# PLANAR RESISTIVE TOMOGRAPHY: AN INVERSE PROBLEM

by

Dharmesh M. Mehta

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May 22, 2000
[June 2000]



Author .....
Department of Electrical Engineering and Computer Science
June 22, 2000

Approved by ...
.....
Professor John L. Wyatt, Jr.
Thesis Supervisor

Accepted by .....
.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

PLANAR RESISTIVE TOM OGRAPHY:
AN INVERSE PROBLEM

by
Dharmesh M. Mehta

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 2000

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

## ABSTRACT

The primary motivation behind this thesis is to develop a more efficient algorithm for solving inverse resistive problems. Efficiency in this context refers to using a very coarse grid of measurements to solve backwards for a very fine grid of values that are both accurate and useful in modeling physical situations. This problem is modeled, simulated, and solved using MATLAB. Chapters 1 and 2 provide a background on numerical algorithms and the conventions that will be used in this document. Chapter 3 develops two building blocks for all other simulations: a "forward solver" and the associated "inverse solver." The solvers are written so that they can be used on any finite 2-dimensional, rectangular resistive grid. Given resistor values along the grid, the forward solver determines voltage values at all nodes for any specified current drive. The inverse problem does the opposite. Given voltage measurements under a specified current drive, it solves for the resistive grid that created such values. This paired set of solvers allows for both the generation of and the solution to any given situation. These solvers also help analyze the impact of noise, grid size, and quantity of current drives on efficiency. The next chapter focuses on developing a method that serves as a "Sparse data In, Dense grid Out" (SIDO) inverse solver. The basic idea behind such a solver is to use voltage measurements along a coarse grid to solve for a fine grid of resistances. There are a number of ways in which this can be accomplished. Chapter 6 will look at using an alternating technique, and Chapter 7 will focus on an application of Newton's Method. The alternating technique switches between solving for node voltages and solving for resistor values. This SIDO solver begins by using the given coarse voltage measurements

to infer both a fine grid of voltage values and a fine grid of resistors. This involves both the aforementioned inverse solver as well as some circuit approximation techniques. The alternating part of the algorithm involves switching between the update of the estimates for the voltage values using the estimate for the resistor values and the update of the estimates for the resistor values using the estimate for the voltage values. The second SIDO solver applies Newton's method. It also initializes the system with approximations for both the fine grid of voltage values and the fine grid of resistors. From there, this SIDO solver proceeds differently. The next step creates a linear approximation for the system by means of a Taylor series expansion about the current estimate. Instead of alternating between solving for voltage and resistor values, this solver simultaneously solves for the necessary update to both unknowns. The final chapter of this thesis presents careful analysis of each of these two different SIDO solvers. Factors to consider include their computation times, the accuracy of their final results, the coarseness of the original voltage measurements, the number of current drives used, and the advantages and shortcomings of each solver. It is also important to look at possible applications for these SIDO solvers. The focus will be on how to apply a SIDO solver to different situations as well as the benefits and advantages of using a SIDO solver as opposed to the original inverse solver.

Thesis Advisor: Professor John L. Wyatt, Jr.
Title: Professor of Electrical Engineering

TABLE OF CONTENTS

# LIST OF FIGURES

# ACKNOWLEDGMENTS

I would like to thank my thesis advisor, John L. Wyatt, Jr. for the invaluable technical support that he provided throughout this project. It was Professor Wyatt's enthusiasm and interest that encouraged me to pursue this project and to become engrossed in and excited about the material. Furthermore, I greatly appreciate that Professor Wyatt did not give up on this thesis in the face of many very complex obstacles.

I am also deeply indebted to three of my closest friends – Jae Kyoung Ro, Richa Shyam, and Rodney Eric Huang. Thanks to Rodney for being a great roommate and providing the space I needed to work on this thesis and for not complaining when things got a little crazy. I'd like to thank Richa for always caring, for knowing how to make a person laugh, and for also knowing when I needed to be left alone to get work done. And I'd like to thank Jae for always being there to talk, for being comforting in every crisis, and for helping me get through it all. And thanks to all three for their help in proofreading this thesis.

I would finally like to thank my family. My parents, Madhuker A. and Gita M. Mehta, have always provided immense amounts of support in every endeavor that I have undertaken. They have always tried to provide guidance, direction, and more than anything their love. Thanks to my brother and sister, Rajiv and Dina, for their wisdom, love, and their senses of humor.

This thesis was prepared at the Massachusetts Institute of Technology Research Laboratory for Electronics. Publication of this thesis does not constitute approval by the Massachusetts Institute of Technology of the findings or conclusions herein. It is published for the exchange and stimulation of ideas.

Permission is hereby granted to the Massachusetts Institute of Technology to reproduce this thesis.

Dharmesh M. Mehta

*Chapter 1*

# NUMERICAL ALGORITHMS FOR SOLVING OVERDETERMINED SYSTEMS

Scientists and engineers frequently have to solve overdetermined problems. Physical situations are described by a set of constitutive relations, but more measurements than required are usually taken to describe the situation. Many of these equations would be redundant if they were accurate, but because they are imprecise, these extra measurements become valuable. Systems where there are more equations than unknowns are known as overdetermined, and because of imprecise measurements or other external factors[1], these systems generally do not have an exact solution.

More precisely, this situation can be described as solving the equation:

$$\mathbf{Ax} = \mathbf{b}, \tag{1.1}$$

for the vector $\mathbf{x}$ where $\mathbf{A}$ has dimensions m x n with $m > n$. The residual error $\mathbf{r}$ of this equation is defined as $\mathbf{Ax} - \mathbf{b}$. In order to solve an overdetermined system, it is necessary to find a solution that minimizes some function of the residual error, $\mathbf{f(r)} = \mathbf{f(Ax - b)}$. The possibilities for the function $\mathbf{f(r)}$ are endless, but some of the most common include the one-norm,

[1] Other external factors can include noise in the system, varying physical conditions, or any approximation that is made (e.g. generally when solving an equation using Kirchoff's Voltage Law, current dissipation in wires connecting circuit elements is ignored and therefore a factor leading to an imprecise equation)

$f(\mathbf{r}) = \|\mathbf{r}\|_1 = \sum_{i=1}^{m} |r_i|$, the two-norm $f(\mathbf{r}) = \|\mathbf{r}\|_2 = \left( \sum_{i=1}^{m} |r_i|^2 \right)^{1/2}$, and the $\infty$-norm, $f(\mathbf{r}) = \|\mathbf{r}\|_\infty = \max_i |r_i|$.

## Section 1: Linear Least Squares

Linear least squares problems are those that solve the overdetermined system $\mathbf{Ax} = \mathbf{b}$ for the vector $\mathbf{x}$ by minimizing $\|\mathbf{Ax} - \mathbf{b}\|_2$. Since this is the only norm used in the rest of this document, it will be referred to as simply $\|\mathbf{Ax} - \mathbf{b}\|$. Because of the non-negative monotonic nature of this norm, this is equivalent to minimizing $\|\mathbf{Ax} - \mathbf{b}\|^2$.

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} - \mathbf{b}^T\mathbf{Ax} + \mathbf{b}^T\mathbf{b} \tag{1.2}$$

Taking the gradient of (1.2), setting it equal to zero, and rearranging terms:

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b} \Rightarrow \mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b} \tag{1.3}$$

This could be computed directly, but there are many other algorithms that solve this problem more efficiently. MATLAB makes use of a QR factorization[2] so that:

$$\mathbf{A} = \mathbf{QR}, \text{ where } \mathbf{R} \text{ is upper triangular} \tag{1.4}$$

$$\mathbf{R}^T\mathbf{Q}^T\mathbf{QRx} = \mathbf{R}^T\mathbf{Q}^T\mathbf{b} \tag{1.5}$$

[2] The choice among different QR factorization methods is dependent on the nature of the matrix **A**.

$$\mathbf{Rx} = \mathbf{Q}^{\mathbf{T}}\mathbf{b} \tag{1.6}$$

Solving the final upper-triangular system in (1.6) is much simpler than the original system. The longest step is the actual QR factorization. The computation time for this method is on the order of $2mn^2 - \frac{2}{3}n^3$ flops.

## Section 2: Singular Value Decomposition

The singular value decomposition (SVD) of a system of equations translates the original **A** matrix into one that is diagonal by choosing the correct bases for the domain and range spaces. In order to solve the equation $\mathbf{Ax} = \mathbf{b}$, new bases defined as $\mathbf{U}^{\mathbf{T}}\mathbf{b}$ and $\mathbf{V}^{\mathbf{T}}\mathbf{x}$ are chosen so that $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$, where **U** and **V** are square unitary matrices and $\Sigma \in \Re^{mxn}$ is diagonal[3]. The original equation then becomes:

$$\mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}\mathbf{x} = \mathbf{b} \tag{1.7}$$

$$\mathbf{U}^{\mathbf{T}}\mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}\mathbf{x} = \mathbf{U}^{\mathbf{T}}\mathbf{b} \tag{1.8}$$

$$\Sigma\left(\mathbf{V}^{\mathbf{T}}\mathbf{x}\right) = \left(\mathbf{U}^{\mathbf{T}}\mathbf{b}\right) \tag{1.9}$$

Implementation of this method involves first solving the equation:

$$\Sigma\mathbf{w} = \mathbf{U}^{\mathbf{T}}\mathbf{b} \tag{1.10}$$

for the vector **w**. The value for **x** is then equal to **Vw**. The diagonal entries $\sigma_i$ of $\Sigma$ are non-negative and in non-increasing order. This format is very convenient

[3] Trefethen and Bau prove that such a decomposition for any matrix **A** is always possible.

for numerous calculations, including computing the condition number of the system[4]. Another advantage of the SVD is that although it is not the fastest method, it is one of the best algorithms for rank-deficient matrices. Other methods that use normal equations or QR factorization have undesirable instabilities for certain systems. Calculating the SVD has a computation time[5] on the order of $4mn^2 - \frac{4}{3}n^3 - \frac{2}{3}(m-n)^3$.

## Section 3: Condition Numbers

The condition of a problem refers to the response of the system function $\mathbf{f}(\mathbf{x})$ to small perturbations. A well-conditioned problem is one in which a small change in $\mathbf{x}$ will result in a small change in $\mathbf{f}(\mathbf{x})$, and an ill-conditioned problem is one in which a small change in $\mathbf{x}$ will result in a large change in $\mathbf{f}(\mathbf{x})$.

There are several different types of condition numbers. The absolute condition number is defined as:

$$\hat{\kappa} = \sup_{\delta x} \frac{\|\delta f\|}{\|\delta x\|} \tag{1.11}$$

where $\delta \mathbf{f} = \mathbf{f}(\mathbf{x} + \delta \mathbf{x}) - \mathbf{f}(\mathbf{x})$. Note that if this is expressed in terms of the Jacobian of $\mathbf{f}$ at $\mathbf{x}$, this simplifies to $\hat{\kappa} = \|\mathbf{J}(\mathbf{x})\|$.

Relative condition numbers are concerned with the magnitude of these changes relative to their current values. So, the relative condition number of a function is defined as:

---

[4] Conditions numbers are covered in more detail in Section 3.

$$\kappa = \left( \frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right) \tag{1.12}$$

Putting this in terms of the Jacobian, $\kappa = \frac{\|J(x)\|}{\|f(x)\|/\|x\|}$. Relative condition numbers are generally far more important than absolute condition numbers because most errors introduced by computers that use floating point arithmetic are relative errors, not absolute errors. In terms of relative condition numbers, a value on the order of 1, 10, or 100 is generally indicative of a well-conditioned problem while a value on the order of $10^9$ or $10^{12}$ is considered ill-conditioned.

The final type of condition number that will be mentioned is the one used for the rest of this project. It is referred to as the condition number of a matrix. It is defined as:

$$\kappa(A) = \|A\| \cdot \|A^{-1}\| \tag{1.13}$$

The norm used is generally the two-norm because then this equation becomes[6]:

$$\kappa(A) = \frac{\sigma_1}{\sigma_n} \tag{1.14}$$

This ratio of the largest to smallest singular value is easily obtained when singular value decomposition is used to solve a linear least squares system. Note that when a system is singular, the smallest singular value is zero, and the condition number is infinity. Condition numbers will be very valuable in this project

---

[5] This formula is found by Trefethen and Bau as the amount of work for a three-step bidiagonalization.

[6] Trefethen and Bau derive this result in their analysis of condition numbers.

because they indicate whether a simulation will produce accurate or erratic results as a result of being either well-conditioned or ill-conditioned.

*Chapter 2*

# TWO-DIMENSIONAL RESISTIVE CIRCUIT MODELS

There are a variety of physical situations that can be modeled using a two-dimensional grid. From simple situations such as a sheet of metal to something as complex as the steam pipe system of a power plant, the methods developed here are very valuable in solving for unknown characteristics in a wide variety of systems.

## Section 1: Conventions

This research project will generally run simulations on a grid that measures 7 units by 11 units. Each vertex on the grid represents a node at which voltage measurements will be taken. Under any particular current drive, one node will serve as the current source, and some other node will serve as the sink. A resistor connects every pair of vertically and horizontally adjacent nodes. A typical physical problem involves using voltage measurements under different current drives to solve for all conductance (or equivalently, resistor) values.

There are N nodes in the grid, M of which will be points at which measurements are taken[7]. D different current drives will be run across the B conductances. For the sake of clarity, all matrices and vectors will appear in bold. Uppercase letters will be used for matrices and constants while lowercase letters will be used for vectors and individual variables.

[7] For the initial inverse solver M = N, but in the SIDO solver M << N.

Nodes are numbered starting in the upper left corner and proceeding down each successive column. Conductances are numbered beginning with the vertical resistors by proceeding downwards and then to the right. The horizontal resistors are then numbered proceeding to the right and then downward. The two figures below help clarify this explanation.

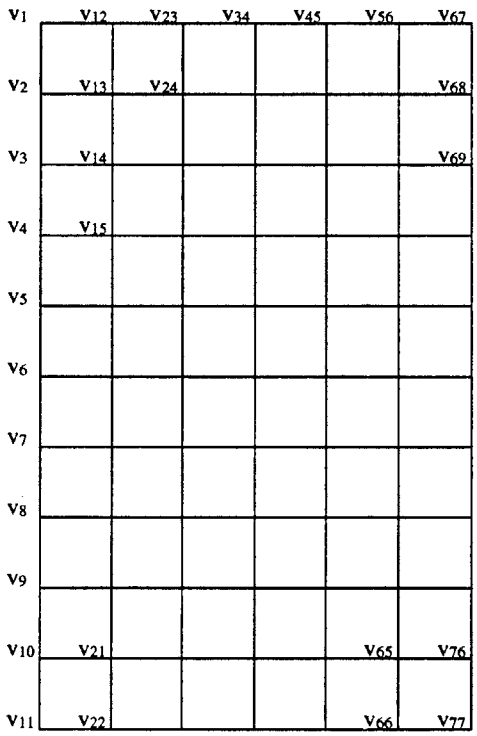



Figure 1: Numbering system for voltage measurements at each node in the fine grid.

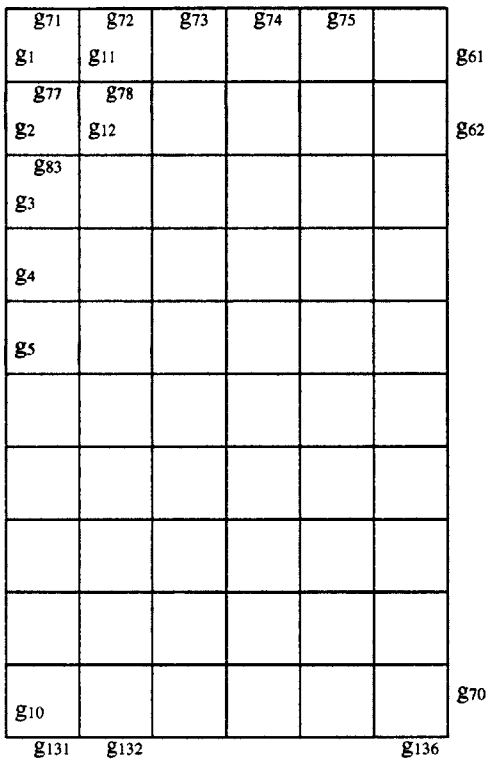



Figure 2: Numbering system for conductances within the fine grid.

In dealing with a coarse grid, only every $k^{th}$ voltage measurement in every $k^{th}$ row will be used. For the majority of the simulations, $k = 2$, and so the coarse grid uses about one out of every four nodes in the fine grid. This results in a grid that measures 6 nodes by 4 nodes. This grid will then have 24 voltage measurements, leaving 53 fine grid nodes for which no measurements are provided.

Because of the manner in which the resistors are situated, it is difficult to plot both the horizontal and vertical resistors on the same graph. To make the resistor plots both complete and understandable, three different plots will be made for each set of resistors. The first will show only vertical resistors, the

second will show only horizontal resistors, and the third will attempt to combine the two. Plotting a resistor's conductance value at the center of the physical resistor results in the data points shown below.

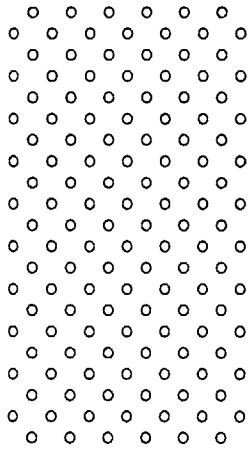


Figure 3: Locations at which resistor values can be considered centered.

The grid above is missing alternate locations in each row and column, which presents a problem for the plotting routine. To create a full grid of values for the third plot, it is necessary to approximate values, indicated as ⊗ in the figure below, by averaging their nearest neighbors.

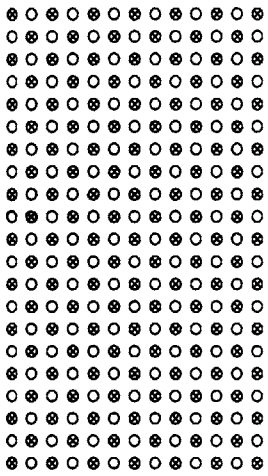


Figure 4: Full grid from which a 3-D plot of all resistors can be made.

Current is injected into one node and drained at any other node. There are many different ways that the source and sink can be chosen. This research endeavor will generally use eight different current drives. They are shown below.

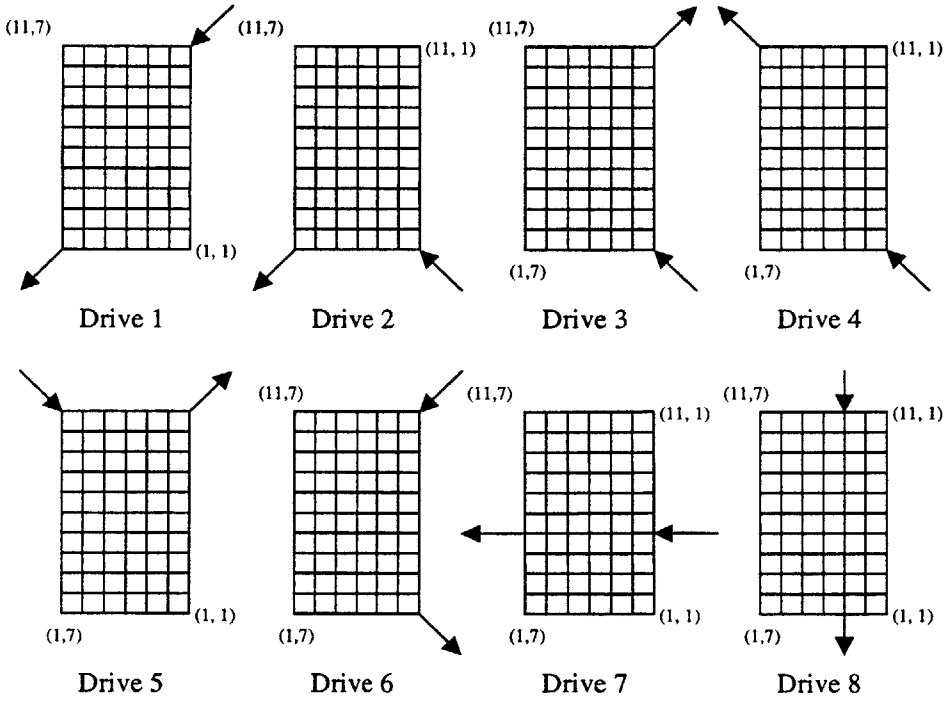



Figure 5: Arrows indicate flow for the eight current drives used in simulations.

Given a set of voltage measurements, these simulations try to solve for the conductances. However, the voltage measurements are generated from resistor values specified within the forward solver. Two resistive grids will be used. The first is a uniform grid of resistors with value 0.1 Ohms, while the second grid has a small "bump" of resistors with value 0.25 Ohms in the center. The resistor plots for the bump grid are shown in the figure below.

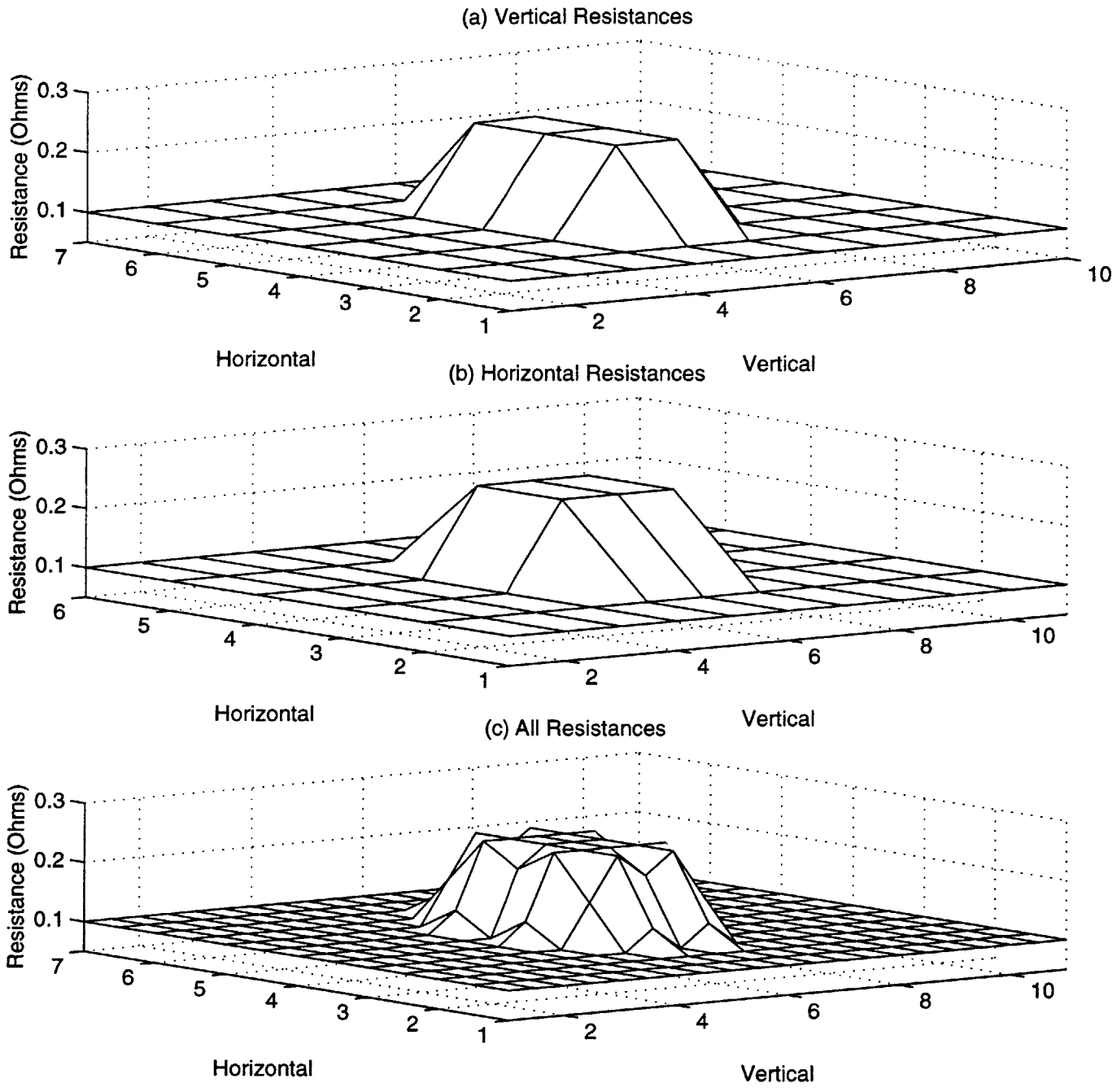



Figure 6: Resistor plots of "bump" grid.

Section 2: Kirchoff's Laws

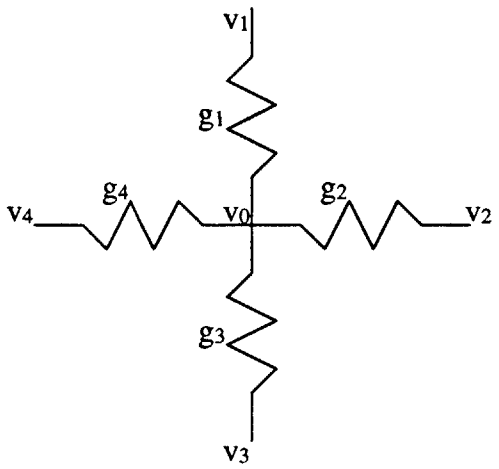



Figure 7: Typical node in the center of the grid.

Kirchoff's Current Law (KCL) is the fundamental equation for all systems in this project. In simplest terms, the law states that the sum of the currents entering a node must be equal to the sum of the currents leaving that node. Given a typical node that is not the current sink or source (see figure above), the relation between voltages and conductances can be written as:

$$\mathbf{g}_1(\mathbf{v}_1-\mathbf{v}_0)+\mathbf{g}_2(\mathbf{v}_2-\mathbf{v}_0)+\mathbf{g}_3(\mathbf{v}_3-\mathbf{v}_0)+\mathbf{g}_4(\mathbf{v}_4-\mathbf{v}_0)=0 \qquad (2.1)$$

If the node happens to be one of the drive nodes, then the right hand side of the above equation becomes the amount of leaving the node. KCL equations are written at all but one of the nodes. The last KCL equation is not used because it can be derived from the other $N-1$ KCL equations.

To this point, the model has been referred to as a circuit where the primary components are node voltages, drive currents, and conductances. However, the

only requirement for the model is to have components that can be described as follows. One component must be a flow variable, which has so far been referred to as "current." The flow element is injected at one node and drained at another. At all other nodes, the amount of flow inwards must be equal to the amount of flow outwards. Another component must describe the characteristics of the path between two adjacent nodes. This variable has been referred to as resistance, but it could easily be thought of as a velocity along a path, or any other item that physically describes that connection. The last component is an across variable that quantifies the product of the flow and the "resistance." Measurements are taken at each node so that the difference between two node measurements is equal to a sum of the products of the "resistances" with the "currents" passing through each resistance or in other words a sum of the across variables along the path. The situation is generalized to emphasize that this model can be used for a wide array of situations beyond the obvious circuit model.

## Section 3: Cut-Set Equations and Reference Voltage Node

Depending on the physical situation, a KCL equation may or may not be able to be written at the two drive nodes. The physical device supplying current may not allow for a connection from which voltage can be measured. However, a node's voltage is needed to write its KCL equation. If this should happen, the entire system becomes a set of equations with the right sides set to zero. This has two implications as seen from (2.1). First of all, there is no way to calculate the value of a particular conductance because if all the conductances were scaled by a factor k, the KCL equations would still hold. The second problem is that only voltage differences and not actual voltages enter the equations. So it becomes impossible to determine the actual voltage values.

Setting one node voltage to zero, and using this node as a reference point remedies the latter problem. For the sake of this paper, the last node ($v_{77}$) is designated as ground (voltage equal to zero) and will also be the one node where a KCL equation is not written. The former problem is ameliorated by means of a cut-set equation. The cut-set is a group of resistors through which a line can be drawn that splits the grid into two not necessarily equal halves. The only stipulation is that the current source must lie on one side of the line and the current sink must lie on the other. The figure below shows one example of a valid cut-set line for drive 4.

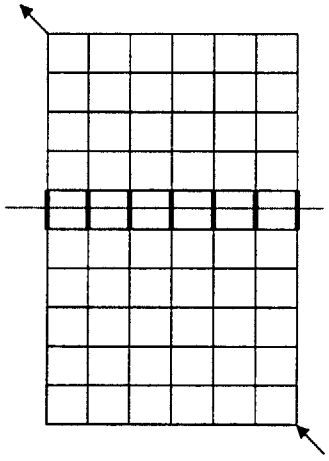


Figure 8: Example of cut-set line.

The resistors indicated with bold lines form the cut-set. The total current flow through the cut-set resistors must be equal to the total amount of current injected at the current source. This equation actually has a non-zero right side and will therefore enable the calculation of resistor magnitudes.

*Chapter 3*

# DISCRETIZED FORWARD AND INVERSE PROBLEMS

Before working with sparse matrices or considering how solutions may be affected by noise, it is essential to understand and solve the simplest case of this problem. Using the equations $\mathbf{Gv} = \mathbf{i}$ and $\mathbf{Vg} = \mathbf{i}$ to solve the forward and inverse problems, respectively, is not as straightforward as it seems. The complication lies in the calculation of the $\mathbf{G}$ and $\mathbf{V}$ matrices since both are complex functions of the actual conductances and voltages. Branch incidence matrices will be used to create a compact and clear method for solving these systems.

## Section 1: Branch Incidence Matrix

Consider the grid of six nodes and seven conductances below.

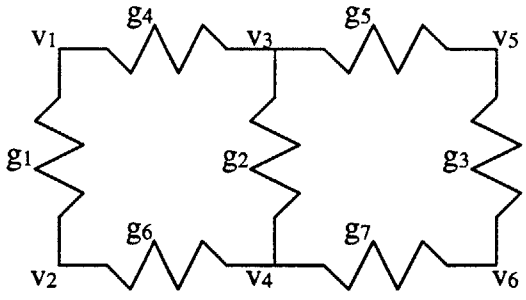



Figure 9: Simple example used to understand branch incidence matrices

Each conductance can be thought of as a branch between two nodes. One node in the grid is set as the reference or datum node and always has a voltage of zero. In this example, $\mathbf{v}_6$ is the reference node: $\mathbf{v}_6 = 0$. As stated earlier, this is necessary because all voltage values in the KCL equations are relative and not absolute. Across the $k^{th}$ branch, the direction of positive current flow relative to the orientation of positive voltage is as shown below:

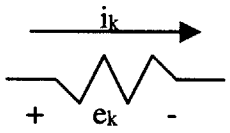



Figure 10: Orientation of current and voltage along each branch

For all further calculations, vertical resistors are oriented so that the branch travels downwards, and horizontal resistors are oriented so that the branch travels to the right. A branch incidence matrix has one row for each non-reference node and one column for each branch. So in this example, **A** is a 5x7 matrix. Each element $\mathbf{a}_{nb}$ of the matrix is defined as:

$$\mathbf{a}_{nb} = \begin{cases} +1, & \text{if branch b leaves node n} \\ -1, & \text{if branch b enters node n} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

For the given circuit,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \end{bmatrix} \tag{3.2}$$

Let

$$\mathbf{i} = \begin{bmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{i}_4 \\ \mathbf{i}_5 \\ \mathbf{i}_6 \\ \mathbf{i}_7 \end{bmatrix} \tag{3.3}$$

where $\mathbf{i}$ represents the vector of currents through each branch. Also let

$$\left(\mathbf{i}_{(d)}\right)_k = \begin{cases} +\mathbf{i}_{in}, & \text{if the } k^{th} \text{ node in the } d^{th} \text{ drive is the current source} \\ -\mathbf{i}_{in}, & \text{if the } k^{th} \text{ node in the } d^{th} \text{ drive is the current sink} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

where $\mathbf{i}_{(d)}$ represents the vector of currents driven into (or out of) each node under the $d^{th}$ drive. Then, Kirchoff's current law can be written as

$$\mathbf{A}\mathbf{i} = \mathbf{i}_{(d)}. \tag{3.5}$$

Let

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \\ \mathbf{v}_5 \end{bmatrix} \text{ and } \mathbf{e} = \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \\ \mathbf{e}_4 \\ \mathbf{e}_5 \\ \mathbf{e}_6 \\ \mathbf{e}_7 \end{bmatrix} \tag{3.6}$$

represent the node potentials relative to the reference node and the voltage drops across each branch. Then

$$\mathbf{e} = \mathbf{A}^{\mathrm{T}}\mathbf{v} \tag{3.7}$$

as a result of Kirchoff's Voltage Law, which states that the voltage drop across a particular branch is equal to the voltage at the start of the branch minus the voltage at the end of the branch.

The final necessary relationship involves the branch currents and voltage drops across each branch. The branch current is equal to the product of the voltage drop across and the conductance along that branch:

$$\mathbf{i} = \begin{bmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{i}_4 \\ \mathbf{i}_5 \\ \mathbf{i}_6 \\ \mathbf{i}_7 \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{g}_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{g}_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{g}_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{g}_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{g}_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{g}_7 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \\ \mathbf{e}_4 \\ \mathbf{e}_5 \\ \mathbf{e}_6 \\ \mathbf{e}_7 \end{bmatrix} = \mathbf{Ge} \tag{3.8}$$

Combining these equations,

$$\mathbf{AGA}^{\mathrm{T}}\mathbf{v}_{(\mathbf{d})} = \mathbf{i}_{(\mathbf{d})} \tag{3.9}$$

which is the relationship that will be used in both solvers.

Section 2: Mathematical Formulation of Solution to Forward Problem

In the forward problem, all resistor values are given, and the goal of the solver is to determine the voltage values at all nodes under a specified current drive. This is equivalent to solving (3.9) for **v**.

$$\mathbf{v}_{(d)} = \left(\mathbf{AGA}^{\mathbf{T}}\right)^{-1}\mathbf{i}_{(d)} \tag{3.10}$$

Since there are as many[8] unknowns as equations in (3.9), the solution for **v** is exact. The forward problem can be solved for each of the eight current drives using both the uniform and the bump grids as the values for the resistors. While there is no need for so many current drives at this point, the SIDO solver will require them, so they are also included. The figures below show the results of using the forward solver with the two different grids each of which have eight different current drive plots.

