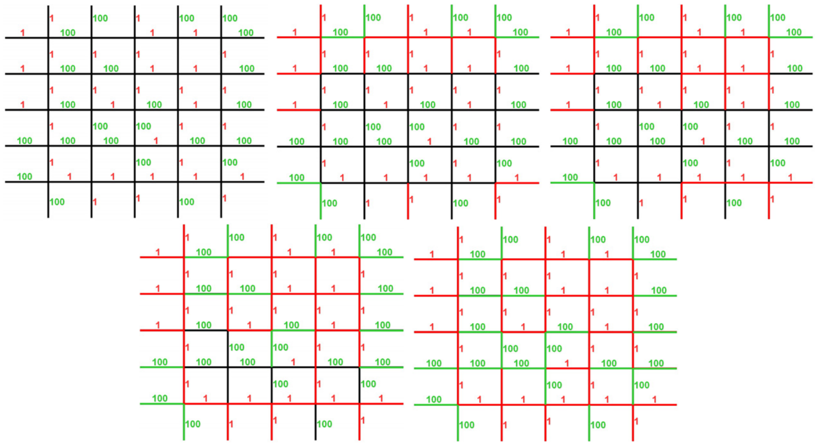

Figure 1. An example of the rounding algorithm for a 5x5 square lattice. Each frame shows an iteration of the rounding algorithm, where the colored edges represent determined conductances. As the algorithm progresses, more of the edges are determined, until the graph is completely determined.

3.1. **Rounding.**

```
additional input: set of known conductances
fixed edges = nothing
while some conductances are not fixed
  minimize with other algorithm to low precision
  candidate edges = boundary edges + edges adjacent to fixed edges
  if a candidate edge is within epsilon percent of a resistor, fix
      that edge to that resistor
end
```

When working with a finite set of known conductances, we could observe when a conductance approached one of the possible conductances during the process of minimization. When a varying conductance was sufficiently close to one of the possible conductances, we could round the conductance. We thought that this would help the minimization converge faster because it would avoid excessive computation. However we ran into difficulties if, during the process of minimization, a conductance was rounded incorrectly. We noticed that this occurred most frequently in the interior of the graph. We also observed that during minimization, the conductivities on the edge of the graph converged to the correct values relatively rapidly. This intuitively made sense because it is easier to recover conductances on the boundary of the graph. To take advantage of these observations we developed a more sophisticated approach.

The algorithm we developed operates by determining certain edges during the optimization process. Once an edge is determined it is not varied during subsequent minimization. Thus the number of free variables is reduced throughout the algorithm until all edges are determined. An iteration of the algorithm begins with the use of another minimization algorithm such as Levenberg-Marquardt (see section 5). We run this algorithm for only a small number of iterations and then try to fix conductances. An edge is a candidate for fixing if it is a boundary edge or it is adjacent to an edge that is already determined. After generating the set of candidate edges, we compare their values to the set of possible conductances; if they are within a relative ϵ, the edge is rounded to that conductance and considered determined in the following iterations. This served the dual purposed of increasing accuracy and reducing computation because the optimizer no longer needed to consider that edge. However, the algorithm suffered a serious flaw because if an edge was fixed to an incorrect value, there was no hope for recovery. We tried improving this method by adding extra conditions before an edge was rounded. These included examining the gradient to see if it indicated the edge was moving toward the value to which it was being rounded and requiring the conductance lies near the possible conductance for several iterations before rounding. We were unable to make the algorithm successful in all cases. However when the algorithm runs successfully, it can recover a graph much faster than other methods.

For comparison, in a 10x10 square lattice network with conductances chosen randomly from possibilities of 1 and 10, the rounding algorithm ran in 25.063 seconds while the normal Levenberg-Marquardt algorithm took 608.5061 seconds.

3.2. **Genetic Algorithm.**

```
additional input: set of known conductances
randomly generate a population, where each organism is guess for
   conductances
while iterationsDone < maxIterations
   evaluate fitness for each organism in population
   save most fit organism encountered
   evolve a new population:
      crossbreed organisms, with more fit organisms selected more
         often
      mutate organisms
   population = new population
end
output the most fit organism
```

A genetic algorithm models the process of evolution and survival of the fittest in an attempt to find good solutions to a problem. In our implementation, each organism was the conductances of all edges in the graph. Thus a population of organisms was a collection of possible solutions to minimizing the functional (Eq. 1).

