UNIVERSITY OF CALIFORNIA, MERCED

Graph Based Scalable Algorithms with Applications

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

in

Applied Mathematics

by

Garnet Jason Vaz

Committee in charge:

Professor Harish S. Bhat, Chair

Professor Mayya Tokman

Professor Arnold D. Kim

2014

The Dissertation of Garnet Jason Vaz is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Mayya Tokman

Arnold D. Kim

Harish S. Bhat, Chair

University of California, Merced

2014

To my Aunt, Lynette

Contents

0.1 Acknowledgements

First and foremost I would like to thank my advisor Harish Bhat for his support and guidance throughout my PhD career. He has been the advisor that I wanted and the one that I needed equally well and has helped me grow both professionally and personally in ways beyond description and I will forever be in his debt.

I would like to thank the guidance provided by my other committee members Mayya Tokman and Arnold Kim. Their patience in listening towards my concerns and providing guidance in my research has been invaluable.

I would like to thank the helpful staff at the School of Natural Sciences office and especially Carrie King for making all the paper work disappear. The staff at the International Office have been extremely helpful in making my stay here hassle free.

My studies would not have been possible without the support and love from my Dad and my aunt. They have always believed in me and encouraged me to search my own path.

Being away from my school friends was hard but my colleagues here including Nitesh Kumar, Jane Hyojin Lee and Derya Şahin have taught me how to smile. They have been around to share in my laughter and more importantly supported me when I was down. It would be unfair to call them friends and so to me they will always be family.

I would also like to thank the UC Merced Open Access Fund Pilot and U.S. Department of Energy (Contract No. DE-AC02-05CH11231, Subaward 7041635) for supporting my research.

Graph Based Scalable Algorithms with Applications.

by

Garnet Jason Vaz

University of California, Merced, 2014

Prof. Harish S. Bhat, Chair

ABSTRACT OF THE DISSERTATION

In this thesis, we propose various algorithms for problems arising in nonlinear circuits, nonlinear electromagnetics and data mining. Through the design and implementation of these algorithms, we show that the algorithms developed are scalable.

In the first part of the thesis we provide two solutions to the forward problem of finding the steady-state solution of nonlinear RLC circuits subjected to harmonic forcing. The work generalizes and provides a mathematical theory bridging prior work on structured graphs and extending it to random graphs. Both algorithms are shown to be orders of magnitude faster than time stepping. We introduce an inverse problem of maximizing the energy/voltage at certain nodes of the graph without altering the graph structure. By altering the eigenvalues associated with the weighted graph Laplacian of the underlying circuit using a Newton-type algorithm, we solve the inverse problem. Extensive results verify that a majority of random graph circuits are capable of causing amplitude boosts.

Next, we connect nonlinear Maxwell's equations in 2D to the RLC circuit problem. This relationship is achieved by considering the finite volume decomposition of nonlinear Maxwell's equations. When we consider a discretization of the domain, the dual graph of this discretization provides us with a planar random graph structure very similar to our previous work. Thus, algorithms developed in the previous work become applicable. Using distributed computing, we develop an implementation of one of the algorithms that scales to large-scale problems allowing us to obtain accurate and fast solutions. Simulations are conducted for structured and unstructured meshes, and we verify that the method is first-order in space.

Our final application is in the field of supervised learning for regression problems. Regression trees have been used extensively since their introduction and form the basis of several state-of-the-art machine learning methods today. Regression trees minimize the loss criterion (objective function) using a greedy heuristic algorithm. The usual form of the loss criterion is the squared error. While it has been known that minimizing the absolute deviation provides more robust trees in the presence of outliers trees based on absolute loss minimization have been ignored because they were believed to be computationally expensive. We provide the first implementation which has the same algorithmic complexity as compared to trees built with the squared error loss function. Besides computing absolute deviation trees, our algorithm generalizes and can be used as a non-parametric alternative to quantile regression.