[8] This number of is equal to one less than the number of nodes since the voltage at the reference node is already specified

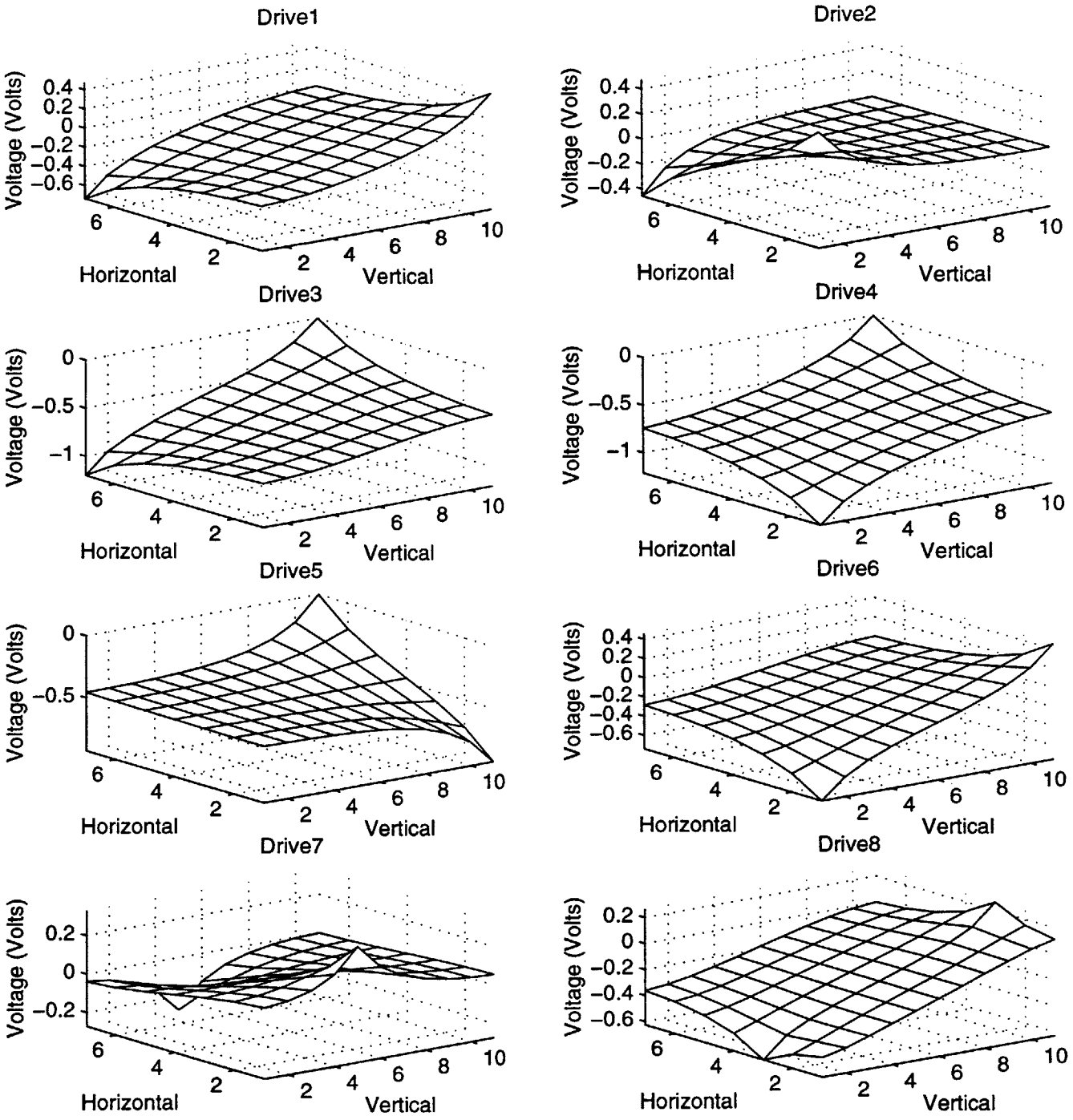



Figure 11: Solutions found by forward solver under different current drives using the uniform grid

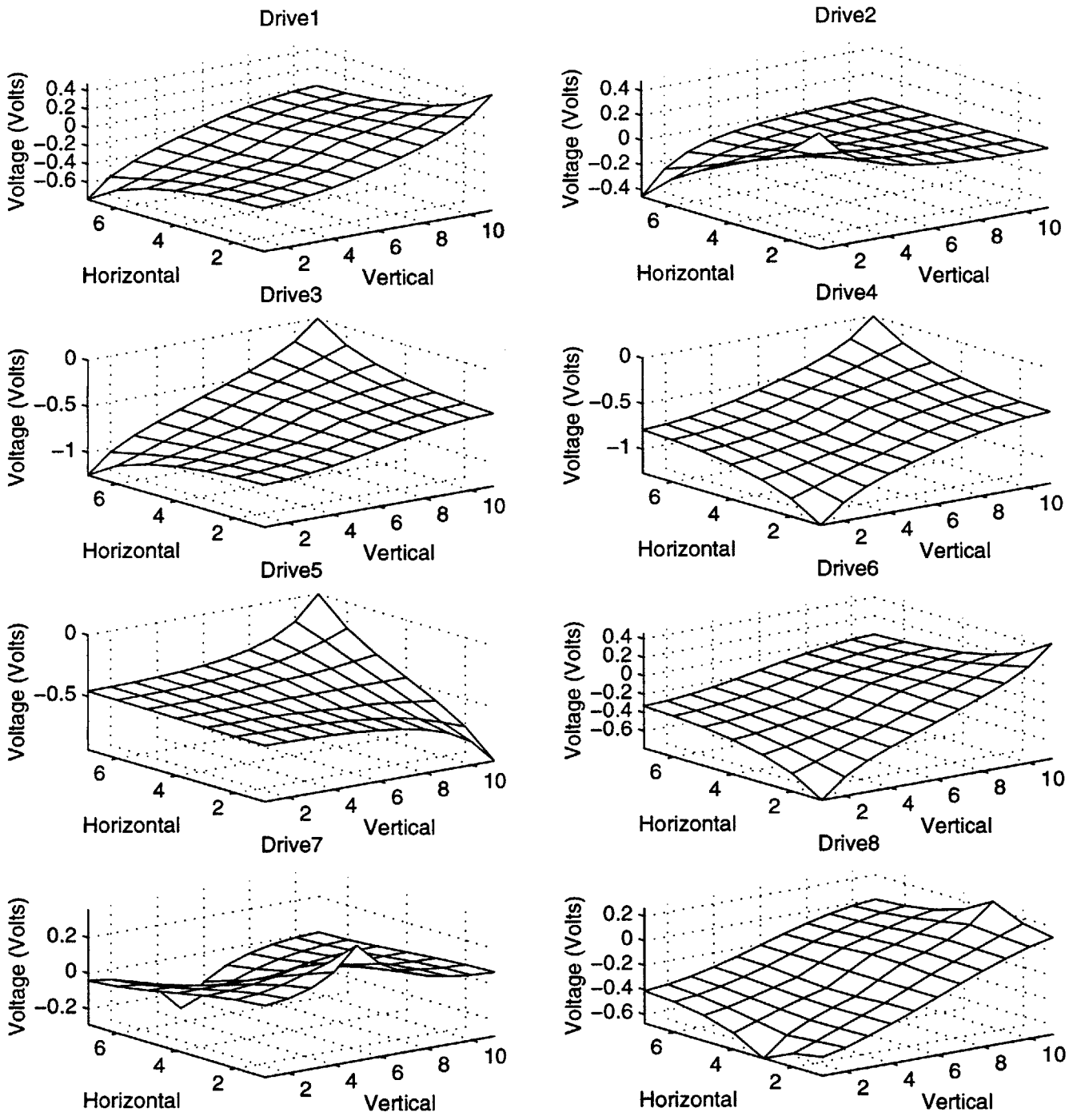



Figure 12: Solutions found by forward solver under different current drives using the bump grid

## Section 3: Mathematical Formulation of Overdetermined Inverse Problem

The inverse solver is given voltage measurements under different current drives. Each current drive provides $(N-1)$ equations,[9] and there are a total of B unknown conductances. If a grid's dimensions are L nodes by W nodes, then $N = L * W$, and $B = (2 * L * W) - (L + W)$. If D drives are used, then in order to have at least as many equations as unknowns, the following inequality must be satisfied:

$$D * (L * W - 1) \geq (2 * L * W) - (L + W) \tag{3.11}$$

This means that at least two drives must be used. As mentioned before, eight drives are used for now because this number is required by the SIDO solver.

The basic equation for the inverse problem is:

$$\mathbf{Vg} = \mathbf{i} \tag{3.12}$$

which is derived from (3.9). $\mathbf{V}$ and $\mathbf{i}$ incorporate information for all drives so that:

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_{(1)} \\ \mathbf{V}_{(2)} \\ \vdots \\ \mathbf{V}_{(D)} \end{bmatrix} \text{ and } \mathbf{i} = \begin{bmatrix} \mathbf{i}_{(1)} \\ \mathbf{i}_{(2)} \\ \vdots \\ \mathbf{i}_{(D)} \end{bmatrix} \tag{3.13}$$

To translate $\mathbf{AGA}^{\mathrm{T}}\mathbf{v}_{(k)} = \mathbf{i}_{(k)}$ into the form $\mathbf{V}_{(k)}\mathbf{g} = \mathbf{i}_{(k)}$, the matrix multiplication is broken down step by step.

---

[9] N-1 equations because no KCL equation is written at the reference node.

$$(\mathbf{AG})_{i,j} = \sum_m \mathbf{A}_{i,m}\mathbf{G}_{m,j} = \mathbf{A}_{i,j}\mathbf{G}_{j,j} = \mathbf{A}_{i,j}\mathbf{g}_j \text{, because } \mathbf{G} \text{ is diagonal} \quad (3.14)$$

$$\left(\mathbf{AGA}^T\right)_{i,j} = \sum_m (\mathbf{AG})_{i,m}\left(\mathbf{A}^T\right)_{m,j} = \sum_m \mathbf{A}_{i,m}\mathbf{g}_m\mathbf{A}_{j,m} \quad (3.15)$$

$$\begin{aligned}\left(\mathbf{AGA}^T\mathbf{v}_{(k)}\right)_i &= \sum_l\sum_m \mathbf{A}_{i,m}\mathbf{g}_m\mathbf{A}_{l,m}\left(\mathbf{v}_{(k)}\right)_l \\ &= \sum_m \mathbf{g}_m\mathbf{A}_{i,m}\sum_l \mathbf{A}_{l,m}\left(\mathbf{v}_{(k)}\right)_l \\ &= \sum_m \mathbf{g}_m\left(\mathbf{A}\left(\mathbf{diag}\left(\mathbf{v}_{(k)}^T\mathbf{A}\right)\right)\right)_{i,m} \\ &= \left(\mathbf{A}\left(\mathbf{diag}\left(\mathbf{v}_{(k)}^T\mathbf{A}\right)\right)\mathbf{g}\right)_i\end{aligned} \quad (3.16)$$

$$\mathbf{V}_{(k)} = \mathbf{A}\left(\mathbf{diag}\left(\mathbf{v}_{(k)}^T\mathbf{A}\right)\right) \quad (3.17)$$

By calculating $\mathbf{V}_{(k)}$ under each current drive, the full $\mathbf{V}$ matrix can be determined. The solution for the conductance values is then the linear least squares solution to equation (3.12). This can be accomplished by a variety of methods, but this thesis will use the singular value decomposition method because of its stability and ease of use for tracking condition numbers.

Using this method with a sufficient number of drives provides a near exact solution for $\mathbf{g}$. Imprecision in the final solution for $\mathbf{g}$ arises only as a result of noise in the voltage values used by the solver. The figures below show the resistor values found by the inverse solver when provided with voltage measurements from the forward solver under each of the aforementioned eight drives for the two different resistive grids.

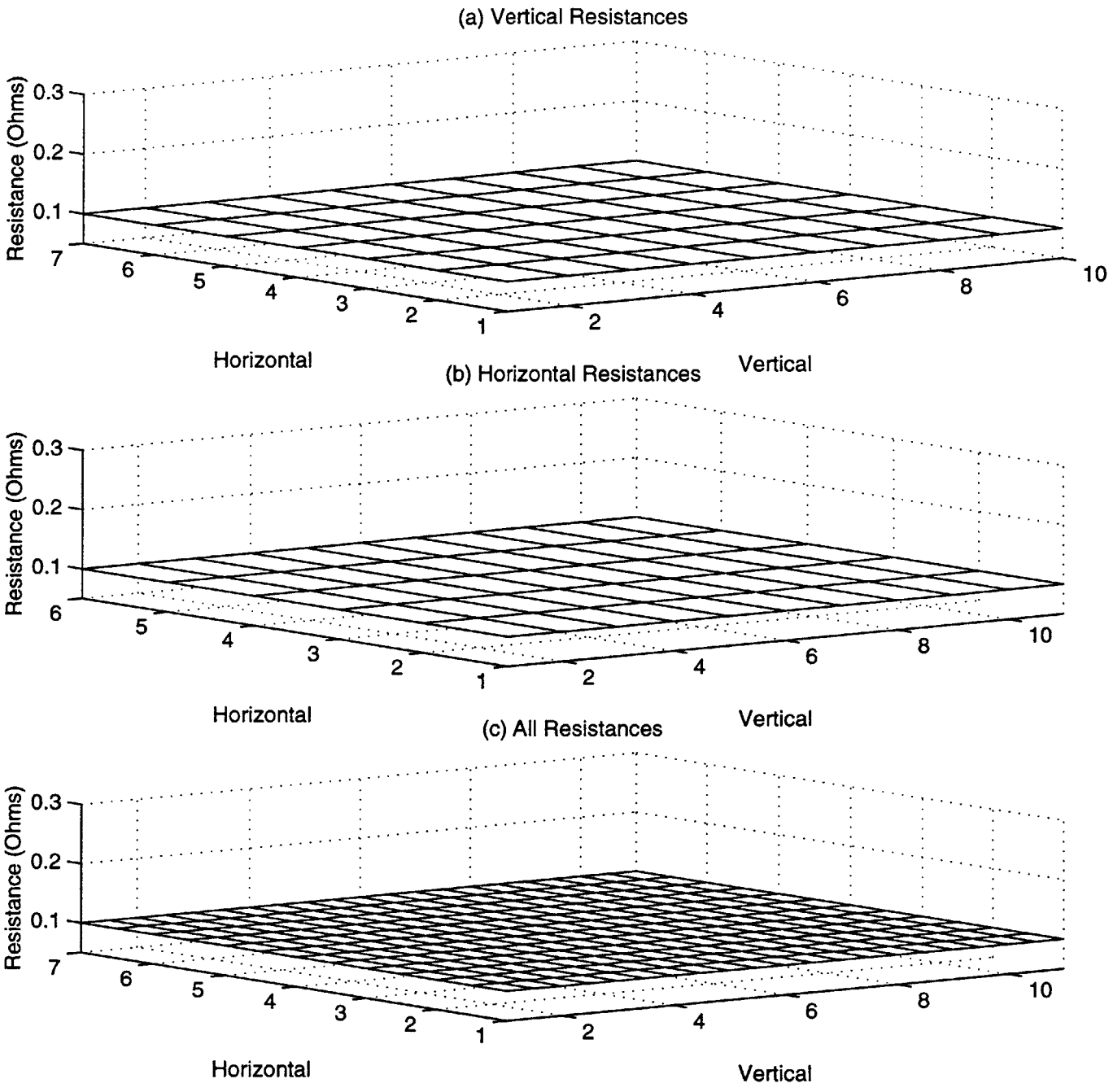



Figure 13: Inverse solver result from voltage grids calculated by forward solver with flat grid.

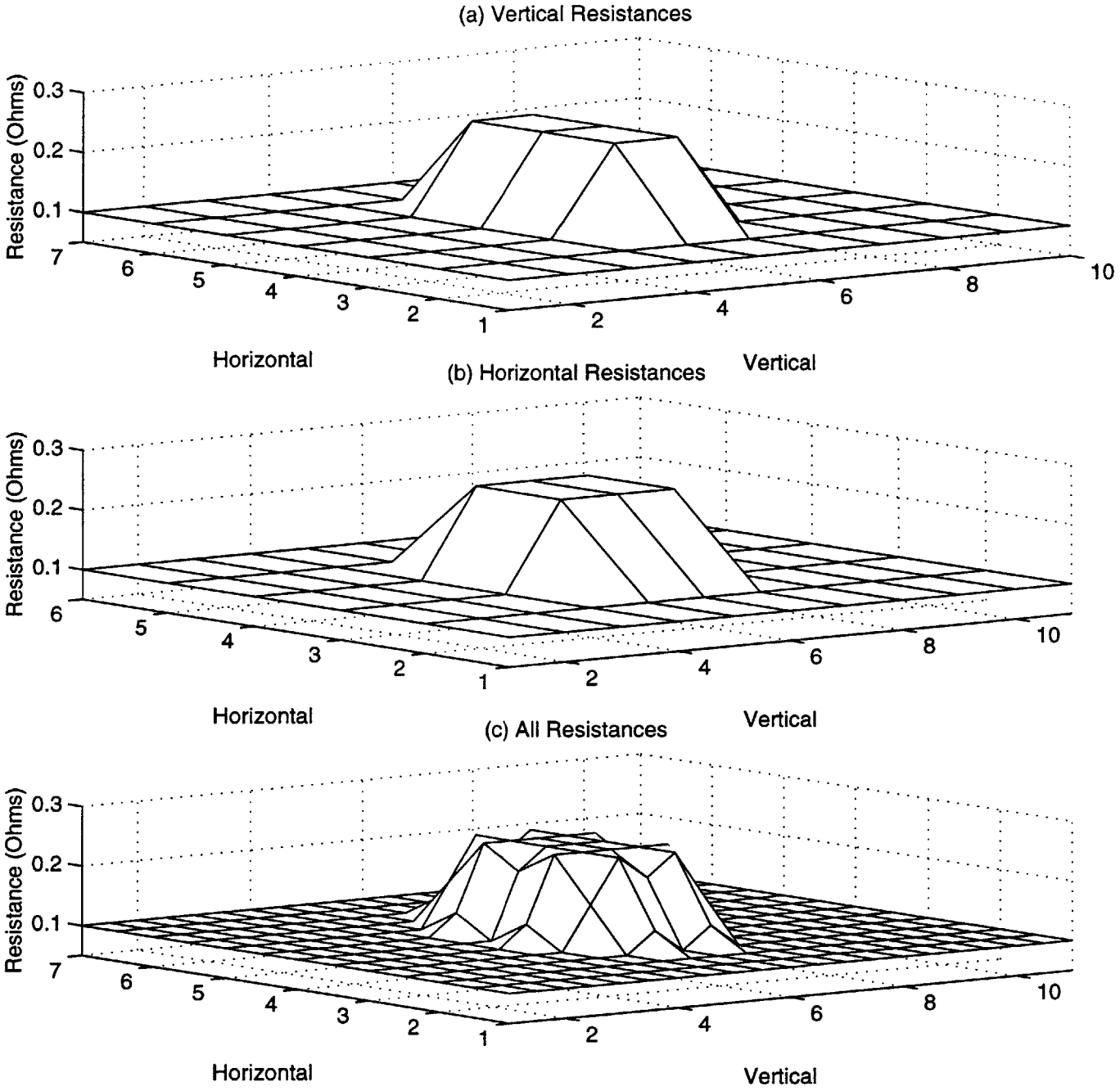



Figure 14: Inverse solver result from voltage grids calculated by forward solver with bump grid.

## Section 4: Analysis of Forward and Inverse Methods

The biggest difference between the above inverse solver and an actual physical situation is that real measurements are filled with noise. Without noise, both the forward and inverse solvers would have no problem producing an exact solution. To analyze the effectiveness of the inverse solver, it is necessary to see how accurately it determines resistor values under varying levels of noise. The relative magnitude of the noise is in proportion to the difference in adjacent voltage measurements since it is this difference that enters KCL equations. Adding in white noise of standard deviation $10^{-4}$ and $10^{-3}$ alters the voltage values generated by the forward solver under each of the eight current drives. These standard deviations correspond to measurement error on the order of 1% and 10%, respectively. The following figures show the results. As would be expected, an increase in the magnitude of noise causes the plots to move further away from the exact solution.

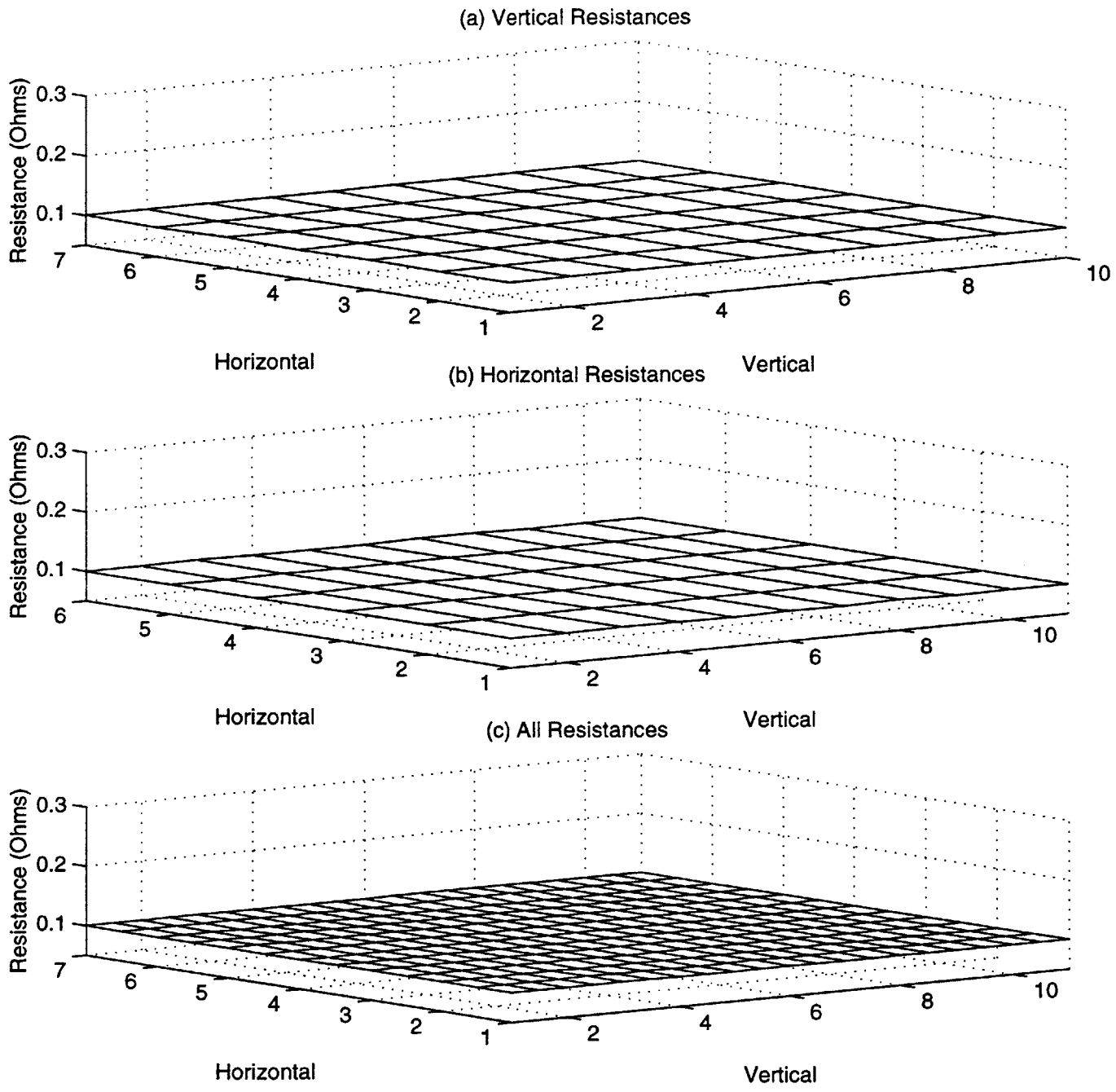



Figure 15: Result of solving noisy uniform grid problem with 1% measurement error

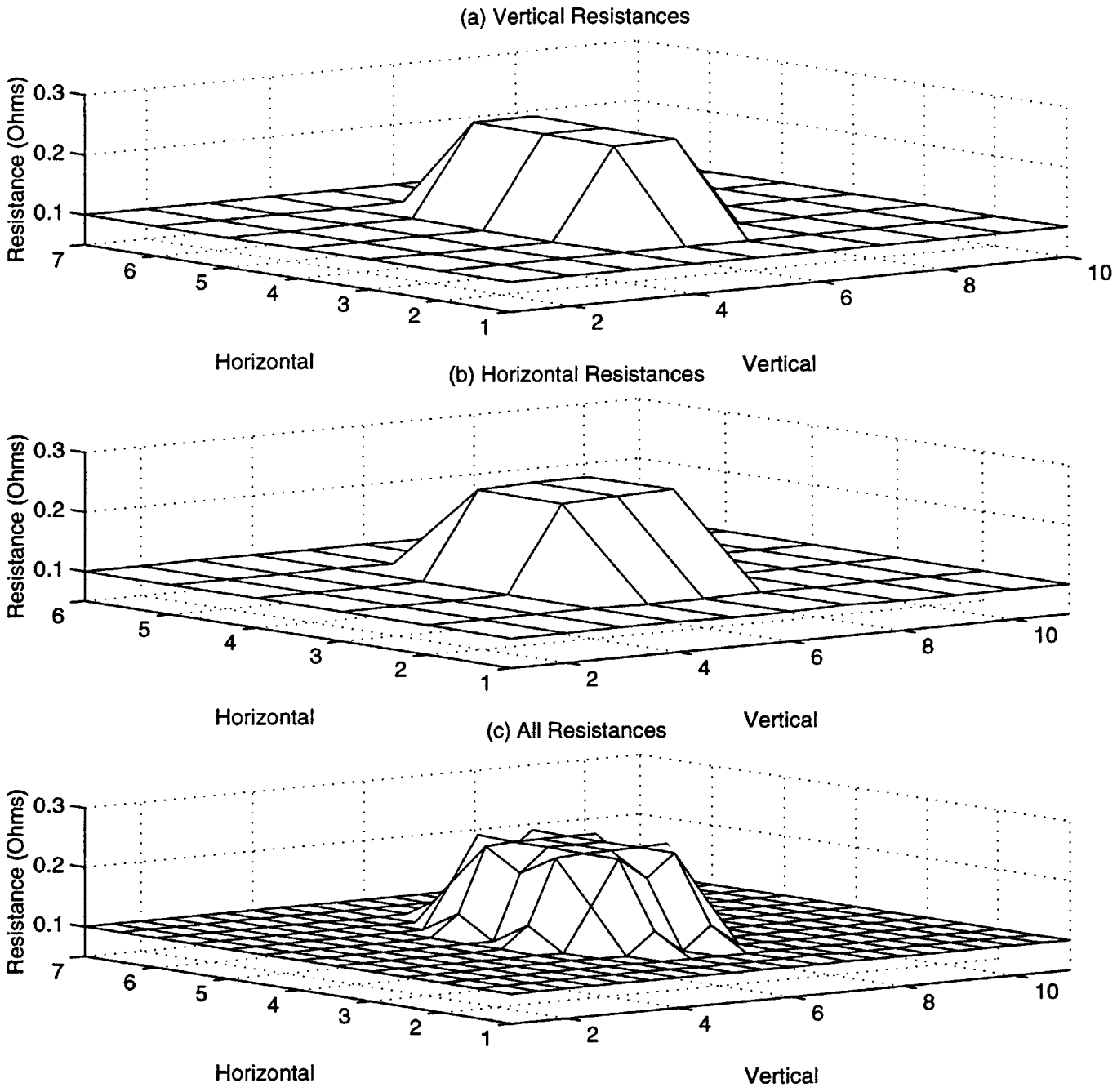



Figure 16: Result of solving noisy bump grid problem with 1% measurement error

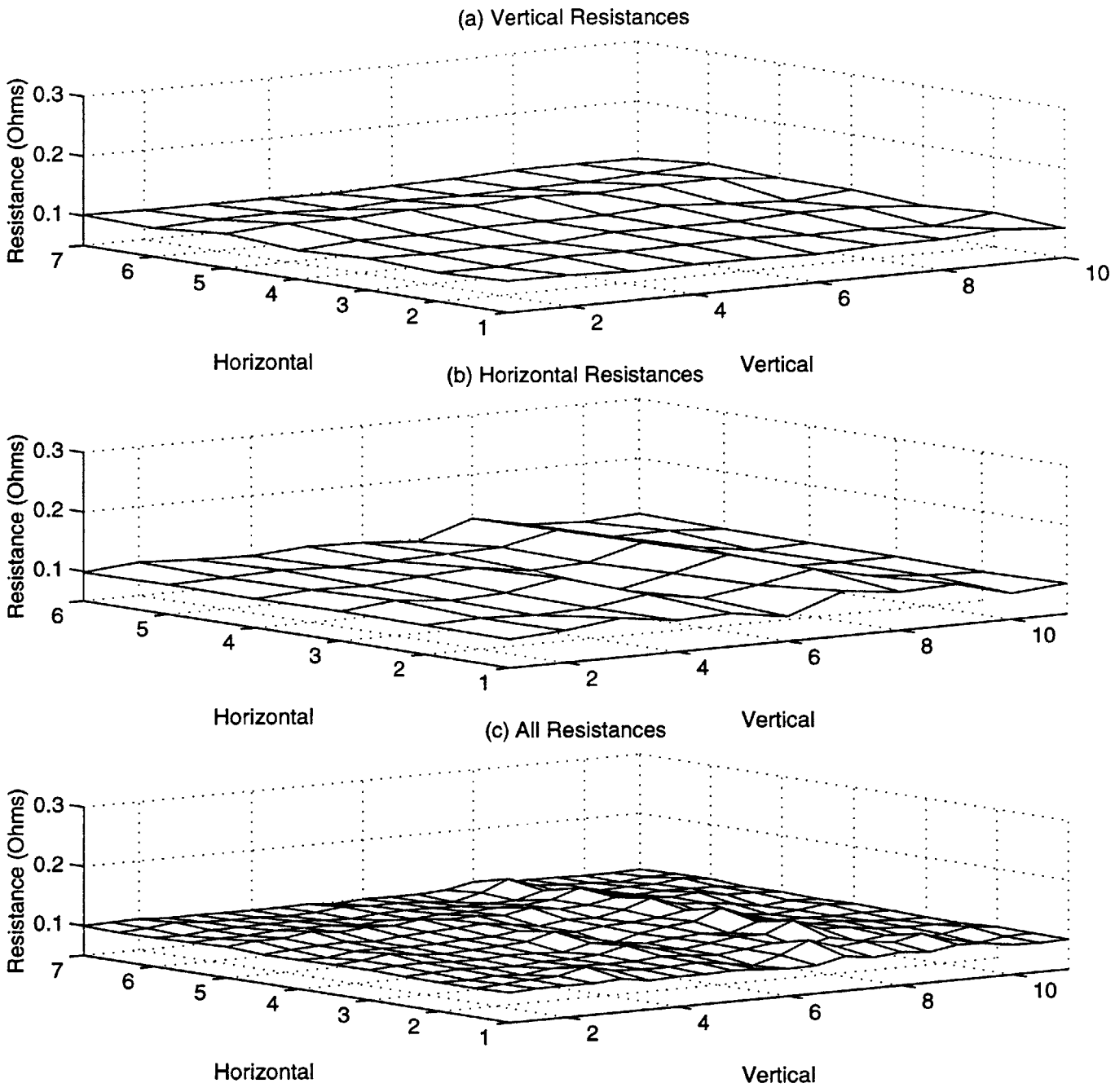



Figure 17: Result of solving noisy uniform grid problem with 10% measurement error

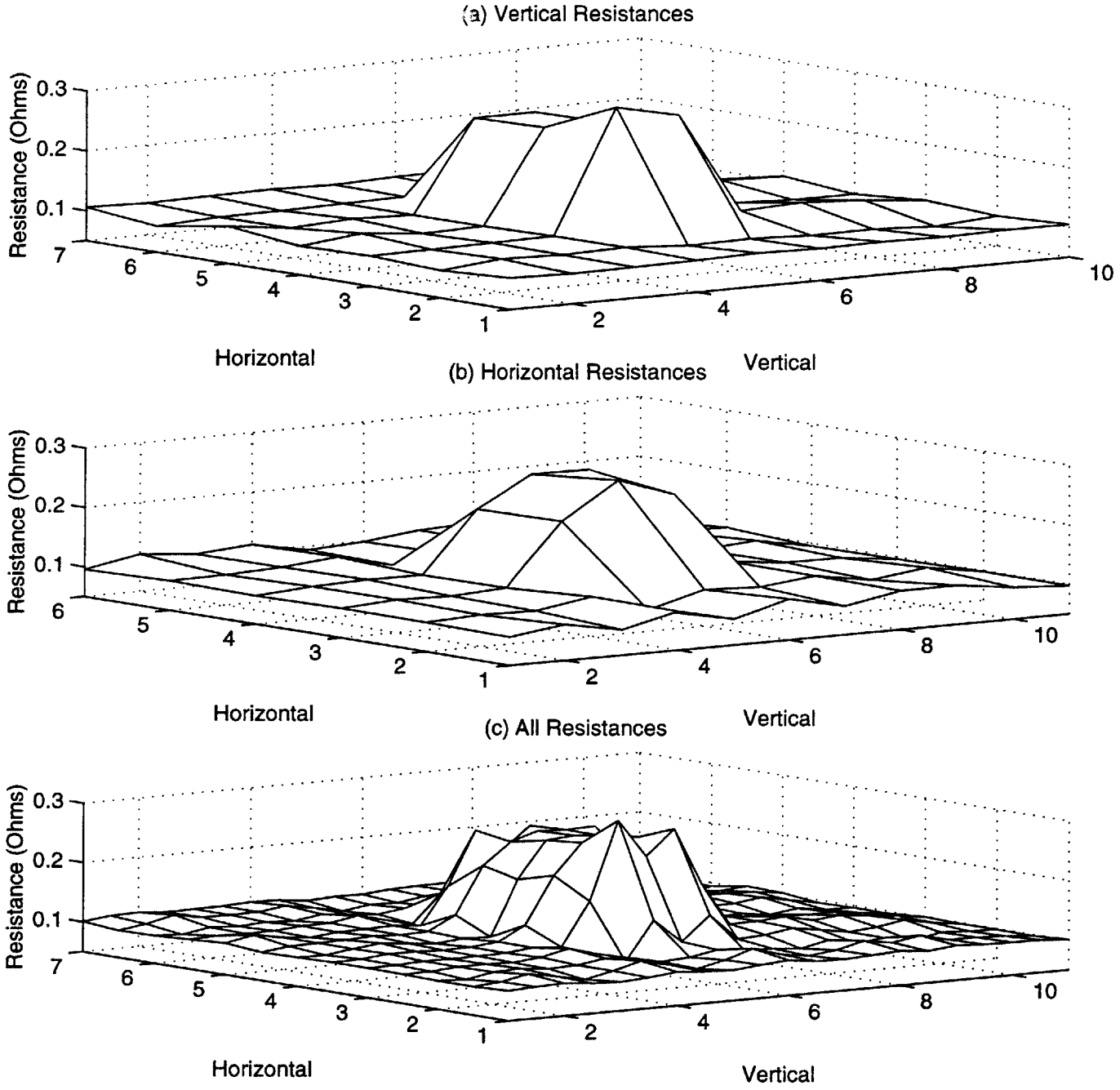



Figure 18: Result of solving noisy bump grid problem with 10% measurement error

*Chapter 4*

# SPARSE DATA IN / DENSE DATA OUT (SIDO) SOLVER

The goal in this project is not only to develop a forward and inverse solver, but also to create an inverse solver that can determine an accurate grid of conductances given as little information as possible. "As little information as possible" refers to developing a solver that is provided with voltage measurements at a small subset of the nodes. In a physical application, "voltage" measurements would be determined by means of some physical device tied to each node. Decreasing the number of such devices would help lower the cost of implementing the system. It is often more expensive to take a greater number of node measurements under fewer current drives that it is to take fewer node measurements under a greater number of current drives. The fundamental objective behind the SIDO solver is to take measurements under more current drives but on a coarser grid and still be able to infer an accurate fine grid of resistances.