The algorithm begins with an initial population of random organisms. Each organism consists only of conductivities specified in the given set. The organisms are then ranked according to the value of the functional evaluated at their conductances. An organism is defined to be more fit than another if it has a lower functional value. A new population is then generated through crossbreeding and mutation.

Crossbreeding is accomplished by taking two organisms and swapping some of their conductances. The probability of an organism being selected as a parent in crossbreeding corresponds to its fitness rank, hopefully resulting in fitter offspring. We implemented crossbreeding by first ordering the nodes in the graph by their depth. The depth of an edge was defined to be the minimum depth of its end nodes. The edges in the graph were then ordered by their depth. To crossbreed two organisms, we then selected a random pivot point and exchanged all conductances corresponding to nodes following the pivot point in the edge-depth ordering. This can be intuitively thought of as swapping the interior portion (the size of which depended on the pivot) of the two parents. The resulting two offspring were then added to the new population.

In addition to those organisms produced by crossbreeding, the new population also contained some of the previous fittest organisms and some new randomly generated organisms. This kept fit organisms in the population while also providing some diversity. After producing a new population, we then mutated organisms by changing some conductances randomly. By doing so, we evolved the solutions and beneficial mutations were rewarded in the next iteration.

This process was repeated for a certain number of iterations or until the actual conductances were reached. At each iteration, the fittest organism in the current population was checked if it as the fittest organism encountered so far. Thus the best answer encountered was stored until the end of the algorithm, even if it was not within the ending population.

Genetic algorithms have proved successful in finding good solutions to difficult problems. In our experience, this approach worked well for small (5 by 5) lattice networks. A problem arose with larger networks related to our functional. The algorithm found local minima, at which the functional value was very small. However, in such a solution many conductances in the graph were incorrect. Because so many conductances were different from the actual solution, the amount of mutation needed to jump to near the actual solution would have been large. If the mutation amount was too large however, the algorithm would degenerate to random searching. Therefore genetic algorithms unfortunately did not apply well to our problem.

3.3. **Regularization.**

In [2], the inverse problem is solved for the continuous case with a piecewise constant (in two pieces) conductance function. To deal with the ill-posedness of the problem, Chan et al. proposed minimizing

$$f(\phi, q_1, q_2) = \frac{1}{2}\sum_{i=1}^{N}\int_{\partial\Omega} |u_i(s,q) - m_i(s)|^2 ds + \beta \int_{\Omega} |\nabla q(\phi, q_1, q_2)| dx$$

where ϕ is the level set function defining the two regions, q_1 and q_2 are the conductivities, N is the number of measurements, u is the predicted function, and m is the measured function. The regularization term $\beta \int_{\Omega} |\nabla q| dx$ measures the jump between q_1 and q_2 and the length of the interface between the two regions. So the new equation seeks to minimize the functional as well as minimizing the interface and the difference. In the context of tomography, where the areas of conductance will be clumped in roughly smooth regions, this regularization term is practical. They had favorable results when recovering the interface even with 1 percent noise.

To discretize the regularization term, we fixed an interior node and summed the squared difference between adjacent pairs of edges and then summed over all interior nodes. So our discretized functional was

$$f(\gamma) = \frac{1}{2}||F(\gamma)||^2 + \beta \sum_{i \in \text{int}\Omega} \left(\sum_{j\sim i, k\sim i} (\gamma_{ij} - \gamma_{ik})^2 \right).$$

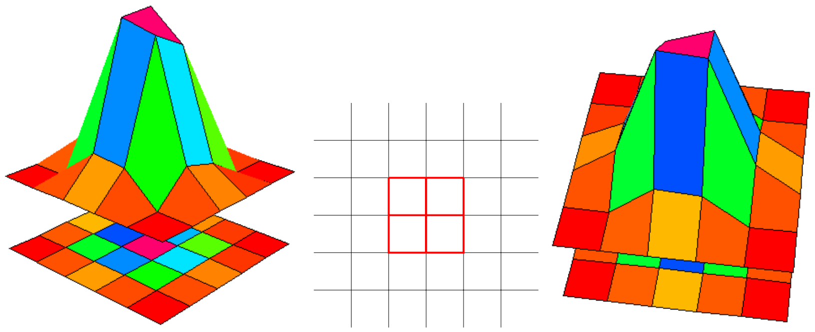

FIGURE 2. Minimization with regularization term on a 5x5 network with all ones except for hundreds in a 2x2 clump in the middle and measurements simulated with 1% error. Each square in the graph corresponds to an average of the edges along that square in the circuit.