Chapter 1

Introduction

The increase in computational power over the last two decades has led to massive advances in our ability to solve a variety of mathematical problems. The growing computational power in turn has resulted in a desire to solve even larger problems. The size of the problems we solve routinely nowadays might have seemed impossible 2–3 decades ago. With such impressive advances it may seem that in order to solve problems of current interest, we may just have to wait for another decade. This line of reasoning is flawed. The ability of hardware to speed up computations has stalled due to the inability to increase computational speed beyond its current limit while providing energy efficient processors. For almost a decade now processor speeds have not increased according to prior trends. This has impeded our ability to speed up computations. Instead of merely relying on hardware advances to speed our work, the scientific community has branched out towards alternate methods to feed our computational hunger. Rather than rely on a single technique there now exist a variety of methods depending on our needs. For example, if our applications lie in an area which includes high structured computing like BLAS based operations we can now use GPU's. An alternative to GPU's is the Intel Xeon Phi co-processor, which allows one to execute X86 instruction sets that do not require any modifications unlike CUDA or OpenCL codes for GPU's. For applications which include large data manipulations, the MapReduce framework is very popular and depending on the problem allows one to handle terabytes of data.

While it may seem that these new technologies and frameworks allow us to satisfy any computational needs we might have, the truth is that there is a lot of work required to leverage their full potential. Existing algorithms do not directly transfer over to newer technologies. The LAPACK set of libraries that form the essence of linear algebra computations cannot be used directly on GPU's since code written for GPU's requires us to micro-manage several layers of memory very carefully. A naive implementation would result in very bad performance. Similarly, the MapReduce framework used very widely in data analysis and machine learning requires one to rethink many algorithmic concepts of existing methodologies. As a result, much of current research is focused on developing new algorithms that can efficiently use these technologies.

There are two approaches to building scalable methods for mathematical problems: either start with an existing algorithm and improve it to make it faster and scalable, or develop an algorithm with scalability in mind. The current work focuses on the latter. We design and implement new algorithms for some problems of interest, verifying their potential to scale as desired. All the algorithms involve graphs, either random or in the form of

binary trees. Graphs provide a very powerful and abstract concept towards a wide variety of problems. Recently there has been much interest in random graphs for computations and network analysis. Graphs also form a crucial part of the underpinnings of computer science. As a result, data structures and algorithms for graph based approaches to a variety of problems are well-known and many efficient libraries exist in every major language. The current work builds algorithms for three problems.

In Chapter 2, we present the problem of solving for the frequency response of nonlinear electrical oscillator networks. The networks are formed by interconnecting capacitors and resistors using inductors. The circuit is forced with a harmonic frequency and the aim it to find its steady-state response. If we construct a single chain circuit we obtain an approximation of transmission lines. While transmission lines have been studied extensively, most recent work has focused on the use of saturating, voltage-dependent capacitors yielding nonlinear transmission lines. Nonlinear transmission lines thus created using semi-conductor materials offer a wide range of possibilities in applications for signal processing and filtering [57], [48], [4], [1]. Their success in one-dimensional applications naturally led to their extension towards two-dimensional theory and applications [76], [1], [2], [9]. In all the above cases involving 1D and 2D studies, the underlying circuit was always assumed to be regular and structured. Given the success of these prior applications, Chapter 2 extends the mathematical framework to cover a random network. This provides a unifying approach in the study of nonlinear oscillator networks, encompassing both structured and unstructured networks. In Section 2.3, we develop two different algorithms for solving the forward problem. The first approach uses a perturbative expansion which offers flexibility in computing higher order solutions as desired. This method shows that the solution only consists of higher harmonics leading to an alternate algorithm to obtain the solution using a fixed-point method. Using extensive numerical results in Section 2.5.1, we verify the accuracy of the two new algorithms by contrasting them with traditional time-stepping methods. Since the two algorithms solve for the steady-state solution directly without resorting to any computation in the time-domain, they are orders of magnitude faster than explicit time-stepping methods as demonstrated.