## Section 1: Overview

The last section of Chapter 7 will look at how the solver is affected by using a even sparser set of voltage measurements, but everywhere else, the coarse grid will contain about one out of every four nodes in the fine grid. The figure below shows where voltage measurements are made in relation to the complete fine grid.

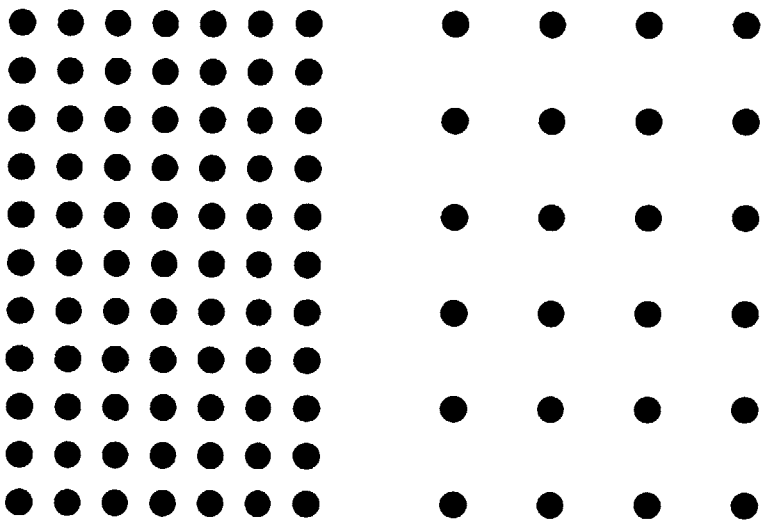



Figure 19: (a) All nodes in the fine grid. (b) Nodes where voltage measurements are made.

If the original grid measures 11 nodes by 7 nodes, then the coarse grid measures 6 nodes by 4 nodes. Voltage measurements are taken at 24 nodes, but only 23 KCL equations [10] can be written for each drive, and so D drives will create 23D KCL equations. The other 53 node voltages under each drive, as well as the conductances, are not known. The goal of the solver is to determine the value of the 136 unknown conductances. To be able to solve the system, there must be at least as many equations as unknowns. In other words:

$$23D \geq 136$$
$$D \geq 5\frac{21}{23} \tag{4.1}$$

[10] The equation at the datum node is redundant and does not provide useful information

Eight drives are used because this more than satisfies the inequality in (4.1). To distinguish between nodes where voltage measurements are taken and those where they are not, the voltage vector, originally denoted **v**, will be broken into:

$$\mathbf{v} = \begin{bmatrix} \mathbf{e} \\ \mathbf{u} \end{bmatrix} \tag{4.2}$$

where **e** contains the voltage values at the measured coarse nodes, and **u** contains the voltage values at the unmeasured nodes.

The SIDO solver begins by making an initial estimate for the fine grid of conductances. Then, using the information[11] it has, the solver iterates in order to improve this approximation.

The initialization of the fine grid of conductances can be accomplished by one of two ways. One method is to take the coarse voltage measurements, interpolate fine grids of voltage measurements, and then solve for a fine grid of conductances. The other method is just the opposite in that it first solves for a coarse grid of conductances and then interpolates a fine grid of conductances. In this thesis, the latter method will be used.

The decisions that must now be made include how to interpolate a fine grid of conductances and how to iterate through fine grids using the information currently available.

[11] The "information" is actually the estimate of the conductances and also estimates for the values of the voltages at nodes not along the coarse grid.

## Section 2: Error Metrics

In each iteration, the error in the current estimate of the unknowns is recorded to measure performance. However, the measured voltages along the coarse grid are the only exact values by which to gauge the present estimate. After each update of the conductances in the fine resistive grid, the forward solver is run to determine all voltage values (not just the unmeasured ones). An average absolute percent difference between the coarse voltage measurements and those that are calculated in this forward solver is determined. This average percent error is the first method of quantifying error. The next error metric, RMS current error $\sqrt{\frac{\|\mathbf{Gv}-\mathbf{i}\|^2}{\mathbf{n}}}$, quantifies the residual current errors in each iteration. This includes the errors in all KCL equations at all nodes under every current drive. RMS error has units of amperes and if the error in every KCL equation is equal, then the RMS error will be equal to that uniform value. The third error metric calculated is in the actual resistor values. Since all simulations start from specified resistor values, it is straightforward to calculate $\left|\frac{\mathbf{r}_{actual}-\mathbf{r}^{(k)}}{\mathbf{r}_{actual}}\right|$, where $\mathbf{r}_{actual}$ is the vector of resistors specified, and $\mathbf{r}^{(k)}$ is the current estimate of $\mathbf{r}$. Lastly, the condition number is tracked throughout the iterations. There is only one condition number in the inverse part of the algorithm. However, there are D condition numbers in the forward step since this stem involves solving D sets of equations, one for each current drive. Only the worst-case (highest) condition number is recorded in the forward step.

The coarse nodes voltage error gives the average over all the coarse nodes of the absolute value of the percentage error at the end of the $k^{th}$ iteration. It is obtained by solving:

$$\mathbf{G}^{(k)}\mathbf{v}_{(d)}^{(k)} = \mathbf{i}_{(d)},\ 1 \le d \le D \tag{4.3}$$

for the voltages that would result if current drive d were applied to the fine grid of conductances $\mathbf{G}^{(k)}$. Only the calculated voltages at the coarse nodes, referred to as $\mathbf{e}_{calc,(d)}^{(k)}$, are used in the error metric because exact values are only known at the coarse nodes. The error metric is:

$$\text{Coarse nodes voltage error} = \frac{100}{ND}\sum_{d=1}^{D}\sum_{n=1}^{N}\left|\frac{\mathbf{e}_{calc,(d),n}^{(k)} - \mathbf{e}_{(d),n}}{\mathbf{e}_{(d),n}}\right| \tag{4.4}$$

where n indexes the N coarse nodes, $\mathbf{e}_{(d),n}$ is the measured voltage at coarse node n resulting from drive d, and $\mathbf{e}_{calc,(d),n}^{(k)}$ is the value predicted for $\mathbf{e}_{(d),n}$ by using fine grid estimate $\mathbf{G}^{(k)}$.

The RMS current error calculates the square root of the average squared residual current error over all KCL equations under all drives at the end of the $k^{th}$ iteration. It is obtained using both the conductance estimate $\mathbf{G}^{(k)}$ and the unmeasured (fine node) voltage estimate $\mathbf{u}_{(d)}^{(k)}$. The unmeasured voltages $\mathbf{u}_{(d)}^{(k)}$ are combined with the measured coarse voltages $\mathbf{e}_{(d)}$ to obtain the full fine grid of voltages $\mathbf{v}_{(d)}^{(k)}$. The error metric is:

$$\text{RMS current error} = \sqrt{\frac{\sum_{d=1}^{D}\sum_{n=1}^{N}\left(\left\|\mathbf{G}^{(k)}\mathbf{v}_{(d)}^{(k)} - \mathbf{i}_{(d)}\right\|^2\right)_n}{ND}} \tag{4.5}$$

where n indexes the N coarse nodes.

The exact resistor values are known[12]. The resistor error calculation creates an average absolute percent difference between the actual values for the resistors, $\mathbf{r}_{\text{actual}}$, and the present estimate, $\mathbf{r}^{(k)}$. The error metric is:

$$\text{Resistor error} = \frac{100}{N}\sum_{n=1}^{N}\left|\frac{\mathbf{r}^{(k)} - \mathbf{r}_{\text{actual}}}{\mathbf{r}_{\text{actual}}}\right| \tag{4.6}$$

where n indexes the N coarse nodes.

[12] These resistor values are known because they are the ones originally specified in the forward problem

*Chapter 5*

# COARSE TO FINE INTERPOLATION

The interpolation of a fine grid of conductances from a coarse grid is much more complicated than it first seems. Two major complications arise. The first involves the representation of a coarse square as a set of four fine squares, as shown below.

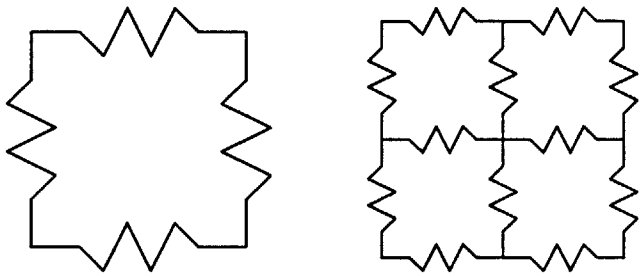



Figure 20: (a) Typical coarse grid block. (b) Same physical area in the fine grid.

The second complication involves deciding how to connect such blocks and how properties will change or remain the same at the boundaries.

## Section 1: Constitutive Laws

There are a few important issues to consider when interpolating from a coarse to fine grid. First, the two grids should provide the same voltage measurements at the coarse nodes under any current drive. Another factor to consider is the

degree to which horizontal and vertical resistors should affect each other. There is also the question of how the proximity of fine resistors to a coarse resistor affects their values. Other logical requirements are that a uniform coarse grid should result in a uniform fine grid, and that a ramped coarse grid should result in a ramped fine grid.

The choice of an interpolation method is quite complex for a number of reasons. First of all, there are not enough theoretically proven physical equations to exactly specify the values of the fine resistors. Furthermore, there are a number of facts that seem like logical attributes, but are not mathematical theorems. A decision must be made about which properties to turn into equations and which attributes to ignore. After deciding how a single block in the coarse grid translates into a set of four blocks in the fine grid, it is also necessary to determine how two adjacent blocks affect each other. The simplest solution may be to average the two overlapping values.

## Section 2: "Twelve to Three" Approximation

The first method used is referred to as a "12 to 3" interpolation. As shown in the figure below, twelve resistors in the fine grid replace the four in the coarse grid.

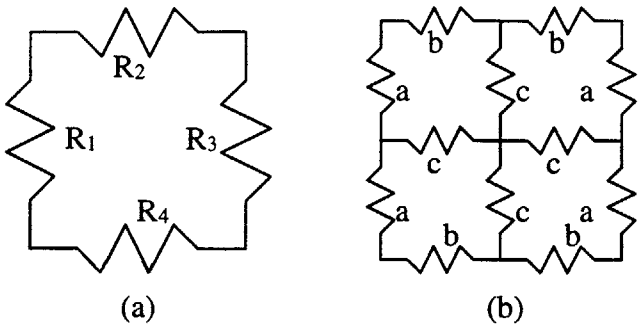



Figure 21: (a) Four resistors in a coarse grid block
(b) Same physical area in fine grid

The twelve resistors are approximated as having one of three values:

$$\mathbf{a} = \frac{\mathbf{R}_1 + \mathbf{R}_3}{2} \quad \mathbf{b} = \frac{\mathbf{R}_2 + \mathbf{R}_4}{2} \quad \mathbf{c} = \frac{\mathbf{R}_1 + \mathbf{R}_2 + \mathbf{R}_3 + \mathbf{R}_4}{4} \tag{5.1}$$

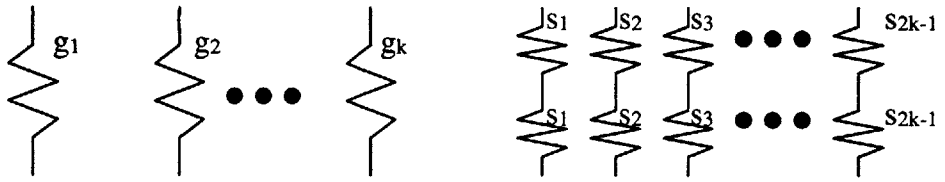
When two blocks are placed side by side, this creates overlapping resistors. Their values are calculated by averaging the approximations from each individual block.

This method has a number of shortcomings. If both grids are driven by the same current drive, then the fine and coarse grids will produce different voltage values at the coarse nodes. This approximation also arbitrarily decides which resistors affect current flow in the horizontal and vertical directions. These decisions are not the result of concrete formulas or theorems.

## Section 3: Linear Interpolation

The linear interpolation method chooses some desirable properties and makes them into constraints seeking just enough constraints to uniquely determine the method. The first stipulation is that fine horizontal conductances are a linear

function of coarse horizontal conductances. Similarly, the fine vertical conductances are a linear function of the coarse vertical conductances. Furthermore, the relationship is the same in each case, so by specifying the relationship in one direction, the entire interpolation method is defined. Looking at just the vertical direction, the goal is then to turn the situation on the left into the situation on the right as shown below.




(a) (b)

Figure 22: (a) Vertical conductances in coarse grid.
(b) Vertical conductances in fine grid.

In order for the fine conductances to be a linear function of the coarse conductances, there must exist some matrix L such that:

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{2k-1} \end{bmatrix} = \mathbf{A} \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} \tag{5.2}$$

where

$$\frac{1}{c_i} = \frac{1}{s_i} + \frac{1}{s_i} \tag{5.3}$$

In other words, c is the conductance of the series combination of two resistors, each with conductance s. The matrix **A** has 2k-1 rows and k columns. In order for the sum of all the fine conductances to be equal to the sum of all the coarse conductances, the sum of each column of **A** must be equal to 1. Another restriction imposed in this interpolation is that if the coarse conductances are uniform, then so are the fine conductances. This implies that the sum of each row in **A** is the same.

The sum of all the elements in **A** is equal to the number of columns times the sum of each column, which is the same as k times 1, or just k. If the sum of each row is **x**, then **x** times $2k-1$ must equal k. In other words, **x** must be equal to $\frac{k}{2k-1}$. A narrowness assumption is made stating that the conductance of any particular coarse resistor only affects the values of the fine resistor in its place and the fine resistors to its immediate left and right. This makes a number of entries in **A** equal to 0. Precisely, **A** can be written as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_{1,1} & 0 & \cdots & 0 & 0 \\ \mathbf{a}_{2,1} & \mathbf{a}_{2,2} & 0 & 0 & 0 \\ 0 & \mathbf{a}_{3,2} & \vdots & 0 & 0 \\ 0 & \mathbf{a}_{4,2} & 0 & 0 & 0 \\ \vdots & 0 & \mathbf{a}_{2i-2,i} & \vdots & \vdots \\ \vdots & \vdots & \mathbf{a}_{2i-1,i} & \vdots & \vdots \\ \vdots & \vdots & \mathbf{a}_{2i,i} & 0 & \vdots \\ 0 & 0 & 0 & \mathbf{a}_{2k-4,k-1} & \\ 0 & 0 & \vdots & \mathbf{a}_{2k-3,k-1} & \\ 0 & 0 & 0 & \mathbf{a}_{2k-2,k-1} & \mathbf{a}_{2k-2,k} \\ 0 & 0 & \cdots & 0 & \mathbf{a}_{2k-1.k} \end{bmatrix} \tag{5.4}$$

From the information above, the elements of **A** are of the form:

$$\mathbf{a}_{i,j} = \begin{cases} \dfrac{j-1}{2k-1}, & i = 2j-2, j = 2..k \\ \dfrac{k}{2k-1}, & i = 2j-1, j = 1..k \\ \dfrac{k-j}{2k-1}, & i = 2j, j = 1..k-1 \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

## Section 4: Interpolation by Averaging

The final interpolation method used is simply an averaging function. All coarse resistor values in the grid are averaged, and this value is assigned to each resistor in the fine grid. Although crude, it will later be shown that this method is sufficient in most cases. Furthermore, because of its simplicity, it is the preferred method.

*Chapter 6*

# ALTERNATING METHOD

Once a fine grid of resistor values is initialized, the SIDO solver runs through iterations that try to improve the initial estimate. Each iteration in the alternating method has two parts: a forward and inverse step. The forward half solves the equation $\mathbf{Gv} = \mathbf{i}$ for $\mathbf{v}$, but not in the usual manner. Since some of the voltage values are known from the given coarse measurements, the equation is only solved for the unknown values in the fine grid. The voltage values are determined such that the solution minimizes $\|\mathbf{Gv} - \mathbf{i}\|^2$. For the second part of each iteration, the measured voltage values (coarse grid) are combined with the values calculated in the forward half of the iteration. This provides a complete fine grid of voltage values. The inverse solver can now be used to determine a fine grid of resistors by again minimizing $\|\mathbf{Gv} - \mathbf{i}\|^2$.

## Section 1: Mathematical Formulation of SIDO Solver

The forward and inverse methods that are used on this fine grid are for the most part the same as those that have been used earlier. However, the forward solver incorporates a few changes since the values of voltages that have been measured are not updated. In solving $\mathbf{Gv} = \mathbf{i}$, an exact solution can normally be found because there are as many equations (one Kirchoff current equation per non-reference node) as voltages. Since some values in the $\mathbf{v}$ vector are known, it is necessary to alter the system to move all known information to the right while leaving unknown values to the left. The columns in the $\mathbf{G}$ matrix that correspond

to the known voltage values are deleted. The vector **i** must be adjusted to compensate for the missing columns of **G**. For example, if the $k^{th}$ element of the voltage vector is a known value, then the $k^{th}$ column of **G** is deleted and the vector **i** is decreased by the value of this $k^{th}$ column multiplied by the known $k^{th}$ element of the voltage vector. This results in a linear least squares problem because the matrix **G** will end up with more rows than columns.

Using the notation mentioned earlier, the equation $\mathbf{Gv} = \mathbf{i}$ can be rewritten as:

$$[\mathbf{G}_1 \quad \mathbf{G}_2] \cdot \begin{bmatrix} \mathbf{e} \\ \mathbf{u} \end{bmatrix} = \mathbf{i} \tag{6.1}$$

This is equivalent to:

$$\mathbf{G}_1\mathbf{e} + \mathbf{G}_2\mathbf{u} = \mathbf{i} \tag{6.2}$$

Therefore, the forward solver will solve the equation:

$$\mathbf{G}_2\mathbf{u} = \mathbf{i} - \mathbf{G}_1\mathbf{e} \tag{6.3}$$

for the unknown voltage vector **u**.

An exact mathematical description of the entire SIDO solver may help elucidate this process. Parenthesized subscripts and superscripts are used to indicate the current drive and round of iteration, respectively. The procedure is then:

1) Initialize **G** to $\mathbf{G}^{(0)}$, some initial estimate of all the conductances using coarse to fine interpolation.

2) Set $k = 0$.

3) Using the existing estimate $\mathbf{G}^{(k)}$ for the conductance matrix, solve each of these over determined equations in the least squares sense:

$$\begin{aligned}
\mathbf{G}^{(k)}\begin{bmatrix} \mathbf{e}_{(1)} \\ \mathbf{u}_{(1)}^{(k+1)} \end{bmatrix} &= \mathbf{i}_{(1)} \\
\mathbf{G}^{(k)}\begin{bmatrix} \mathbf{e}_{(2)} \\ \mathbf{u}_{(2)}^{(k+1)} \end{bmatrix} &= \mathbf{i}_{(2)} \\
&\vdots \\
\mathbf{G}^{(k)}\begin{bmatrix} \mathbf{e}_{(D)} \\ \mathbf{u}_{(D)}^{(k+1)} \end{bmatrix} &= \mathbf{i}_{(D)}
\end{aligned} \tag{6.4}$$

for the vectors $\mathbf{u}_{(1)}^{(k+1)}, \mathbf{u}_{(2)}^{(k+1)}, \ldots \mathbf{u}_{(D)}^{(k+1)}$ of node voltages on the fine grid using the method shown in (6.3).

4) Increment k.

5) Combine the most recent estimates $\mathbf{u}_{(1)}^{(k)}, \mathbf{u}_{(2)}^{(k)}, \ldots \mathbf{u}_{(D)}^{(k)}$ with the measured voltages $\mathbf{e}_{(1)}, \mathbf{e}_{(2)}, \ldots \mathbf{e}_{(D)}$ to produce the set $\mathbf{v}_{(1)}^{(k)}, \mathbf{v}_{(2)}^{(k)}, \ldots \mathbf{v}_{(D)}^{(k)}$. Solve the combined set of equations:

$$\begin{bmatrix} - \mathbf{V}_{(1)}^{(k)} - \\ - \mathbf{V}_{(2)}^{(k)} - \\ \vdots \\ - \mathbf{V}_{(D)}^{(k)} - \end{bmatrix} \mathbf{g}^{(k)} = \begin{bmatrix} \mathbf{i}_{(1)} \\ \mathbf{i}_{(2)} \\ \vdots \\ \mathbf{i}_{(D)} \end{bmatrix} \tag{6.5}$$

for the conductance vector $\mathbf{g}^{(k)}$ that minimizes the sum of squared errors using the formula shown in (3.17).

6) Go to step 3.

## Section 2: Simulation Results and Analysis

Iterations of this method are performed in several different situations. The solver is used on both the uniform and the bump grid. The flat grid is relatively uninteresting compared to the bump grid, and so the simulations run on the flat grid are not included in this document. Both the 12 to 3 and the averaging interpolation methods are used to initialize the conductances, and the number of current drives used is also changed to six[13] for some of the simulations. The 12 to 3 approximation takes far more time than the averaging interpolation method, and this algorithm quickly infers a good estimate for the conductances even if it begins with a uniform estimate. For these reasons, the 12 to 3 simulations are left out of this document. The final factor that is altered is the amount of noise added to the voltage measurements provided at the coarse nodes. The first two simulations are for a noiseless system with six and eight current drives, respectively. Eventually the results of the alternating algorithm will be compared with the results of the Newton algorithm. For this reason, the remaining two simulations use eight current drives. The first adds in measurement error on the order of 1% and the second adds error on the order of 10%.

For each set of simulations, two plots are shown. The first plot shows the resistor estimates after one thousand iterations. The second plot in each set illustrates the progression of four values over the first one thousand iterations. The first three values are the coarse voltage error, RMS current error, and resistor error mentioned earlier. The fourth value is the condition number within both the forward and inverse steps.

These plots are shown in the following figures.

---

[13] Drives 1 through 6 as indicated in Figure 5

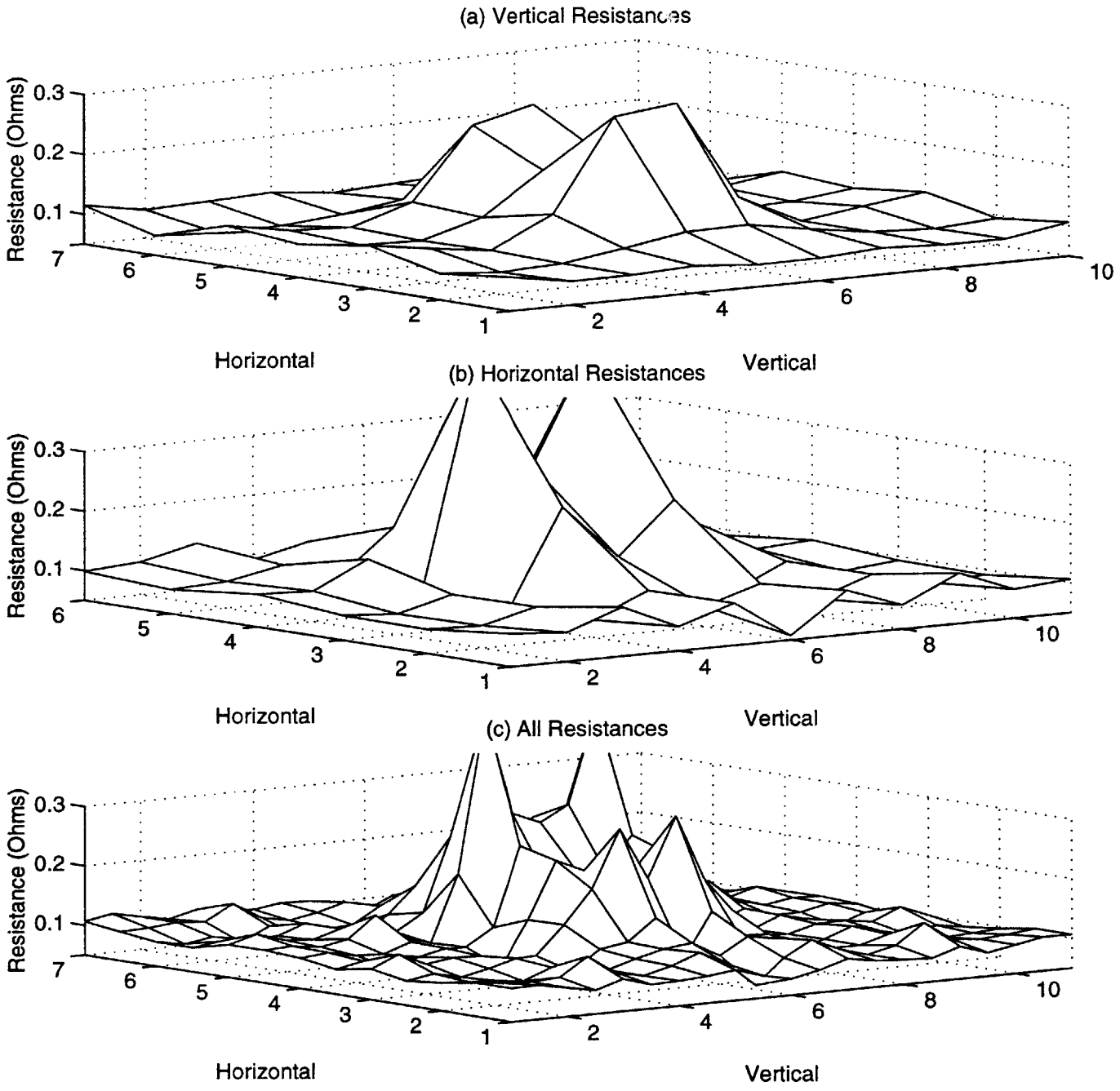



Figure 23: Resistor values after 1000 iterations using six drives with no measurement error.

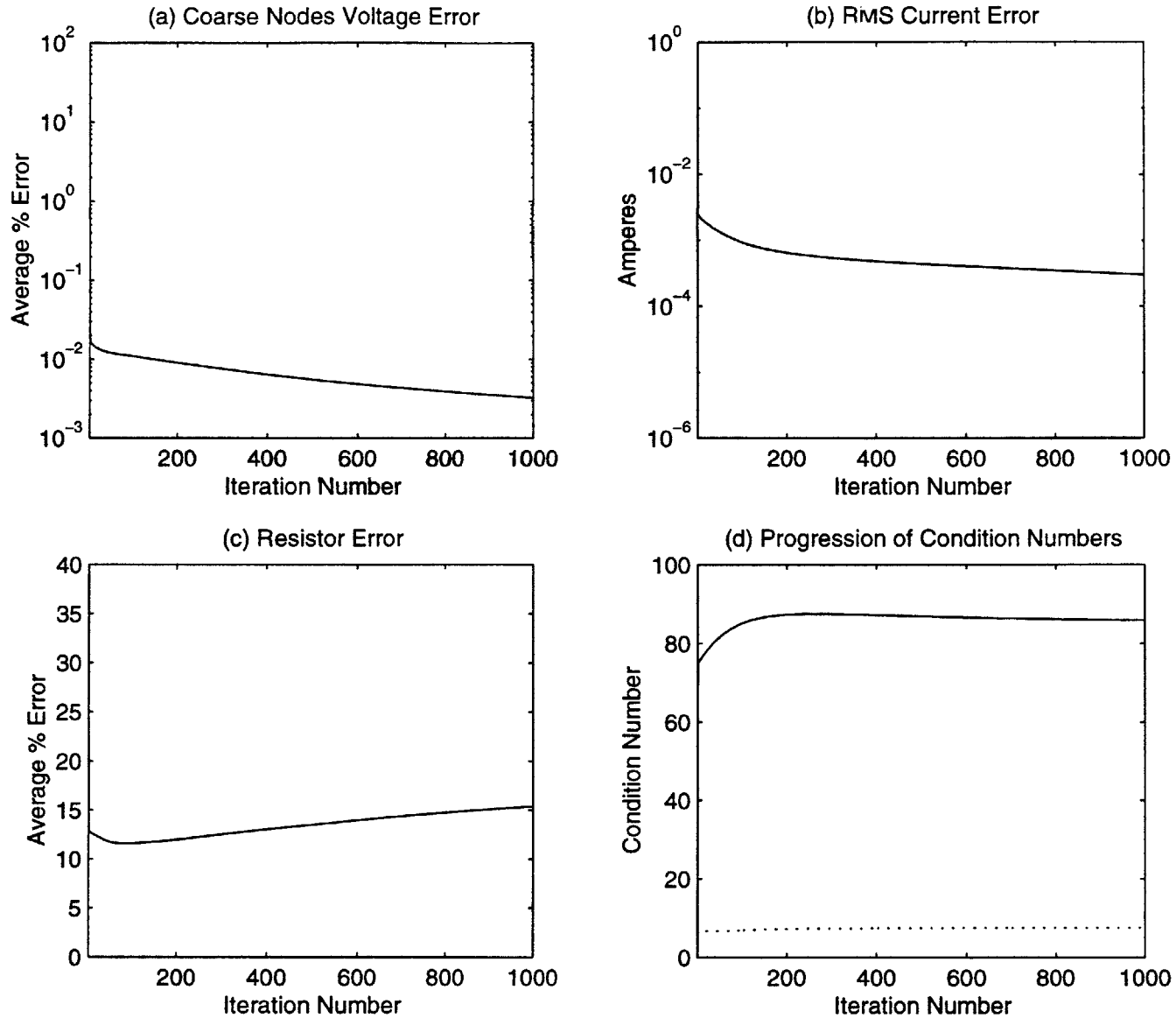



Figure 24: Error metrics for first 1000 iterations
using six drives with no measurement error.