Unlike the regularization term in the continuous case, we decided to square the differences in the regularization. This way we could still apply least square techniques to the functional.

We were able to recover the interface on a 5x5 square lattice with a clump of 100 conductance edges surrounded by 1 conductance edges with 1 percent noise in the measurements (refer to Fig. 2). This particular setup is extremely difficult to solve because the high conductance region is surrounded by low conductance. Therefore the currents do not penetrate the middle area, so the measurements do not provide much information on the middle conductances. The problem was recovering the exact value of the conductance of the clump of resistors. The algorithm returned a constant conductance (within some error) for both regions, but the actual value of the middle conductance region varied as we changed the order of magnitude of β. Perhaps an appropriate value for β could be determined by knowing an estimate for the error in the measurements.

4. Analytic Derivative

Let K be a Kirchhoff matrix with block structure $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$, let $D = \begin{bmatrix} I \\ -C^{-1}B^T \end{bmatrix}$ where D can be thought of as a map from boundary potentials to γ-harmonic functions, and let $\partial_i x$ mean the partial derivative of x with respect to edge i. Then

$$\begin{aligned} \Lambda = A - BC^{-1}B^T &= \begin{bmatrix} I & -BC^{-1} \end{bmatrix} \begin{bmatrix} A - BC^{-1}B^T \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} I & -BC^{-1} \end{bmatrix} \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} I \\ -C^{-1}B^T \end{bmatrix} \\ &= D^T K D. \end{aligned}$$

Therefore, $\partial_i \Lambda = \partial_i D^T K D + D^T \partial_i K D + D^T K \partial_i D$. However, $\partial_i D^T K D = \begin{bmatrix} 0 & \partial_i(-BC^{-1}) \end{bmatrix} \begin{bmatrix} \Lambda \\ 0 \end{bmatrix} = 0$ and similarly $D^T K \partial_i D = (\partial_i D^T K D)^T = 0$. So,

$$\partial_i \Lambda = D^T \partial_i K D.$$

Thus,

$$\begin{aligned}\partial_j\partial_i\Lambda &= \partial_j D^T\partial_i K D + D^T\partial_j\partial_i K D + D^T\partial_i K\partial_j D\\ &= \partial_j D^T\partial_i K D + (\partial_j D^T\partial_i K D)^T\end{aligned}$$

because K is linear in γ.

Consider the definition of the differential in [1],

$$D_{\vec{\kappa}}\Lambda = \frac{d}{dt}B_{\gamma+t\kappa}(x,y)|_{t=0} = \sum \kappa_{i,j}\nabla_{i,j}u\nabla_{i,j}w$$

which can be thought of as a directional derivative in the direction κ, so not surprisingly, we can relate the two forms of the differential and show that they are in fact equivalent.

4.1. **Rank of the Differential.** If we could show the differential had full rank, then by the implicit function theorem, the map itself would be locally invertible, which is one step closer to showing it is globally invertible. To show that the differential has full rank, we need to show that the kernel is trivial.

So we need to show that $D^T(D_\kappa K)D = 0$ implies that $D_\kappa K = 0$. First we will prove a motivating lemma.

Lemma 4.1. *If X is a real matrix, and $X^TX = 0$, then $X = 0$.*

Proof. Let y be a vector of appropriate length, then $y^TX^TXy = (Xy)^T(Xy) = ||Xy||^2 = 0 \Rightarrow Xy = 0 \Rightarrow X = 0$. □

So if we can factor $D_\kappa K$, then we can apply the lemma. The following definition motivates such a factorization.