In prior work involving 2D RLC networks, it has been shown that the nonlinear effects of the circuit on a structured lattice could significantly boost small-amplitude inputs [9]. A second major contribution of Chapter 2 is to answer the analogous question with regards to random networks. While previous work on amplitude boosting relied on a geometric approach we use a completely approach: we enhance the given network's nonlinear behavior by altering the eigenvalues of the graph Laplacian, i.e., the resonant frequencies of the linearized system. These results demonstrate a relationship between the network's structure, encoded in the graph Laplacian, and its function, which in this case is defined by the presence of nonlinear effects in the frequency response. In order to achieve these results, we have developed a Newton-type method that solves for the network inductances such that the graph Laplacian achieves a desired set of eigenvalues; this method enables one to move the eigenvalues while keeping the network topology fixed. Results in Section 2.5.2 show detailed results across three different random graph models by just altering the graph Laplacian's first two eigenvalues. By altering the inductance values and retaining the underlying graph topology, we are able to improve the network's ability to (i) transfer energy to higher harmonics and (ii) generate large-amplitude signals.

In Chapter 3, we focus on nonlinear Maxwell's equations. We develop a new numerical

method for the planar Maxwell's equations for the (H_1,H_2,E) polarized modes in nonlinear inhomogeneous media subjected to time-harmonic forcing. Maxwell's equations form the basis of the vast field of electromagnetic theory. In this chapter, we provide a numerical method for the solution of the nonlinear form of these equations. The nonlinearity arises due to the dependence of the permittivity of the material on the applied electric field. By far, the most widely used approach to numerically solving Maxwell's equations is the finite difference time domain approach using the Yee scheme developed by Kane Yee [87]. Though the algorithm was introduced in the 1960's it only gained recognition when it was shown to correspond to the finite difference method by Taflove [80]. Due to their time domain nature they are very good in studying transient solutions in models. Since the Yee scheme is only conditionally stable, numerical dispersion is a major concern. This leads to the use of advanced time stepping solvers [70]. Finite difference time domain methods use a grid discretization which is known to introduce further complications when trying to model curved boundaries or arbitrary material inhomogeneity. Finite element and Finite volume methods get past this problem due to their flexibility in handling unstructured meshes.

The application which is the focus of Chapter 3 involves obtaining solutions to the steady-state response of the model under harmonic forcing. Since we do not require the transient part of the solution, we study the problem in the Fourier domain and obtain the steady-state solution directly. We begin with a finite volume discretization of the domain. By converting the problem to a Fourier domain we are able to forego time stepping. The use of finite volume allows us to model complex inhomogeneity in the material. A major contribution of the work is to reduce the finite volume Fourier domain method to a nonlinear RLC circuit problem, exactly the same as was studied in Chapter 2. By providing analogues to the permittivity of the material with capacitance and permeability of the material with inductance, the connection become obvious. Once this relationship is obtained, convergence results from Chapter 2 become an integral part in showing the validity of the method. By ensuring that Kirchoff's laws on the circuit hold, we can verify energy conservation easily thereby ensuring physically correct solutions. While the algorithms developed in Chapter 1 apply directly without any modifications, solving partial differential equations usually require us to scale problems to large sizes. By providing a working implementation in PETSc we demonstrate the scalable power of our method. A major benefit of our approach is the reduction of the solution to solving linear systems which allows us to leverage decades worth of large-scale linear solvers.