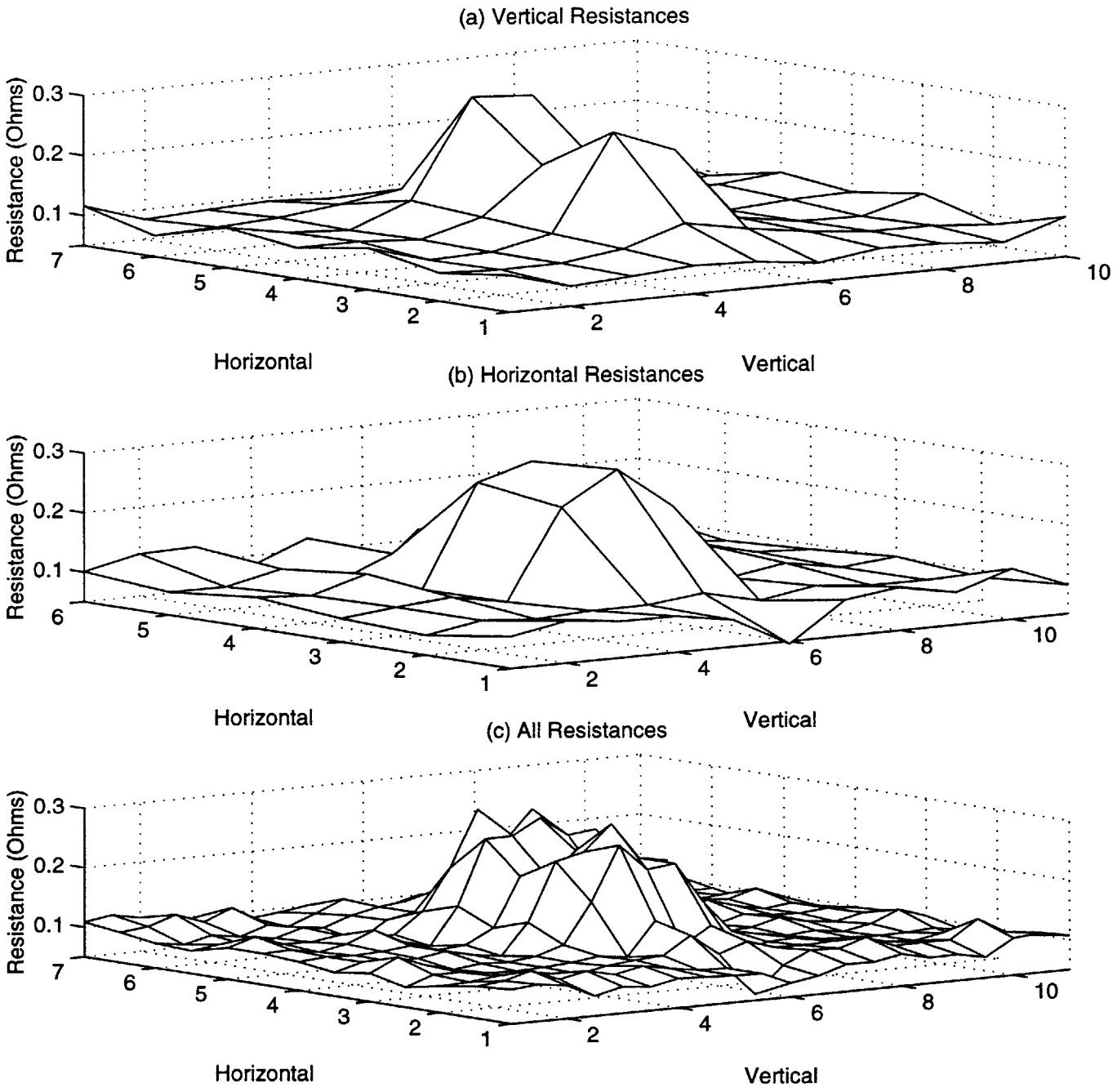



Figure 25: Resistor values after 1000 iterations using eight drives with no measurement error.

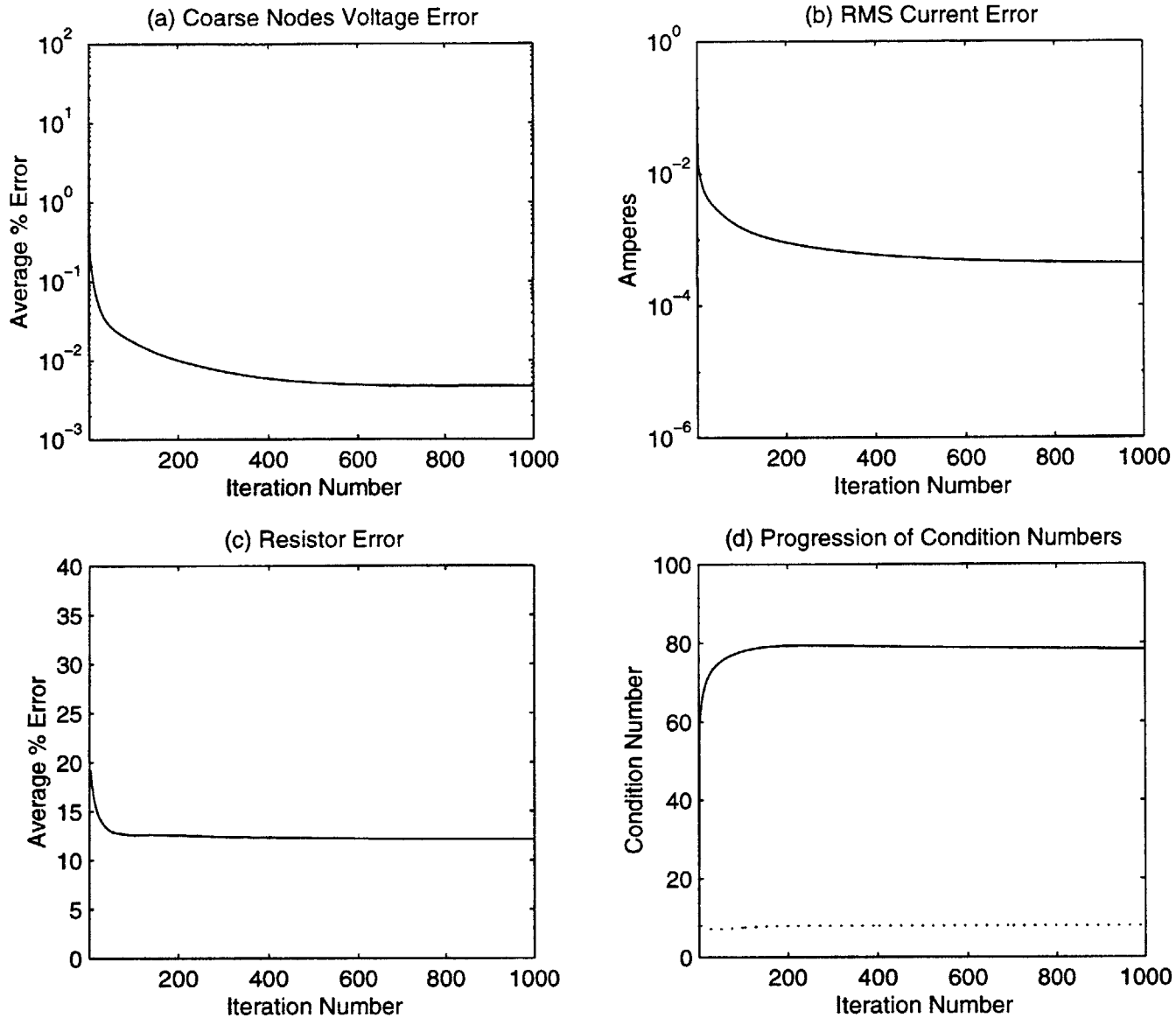



Figure 26: Error metrics for first 1000 iterations
using eight drives with no measurement error.

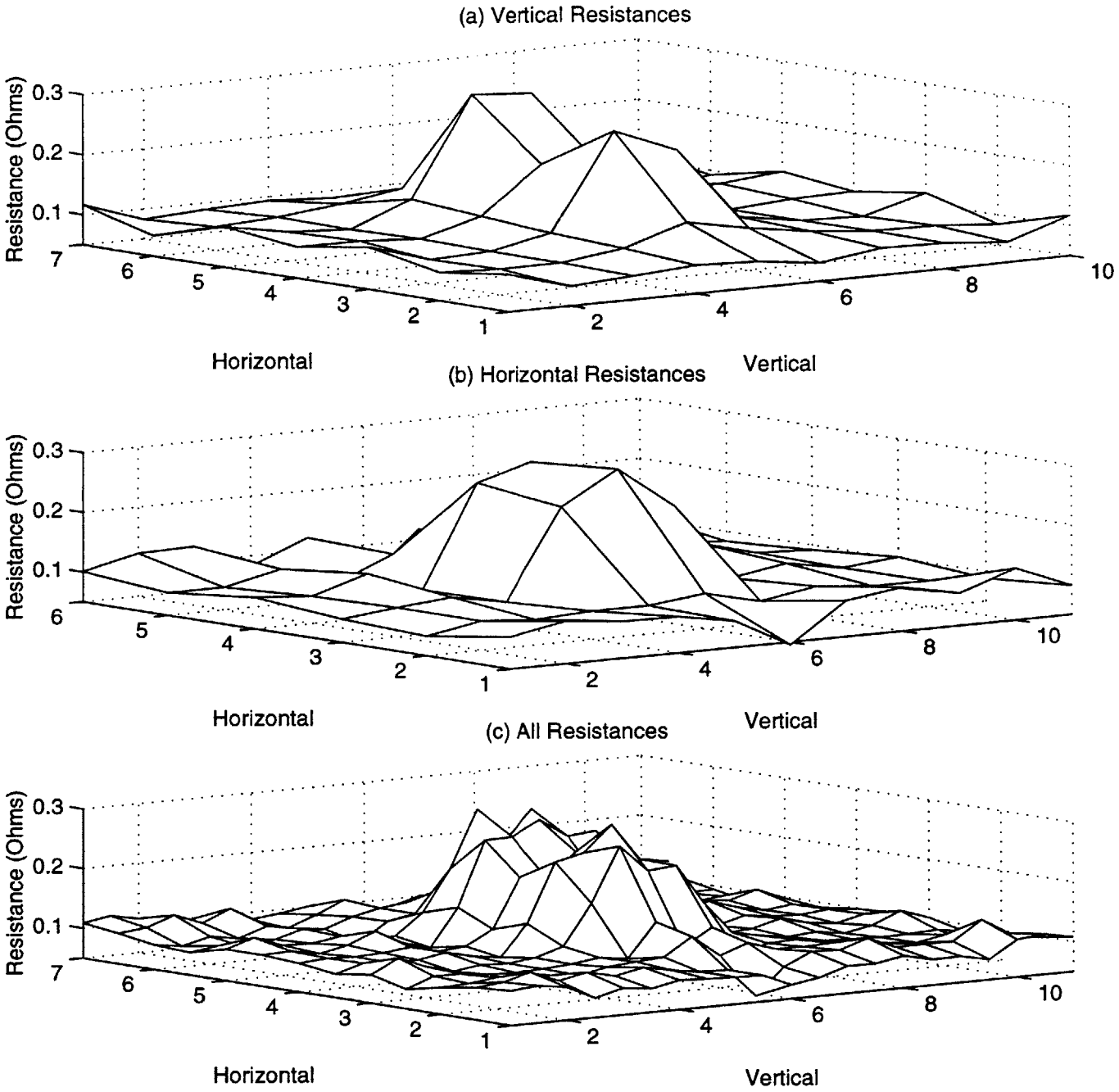



Figure 27: Resistor values after 1000 iterations using eight drives with 1% measurement error.

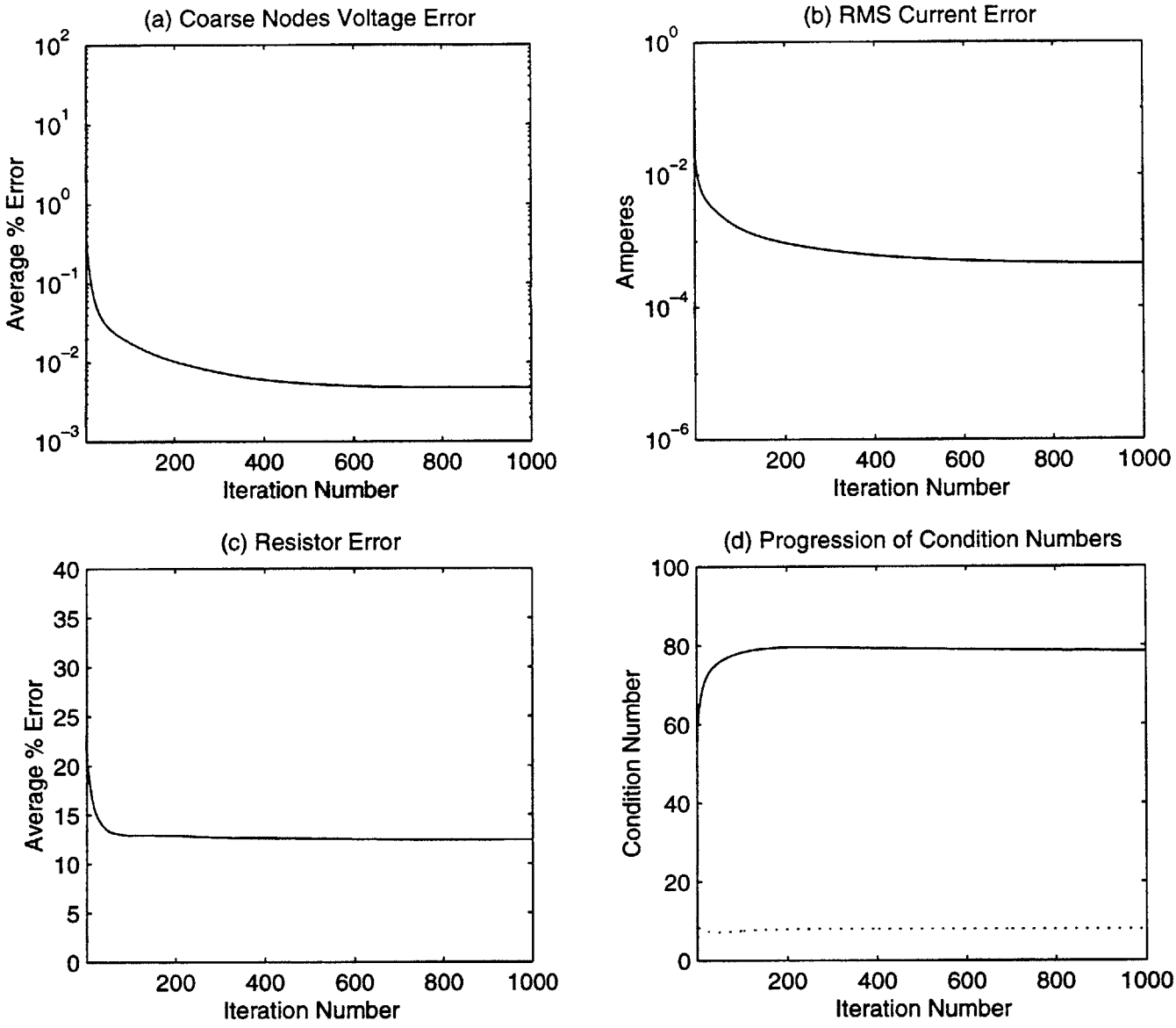



Figure 28: Error metrics for first 1000 iterations using eight drives with 1% measurement error.

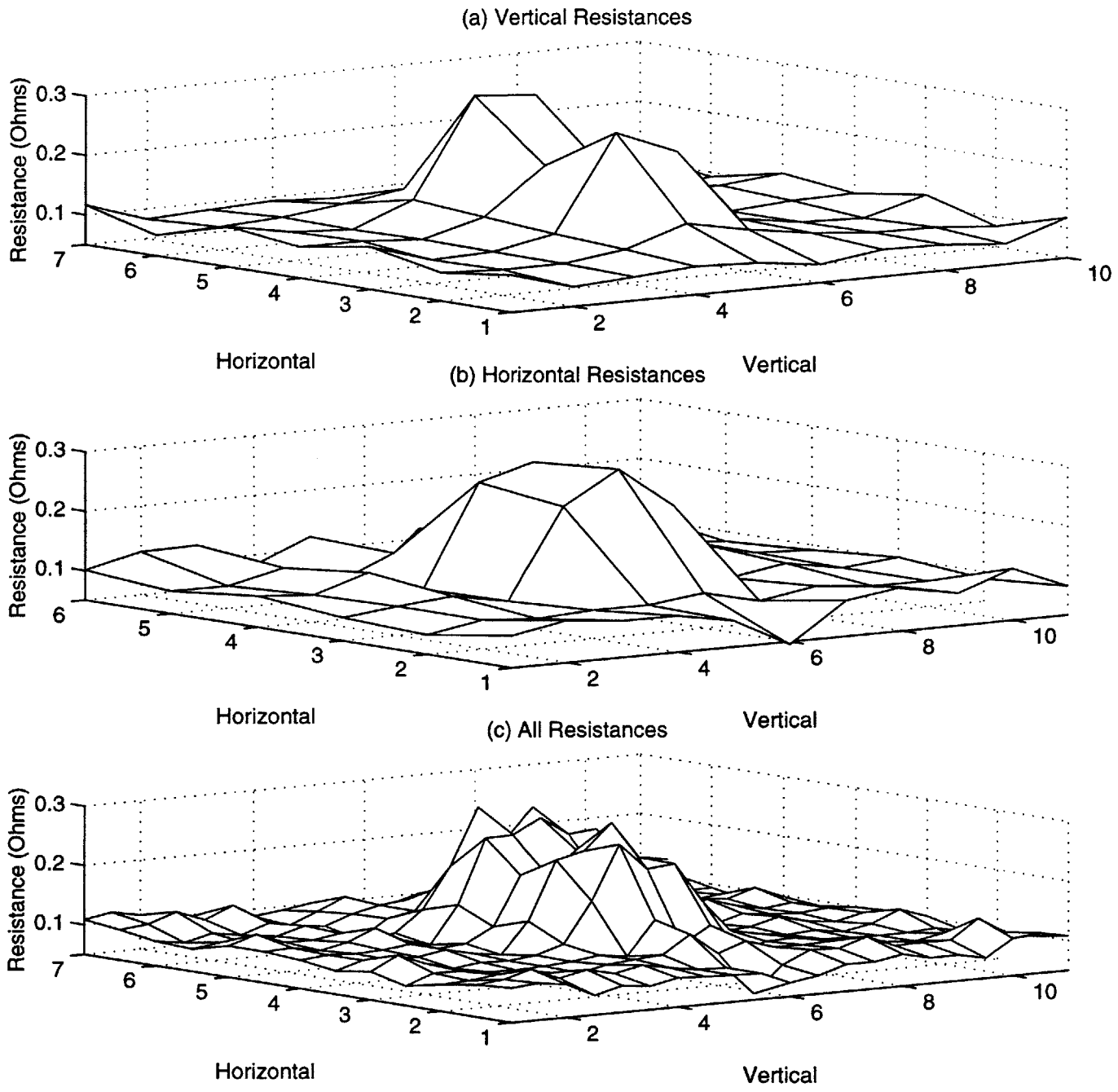



Figure 29: Resistor values after 1000 iterations
using eight drives with 10% measurement error.

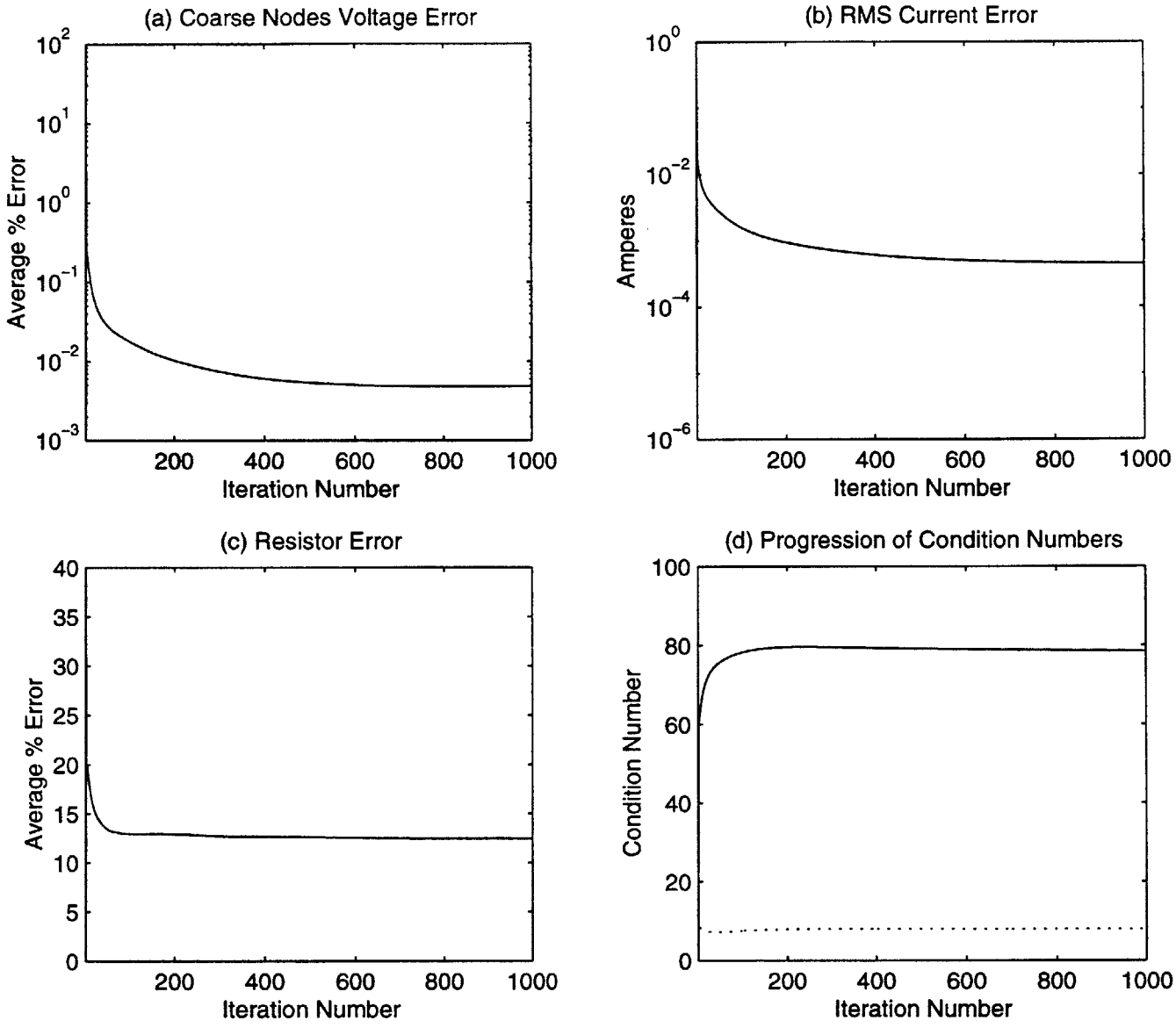



Figure 30: Error metrics for first 1000 iterations using eight drives with 10% measurement error.

The error metrics show some very interesting results. The alternating method is designed so that during both halves of each iteration, a linear least squares problem is solved. This guarantees that the RMS current error decreases at each step[14] since it is a monotonic function of residual current error. The simulations show that the RMS current error is in fact a decreasing function of iteration number. Furthermore, the average absolute percentage coarse voltage error also decreases with each iteration. The coarse voltage and RMS current error plots for each simulation seem to suggest that the solver is iterating towards the correct solution. However, the progression of the resistor error shows that this is not always true. The equation being modeled is highly non-linear and the plots show that it is possible for the resistor estimates to become progressively worse even though the RMS current error and the coarse nodes voltage error decrease at each step[15].

As a general trend, increasing the number of current drives does insure that the average absolute resistor error decreases. With more drives, the condition numbers also become smaller. Again, the addition of noise causes the system to move away from the exact solution, but it does not cause erratic, unpredictable results.

---

[14] To be completely precise, a linear least squares solution is guaranteed to be non-increasing. So, the solution must either decrease or at worst, remain constant.

[15] The simulation using six current drives with no measurement error is the best example of the simulations that were run of a situation where the resistor error increases although the RMS current error is decreasing

*Chapter 7*

# NEWTON'S METHOD

Newton's method looks at how the value of a function changes as a result of small changes in the estimate of its solution. The method forms a Taylor series expansion for the system about the current estimate, removes all non-linear terms, and then solves this relatively simpler equation. There are several unknowns in this system. The most important ones are the values of the resistors. However, the voltage values at all non-coarse nodes are also unknown. The alternating method solves for the each of these in turn, but the Newton algorithm is different in that it simultaneously updates all unknowns.

## Section 1: Mathematical Formulation of SIDO Solver

The state of the system on the $k^{th}$ iteration is:

$$\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix}=\mathbf{i}^{(k)} \tag{7.1}$$

Newton's method proceeds by replacing each unknown with its updated value. This results in:

$$\mathbf{A}\left(\mathbf{G}^{(k)}+\delta\mathbf{G}^{(k)}\right)\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}+\delta\mathbf{u}^{(k)}\end{bmatrix}=\mathbf{i}^{(d)} \tag{7.2}$$

$$\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix}+\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}0\\ \delta\mathbf{u}^{(k)}\end{bmatrix}$$
$$+\mathbf{A}\delta\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix}+\mathbf{A}\delta\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}0\\ \delta\mathbf{u}^{(k)}\end{bmatrix}=\mathbf{i}^{(d)} \tag{7.3}$$

Three of these terms are linear in delta, but one is quadratic. Because it is assumed that the delta values are relatively small, the quadratic term is dropped and the known term is moved to the right, resulting in the following equation that can be solved for the delta values.

$$\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}0\\ \delta\mathbf{u}^{(k)}\end{bmatrix}+\mathbf{A}\delta\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix}=\mathbf{i}^{(d)}-\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix} \tag{7.4}$$

In order to solve this equation, it is simplified to the form:

$$\begin{bmatrix}\mathbf{P}^{(k)} & \mathbf{W}^{(k)}\end{bmatrix}\cdot\begin{bmatrix}\delta\mathbf{g}^{(k)}\\ \delta\mathbf{u}^{(k)}\end{bmatrix}=\mathbf{i}^{(d)}-\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix} \tag{7.5}$$

where **P** and **W** are functions of **A**, $\mathbf{G}^{(k)}$, and $\mathbf{u}^{(k)}$. From the previous explanations of branch incidence matrices, the precise values are:

$$\mathbf{P}^{(k)}=\begin{bmatrix}\mathbf{A}\left(\mathrm{diag}\left(\left(\mathbf{v}_1^{(k)}\right)^{T}\mathbf{A}\right)\right)\\ \mathbf{A}\left(\mathrm{diag}\left(\left(\mathbf{v}_2^{(k)}\right)^{T}\mathbf{A}\right)\right)\\ \vdots\\ \mathbf{A}\left(\mathrm{diag}\left(\left(\mathbf{v}_D^{(k)}\right)^{T}\mathbf{A}\right)\right)\end{bmatrix} \tag{7.6}$$

$$\mathbf{W}^{(k)}=\begin{bmatrix}\left(\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\right)_{N-M} & 0 & 0 & 0\\ 0 & \left(\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\right)_{N-M} & 0 & 0\\ 0 & 0 & \ddots & 0\\ 0 & 0 & 0 & \left(\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^{T}\right)_{N-M}\end{bmatrix} \tag{7.7}$$

where the **N-M** subscript indicates using the **N-M** columns of $\mathbf{AGA}^T$ that correspond to the unmeasured voltages. Solving for the deltas in the unknown values is now a linear least squares problem.

Newton's method uses the current estimates for **g** and **u** to solve for the delta values. Then it updates the estimates, and finally, the above equation is reformulated with the new estimates so that the process can be repeated.

## Section 2: Need for a Regularization Term

If the conductance and unmeasured voltages are exact, they are linear functions of each other, and this will make **P** and **W** linear functions of each other as well. This means that $\left[\mathbf{P}^{(k)} \quad \mathbf{W}^{(k)}\right]$ is rank deficient and has more unknowns than equations. Furthermore, it has been experimentally observed that even if the conductance and unmeasured voltage values are not exact, as they get closer to being correct, the system will approach singularity and produce unpredictable results.

In order to avoid this situation, a regularization term is added. This translates into solving the equation $\mathbf{Ax} = \mathbf{b}$ for the vector **x** by minimizing the value of:

$$\|\mathbf{Ax} - \mathbf{b}\|^2 + \varepsilon\|\mathbf{x}\|^2 \tag{7.8}$$

for some value of $\varepsilon$. This is equal to:

$$\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} - \mathbf{b}^T\mathbf{Ax} + \mathbf{b}^T\mathbf{b} + \varepsilon\mathbf{x}^T\mathbf{x} \tag{7.9}$$

Taking the derivative of this value and setting it equal to 0, results in:

$$2\mathbf{A}^T\mathbf{Ax} - \mathbf{A}^T\mathbf{b} - \mathbf{A}^T\mathbf{b} + 2\varepsilon\mathbf{x} = 0 \tag{7.10}$$

$$\left(\mathbf{A}^T\mathbf{A}+\varepsilon\mathbf{I}\right)\mathbf{x}=\mathbf{A}^T\mathbf{b} \tag{7.11}$$

The vector **x** in (7.11) is now multiplied by a square, positive definite, symmetric matrix. With an exact solution for **x**, there is no need to worry about a rank-deficient system. The only decision left is to choose a value for $\varepsilon$. The purpose of the regularization term is to change **A** because $\mathbf{A}^T\mathbf{A}+\varepsilon\mathbf{I}$ is not singular. Singularity for **A** implies that its smallest singular value is equal to zero. Let $\sigma_1$ and $\sigma_n$ be the largest and smallest singular values of the matrix **A**. This implies that the largest and smallest singular values of $\mathbf{A}^T\mathbf{A}+\varepsilon\mathbf{I}$ are equal to $(\sigma_1)^2+\varepsilon$ and $(\sigma_n)^2+\varepsilon$. Therefore, the choice of $\varepsilon$ is made so that the condition number of the resulting matrix will represent a well-posed system. In other words, $\varepsilon$ is chosen so that:

$$\frac{(\sigma_1)^2+\varepsilon}{(\sigma_n)^2+\varepsilon}\approx\frac{(\sigma_1)^2}{\varepsilon}<<10^{12} \tag{7.12}$$

Note that in the (7.11), the values for **A**, **x**, and **b** are actually:

$$\begin{aligned}\mathbf{b}&=\mathbf{i}^{(d)}-\mathbf{A}\mathbf{G}^{(k)}\mathbf{A}^T\begin{bmatrix}\mathbf{e}\\ \mathbf{u}^{(k)}\end{bmatrix}\\ \mathbf{A}&=\begin{bmatrix}\mathbf{P}^{(k)} & \mathbf{W}^{(k)}\end{bmatrix}\\ \mathbf{x}&=\begin{bmatrix}\delta\mathbf{g}^{(k)}\\ \delta\mathbf{u}^{(k)}\end{bmatrix}\end{aligned} \tag{7.13}$$

## Section 3: Simulation Results and Analysis

Iterations of this method are performed in several different situations. The simulation is only applied to the bump grid and always uses the averaging

interpolation method with eight current drives. The factors that are altered are the amount of white noise added to the system and the value of $\varepsilon$ used for regularization.

For each set of simulations, two plots are shown. The first plot shows the resistor estimates immediately after the final iteration. The second plot in each set shows the progression of three values over the first twenty-five iterations. The three values are the coarse voltage error, RMS current error, and resistor error mentioned earlier. These plots are shown in the following figures. Note that the simulations where $\varepsilon = 0.1$ are run for one hundred iterations whereas for other values of $\varepsilon$, there are only twenty-five iterations. For the sake of comparison, it is best to look at the plots that either have the same value for $\varepsilon$ or have the same amount of noise.