Definition 4.2 (Oriented Incidence Matrix)**.** An oriented incidence matrix M of an undirected graph is a matrix with row dimension equal to the number of vertices and column dimension equal to the number of edges and

$$M_{ij} = \begin{cases}\pm 1, & \text{if vertex i is incident to edge j}\\ 0, & \text{otherwise}\end{cases}$$

where the sign is chosen so that the two entries corresponding to the two vertices of an edge have opposite sign.

Now, $D_\kappa K$ can be factored as $D_\kappa K = H^TEH$, where H^T is an oriented incidence matrix for the graph, and E is a diagonal matrix with the diagonal equal to κ.

Lemma 4.3. *If $\kappa \geq 0$ (by which we mean all entries in the κ vector are ≥ 0) and none of the rows in D are equivalent, then $D^T(D_\kappa K)D = 0$ implies that $D_\kappa K = 0$.*

Proof. $\kappa \geq 0$, so all of the diagonal entries of E are positive so, if we let F be the diagonal matrix with the square root of the diagonal entries of E, then $E = F^TF$ and $D^T(D_\kappa K)D = D^TH^TF^TFHD = (FHD)^T(FHD) = 0$, so by the previous lemma, $FHD = 0$. Each row in H corresponds to an edge in the graph, and every edge only has two vertices, so every row of H has only two non-zero entries, a 1 and a -1. Multiplying on the left by a matrix can be interpreted as saying a row in HD is a linear combination of the rows of D with weights from H. Because of the structure of H, then a row in the final product can only be 0 if two rows in D are equivalent. Since we assumed that none of the rows of D were equivalent, then F must be 0, so $H^TF^TFH = D_\kappa K = 0$. □

The problem is that we must assume that $\kappa \geq 0$. When considering directional derivatives, we can take a partial derivative in each variable and combine them appropriately to get any directional derivatives. However, in this case we have only shown that there is no positive linear combination of the columns that sums to 0, when we need to show that only the trivial linear combination sums to 0 to prove full rank. To do this we would have to allow κ to be both positive and negative. However then problems arise from decomposing E because we get a complex-valued F, and Lemma 4.1 does not hold for complex matrices. It would then be necessary to determine when $X^T X = 0$ implies $X = 0$ for complex matrices, and then the rest of the argument would hold. Note that using what would seem more appropriate for complex matrices, the conjugate transpose, would not work either because $X^\dagger X$ is positive semi definite and $D_\kappa K$ is not necessarily positive semi definite if κ is positive and negative.

5. Non-Linear Least Squares - Gauss-Newton and Levenberg-Marquardt

```
Levenberg-Marquardt( J, F, x )
initialize u
let f( x ) = 1/2*F(x)^T*F(x)
while iterationsDone < maxIterations
  solve for d in ( J(x)^T*J(x) + u*I )d = J(x)^T*F(x)
  if f( x + d ) < f( x )
    x = x + d;
    decrease u
    iterationsDone++
  else
    increase u
  end
end
```

We are looking for a point x^* such that f is minimal, which will necessarily be a stationary point meaning $\nabla f(x^*) = 0$. Once such a x^* is found we can recast the minimization as a root finding problem. If our initial guess is sufficiently close to x^*, we can use Newton's method to find the solution to $\nabla f(x) = 0$. If $d_k = x_{k+1} - x_k$, then by Newton's method, d_k should satisfy $Hf(x_k)d_k = -\nabla f(x_k)$, where Hf is the Hessian. The advantage of using Newton's method is that it has quadratic convergence, which roughly means a doubling of precision at each iteration, however computing the Hessian is generally impractical.