In Chapter 4, we turn our attention to a problem arising in supervised learning. Our focus is on decision trees, one of the most influential machine learning techniques. Since their introduction in the 1980's, they have been successfully used in tackling both inference and prediction when analyzing data [72], [20]. In a supervised learning problem, we have a data set consisting of predictor variables $X_1, X_2, \ldots, X_n$ along with a response variable Y. The number of samples can be anywhere between a few hundred to millions. The aim of supervised learning is to construct a model $y = \hat{f}(x)$ while ensuring that such a model is close to the unknown true model $y = f(x)$. To make this requirement mathematically precise we usually rewrite it as: find $\hat{f}$ to

$$\text{Minimize} \parallel y - \hat{f}(x) \parallel . \tag{1.1}$$

We now have to decide over which norm we would like to minimize this function. Traditional statistics usually considers the 2-norm, since it is differentiable allowing for the possibility

of closed form solutions. Linear regression is one such example. While linear regression is extremely popular and simple to implement, its simplicity and strong model assumptions usually leads to a variety of problems which make it unsuitable for real world data without regularization. Most important among these are (i) underlying assumptions of normality of the data, (ii) heterogeneous error models, (iii) complications with noisy data and, (iv) non-sparse models. As a result there are a number of alternate modeling frameworks developed to counteract these problems.

Decision trees form the basis of many advanced state-of-the-art machine learning methods today like bagging [18], random forests [19] and various forms of boosting [37] [38]. While minimizing the 2-norm is analytically convenient, recent trends have shifted towards the use of the 1-norm. The change is largely attributed towards the sparse solutions provided by the 1-norm. Replacing the 2-norm minimization at each step with 1-norm minimization has the potential to make the trees far more robust to outliers, and also to provide a direct approach to minimizing the absolute deviation. This observation is not new, and it has appeared in the data mining research community for over 3 decades [20]. However, the research community has largely relied on 2-norm minimization due to its efficient scalable nature. Decision trees minimizing the 2-norm will be called OLS trees and trees which minimize the 1-norm will be denoted LAD trees in this work. Our work provides the first working implementation of LAD trees that have the same algorithmic complexity as that of OLS trees.

Our algorithm generalizes decision tree construction and can construct quantile trees of which LAD trees are a special case. This flexibility allows us to provide non-parametric alternatives to linear quantile regression [53]. Linear quantile regression is an alternative to linear regression that can provide either the median or other quantiles of interest. They have been successfully applied to problems in economics and many other fields. The benefit of using quantile regression is that the models allow us to determine weak relationships among the predictor and response variables. This is very hard to do using regular linear regression without knowing anything about the underlying model. While quantile regression offers a better approach in this case, it is still a linear model. This is a severe restriction, one which we tackle by providing a completely non-parametric model with no assumptions on linearity.

While designing and implementing a scalable algorithm was the first part, our present work will enable us to extend LAD trees to a wide variety of situations including random forests, boosting and quantile regression.

Since all the work included in the thesis has revolved around the design and implementation of algorithms, in the spirit of reproducible research, all codes are available through the author's Github repository at `https://github.com/GarnetVaz`.

Chapter 2

Frequency Response and Gap Tuning for Nonlinear Electrical Oscillator Networks

2.1 Introduction

Networks of nonlinear electrical oscillators have found recent application in several microwave frequency analog devices [61, 63, 58, 59, 44, 45]. The fundamental unit in these networks is a nonlinear oscillator wired as in Figure 2.1; this oscillator consists of one inductor, one voltage-dependent capacitor, one source, and one sink (a resistor). While many nonlinear oscillatory circuits have been studied for their chaotic behavior, the particular oscillator in Figure 2.1 does not exhibit sensitive dependence on initial conditions in the regime of operation that we consider [10]. Instead, assuming the source is of the form $A\cos(\omega t + \phi)$, the oscillator reaches a steady-state consisting of a sum of harmonics with fundamental frequency ω [10].

When networks of these oscillators have been studied, the network topology has either been a one-dimensional linear chain, in which case the circuit is called a nonlinear transmission line [56, 47, 23, 22, 4]–see Figure 2.2, or a two-dimensional rectangular lattice [65, 78, 79, 3, 9]—see Figure 2.3. Even if each individual block in the chain/lattice is weakly nonlinear, the overall circuit can exhibit strongly nonlinear behavior. It is this property that is exploited for microwave device applications, enabling low-frequency, low-power inputs to be transformed into high-frequency, high-power outputs.