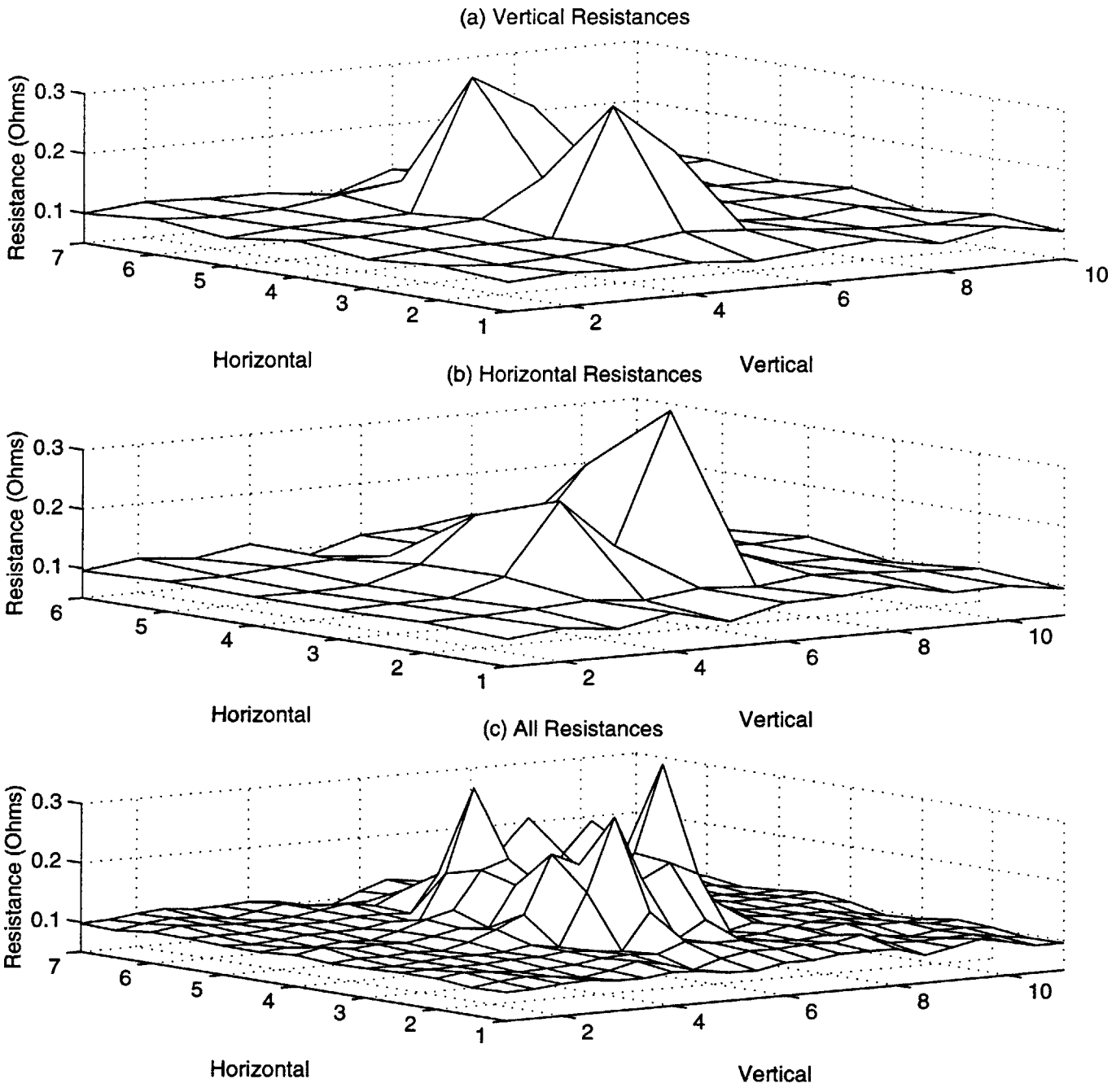



Figure 31: Resistor values after 100 iterations of a noiseless system with $\varepsilon = 0.1$

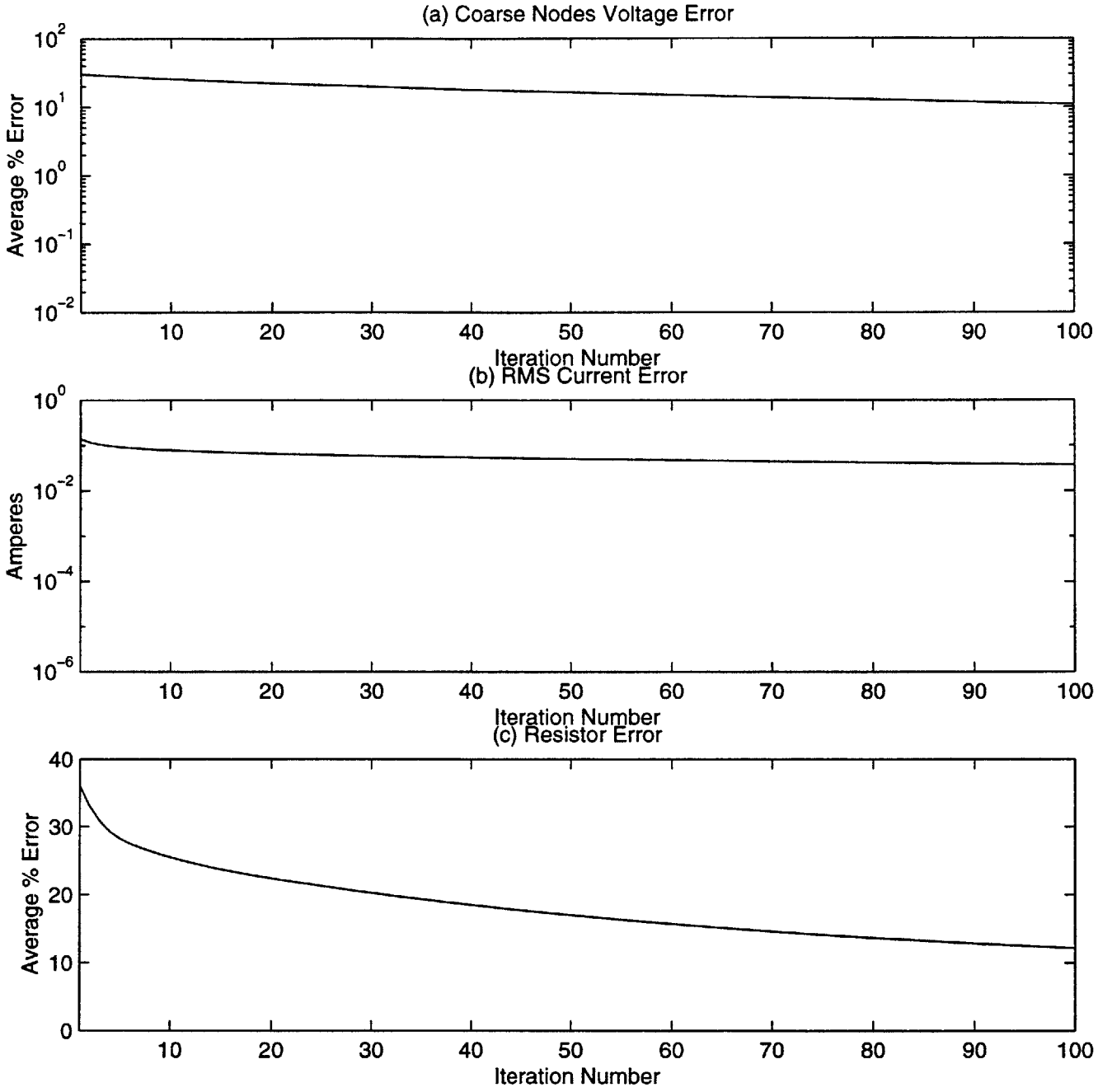



Figure 32: Error metrics for 100 iterations of a noiseless system with $\varepsilon = 0.1$

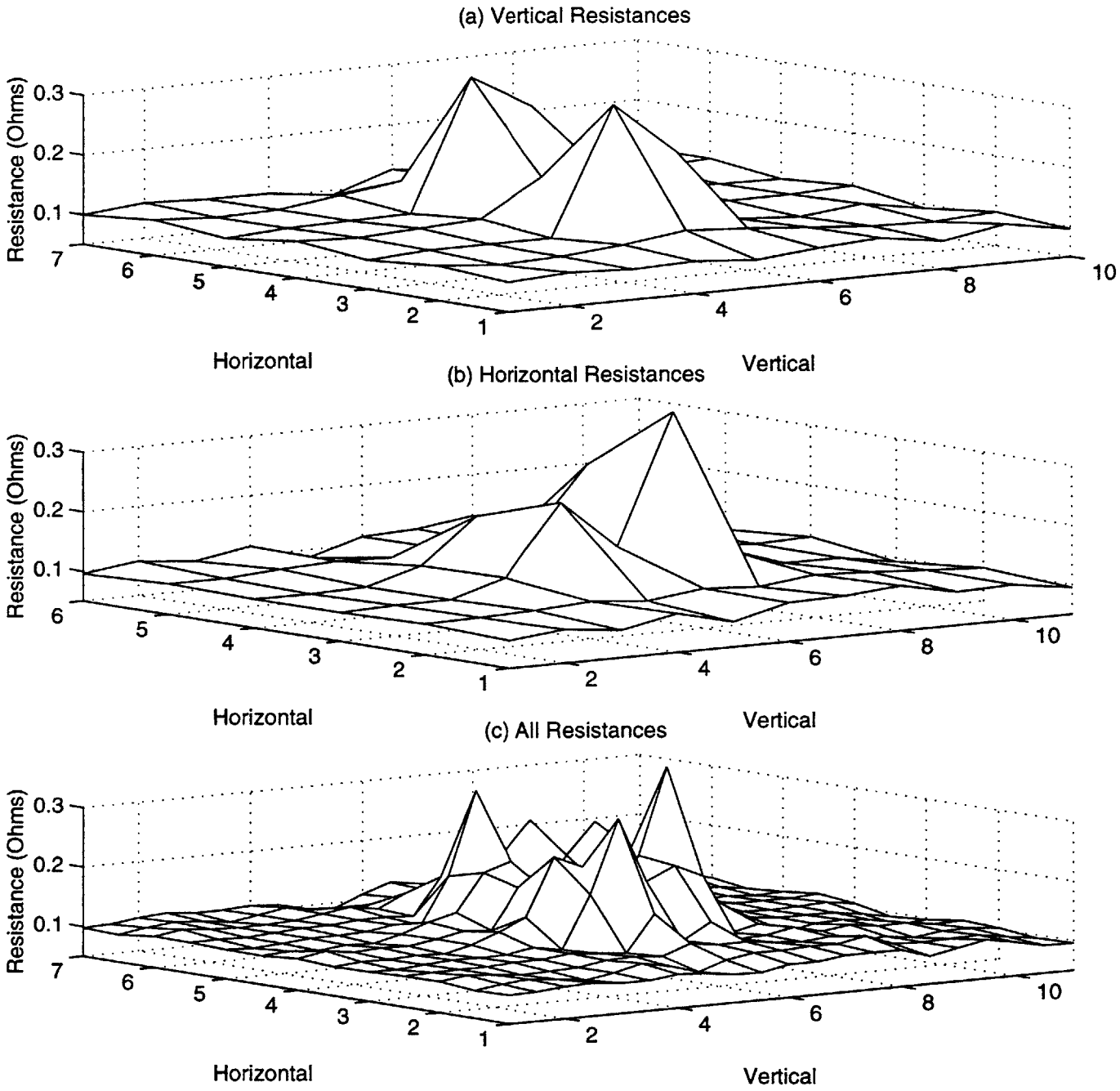



Figure 33: Resistor values after 100 iterations of a system with 1% measurement error and $\varepsilon = 0.1$

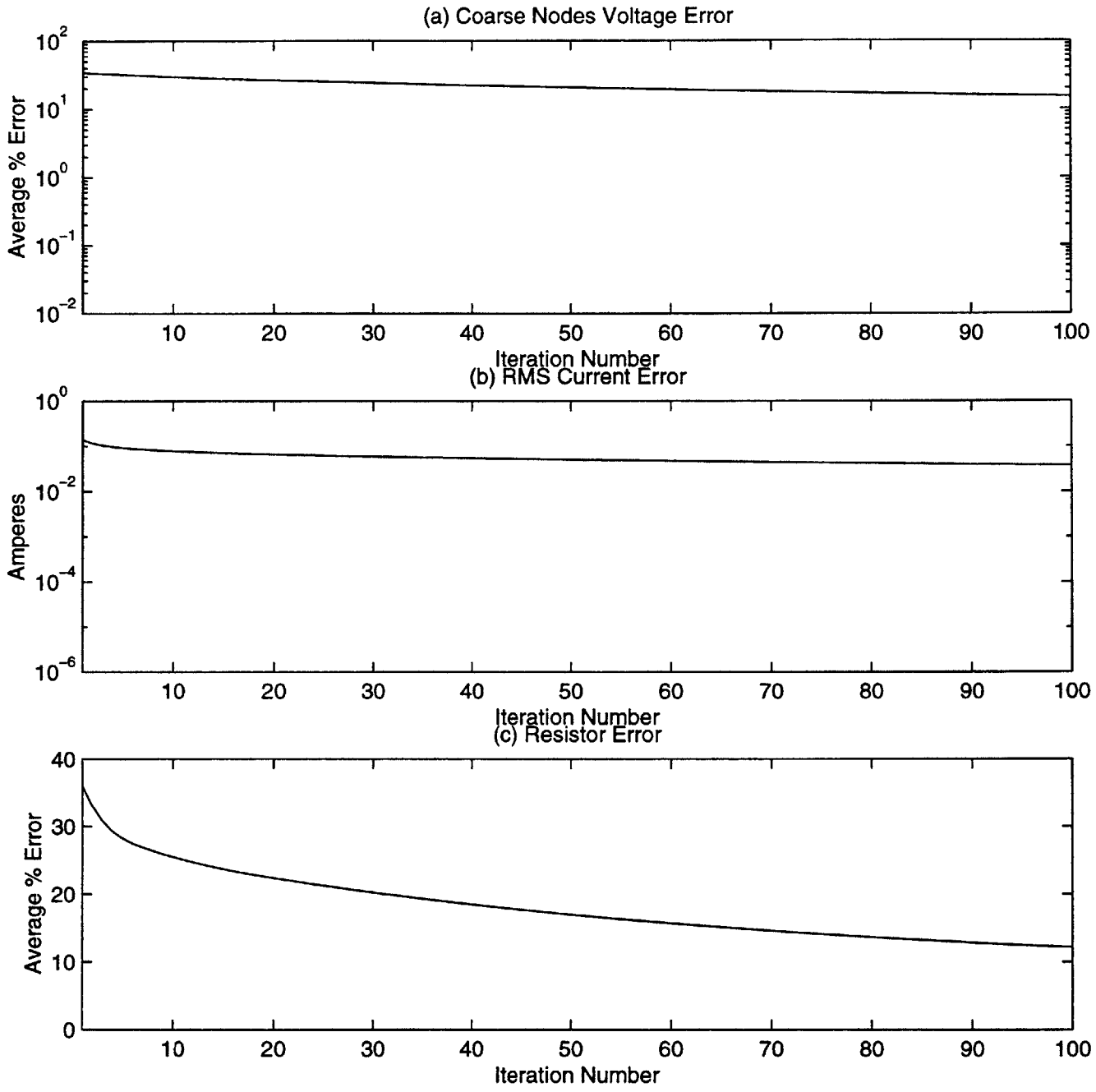



Figure 34: Error metrics for 100 iterations of a system with 1% measurement error and $\varepsilon = 0.1$

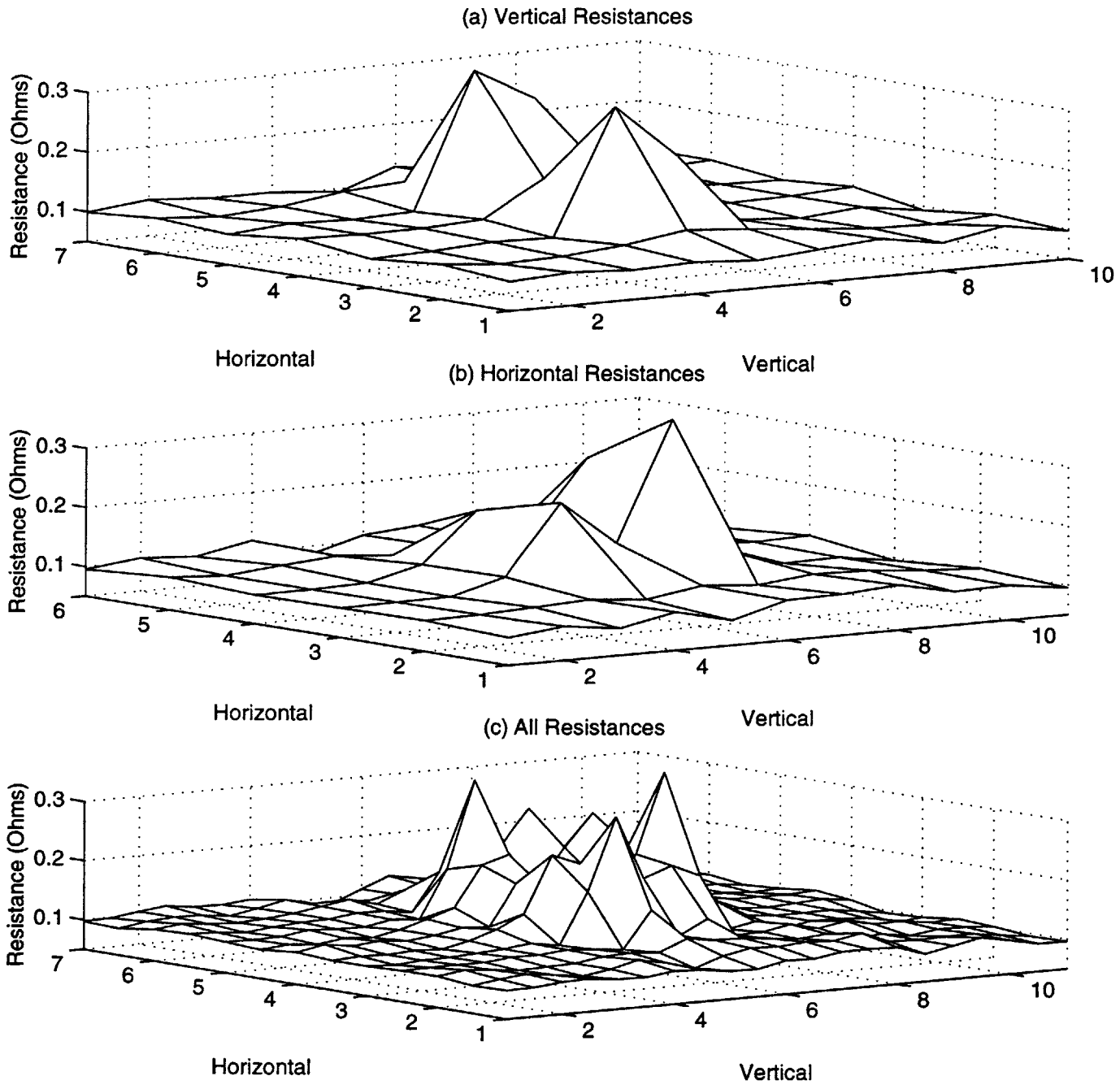



Figure 35: Resistor values after 100 iterations of a system with 10% measurement error and $\varepsilon = 0.1$

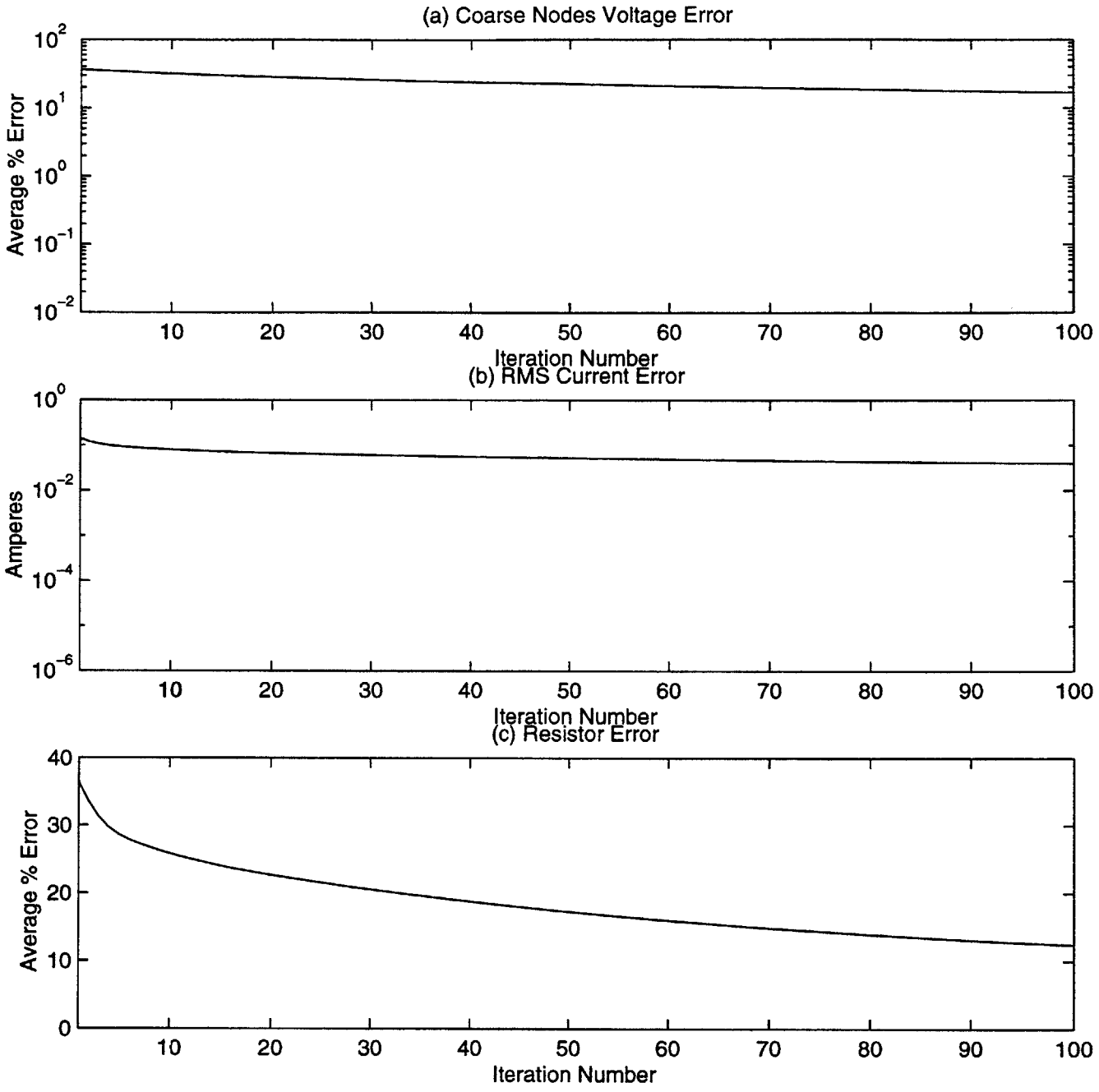



Figure 36: Error metrics for 100 iterations of a system with 10% measurement error and $\varepsilon = 0.1$

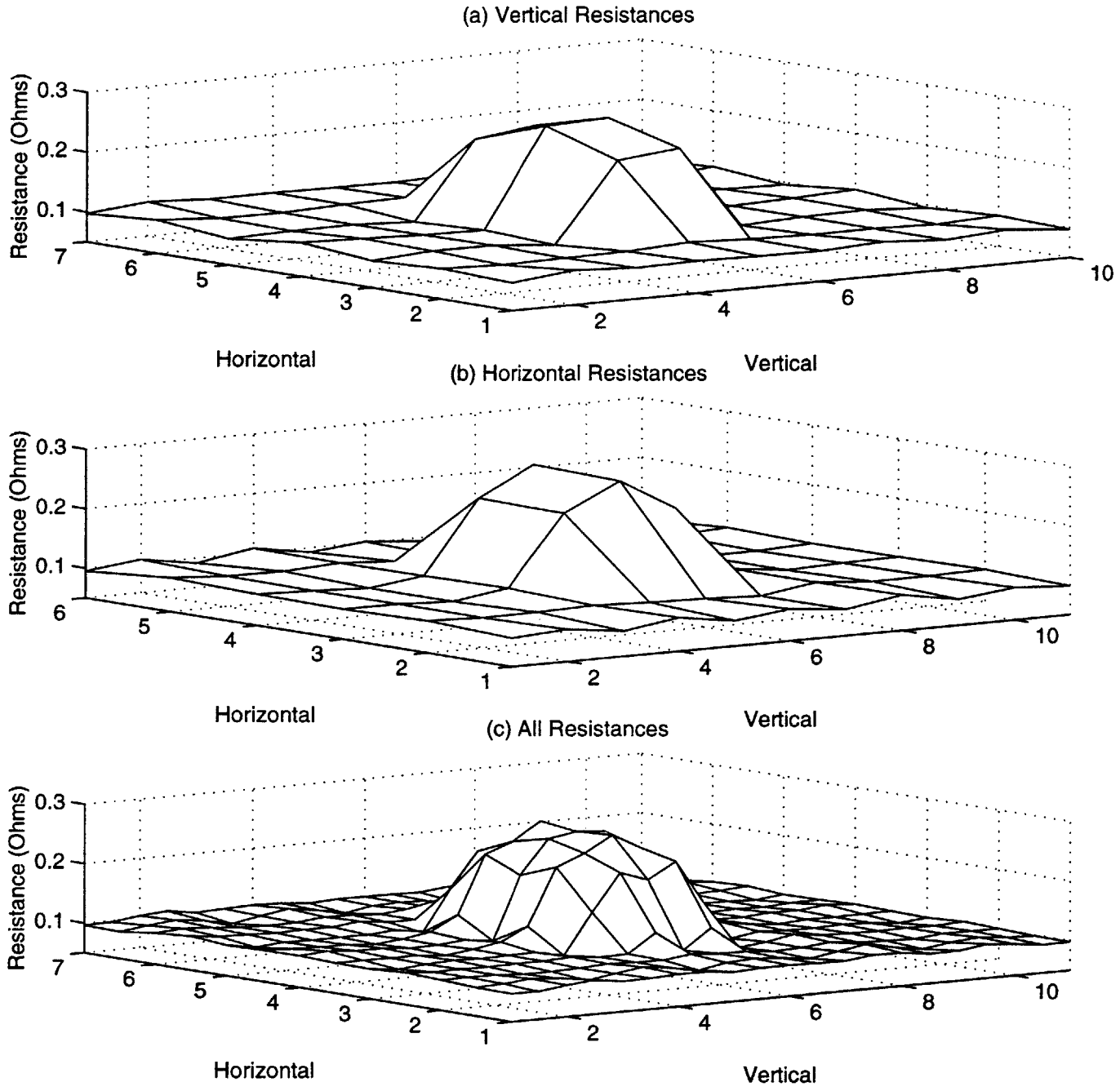



Figure 37: Resistor values after 25 iterations of a noiseless system with $\varepsilon = 10^{-4}$

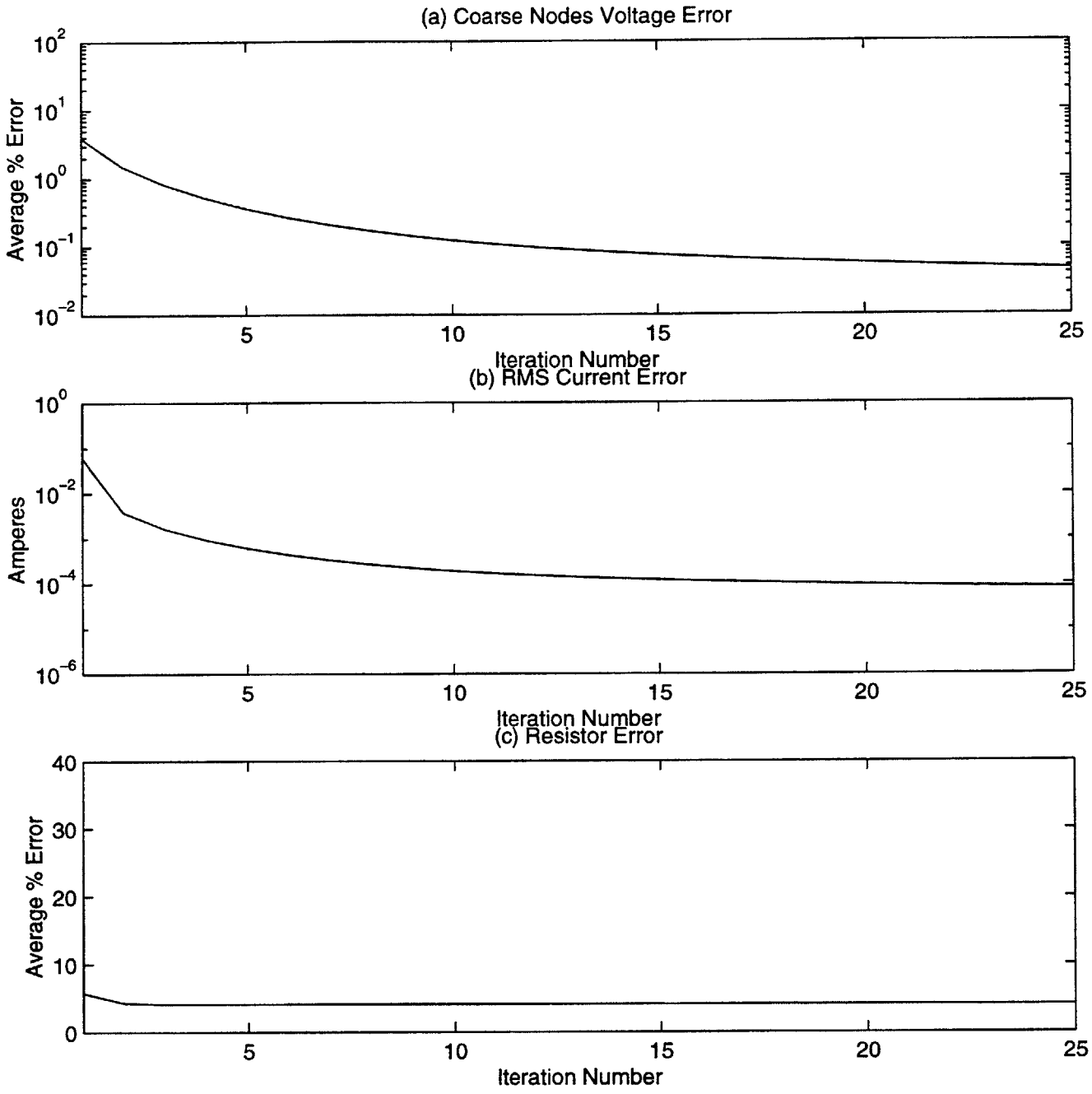



Figure 38: Error metrics for 25 iterations of a noiseless system with $\varepsilon = 10^{-4}$

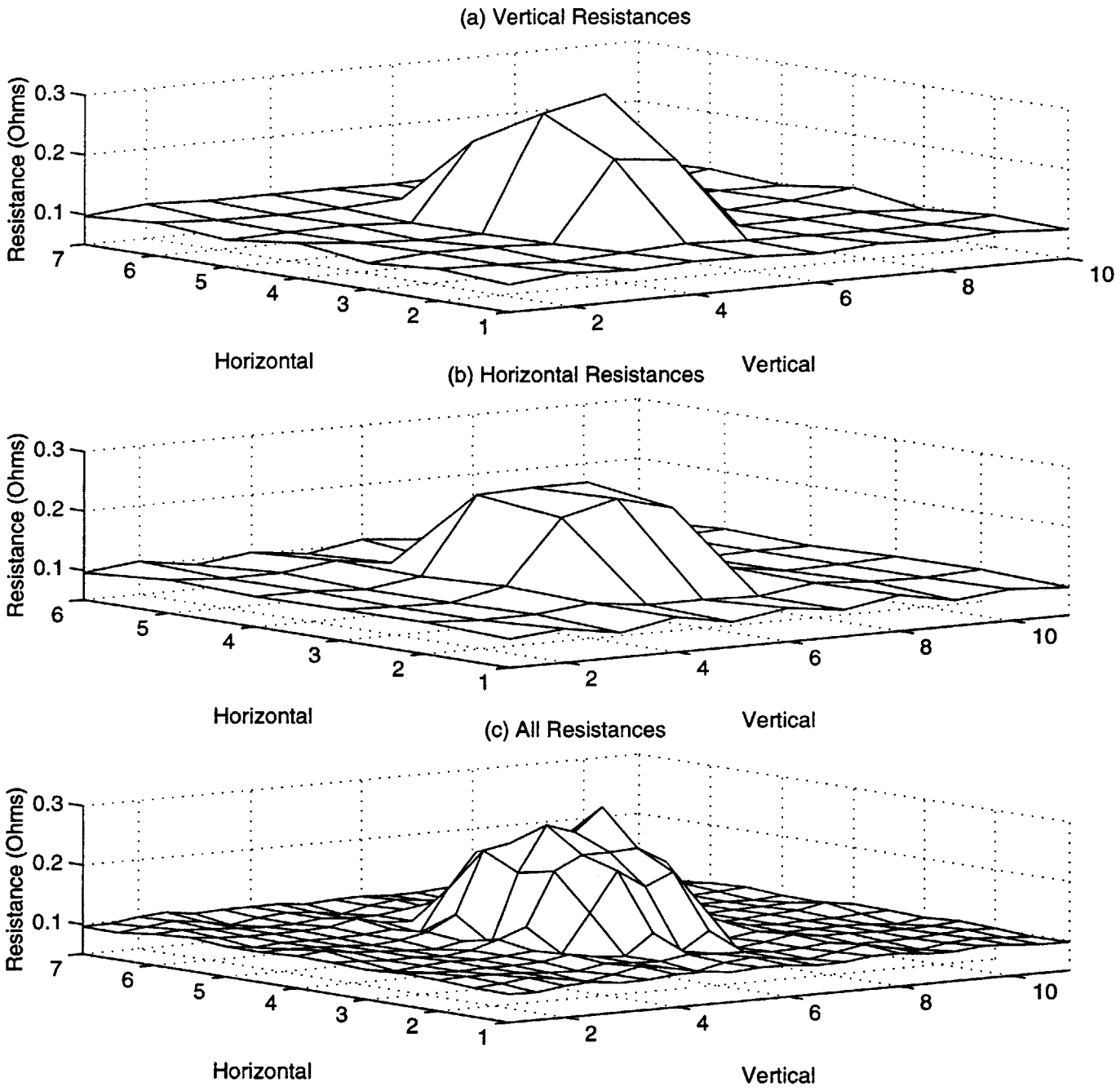



Figure 39: Resistor values after 25 iterations of a system with 1% measurement error and $\varepsilon = 10^{-4}$

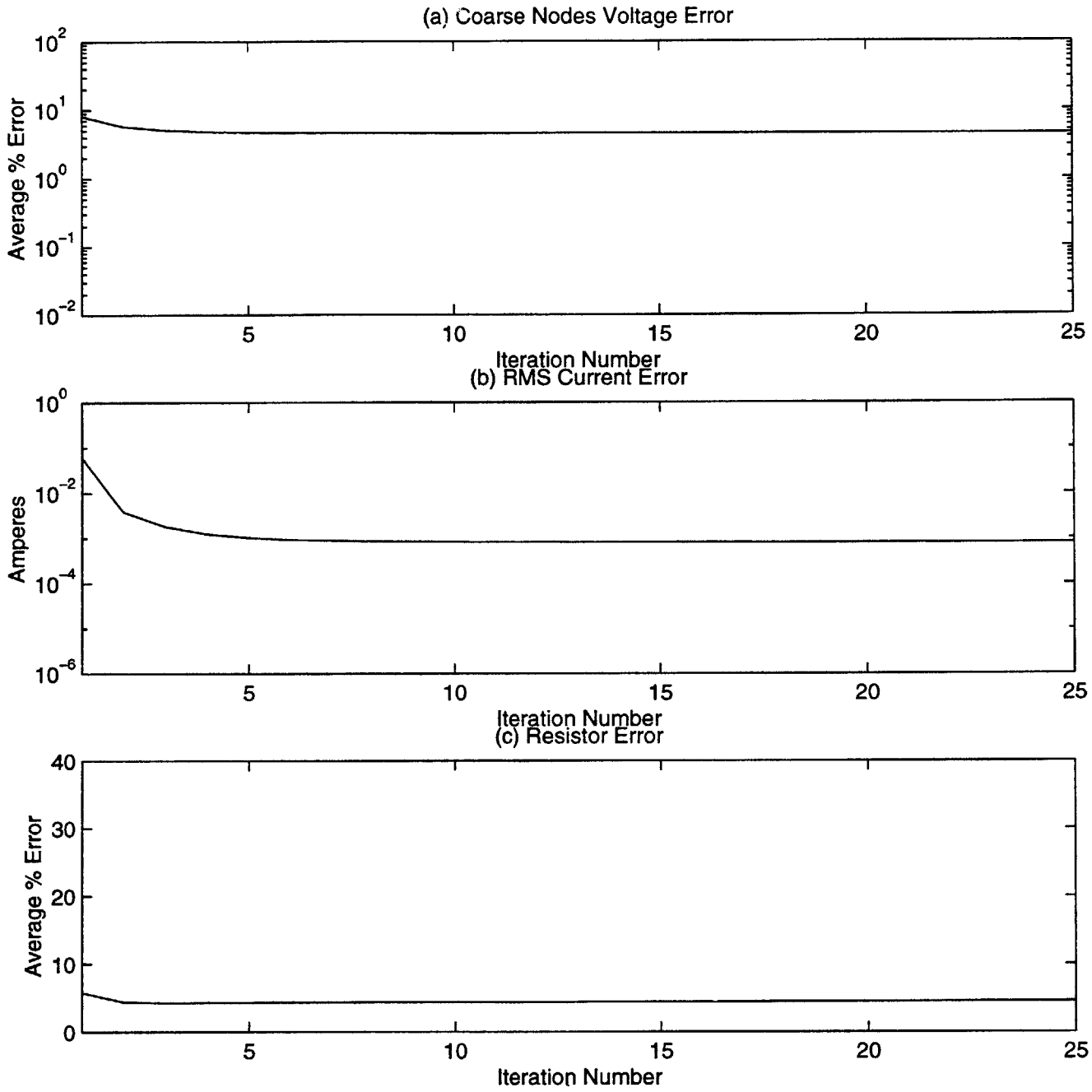



Figure 40: Error metrics for 25 iterations of a system with 1% measurement error and $\varepsilon = 10^{-4}$