Fortunately, an unconstrained nonlinear least squares minimization problem has several advantages over a general unconstrained nonlinear minimization problem. Recall that

$$f = \frac{1}{2}\sum F_i^2.$$

So the functional can be expressed in terms of the individual terms and taking the derivative results in

$$\frac{\partial f}{\partial x_j} = \sum \frac{\partial F_i}{\partial x_j} F_i \Rightarrow f' = J^T F.$$

We see that to compute the gradient of f we simply need the Jacobian of F, which can be computed efficiently as in Section 4. And then similarly the Hessian matrix can be computed as

$$\frac{\partial^2 f}{\partial x_j x_k} = \sum_i \left(\frac{\partial F_i}{\partial x_k}\frac{\partial F_i}{\partial x_j} + F_i \frac{\partial^2 F_i}{\partial x_j \partial x_k} \right) \Rightarrow f'' = J^T J + \sum_i (F_i F_i'').$$

At first it does not seem like we have reduced the problem because we still need the Hessian matrices of F_i; but when we are near a solution, the residuals or F_i values $\to 0$, so $\sum_i (F_i F_i'') \to 0$, and $J^T J$ is a good approximation to the Hessian. So the approximate search direction d_k is the solution of the linear system $J^T J d_k = -J^T F$. This method is referred to as the Gauss-Newton method, and an additional constraint is usually imposed that forces the functional values to be decreasing.

Problems arise when the second order term of the Hessian is significant. The Levenberg-Marquardt algorithm avoids this problem by alternatively solving the following linear system $(J^T J + \mu_k I)d_k = -J^T F$ where $\mu_k > 0$. This method has several other advantages that result from $(J^T J + \mu I)$ being positive definite, thus the system always has a unique solution.

Definition 5.1 (Descent direction)**.** A direction h such that $h^T \nabla f < 0$.

Lemma 5.2. *d_k is a descent direction.*

Proof. $d_k = -(J^T J + \mu_k I)^{-1} J^T F$, so $(d_k)^T \nabla f = -(J^T F)^T (J^T J + \mu_k I)^{-1} (J^T F)$ and $(J^T J + \mu_k I)$ is positive definite, so $(J^T J + \mu_k I)^{-1}$ is positive definite. Therefore $-(J^T F)^T (J^T J + \mu_k I)^{-1} (J^T F) < 0$, so d_k is a descent direction. □

Because d_k is a descent direction, there exists an α such that $f(x_k + \alpha d_k) < f(x_k)$ and we can always move in a decreasing direction. As $\mu_k \to 0$, the algorithm reduces to Gauss-Newton and as μ_k gets large the equation simplifies to $d_k = -\frac{J^T F}{\mu_k}$, which is simply the steepest descent direction as in algorithms in Section 2. Thus μ can be adjusted at each step to avoid the problems inherent in Gauss-Newton. The key to the algorithm is controlling μ. Unfortunately we did not have time to try out different methods for controlling μ, but our program can be easily modified to use a different algorithm.

6. Physical Network

To see how the algorithm stands up to a real life situation, we created several physical networks and made measurements to put into our program. As a start, we built a $Y - Network$ using resistances of 1 Ohm, 47 Ohms, and 100 Ohms seen in Figure 3.

After making the measurements and running it through the program, we were able to clearly distinguish which edge corresponded with the right resistance (basically recovering the network). The largest error was about 15%, which is quite reasonable given the fact that the resistors had a 5% and the equipment added an additional 1.2% to 2% error.

The next step up from solving the Y-network was to work out the 2-by-2 grid network. For this problem, we modeled situations like the human body where there are lumps of resistances. In our case, we used 1 Ohm and 10 Ohm resistors with a lump in the middle of the grid as shown in Figure 4.