The first objective of this work is to develop numerical algorithms to compute the frequency response of a nonlinear electrical network with topology given by an arbitrary connected graph. Here we are motivated by the successful application of computational

Figure 2.1: Schematic of a single nonlinear oscillator.
This oscillator is the basic building block of the networks considered in this chapter. The circuit contains one inductor, one voltage-dependent capacitor, one source, and one resistor.

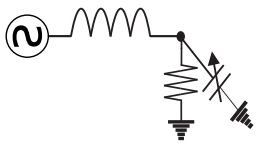


Figure 2.2: An example of a nonlinear transmission line.
A nonlinear transmission line is a nonlinear electrical network on a one-dimensional linear graph.

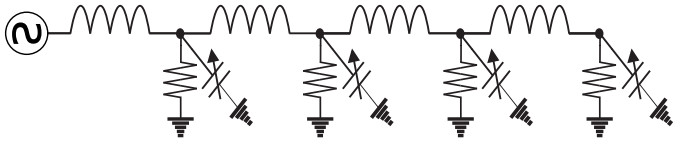
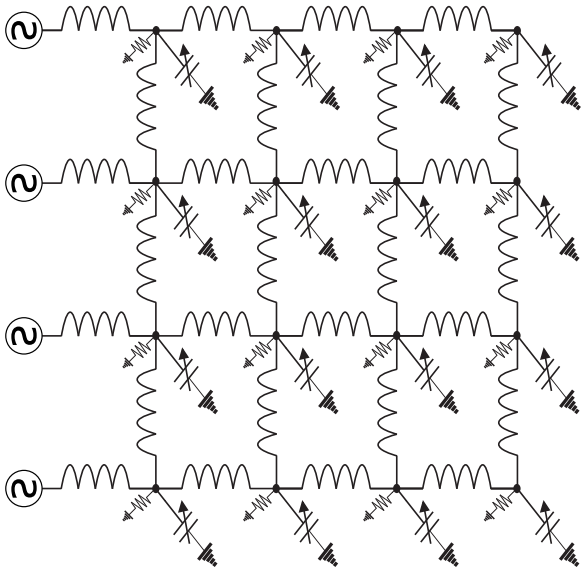


Figure 2.3: An example of a nonlinear lattice.
A nonlinear lattice is a nonlinear electrical network on a two-dimensional rectangular grid graph.



techniques in the design of the high-frequency analog devices referenced above. As we show, to compute steady-state solutions with comparable accuracy, both the perturbative and iterative algorithms developed in this chapter require orders of magnitude less computational time than standard numerical integration. While the perturbative algorithm generalizes derivations given in prior work [9, 10], the iterative algorithm has not been previously applied to nonlinear electrical networks. Both new algorithms show exponential convergence in the number of iterations, and for a test problem on a network with $N = 400$ nodes, less than 20 iterations are required to achieve machine precision errors.

The second objective of this work is to relate structural properties of the network to the dynamics of the nonlinear oscillator system. The derivation of the perturbative algorithm indicates that nonlinearity in the electrical network manifests itself through energy transfer from the fundamental forcing frequency to higher harmonics. This helps us understand why properties such as amplitude boosting [9, 10] and frequency upconversion [61], observed in nonlinear electrical networks with regular lattice topologies, can be expected when the topology is that of a random, disordered network. Additionally, we observe that an inductance-weighted graph Laplacian matrix features prominently in both algorithms for computing the steady-state solution. This graph Laplacian matrix encodes the structure of the network, and its eigenvalues are the squares of the resonant frequencies for the undamped, linear version of the circuit. Driving the damped, linearized circuit at one of these resonances results in large amplitude outputs. It is reasonable to hypothesize that the locations of these resonances