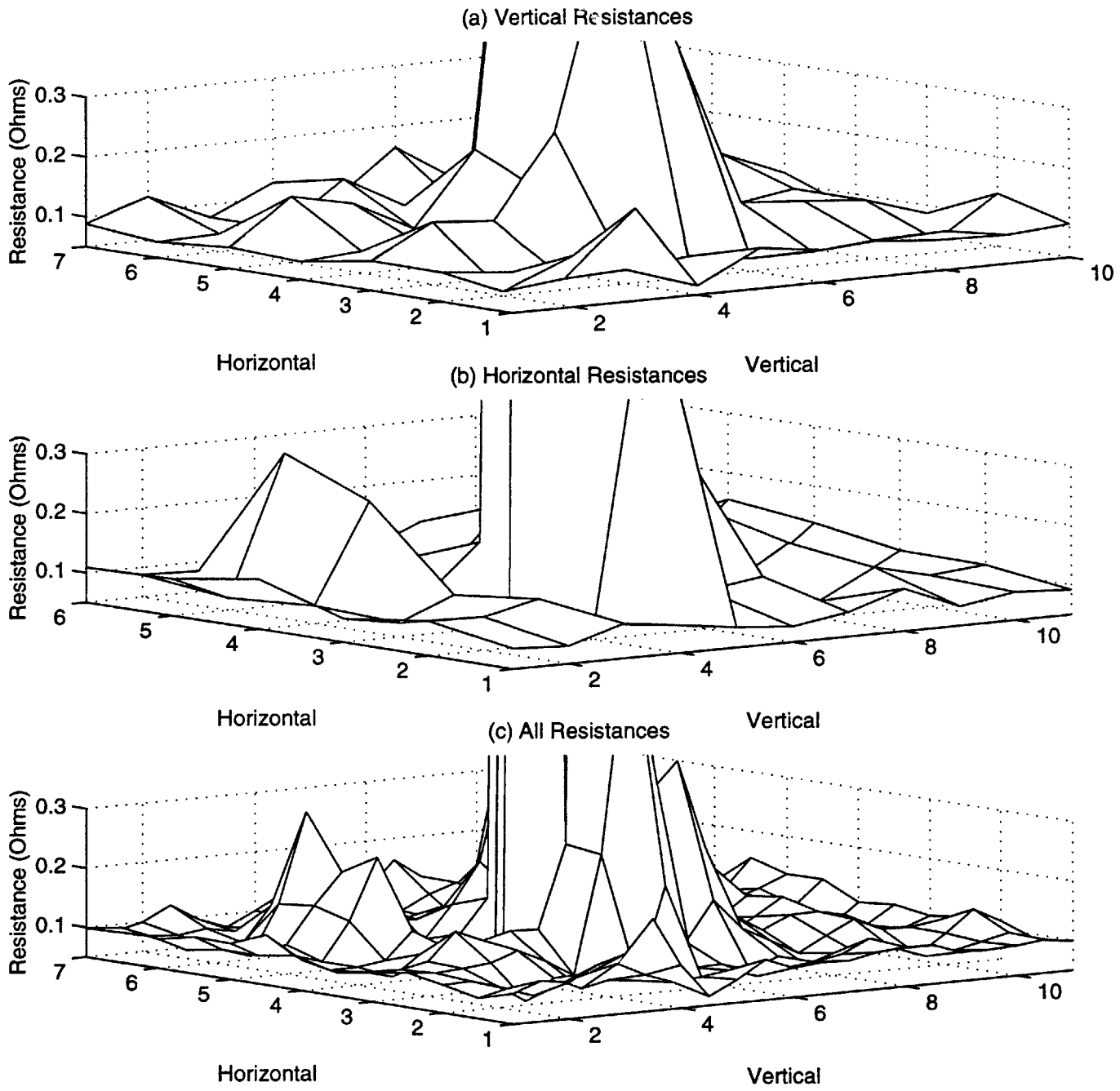



Figure 41: Resistor values after 25 iterations of a system with 10% measurement error and $\varepsilon = 10^{-4}$

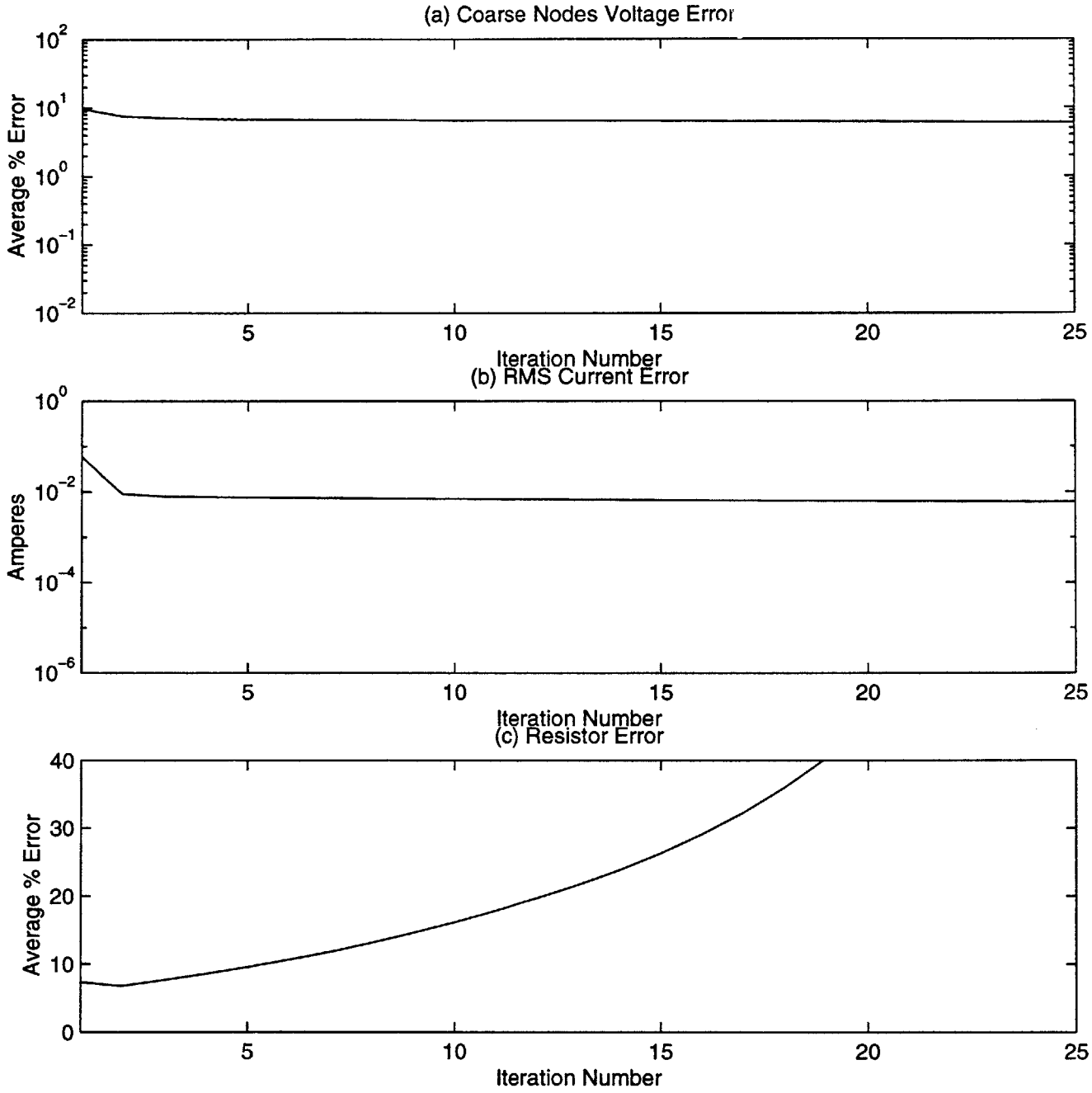



Figure 42: Error metrics for 25 iterations of a system with 10% measurement error and $\varepsilon = 10^{-4}$

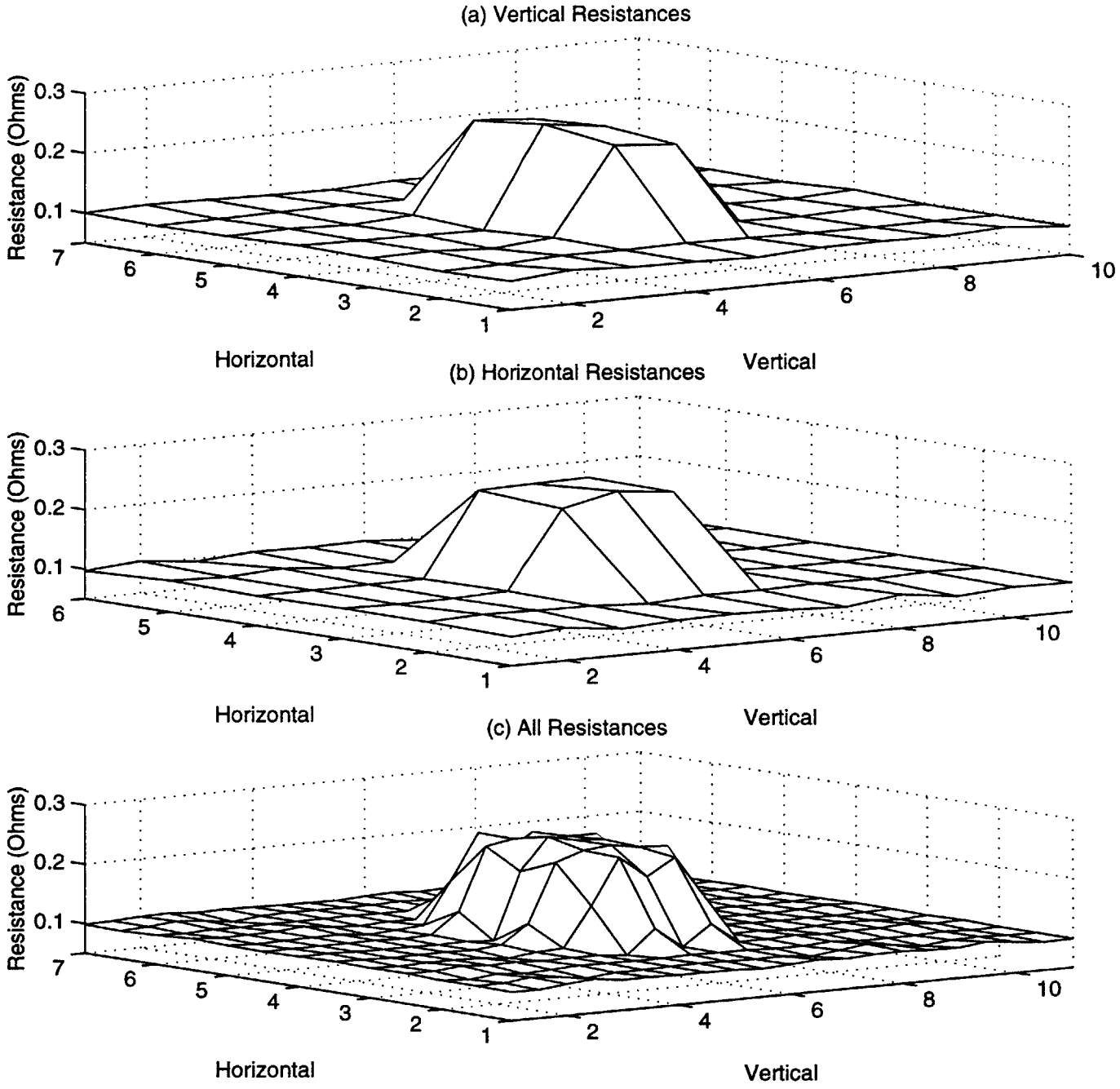



Figure 43: Resistor values after 25 iterations of a noiseless system with $\varepsilon = 10^{-7}$

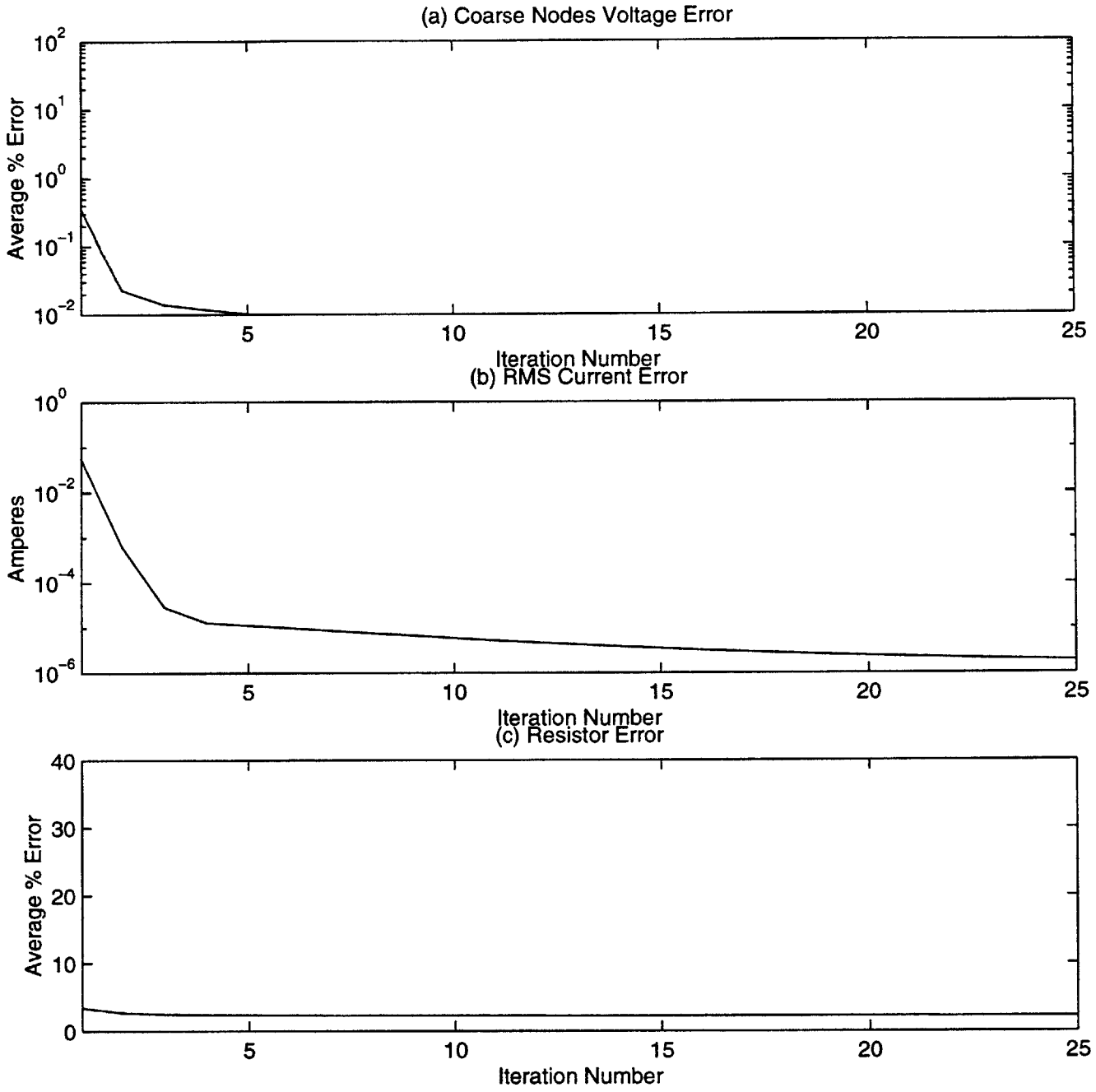



Figure 44: Error metrics for 25 iterations of a noiseless system with $\varepsilon = 10^{-7}$

For noiseless simulations, very small values of $\varepsilon$ can be used and the solver will still produce the correct solution. The plots shown use values as small as $\varepsilon = 10^{-7}$, but in fact values as small as $\varepsilon = 10^{-13}$ can be used and the resistor grid solution will still be obtained accurately and quickly. With such small values, the condition number of the system gets as high as $10^{19}$, but the system is still effective as a solver.[16] However, with the introduction of noise, large condition numbers and ill-posed systems quickly become impediments to the solver and its ability to produce the correct resistor grid. Both noisy systems (1% and 10% error) require that the value of $\varepsilon$ not be much smaller than $10^{-4}$. This would still represent a system with a condition number as high as $10^{10}$. Also, in the simulation with 10% measurement error and $\varepsilon$ equal to $10^{-4}$, the solver actually begins to head off in the wrong direction. This occurrence is not at all surprising for an equation with such a high condition number. Using a value such as 0.1 for $\varepsilon$ will make the solver much more likely to head in the correct direction and much less likely to take some erratic path as a result of an ill-conditioned system. Yet larger values of $\varepsilon$ result in a slower progression towards the final solution, and this is the reason that the simulations with $\varepsilon = 0.1$ are run for 100 iterations. The smaller values of $\varepsilon$ cause the system to move more rapidly, but with large condition numbers, these movements could be in entirely the wrong direction. As is expected, there are a number of factors in these simulations that trade off against each other.

[16] The condition number calculation is based on the fact that the largest singular value for the matrix **A** is on the order of 1000.

## Section 4: A Coarser Set of Measurements

To this point, the SIDO solver has only been used on grids where coarse measurements are provided at about one out of every four nodes. This section will change that factor to about one out of every nine nodes. The fine grid will now measure ten nodes by ten nodes and so the coarse grid measures four nodes by four nodes. The equation for the number of drives needed is:

$$\begin{aligned}(4*4-1)\mathbf{D} &\geq (2*10*10)-(10+10)\\ \mathbf{D} &\geq 12\end{aligned} \tag{7.14}$$

To be sufficiently over this bound, sixteen drives are used. Note that the matrix $[\mathbf{P} \quad \mathbf{W}]$ will now have 1584 rows and 1524 columns. As will be shown later, calculating the singular value decomposition of such a large matrix greatly increases the computation time of the Newton algorithm. There are two plots for each simulation. The first shows the estimate for the grid of resistors after the twenty-fifth iteration, and the second plot shows the progression of the RMS current error and the resistor error. This coarser grid is simulated with and without noise for $\varepsilon = 10^{-4}$ and without noise for $\varepsilon = 10^{-7}$. The plots below show the results.

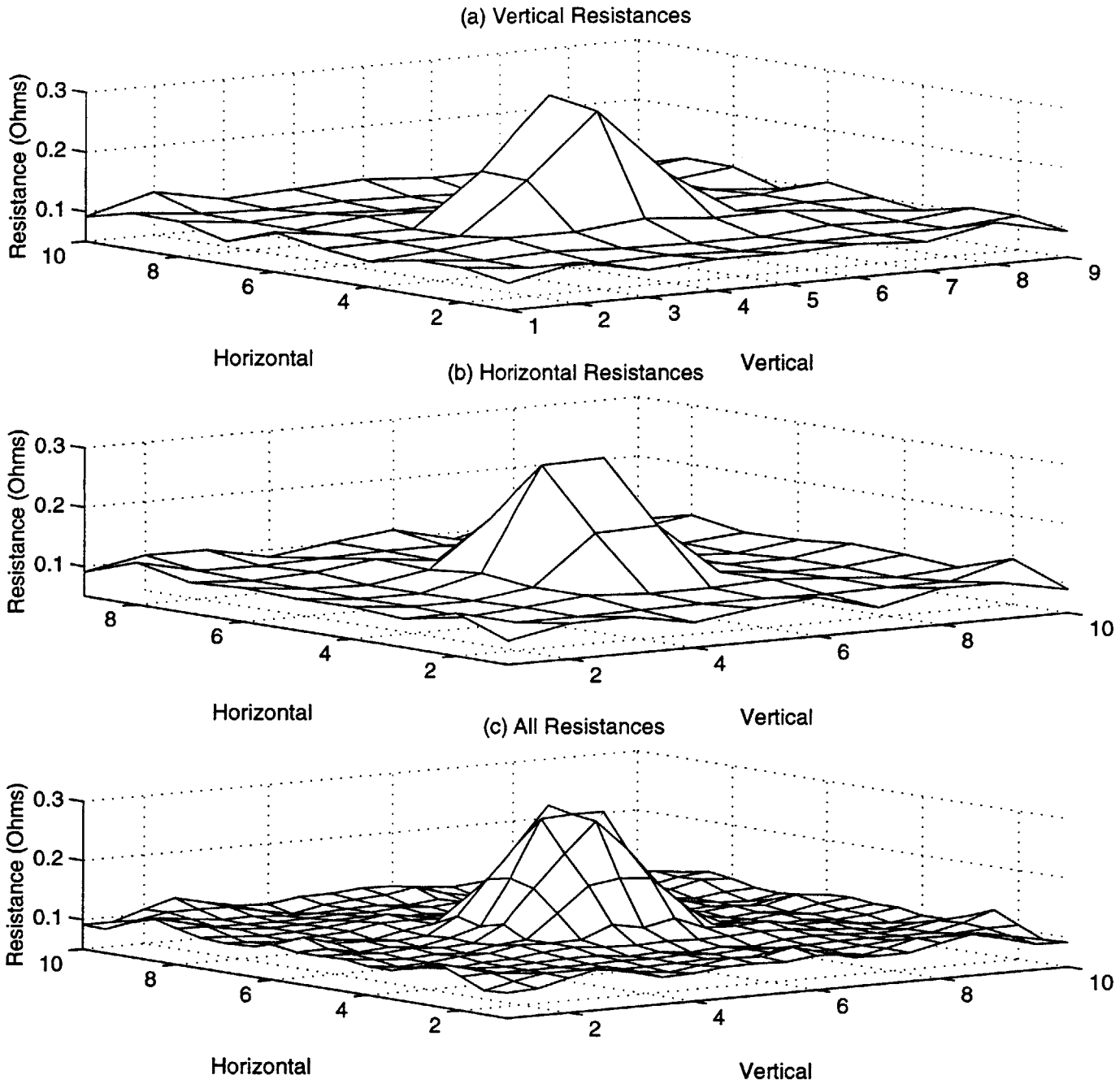



Figure 45: Resistor values after 25 iterations of a noiseless system with $\varepsilon = 10^{-4}$

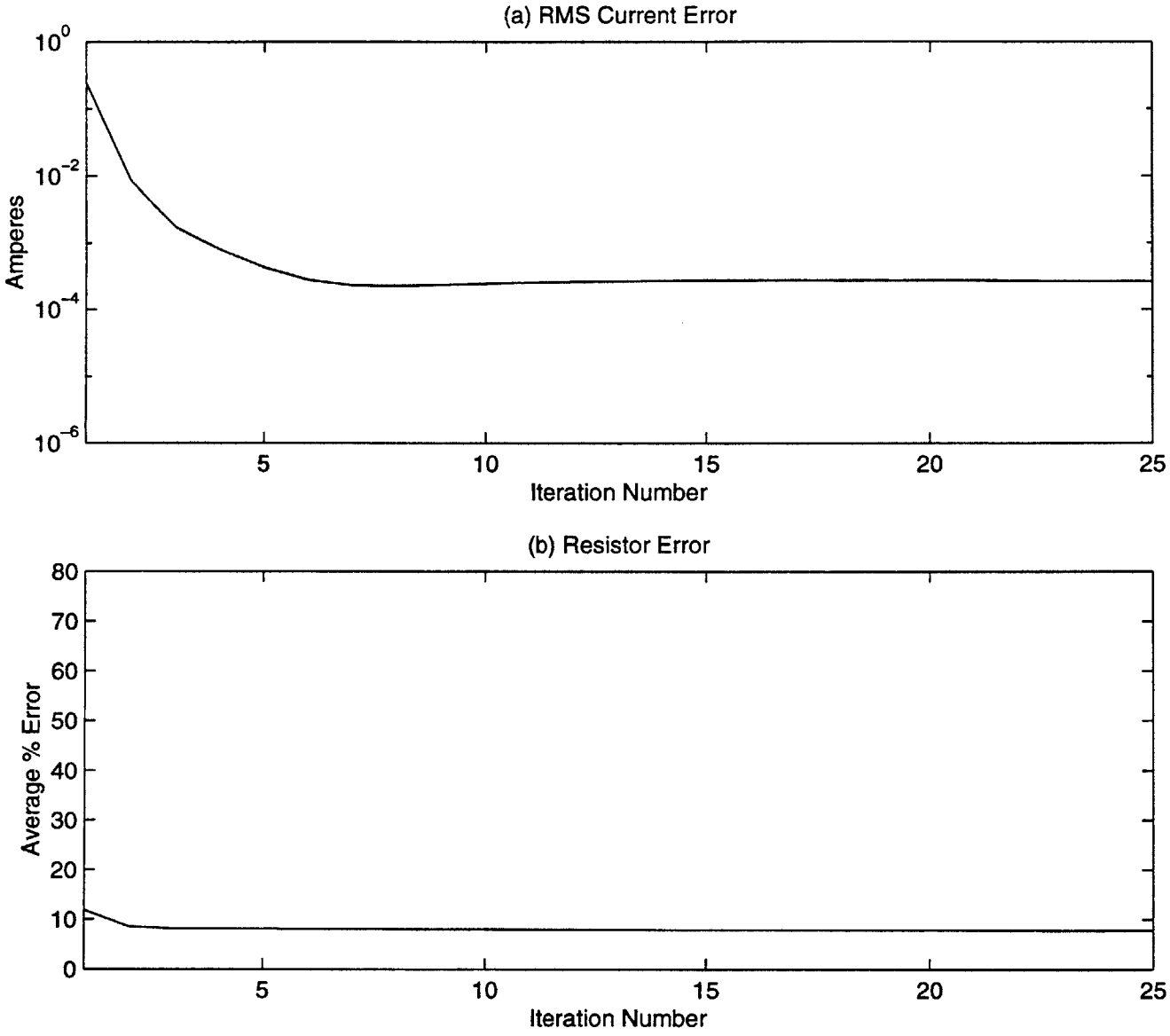



Figure 46: Error metrics for 25 iterations of a noiseless system with $\varepsilon = 10^{-4}$

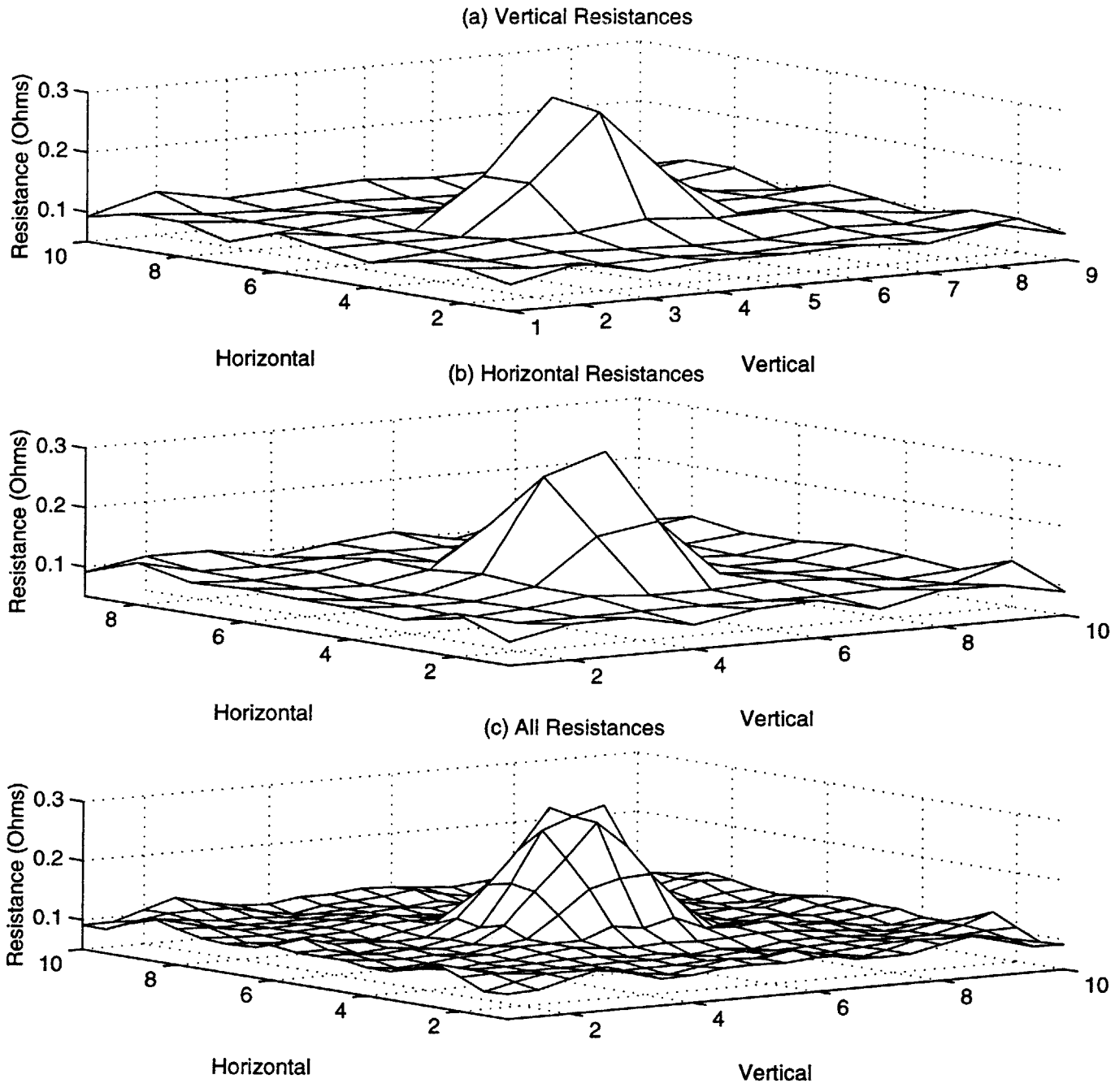



Figure 47: Resistor values after 25 iterations of a system with 1% measurement error and $\varepsilon = 10^{-4}$

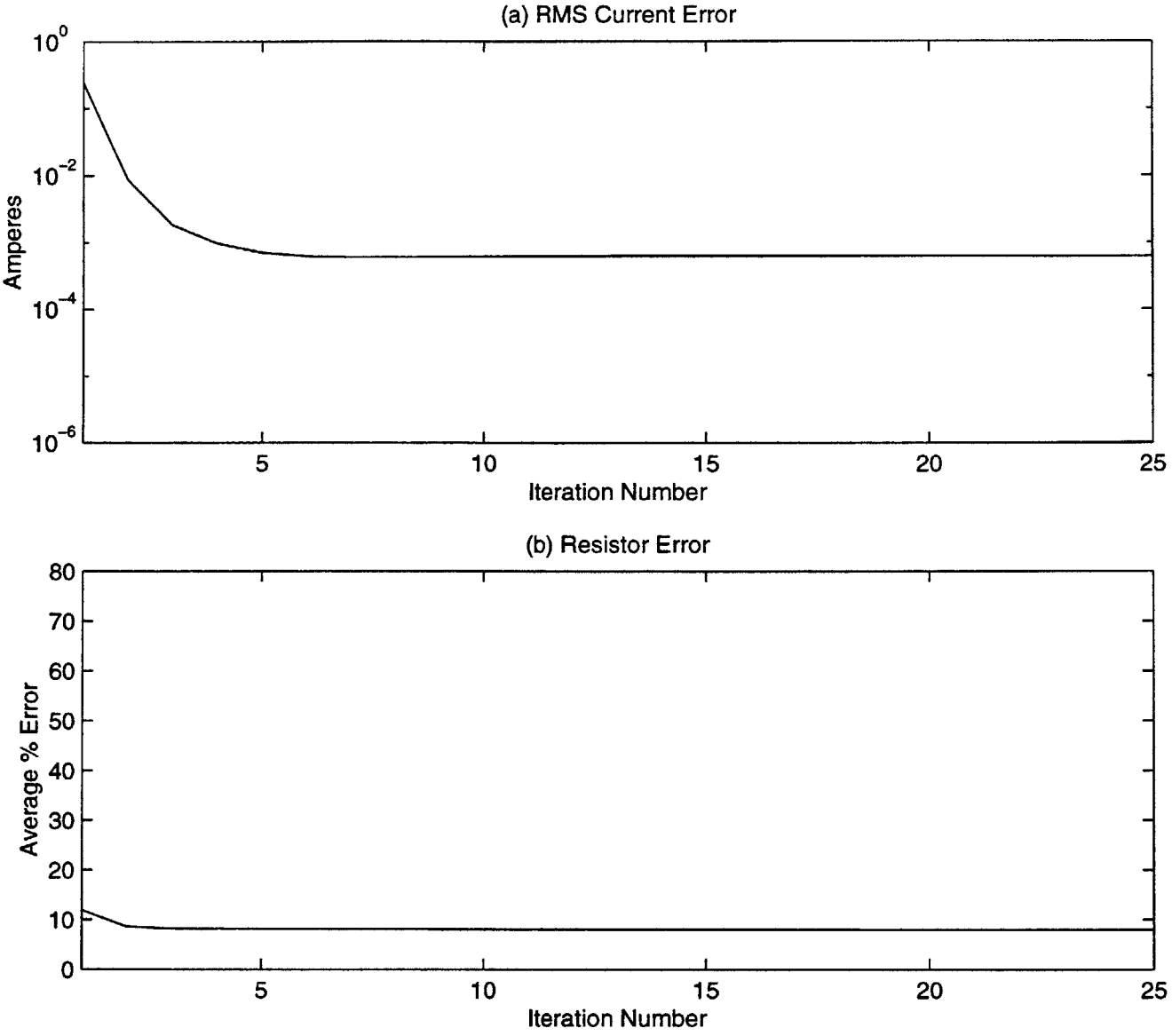



Figure 48: Error metrics for 25 iterations of a system with 1% measurement error and $\epsilon = 10^{-4}$