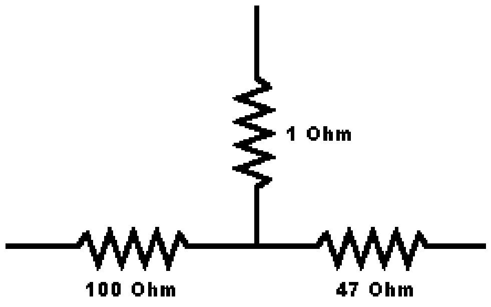

FIGURE 3. The Y-Network

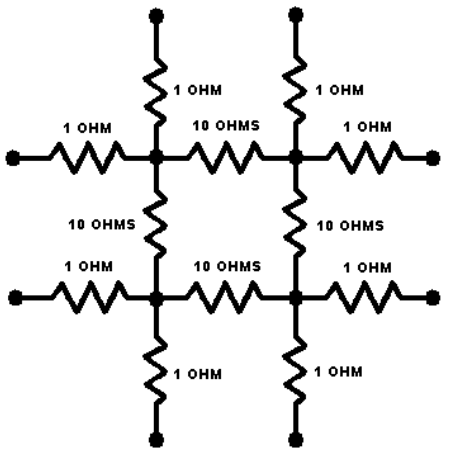

FIGURE 4. The 2-by-2 Grid Network with lumped resistors

On this network we ran into a slight problem. When we ran it through our program, we could definitely match up the conductances with the edges, given we know the only possibly values they could take. So, we were still, in a certain respect, able to recover the network from the response matrix. However, the recovered conductances were twice what the measured values were. Currently, we are unable to develop a concrete explanation for this, but we believe that it is due to systematic error when making the readings. Some possible causes of the error are in the meter readings, loss of power to perform the readings and giving off heat as well, and a poor source of voltage. Further investigation into this issue should be taken so that it can be determined whether it was a systematic error in measurements or a flaw in the algorithm.

7. FUTURE WORK

- Change the criteria for rounding edges, so the edges do not get fixed to the wrong value.

- Change the regularization term to deal with more general cases, instead of just clumped networks.
- Investigate whether it is possible to determine when $X^T X = 0$ implies $X = 0$ for complex X, possibly leveraging information from the graph.
- Implement a hybrid algorithm with Levenberg-Marquardt and a Quasi-Newton method. The Levenberg-Marquardt method has good convergence if $F(x) = 0$ at the minimum, but if there are large residuals, which might be the case with noisy data, the Quasi-Newton method would be more appropriate. In fact, we might even be able to use Newton's method because we can calculate the true Hessian as in Section 4.

8. Results

size	**time**	**iterations**
3	0m0.003s	5
4	0m0.006s	7
5	0m0.019s	8
6	0m0.063s	10
7	0m0.163s	11
8	0m0.403s	14
9	0m0.843s	15
10	0m1.564s	17
11	0m2.748s	18
12	0m5.033s	21
13	0m8.767s	24
14	0m13.976s	26
15	0m21.264s	26
16	0m28.872s	23
17	0m39.829s	22
18	0m56.577s	23
19	1m22.991s	24
20	1m58.056s	26

Figure 5. Square lattice networks of constant conductance one using Levenberg-Marquardt minimization with C implementation and initial guess of all twos.

References

[1] Curtis, B., and James A. Morrow. "Inverse Problems for Electrical Networks." Series on applied mathematics – Vol. 13. World Scientific, ©2000.

[2] Eric T. Chung, Tony F. Chan and Xue-Cheng Tai. "Electrical impedance tomography using level set representation and total variational regularization." November 12, 2003.

[3] K. Madsen, H.B. Nielsen, O. Tingleff. "Methods for Non-Linear Least Squares Problems." Informatics and Mathematical Modelling. Technical University of Denmark. ©2004.

size	Numerical Derivative		Analytic Derivative	
	time (sec)	maximum error	time (sec)	maximum error
3	0.11098	1.33e-015	0.041700	1.11e-015
4	0.11630	1.67e-015	0.097511	1.67e-015
5	0.17119	7.11e-015	0.081613	9.00e-013
6	0.23644	2.44e-014	0.129420	7.88e-012
7	0.35731	1.86e-012	0.164470	1.99e-010
8	0.75801	2.09e-012	0.270460	8.95e-010
9	1.7320	4.57e-011	0.535200	9.01e-013
10	3.6749	1.05e-009	0.788620	2.24e-012
11	9.216	1.53e-009	1.30440	4.26e-011
12	20.585	6.08e-009	2.91440	1.44e-008
13	54.664	5.63e-008	6.64590	1.32e-007
14	200.650	9.61e-007	78.8480	2.53e-006
15	1242.00	4.64e-005	115.910	9.17e-007
16	-	-	183.99	4.2e-005
17	-	-	337.83	5.4e-005
18	-	-	502.10	1.0089e-004
19	-	-	890.36	1.8003e-004
20	-	-	1256.30	7.0156e-004

FIGURE 6. Comparison of Levenberg-Marquardt algorithm on square lattice networks of constant conductance one, using numerical and analytic derivatives. The analytic derivative greatly decreases the running time of the algorithm.