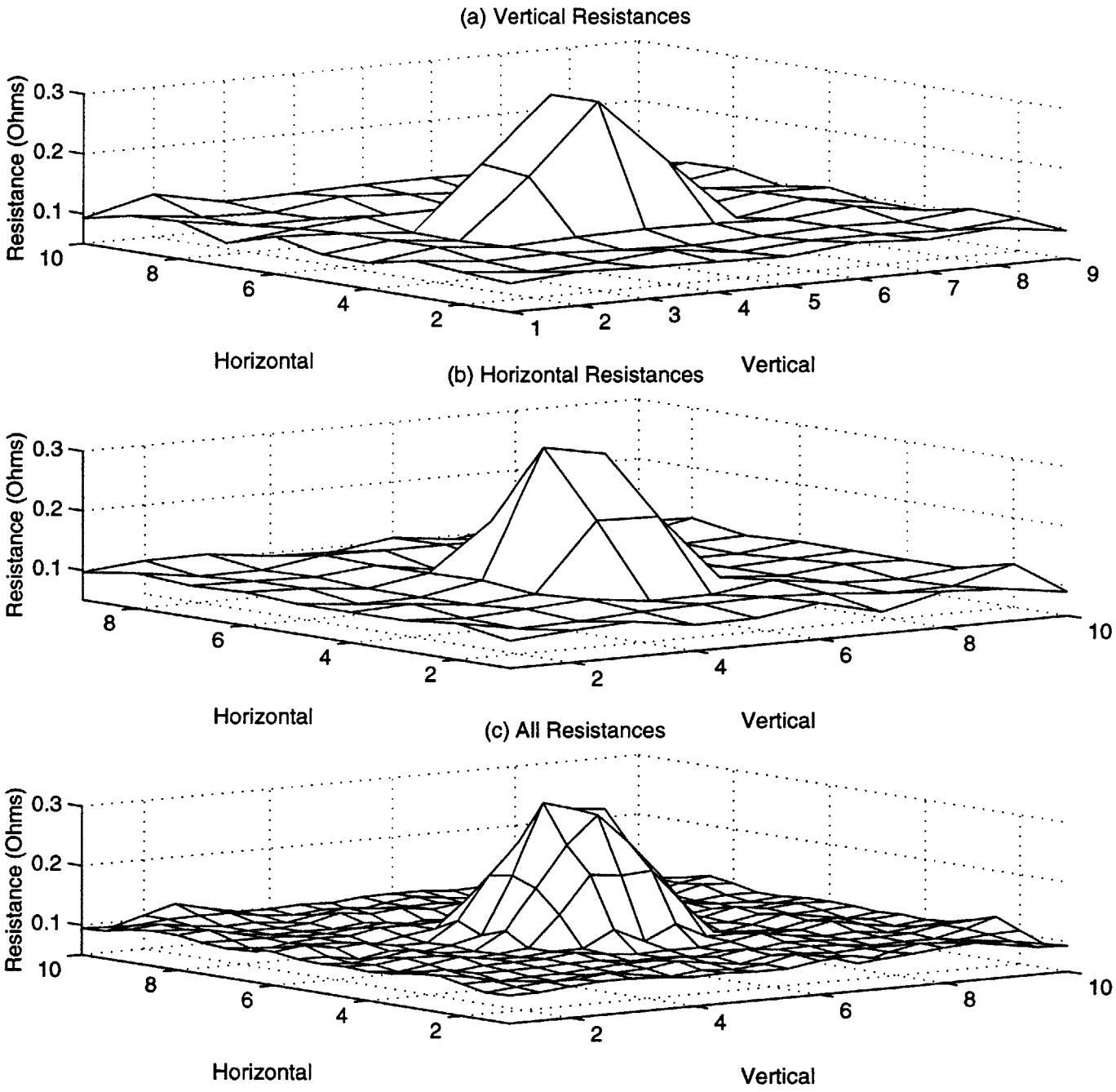



Figure 49: Resistor values after 25 iterations of a noiseless system with $\varepsilon = 10^{-7}$

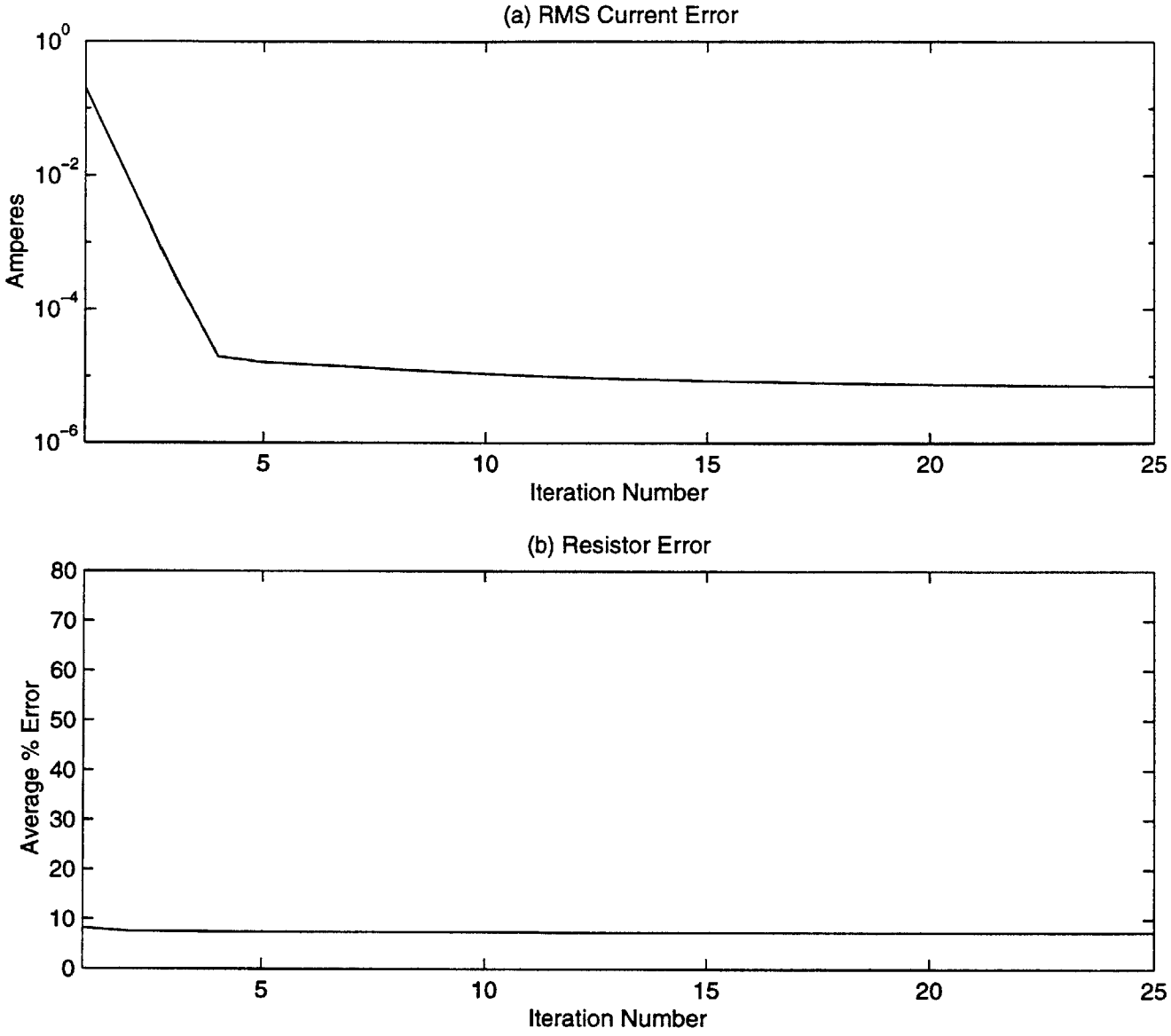



Figure 50: Error metrics for 25 iterations of a noiseless system with $\varepsilon = 10^{-7}$

These plots again show that increasing noise in the system will decrease the accuracy of final results. Also, smaller values of $\varepsilon$ cause the solution to be found more quickly but at the risk of increasing the condition number and causing erratic results for high levels of noise.

## Section 5: Elimination of Regularization

A regularization term is used because the matrix $\left[\mathbf{P}^{(k)} \quad \mathbf{W}^{(k)}\right]$ is rank deficient. Although this matrix has 608 rows[17] and 560 columns[18], the rank of the matrix is 539 and therefore cannot be solved using least squares methods. The deficiency results from the fact that current drives four, five, and six shown in Figure 5 are mirror images of current drives one, two and three. So the KCL equations that are written at the non-reference coarse nodes under current drives four, five, and six provide the same information as the KCL equations from current drives one, two, and three. There are three times 23 redundant equations.[19] The rank of 539 results from removing these 69 dependent equations from the original 608 KCL equations.

The solution to the rank deficiency problem is to designate a set of new current drives that are all linearly independent of each other. In general, nodes close to the current source and sink are "weighted" more than the others in a particular drive because a larger amount of current flows through these nodes. So the new current drives must be selected carefully to attempt to equally weight all nodes in the fine grid.

---

[17] One KCL equation is written at each non-reference node under each of the eight drives. This creates eight times 76 equals 608 equations.

[18] There are 560 unknowns. These are the 136 unknown resistors and 8 times 53 unknown voltage measurements.

[19] There are 23 coarse node KCL equations per drive because no equation is written at the reference node.

The new current drives are used in simulations both with and without regularization. The simulations with the regularization term result in resistor estimates more or less equal to those that resulted from simulations with the eight original current drives. Without the regularization term, the simulations with the new current drives result in resistor estimates that are erratic and unpredictable. The reason for such behavior is not yet understood.

*Chapter 8*

# CONCLUSIONS AND FURTHER EXPLORATIONS

This project began with using a forward and an inverse solver with a full set of voltage measurements and then moved to creating an inverse solver that manages to obtain accurate solutions given minimal information. Many different variations are tried, ranging from changing the number of current drives used to altering the method of grid initialization. Some situations resulted in positive results while others came up short. It is important to not only see where the solvers work, but also understand the shortcomings and how they may possibly be improved.

## Section 1: Successes

As far as successes are concerned, this project did achieve the goal that it set out to accomplish. An inverse solver was developed that solved for a fine grid of resistances when provided with voltage measurements only along a coarse grid. The solver based on Newton's method not only determines the resistive grid, but also manages to do so under a fair amount of noise; however, the ability to solve noisy systems is traded for decreased computational speed. The following table gives the relative amount of work necessary to use each solver. The units of computation are flops, which indicates the number of floating point operations for each calculation. Computation time is measured for the solver initialization, for each iteration, and for the entire system solution.

| | Flops for Initialization | Flops for Each Iteration | Total # of Iterations | Flops for Entire Solution |
|---|---|---|---|---|
| Dense Forward Problem | N/A | $7.3 \times 10^7$ | 1 | $7.3 \times 10^7$ |
| Dense Inverse Problem | N/A | $8.4 \times 10^8$ | 1 | $8.4 \times 10^8$ |
| Alternating Algorithm (6 drives) | $8.0 \times 10^6$ | $2.9 \times 10^8$ | 1000 | $2.9 \times 10^{11}$ |
| Alternating Algorithm (8 drives) | $9.3 \times 10^6$ | $4.7 \times 10^8$ | 1000 | $4.7 \times 10^{11}$ |
| Newton with $\varepsilon = 0.1$; (1 to 4 ratio) | $1.1 \times 10^8$ | $3.0 \times 10^9$ | 100 | $3.0 \times 10^{11}$ |
| Newton for all other $\varepsilon$; (1 to 4 ratio) | $1.1 \times 10^8$ | $3.0 \times 10^9$ | 25 | $7.6 \times 10^{10}$ |
| Newton's Method (1 to 9 ratio) | $6.0 \times 10^{10}$ | $6.3 \times 10^{10}$ | 25 | $1.6 \times 10^{12}$ |

Table 1: Computation times for various solvers developed in this document

Previously, a reason was not stated for choosing the number of iterations for each solver. The above table shows that one thousand iterations of the alternating algorithm require approximately the same number of flops as one hundred iterations of the Newton method. The final solutions of these two algorithms can then be compared to measure the efficiency of each solver as a function of time. Each of the following figures shows a comparison between the alternating algorithm, Newton's method with $\varepsilon = 0.1$, and Newton's method with $\varepsilon = 10^{-3}$. These comparisons are made for the noiseless system, the system with

approximately 1% measurement error, and the system with approximately 10% measurement error. The plots are shown below.

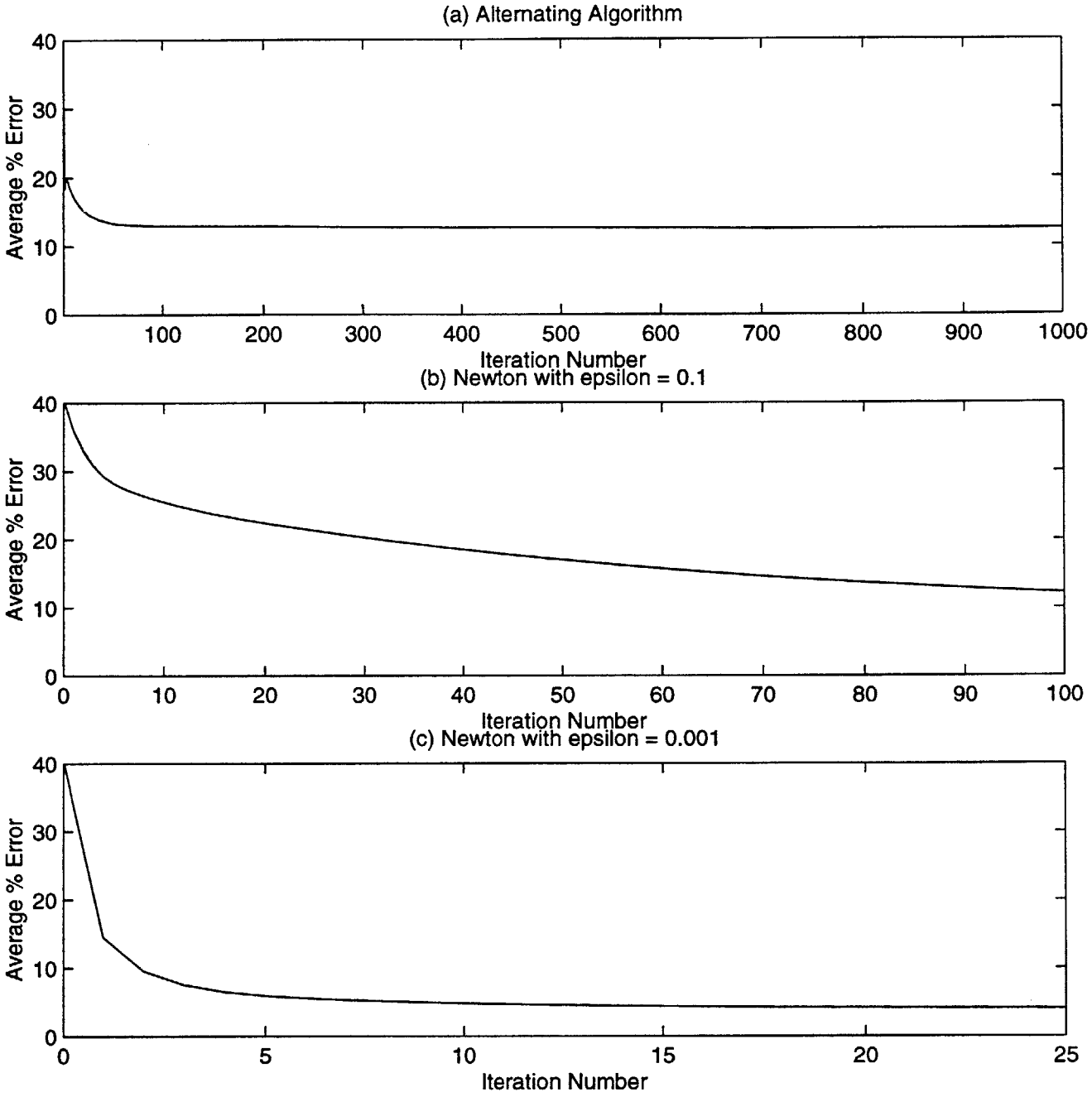



Figure 51: Comparison of resistor error for noiseless systems

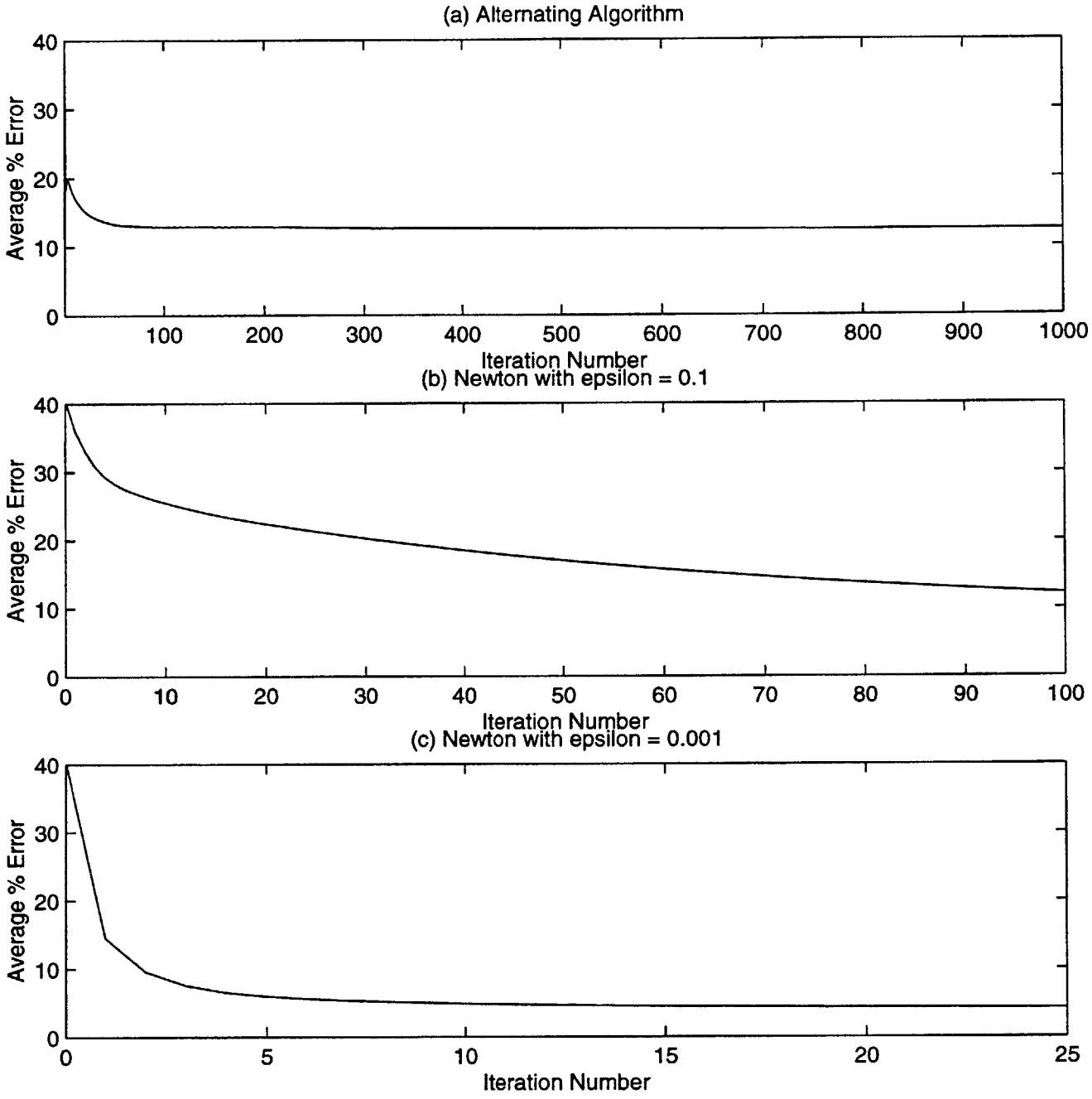



Figure 52: Comparison of resistor error for systems with approximately 1% measurement error

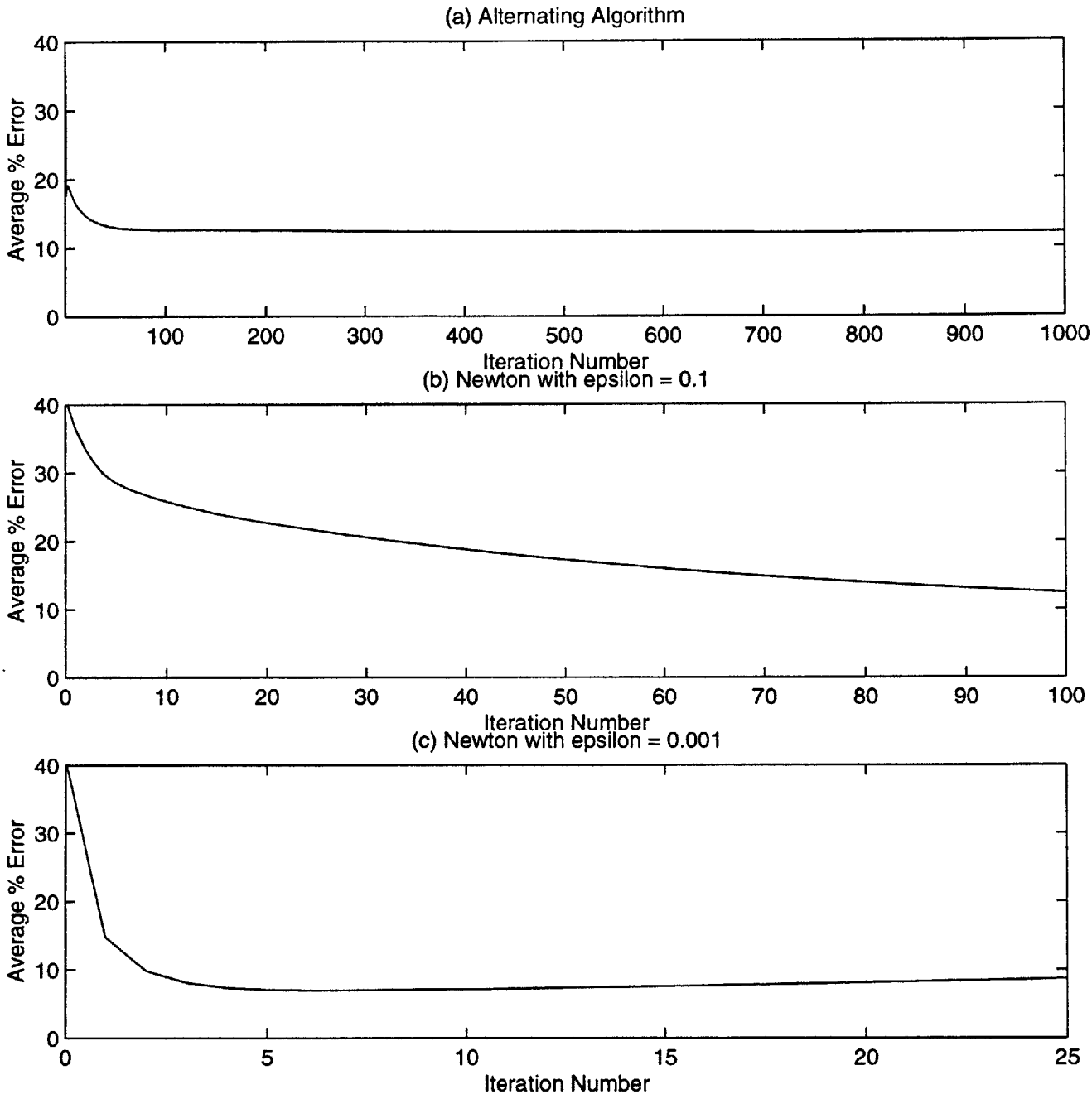



Figure 53: Comparison of resistor error for systems with approximately 10% measurement error.

The plots above show that for the same amount of computation time, Newton's method is generally better than the alternating algorithm. The word "better" refers to having the smallest possible resistor error. The only complication with Newton's method is choosing the proper value of $\varepsilon$ so that the system approaches the correct answer quickly and in a stable progression.

Another success of the solver is that it is very versatile. Parameters such as the number of current drives, their locations, the amount of regularization, the variance of noise in a generated system, the coarseness of the grid, and the actual grid size can all be modified by simply changing the values of certain constants. It is also very easy to change between different interpolation methods, the number of iterations performed, and the algorithms used to calculated the actual linear least squares solution.

## Section 2: Failures

Unfortunately, there are also a few shortcomings to the solvers that were developed. Although not ideal, the averaging method is sufficient for initializing the fine grid and actually saves a great deal both in terms of computation time and complexity. For this reason, the linear interpolation method was never used for initializing a system. It is possible that this or some other interpolation method may actually improve accuracy so vastly that the method is more valuable than the complexity and timesavings of the averaging interpolation. Also, some combination of Newton's method and the alternating algorithm could be optimal to either method on its own. Such a combination was not experimentally or theoretically derived.

A few areas exist where simulations fail. One example is the situation where only six drives are used with the alternating method, resulting in the solution heading

in the wrong direction. Also significant is that if the regularization in Newton's method is too small relative to the amount of noise present in the system, the problem will be ill-conditioned and the resistor estimates will contain unpredictable errors.

Although the reason for the rank-deficiency of the matrix $\left[\mathbf{P}^{(k)} \quad \mathbf{W}^{(k)}\right]$ was found and remedied, the new set of current drives that were used did not create a solution that provided accurate results. The simulations produced resistor estimates that were quite erratic and did not seem logically founded. The reason for such behavior was neither experimentally nor theoretically understood.

## Section 3: Physical Applications and Further Work

The SIDO solver has a number of applications, the most obvious of which is the modeling of the conductance of metal sheets. This area of interest is actually much larger than it sounds. While goals can be as simple as trying to determine the properties of a single sheet of metal, they can also be as complicated as searching for corrosion in a pipe system since corrosion affects the pipe conductances. The basic premise is that any problem that can be modeled as an overdetermined current flow problem can make use of a SIDO solver. The solver is most beneficial in systems where the cost of implementing more voltage measurement devices is higher than the cost of simply running different "drives" or "flows" across a fewer number of "nodes."

There are a number of directions to head from here. Section 2 outlined some of the simulations that fail. It would be beneficial to investigate the exact threshold levels at which a system seems to become unstable and proceed in the wrong direction. An understanding is needed for why the solution to the rank-deficient matrix problem does not result in simulations with accurate results. Furthermore,

a complete understanding of the physical properties of a grid and their relation to a continuous sheet is important. This could allow for an optimal interpolation method since the aforementioned instability could be circumvented with a better initialization method.

Also, it is preferable to construct something other than rectangular grids to better model most physical situations. Furthermore, the coarse nodes where measurements are made do not need to follow a grid pattern and could be a random set of nodes to better model real situations. Real problems have all sorts of abnormalities and kinks and do not follow square patterns.

This thesis has made the first step in showing some of the benefits of a SIDO solver. However this thesis has focused primarily on showing that such benefits do theoretically exist. The next step is to modify the model to better represent real life situations. Once this system is more user-customizable, the possibilities for applications of the SIDO solver are endless.

# BIBLIOGRAPHY


Bau, David and Trefethen, Lloyd N. *Numerical Linear Algebra*, 1st ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

Desoer, Charles A. and Kuh, Ernest S. *Basic Circuit Theory*, 1st ed. New York, NY: McGraw-Hill Book Company, 1969.

Nahib, S. Hamid, Oppenheim, Alan V., and Willsky, Alan S. *Signals and Systems*, 2d. ed. Upper Saddle River, NJ: Prentice Hall, 1997.

Strag, Gilbert. *Introduction to Linear Algebra*, 2d ed. Wellesley, MA: Wellesley-Cambridge Press, 1998.

# APPENDIX A – MATLAB CODE

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                Section 1 - Utilities
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

### * * * branch_incidence.m * * *

```
function A = branch_incidence(plotH, plotW)
% Returns branch incidence matrix for a grid with the dimensions given.

nodes = plotH * plotW; % Total number of nodes
branches = (2*plotH*plotW)-(plotH+plotW); % Total number of conductances

A = zeros(nodes, branches);

% Vertical branches
for j = 1:plotW,
 for i = 1:(plotH-1),
  node_above = ((j-1)*plotH)+i;
  branch_count = ((j-1)*(plotH-1))+i;
  A(node_above, branch_count) = 1; % Branch begins
  A(node_above+1, branch_count) = -1; % Branch ends
 end
end

% Horizontal branches
for i = 1:plotH,
 for j = 1:(plotW-1),
  node_left = ((j-1)*plotH)+i;
  branch_count = ((plotH-1)*plotW)+((i-1)*(plotW-1))+j;
  A(node_left, branch_count) = 1; % Branch begins
  A(node_left+plotH, branch_count) = -1; % Branch ends
 end
end
```

### * * * condense.m * * *

```
function newA = condense(fineA,oldH,oldW, newH, newW)
% Obtains a coarse grid with the dimensions given from a fine grid with the dimensions
% given.

newA = zeros(oldH*oldW,1);
count = 1;

for j = 1:2:newW,
 for i = 1:2:newH,
  newA(count) = fineA(((j-1)*newH)+i);
  count = count+1;
 end
end
```

* * * delcol.m * * *

```
function newA = delcol(A,k)
% Deletes column k from Matrix A

[m, n] = size(A);
newA(:,1:k-1) = A(:,1:k-1);
newA(:,k:n-1) = A(:,k+1:n);
```

* * * delrow.m * * *

```
function newA = delrow(A,k)
% Deletes row k from Matrix A

[m, n] = size(A);
newA(1:k-1,:) = A(1:k-1,:);
newA(k:m-1,:) = A(k+1:m,:);
```

* * * forward.m * * *

```
function voltages = forward(resistors, plot_type, plotH, plotW)
% Takes in the type of plot to create, the plot height, the plot width,
% and the values of the resistors in that grid.  The plot type specifies
% the current drive locations.  The voltages returned are the values
% that solve the equation AGA'v = I

% initialize values in G matrix
A = branch_incidence(plotH, plotW);
cutA = delrow(A, plotH*plotW);
branches = (2*plotH*plotW)-(plotH+plotW);
Gdiag = diag(1./resistors);
I = zeros((plotH*plotW-1),1);
cutG = cutA*Gdiag*(cutA');

if (plot_type == 1)
 I(plotH) = 4;
 I(((plotW-1)*plotH)+1) = -4;
elseif (plot_type == 2)
 I(1) = 4;
 I(((plotW-1)*plotH)+1) = -4;
elseif (plot_type == 3)
% I(plotH*plotW) = 4;
 I(((plotW-1)*plotH)+1) = -4;
elseif (plot_type == 4)
% I(plotH*plotW) = 4;
 I(1) = -4;
elseif (plot_type == 5)
% I(plotH*plotW) = 4;
 I(plotH) = -4;
elseif (plot_type == 6)
 I(plotH) = 4;
 I(1) = -4;
elseif (plot_type == 7)
 I(floor(plotH/2))=4;
 I(((plotW-1)*plotH)+floor(plotH/2))=-4;
elseif (plot_type == 8)
 I(floor(plotW/2)*plotH) = 4;
 I(((floor(plotW/2)-1)*plotH)+1) = -4;
end

voltages = [(cutG\I); 0];
```

* * * index_resistor.m * * *

```
function index = index_resistor(x, y, direction, plotH, plotW)
% Returns the index of the resitor located at (x, y) with the specified
% direction (0 = vertical and 1 = horizontal) for a plot with dimensions
% plotH x plotW

if (direction == 0)
 index = ((y-1)*(plotH-1))+x;
else
 index = (plotW*(plotH-1))+((x-1)*(plotW-1))+y;
end
```

* * * initialize_i.m * * *

```
function new_i = initialize_i(plotH, plotW)
% Returns the value of I in the equation AGA'v = I

i1 = zeros(plotH*plotW,1);
i2 = zeros(plotH*plotW,1);
i3 = zeros(plotH*plotW,1);
i4 = zeros(plotH*plotW,1);
i5 = zeros(plotH*plotW,1);
i6 = zeros(plotH*plotW,1);
i7 = zeros(plotH*plotW,1);
i8 = zeros(plotH*plotW,1);

i1(plotH) = 4;
i1(((plotW-1)*plotH)+1) = -4;

i2(1) = 4;
i2(((plotW-1)*plotH)+1) = -4;

i3(plotH*plotW) = 4;
i3(((plotW-1)*plotH)+1) = -4;

i4(plotH*plotW) = 4;
i4(1) = -4;

i5(plotH*plotW) = 4;
i5(plotH) = -4;

i6(plotH) = 4;
i6(1) = -4;

i7(floor(plotH/2))=4;
i7(((plotW-1)*plotH)+floor(plotH/2))=-4;

i8(floor(plotW/2)*plotH) = 4;
i8(((floor(plotW/2)-1)*plotH)+1) = -4;


new_i = [i1; i2; i3; i4; i5; i6; i7; i8];
```

```
                                * * * space_out.m * * *

function newA = space_out(coarseA,fineA,oldH,oldW)
% Inserts the fine values into a coarse matrix
% oldH and oldW are the coarse dimensions

newH = (2*oldH)-1;
newW = (2*oldW)-1;

newA = zeros(newH*newW,1);
count = 1;

for j = 1:newW,
 for i = 1:newH,
  location = ((j-1)*newH)+i;
  if ((mod(i,2) == 1) & (mod(j,2) == 1)),
   newA(location) = coarseA(floor((i+1)/2) + (oldH * (floor((j+1)/2)-1)));
  else
   newA(location) = fineA(count);
   count = count + 1;
  end
 end
end


                              * * * svd_voltages.m * * *
function [x, sigma_max, sigma_min] = svd_voltages(A, b)
% Performs singular value decomposition for the equation
% Ax=b.  Returns the value for the maximum and minimum singular
% values for this equation as well as the solution x

[Q1,Sigma,Q2] = svd(A);

c = Sigma' * Q1' * b;
y = inv(Sigma' * Sigma) * c;
x = Q2 * y;

i = size(Sigma);
i = i(2);
sigma_max = Sigma(1, 1);
sigma_min = Sigma(i, i);
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                          Section 2 - Interpolation Methods
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                              * * * 12to3_interpolation.m * * *

function fine_resistors = 12to3_interpolation(coarse_resistors, oldH, oldW, newH, newW)
% Function is given a coarse grid of resistors, its dimensions, and the dimensions
% for a new grid of resistors.  Function returns the fine grid of resistors with
% these new dimensions.  New grid is obtained using the 12 to 3 interpolation method.

fine_resistors = zeros((2*newH*newW)-(newH+newW),1);

index = 1;
midValue = oldW*(oldH-1); % Number of vertical resistors in coarse grid.

% Vertical resistors
for x = 1:newH-1,
 fine_resistors(index_resistor(x, 1, 0, newH, newW)) =
(coarse_resistors(index_resistor(floor((x+1)/2), 1, 0, oldH,
oldW))+coarse_resistors(index_resistor(floor((x+1)/2), 2, 0, oldH, oldW)))/2; % Value
of vertical resistor on left edge
 for y = 2:newW-1,
  if (mod(y,2) == 0)
   fine_resistors(index_resistor(x, y, 0, newH, newW)) =
(coarse_resistors(index_resistor(floor((x+1)/2), (y/2), 0, oldH,
oldW))+coarse_resistors(index_resistor(floor((x+1)/2), ((y/2)+1), 0, oldH, oldW)))/2; %
Value of vertical resistor in center & on coarse grid
  else
   fine_resistors(index_resistor(x, y, 0, newH, newW)) =
(coarse_resistors(index_resistor(floor((x+1)/2), floor(y/2), 0, oldH,
oldW))+(2*coarse_resistors(index_resistor(floor((x+1)/2), (floor(y/2)+1), 0, oldH,
oldW)))+coarse_resistors(index_resistor(floor((x+1)/2), (floor(y/2)+2), 0, oldH,
oldW)))/4;% Vertical resistor in center & not on coarse grid
  end
 end
 fine_resistors(index_resistor(x, newW, 0, newH, newW)) =
(coarse_resistors(index_resistor(floor((x+1)/2), oldW-1, 0, oldH,
oldW))+coarse_resistors(index_resistor(floor((x+1)/2), oldW, 0, oldH, oldW)))/2; %
Value of vertical resistor on right edge
end

% Horizontal Resistors
for y = 1:newW-1,
 fine_resistors(index_resistor(1, y, 1, newH, newW)) =
(coarse_resistors(index_resistor(1, floor((y+1)/2), 1, oldH,
oldW))+coarse_resistors(index_resistor(2, floor((y+1)/2), 1, oldH, oldW)))/2; % Value
of horizontal resistor on top edge
 for x = 2:newH-1,
  if (mod(x,2) == 0)
   fine_resistors(index_resistor(x, y, 1, newH, newW)) =
(coarse_resistors(index_resistor((x/2), floor((y+1)/2), 1, oldH,
oldW))+coarse_resistors(index_resistor(floor((x+1)/2), ((y/2)+1), 1, oldH, oldW)))/2; %
Horizontal resistor in center & on coarse grid
  else
   fine_resistors(index_resistor(x, y, 1, newH, newW)) =
(coarse_resistors(index_resistor(floor(x/2), floor((y+1)/2), 1, oldH,
oldW))+(2*coarse_resistors(index_resistor(floor((x/2)+1), (floor(y+1)/2), 1, oldH,
oldW)))+coarse_resistors(index_resistor(floor(x/2)+2, floor((y+1)/2), 1, oldH,
oldW)))/4; % Horizontal resistor in center & not on coarse grid
  end
 end
```

```
 fine_resistors(index_resistor(newH, y, 1, newH, newW)) =
(coarse_resistors(index_resistor(oldH-1, floor((y+1)/2), 1, oldH,
oldW))+coarse_resistors(index_resistor(oldW, floor((y+1)/2), 1, oldH, oldW)))/2; %
Value of horizontal resistor on bottom edge
end
```

* * * avg_interpolation.m * * *

```
function fine_resistors = avg_interpolation(coarse_resistors, oldH, oldW, newH, newW)
% Function is given a coarse grid of resistors, its dimensions, and the dimensions
% for a new grid of resistors.  Function returns the fine grid of resistors with
% these new dimensions.  New grid is obtained by averaging all values in the coarse
% grid and assigning this average value to all the resistors in the fine grid.

avg_value = sum(coarse_resistors)/((2*oldH*oldW)-(oldH+oldW));
fine_resistors = avg_value.*ones((2*newH*newW)-(newH+newW),1);
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                              Section 3 - Error Calculations
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                              * * * error_coarse_voltages.m * * *

function total = error_coarse_voltages(resistors, V1, V2, V3, V4, V5, V6, V7, V8, oldH,
oldW, newH, newW)
% Function that takes in the current estimate of all the conductances.
% Assuming that these resistor measurements are correct, this function
% solves for the coarse voltage values that would have resulted in
% such resistors.  The error metric calculates the value of the average
% absolute value of the percent error between this calculated value
% and the actual given measurement

sum = 0;

% Form AGA' that is used in forward problem
A = branch_incidence(newH, newW);
cutA = delrow(A, newH*newW);
G = diag(1./resistors);
cutG = cutA*G*(cutA');

for plotNumber = 1:8,

 % Initialize value for I depending on current drive
 I = zeros((newH*newW)-1,1);
 if (plotNumber == 1)
  I(newH) = 4;
  I(((newW-1)*newH)+1) = -4;
 elseif (plotNumber == 2)
  I(1) = 4;
  I(((newW-1)*newH)+1) = -4;
 elseif (plotNumber == 3)
 % I(newH*newW) = 4;
  I(((newW-1)*newH)+1) = -4;
 elseif (plotNumber == 4)
 % I(newH*newW) = 4;
  I(1) = -4;
 elseif (plotNumber == 5)
 % I(newH*newW) = 4;
  I(newH) = -4;
 elseif (plotNumber == 6)
  I(newH) = 4;
  I(1) = -4;
 elseif (plotNumber == 7)
  I(floor(newH/2))=4;
  I(((newW-1)*newH)+floor(newH/2))=-4;
 elseif (plotNumber == 8)
  I(floor(newW/2)*newH) = 4;
  I(((floor(newW/2)-1)*newH)+1) = -4;
 end

 voltages_calculated = [(cutG\I); 0];

 voltages_actual = eval(strcat('V',int2str(plotNumber)));

% Iterate through voltage grids and sum up errors only at coarse nodes
 for j = 1:newW,
  for i = 1:newH,
   if ((mod(j,2) == 1) & (mod(i,2) == 1)),
    v_calc = voltages_calculated(((j-1)*newH)+i);
    v_actual = voltages_actual((j-1)*newH+i);
```

```
    if (v_actual ~= 0)
     sum = sum + abs((v_calc-v_actual)./v_actual);
    end
   end
  end
 end

end

total=sum/(8*oldH*oldW);
```

* * * error_lls.m * * *

```
function total = error_lls(resistors, V1, V2, V3, V4, V5, V6, V7, V8, oldH, oldW, newH,
newW)
% Function that takes in the current estimates of both the fine resistor
% grid and all fine voltage grids.  The residual error in AGA'v-I is
% calculated.  The RMS value of this error metric is returned to the user.

A = branch_incidence(newH, newW);
Icalc = [];

for count = 1:8,
 Icalc = [Icalc; A*diag(1./resistors)*(A')*eval(strcat('V',int2str(count)))];
end

actualI = initialize_i(newH, newW);

total = sqrt(sum((abs(Icalc-actualI)).^2)./(8*newH*newW));
```

* * * error_resistors.m * * *

```
function total = error_resistors(resistors)
% Function that takes in the current estimate of the fine resistor grid,
% and because all original resistor values are actually known from their
% specification in the forward problem, this function will calculate
% the average absolute value of the percent error in each of these resistor
% estimates.

% Read in bump grid
data = fopen('data/bump.txt','r');
actual_res = fscanf(data,'%4f');
fclose(data);

Rdiff = abs((actual_res-resistors)./actual_res);

total = sum(Rdiff)/length(Rdiff);
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                Section 4 - Plotting Routines
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                                    * * * fullplot.m * * *

function answer = fullplot(resistors, newH, newW)
% Function that takes all of the conductance values stored in a column vector
% x and graphs them in a 3-D plot.

% counter for current conductance
currentIndex = 0;

% NOTE: since resistors do not lie on actual nodes, we have to
%  use half (.5) values for their locations.  The conductances
%  at other points will be found by averaging the value of the
%  conductances at the nearest resistors.

% Vectors for each dimension
a = 1:0.5:newH;
b = 1:0.5:newW;
C = zeros((2*newH)-1, (2*newW)-1);

% Iterate through vertical resistors
for j=1:newW,
 for i=1:newH-1,
  currentIndex = currentIndex + 1;
  C(2*i,(2*j)-1) = resistors(currentIndex);
 end
end

% iterate through horizontal resistors
for i=1:newH,
 for j=1:newW-1,
  currentIndex = currentIndex + 1;
  C((2*i)-1,2*j) = resistors(currentIndex);
 end
end

% assign values to non-resistor points by averaging the
% nearest resistor values
for j=1:2:(2*newW)-1,
 for i=1:2:(2*newH)-1,
  cursum = 0;
  divisor = 0;
   % add resistor to left if we're not on left edge
   if (j~=1)
    cursum = cursum + C(i,j-1);
    divisor = divisor + 1;
   end
   % add resistor to right if we're not on right edge
   if (j~=(2*newW)-1)
    cursum = cursum + C(i,j+1);
    divisor = divisor + 1;
   end
   % add resistor above if we're not on top edge
   if (i~=1)
    cursum = cursum + C(i-1,j);
    divisor = divisor + 1;
   end
   % add resistor below if we're not on bottom edge
   if (i~=(2*newH)-1)
    cursum = cursum + C(i+1,j);
```

```
    divisor = divisor + 1;
   end
  % average the resistors we've added.
  C(i,j) = cursum/divisor;
 end
end

% assign values to non-resistor points by averaging the
% nearest resistor values
for j = 2:2:2*(newW-1),
 for i = 2:2:2*(newH-1),
  cursum = 0;
  cursum = cursum + C(i,j-1);
  cursum = cursum + C(i,j+1);
  cursum = cursum + C(i-1,j);
  cursum = cursum + C(i+1,j);
  C(i,j) = cursum/4;
 end
end

C = C';
mesh(a,b,C);
fp = 1;
```

```
                              * * * graph_alternate.m * * *

function answer = graph_alternate(iterations, type, drives, sigma, oldH, oldW)
% type 0 = 12 to 3 approximation
% type 1 = averaging approximation
% Graphs results of Alternating Algorithm

newH = (2*oldH)-1;
newW = (2*oldW)-1;

error1 = [];
error2 = [];
error3 = [];
g_k = [];
e_k = [];
resistors = [];

if (type == 0)
 location = strcat('data/alt_123_',int2str(drives));
else
 location = strcat('data/alt_avg_',int2str(drives));
end
if (sigma ~= 0)
 location = strcat(location, '_');
 location = strcat(location, int2str(sigma));
end
location = strcat(location, '.txt');

data = fopen(location,'r');
info =  fscanf(data, '%19f');
fclose(data);

cycle = (2*newW*newH)-(newW+newH)+5;
for i = 0:(iterations-1),
 error1(i+1) = info((cycle*i)+1);
 error2(i+1) = info((cycle*i)+2);
 error3(i+1) = info((cycle*i)+3);
 g_k(i+1) = info((cycle*i)+4);
```

```
 e_k(i+1) = info((cycle*i)+5);
 for j = 1:(2*newW*newH)-(newW+newH),
  resistors(i+1, j) = info((cycle*i)+j+5);
 end
end

figure;
altresistorPlot(resistors(iterations,:)', newH, newW);

figure;
subplot(3, 2, 1);
semilogy(100*error1');
axis([1 iterations 0.001 100]);
axis on;
xlabel('Iteration Number');
ylabel('Average % Error');
title('(a) Coarse Nodes Voltage Error');

subplot(3, 2, 2);
semilogy(error2');
axis([1 iterations 0.000001 1]);
xlabel('Iteration Number');
ylabel('Amperes');
title('(b) RMS Current Error');

subplot(3, 2, 3);
plot(100*error3');
axis([1 iterations 0 40]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(c) Resistor Error');

subplot(3, 2, 4);
plot([1:iterations], g_k, '-', [1:iterations], e_k, ':');
axis([1 iterations 0 100]);
xlabel('Iteration Number');
ylabel('Condition Number');
title('(d) Progression of Condition Numbers');

orient PORTRAIT;
```

* * * graph_compare.m * * *

```
function answer = graph_compare(sigma, oldH, oldW)
% Creates plots comparing alternating algorithm to Newton's method

newH = (2*oldH)-1;
newW = (2*oldW)-1;

error03 = [];
error13 = [];
error23 = [];

l0 = 'data/alt_avg_8';
l1 = 'data/newt_0';
l2 = 'data/newt_2';
if (sigma ~= 0)
 l0 = strcat(l0, '_');
 l0 = strcat(l0, int2str(sigma));
 l1 = strcat(l1, '_');
 l1 = strcat(l1, int2str(sigma));
 l2 = strcat(l2, '_');
```

```
 l2 = strcat(l2, int2str(sigma));
end
l0 = strcat(l0, '.txt');
l1 = strcat(l1, '.txt');
l2 = strcat(l2, '.txt');

data0 = fopen(l0,'r');
info0 =  fscanf(data0, '%19f');
fclose(data0);

data1 = fopen(l1,'r');
info1 =  fscanf(data1, '%19f');
fclose(data1);

data2 = fopen(l2,'r');
info2 =  fscanf(data2, '%19f');
fclose(data2);

cycle = (2*newW*newH)-(newW+newH)+5;
for i = 0:999,
 error03(i+1) = info0((cycle*i)+3);
end

cycle = (2*newW*newH)-(newW+newH)+3;
for i = 0:100,
 error13(i+1) = info1((cycle*i)+3);
 if (i<26)
  error23(i+1) = info2((cycle*i)+3);
 end
end

figure;
subplot(4, 1, 1);
plot(1:1000,100*error03');
axis([1 1000 0 40]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(a) Alternating Algorithm');

subplot(4, 1, 2);
plot(0:100,100*error13');
axis([0 100 0 40]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(b) Newton with epsilon = 0.1');

subplot(4, 1, 3);
plot(0:25,100*error23');
axis([0 25 0 40]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(c) Newton with epsilon = 0.001');

orient PORTRAIT;
```

* * * graph_forward.m * * *

```
function answer = graph_forward(plotH, plotW)
% Plots graphs for dense grid forward problem

data = fopen('data/info_forward.txt','r');
info =  fscanf(data, '%19f');
fclose(data);

V = zeros(16, plotH*plotW);

for i = 1:16,
 for j = 1:(plotH*plotW),
  V(i,j) = info(((i-1)*plotH*plotW)+j);
 end
end

figure;
for count = 1:8,
 subplot(5, 2, count);
 voltagePlot(V(count,:),plotH, plotW);
 xlabel('Vertical');
 ylabel('Horizontal');
 zlabel('Voltage (Volts)');
 title(strcat('Drive ',int2str(count)));
end
orient PORTRAIT;

figure;
for count = 1:8,
 subplot(5, 2, count);
 voltagePlot(V(8+count,:),plotH, plotW);
 xlabel('Vertical');
 ylabel('Horizontal');
 zlabel('Voltage (Volts)');
 title(strcat('Drive ',int2str(count)));
end
orient PORTRAIT;
```

* * * graph_inverse.m * * *

```
function answer = graph_inverse(plotH, plotW)
% Plots graphs for dense grid inverse problem

data = fopen('data/info_inverse.txt','r');
info =  fscanf(data, '%19f');
fclose(data);

branches = (2*plotH*plotW)-(plotH+plotW);

figure;
resistorPlot(info(1:branches), plotH, plotW);

figure;
resistorPlot(info(branches+1:(2*branches)), plotH, plotW);
```

* * * graph_newton.m * * *

```
function answer = graph_newton(iterations, epsilon, sigma, oldH, oldW)
% Plots graphs for Newton Algorithm
```

```
newH = (2*oldH)-1;
newW = (2*oldW)-1;

error1 = [];
error2 = [];
error3 = [];
resistors = [];

location = strcat('data/newt_',int2str(epsilon));
if (sigma ~= 0),
 location = strcat(location, '_');
 location = strcat(location, int2str(sigma));
end
location = strcat(location, '.txt');

data = fopen(location,'r');
info =  fscanf(data, '%19f');
fclose(data);

cycle = (2*newW*newH)-(newW+newH)+3;
for i = 0:iterations,
 error1(i+1) = info((cycle*i)+1);
 error2(i+1) = info((cycle*i)+2);
 error3(i+1) = info((cycle*i)+3);
 for j = 1:(2*newW*newH)-(newW+newH),
  resistors(i+1, j) = info((cycle*i)+j+3);
 end
end

figure;
resistorPlot(resistors((iterations+1),:)', newH, newW);

figure;
subplot(4, 1, 1);
semilogy(delrow(100*error1', 1));
axis([1 iterations 0.01 100]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(a) Coarse Nodes Voltage Error');

subplot(4, 1, 2);
semilogy(delrow(error2',1));
axis([1 iterations 0.000001 1]);
xlabel('Iteration Number');
ylabel('Amperes');
title('(b) RMS Current Error');

subplot(4, 1, 3);
plot(delrow(100*error3',1));
axis([1 iterations 0 40]);
xlabel('Iteration Number');
ylabel('Average % Error');
title('(c) Resistor Error');

orient PORTRAIT;
```

* * * horizResPlot.m * * *

```
function fp = horizResPlot(x, plotH, plotW)
% Function that takes all of the resistor values in x and graphs only the
% horizontal resistors.

currentIndex = plotW*(plotH-1);

b = 1:(plotW-1);
a = 1:plotH;

for i=1:plotH,
 for j=1:(plotW-1),
  currentIndex = currentIndex + 1;
  C(j,i) = x(currentIndex);
 end
end

mesh(a,b,C);
axis tight;
fp = 1;
```

* * * resistorPlot.m * * *

```
function fp = resistorPlot(x, plotH, plotW)
% Function that takes all of the resistor values in x and graphs them
% in three manners: vertical, horizontal, and all

% This first line is inserted in order to prevent there from being a
% graph in the fourth location
subplot(2, 1, 2);

% Vertical Resistors
subplot(4, 1, 1);
vertResPlot(x, plotH, plotW);
axis([1 (plotH-1) 1 plotW 0.05 0.3]);
xlabel('Vertical');
ylabel('Horizontal');
zlabel('Resistance (Ohms)');
title('(a) Vertical Resistances');

% Horizontal Resistors
subplot(4, 1, 2);
horizResPlot(x, plotH, plotW);
axis([1 plotH 1 (plotW-1) 0.05 0.3]);
xlabel('Vertical');
ylabel('Horizontal');
zlabel('Resistance (Ohms)');
title('(b) Horizontal Resistances');

% All Resistors
subplot(4, 1, 3);
fullPlot(x, plotH, plotW);
axis([1 plotH 1 plotW 0.05 0.3]);
xlabel('Vertical');
ylabel('Horizontal');
zlabel('Resistance (Ohms)');
title('(c) All Resistances');

orient PORTRAIT;
```

* * * show_original.m * * *

```
function answer = show_original(plotH, plotW)
% Plots the original resistor grids used by the forward solver

data = fopen('data/flat.txt','r');
flat_grid =  fscanf(data, '%4f');
fclose(data);

data = fopen('data/bump.txt','r');
bump_grid =  fscanf(data, '%4f');
fclose(data);

figure;
resistorPlot(flat_grid, plotH, plotW);

figure;
resistorPlot(bump_grid, plotH, plotW);
```

* * * vertResPlot.m * * *

```
function fp = vertResPlot(x, plotH, plotW)
% Function that takes all of the resistor values in x and graphs only the
% vertical resistors.

currentIndex = 0;
b = (1:plotW)';
a = (1:plotH-1)';

for j=1:plotW,
 for i=1:(plotH-1),
  currentIndex = currentIndex + 1;
  C(j,i) = x(currentIndex);
 end
end

mesh(a,b,C);
axis tight;
fp = 1;
```

* * * voltagePlot.m * * *

```
function fp = voltagePlot(x,plotH,plotW)
% Function that takes all of the voltages values stored in the
% column vector x and graphs them as a 3-D plot

currentIndex = 0;
a = 1:plotH;
b = 1:plotW;
a = a';
b = b';

for j=1:plotW,
 for i=1:plotH,
  C(j,i) = x((plotH*(j-1))+i);
 end
end

mesh(a,b,C);
axis tight;
fp = 1;
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                Section 5 - Main Functions
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                                * * * fine_forward.m * * *

function answer = fine_forward(plotH, plotW)
% This function solves for the voltage grids that result
% from applying the forward solver to the bump and flat grids under
% each of the eight current drives.  The grid size is specified
% by the paraments sent to this function

writeInfo = fopen('data/info_forward.txt','w');

% Read in Resistor Values from flat file
data = fopen('data/flat.txt','r');
flat_resistors = fscanf(data,'%4f');
fclose(data);

% Read in Resistor Values from bump file
data = fopen('data/bump.txt','r');
bump_resistors = fscanf(data,'%4f');
fclose(data);

disp('Writing Info');

% Call forward solver for each of the eight current drives
for count = 1:8,
   current = forward(flat_resistors, count, plotH, plotW);
   for i=1:(plotH*plotW),
    fprintf(writeInfo, '%18.16f ', current(i));
   end
 fprintf(writeInfo, '\n');
end

% Call forward solver for each of the eight current drives
for count = 1:8,
   current = forward(bump_resistors, count, plotH, plotW);
   for i=1:(plotH*plotW),
    fprintf(writeInfo, '%18.16f ', current(i));
   end
 fprintf(writeInfo, '\n');
end

fclose(writeInfo);
```

```
                                * * * fine_inverse.m * * *

function answer = fine_inverse(plotH, plotW)
% This function solves for the resistors that would have created the
% voltage plots that were written to the specified data file.
% The grid size is specified by the paraments sent to this function

% Initialize
drives = 8;
count = 0;
noise_level = 0;
nodes = plotH*plotW;
branches = (2*plotW*plotH)-(plotW+plotH);

A = branch_incidence(plotH, plotW);
Acut = delrow(A, nodes);
fullV1 = zeros(drives*(nodes-1), branches);
fullV2 = zeros(drives*(nodes-1), branches);
voltages = zeros(16, plotH*plotW);

writeInfo = fopen('data/info_inverse.txt','w');

% Read in Voltage Values from file
data = fopen('data/fine_voltages.txt','r');
info = fscanf(data,'%19f');
fclose(data);

for i = 1:16,
 for j = 1:(plotH*plotW),
  voltages(i,j) = info(((i-1)*plotH*plotW)+j) + (noise_level*randn);
 end
end

% Initialize Current
currentI = initialize_i(plotH, plotW);

for i = drives:-1:1,
 currentI = delrow(currentI, i*nodes);
end


% Calculate V1 and V2 - flat and bump
for drive_count = 1:drives,
 currentV1 = voltages(drive_count,:)';
 currentV1 = delrow(currentV1, nodes);
 currentV2 = voltages(8+drive_count,:)';
 currentV2 = delrow(currentV2, nodes);

 for conductance_count = 1:branches,
  for node_count = 1:nodes-1,
   fullV1(((nodes-1)*(drive_count-1))+node_count, conductance_count) = Acut(node_count,
conductance_count)*Acut(:,conductance_count)'*currentV1;
   fullV2(((nodes-1)*(drive_count-1))+node_count, conductance_count) = Acut(node_count,
conductance_count)*Acut(:,conductance_count)'*currentV2;
  end
 end
end

% SVD for flat grid
[G1 g1_sigma_max g1_sigma_min] = svd_voltages(fullV1, currentI);
R1 = 1./G1;

% SVD for bump grid
```

```
[G2 g1_sigma_max g1_sigma_min] = svd_voltages(fullV2, currentI);
R2 = 1./G2;

for i=1:branches,
 fprintf(writeInfo, '%18.16f ', R1(i));
end
fprintf(writeInfo, '\n');

for i=1:branches,
 fprintf(writeInfo, '%18.16f ', R2(i));
end
fprintf(writeInfo, '\n');

fclose(writeInfo);
```

* * * newton.m * * *

```
function answer = newton(oldH, oldW)
% Performs Newton's Method

% Initialize
epsilon = 0.0000001;
drives = 8;
newH = (2*oldH)-1;
newW = (2*oldW)-1;
count = 0;
noise_level = 0.001;
writeR = fopen('data/info_newton.txt','w');

% Read in Resistor Values from file
data = fopen('data/bump.txt','r');
bump_resistors = fscanf(data,'%4f');
fclose(data);

% Determine coarse measurements
coarse1 = condense(forward(bump_resistors, 1, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse2 = condense(forward(bump_resistors, 2, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse3 = condense(forward(bump_resistors, 3, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse4 = condense(forward(bump_resistors, 4, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse5 = condense(forward(bump_resistors, 5, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse6 = condense(forward(bump_resistors, 6, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse7 = condense(forward(bump_resistors, 7, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));
coarse8 = condense(forward(bump_resistors, 8, newH, newW), oldH, oldW, newH, newW) +
(noise_level*randn((oldH*oldW),1));

% Determine Coarse Grid of Conductances
A = branch_incidence(oldH, oldW);
nodes = oldH*oldW;
branches = (2*oldW*oldH)-(oldW+oldH);

% Initialize Coarse current
currentI = initialize_i(oldH, oldW);

for i = drives:-1:1,
 currentI = delrow(currentI, i*nodes);
end
Acut = delrow(A, nodes);

fullQ = zeros(drives*(nodes-1), branches);

% Calculate Q which is used to solve for the coarse conductance grid
for drive_count = 1:drives,
 currentV = eval(strcat('coarse',int2str(drive_count)));
 currentV = delrow(currentV, nodes);

 for conductance_count = 1:branches,
  for node_count = 1:nodes-1,
   fullQ(((nodes-1)*(drive_count-1))+node_count, conductance_count) = Acut(node_count,
conductance_count)*Acut(:,conductance_count)'*currentV;
  end
 end
```

```
end

% SVD for Coarse
[coarse_g g_sigma_max g_sigma_min] = svd_voltages(fullQ, currentI);
coarse_r = 1./coarse_g;
currentR = avg_interpolation(coarse_r, oldH, oldW, newH, newW);
currentG = 1./currentR;

% Initialize for Fine Grid
A = branch_incidence(newH, newW);
nodes = newH*newW;
branches = (2*newW*newH)-(newW+newH);
missing = (newH*newW)-(oldH*oldW);

% Initialize current
totalI = initialize_i(newH, newW);

G = diag(currentG);
fullG = A*G*(A');
totalV = [];

% Fine resistor grid calculuation
for drive_count = drives:-1:1,
 loopG = fullG;
 loopV = eval(strcat('coarse',int2str(drive_count)));
 loopI = totalI((nodes*(drive_count-1))+1:nodes*drive_count);
 for j = oldW:-1:1,
  for i = oldH:-1:1,
   newj = 2*j-1;
   newi = 2*i-1;
   loopI = loopI - (loopV(((j-1)*oldH)+i)*loopG(:,((newj-1)*newH)+newi));
   loopG = delcol(loopG, ((newj-1)*newH)+newi);
  end
 end

 [loopU e_sima_max e_sigma_min] = svd_voltages(delrow(loopG,nodes), delrow(loopI,
nodes));
 newV = space_out(loopV, loopU, oldH, oldW);
 totalV = [newV; totalV];

end

loop1 = totalV((0*nodes)+1:1*nodes,:);
loop2 = totalV((1*nodes)+1:2*nodes,:);
loop3 = totalV((2*nodes)+1:3*nodes,:);
loop4 = totalV((3*nodes)+1:4*nodes,:);
loop5 = totalV((4*nodes)+1:5*nodes,:);
locp6 = totalV((5*nodes)+1:6*nodes,:);
loop7 = totalV((6*nodes)+1:7*nodes,:);
loop8 = totalV((7*nodes)+1:8*nodes,:);

% Calculating Errors
 error1 = error_coarse_voltage(currentR, loop1, loop2, loop3, loop4, loop5, loop6,
loop7, loop8, oldH, oldW, newH, newW);
 error2 = error_lls(currentR, loop1, loop2, loop3, loop4, loop5, loop6, loop7, loop8,
oldH, oldW, newH, newW);
 error3 = error_resistors(currentR);

% Writing Info
 row_val = [error1, error2, error3, currentR'];
 for i=1:(3+branches),
  fprintf(writeR, '%18.16f ', row_val(i));
 end
```

```
 fprintf(writeR, '\n');

count = 0;

while (count < 25),
 count = count + 1;
 disp(strcat('loop # ',int2str(count)));

 A = branch_incidence(newH, newW);
 cutA = delrow(A, nodes);
 totalI = initialize_i(newH, newW);
 cutI = totalI;
 for drive_count = drives:-1:1,
  cutI = delrow(cutI, nodes*drive_count);
 end

 G = diag(currentG);
 cutG = cutA*G*(cutA');

% Remove AGAt from i
 AGAtV = [(cutG*delrow(loop1,nodes)); (cutG*delrow(loop2,nodes));
(cutG*delrow(loop3,nodes)); (cutG*delrow(loop4,nodes)); (cutG*delrow(loop5,nodes));
(cutG*delrow(loop6,nodes)); (cutG*delrow(loop7,nodes)); (cutG*delrow(loop8,nodes))];
 loopI = cutI - AGAtV;

 for j = newW:-2:1,
  for i = newH:-2:1,
   cutG = delcol(cutG, ((j-1)*newH)+i);
  end
 end

% Calculating Delta Matrix for All Voltages

 nil = zeros(size(cutG));

 M = [cutG nil nil nil nil nil nil nil;
      nil cutG nil nil nil nil nil nil;
      nil nil cutG nil nil nil nil nil;
      nil nil nil cutG nil nil nil nil;
      nil nil nil nil cutG nil nil nil;
      nil nil nil nil nil cutG nil nil;
      nil nil nil nil nil nil cutG nil;
      nil nil nil nil nil nil nil cutG];

 N = zeros(drives*(nodes-1), branches);

% Calculating Delta Matrix for G
 for drive_count = 1:drives,
  addition_loop = delrow(eval(strcat('loop',int2str(drive_count))), nodes);
  for conductance_count = 1:branches,
   for node_count = 1:nodes-1,
    N(((drive_count-1)*(nodes-1))+node_count, conductance_count) = cutA(node_count,
conductance_count) * cutA(:,conductance_count)' * addition_loop;
   end
  end
 end

 deltaA = [M N];

% Performing SVD Calculation
 [deltaX X_sigma_max X_sigma_min] =
svd_voltages((deltaA'*deltaA)+(epsilon*eye(branches+(drives*missing))),
(deltaA'*loopI));
```

```
% Separating Deltas
 deltaV1 = space_out(zeros(oldH*oldW), deltaX((0*missing)+1:1*missing), oldH, oldW);
 deltaV2 = space_out(zeros(oldH*oldW), deltaX((1*missing)+1:2*missing), oldH, oldW);
 deltaV3 = space_out(zeros(oldH*oldW), deltaX((2*missing)+1:3*missing), oldH, oldW);
 deltaV4 = space_out(zeros(oldH*oldW), deltaX((3*missing)+1:4*missing), oldH, oldW);
 deltaV5 = space_out(zeros(oldH*oldW), deltaX((4*missing)+1:5*missing), oldH, oldW);
 deltaV6 = space_out(zeros(oldH*oldW), deltaX((5*missing)+1:6*missing), oldH, oldW);
 deltaV7 = space_out(zeros(oldH*oldW), deltaX((6*missing)+1:7*missing), oldH, oldW);
 deltaV8 = space_out(zeros(oldH*oldW), deltaX((7*missing)+1:8*missing), oldH, oldW);
 deltaG = deltaX((drives*missing)+1:(drives*missing)+branches);

% Updating
 loop1 = loop1 + deltaV1;
 loop2 = loop2 + deltaV2;
 loop3 = loop3 + deltaV3;
 loop4 = loop4 + deltaV4;
 loop5 = loop5 + deltaV5;
 loop6 = loop6 + deltaV6;
 loop7 = loop7 + deltaV7;
 loop8 = loop8 + deltaV8;
 currentG = currentG + deltaG;
 currentR = 1./currentG;

% Calculating Errors
 error1 = error_coarse_voltage(currentR, loop1, loop2, loop3, loop4, loop5, loop6,
loop7, loop8, oldH, oldW, newH, newW);
 error2 = error_lls(currentR, loop1, loop2, loop3, loop4, loop5, loop6, loop7, loop8,
oldH, oldW, newH, newW);
 error3 = error_resistors(currentR);

% Writing Info
 row_val = [error1, error2, error3, currentR'];
 for i=1:(3+branches),
  fprintf(writeR, '%18.16f ', row_val(i));
 end
 fprintf(writeR, '\n');

end

fclose(writeR);
```