Table 2.1: Portion of results for graphs with $N = 175$ nodes.
Simulation results for three different types of random graphs with $N = 175$ nodes, averaged over 100 runs. "Pre" and "Post" stand for before and after circuit inductances are changed to reduce the gap between the graph Laplacian's first two eigenvalues. Note that pre *and* post *circuits have the same graph topology and differ only in their inductances.*

	% of energy in higher harmonics		Maximum magnitude voltage	
	Pre	Post	Pre	Post
Barabási-Albert (BA)	0.410	4.063	0.01548	0.31684
Watts-Strogatz (WS)	1.006	8.701	0.03399	0.51157
Erdös-Rényi (ER)	0.033	7.534	0.002956	0.78902

play a large role in the dynamics of the nonlinear network.

This motivates the following question: how do the eigenvalues of the graph Laplacian influence the nonlinear network's properties of frequency upconversion and amplitude boosting? While it is possible to alter the spectrum of the graph Laplacian by changing the node-edge relationships in the graph, we can also change its spectrum by keeping the topology fixed and manipulating the network's inductances. We formulate and solve the inverse problem of finding the inductances such that the graph Laplacian achieves a prescribed spectrum. The solution proceeds via a Newton-type algorithm that takes the desired spectrum as input and iteratively alters the inductances until a convergence criterion is met.

For three types of random graphs, we find that the Newton-type method effectively finds circuit inductances that close the gap between the first two eigenvalues of the graph Laplacian. We conduct a series of numerical experiments to examine the effect of closing this eigenvalue gap on a given circuit's ability (i) to transfer energy from the fundamental driving frequency to higher harmonics, and (ii) to generate high-amplitude output signals. The results indicate that the two metrics (i-ii) can be improved dramatically by closing the gap between the graph Laplacian's first two eigenvalues. Table 2.1 shows results we obtained for graphs with $N = 175$ nodes. Though this a small portion of the results we describe later, this table already illustrates the effect of gap tuning on network performance. Note that each *pre* and *post* circuit have the same graph topology, differing only in their edge inductances.

2.1.1 Connections to Other Systems

We can make several connections between the problem studied in this chapter and other problems of interest:

- **Random elastic networks.** Using a mechanical analogy between inductors/capacitors and masses/springs, the nonlinear electronic network can be transformed into a mathematically equivalent network of masses and anharmonic springs [11, Appendix I]. Such random elastic networks have been of recent interest as models of amorphous solids [85, 64, 82]. For such networks, quartic spring potential energies have been considered [29]. Nonlinear random elastic networks have also been used to model molecular machines; in this context, tuning the gap between the first two eigenvalues of the linearized system enables the construction of networks with properties similar to those of real proteins [83]. Despite this activity, algorithms for computing and manipulating the frequency response of nonlinear elastic networks have not been developed. Our

work addresses this issue directly.

These random elastic networks can be thought of as either (a) spatially inhomogeneous Fermi-Pasta-Ulam [35] systems on connected graphs, or (b) generalizations of Dyson's disordered mass-spring chain [31] to the case of anharmonic spring potential energies combined with more general network topologies.

- **Nonlinear electromagnetic media.** The circuit we analyze, for particular values of the circuit parameters, arises naturally as a finite volume discretization of Maxwell's equations for TE/TM modes in a nonlinear medium [13, 14]. The arbitrary connected graph topology of the circuit corresponds to a finite volume discretization on an arbitrary unstructured mesh. The algorithms developed here can be used to compute and optimize the frequency response of nonlinear electromagnetic media.

- **Coupled phase oscillator networks.** There has been intense interest in nonlinear phase oscillator networks, primarily due to the ability of such networks to model biophysical systems featuring synchronization. Though synchronization is not of primary interest in our system, we may still draw parallels. The effect of network topology on the properties of coupled phase oscillators has been studied extensively [15, 54, 67, 5]. Manipulating eigenvalues of the Laplacian matrix enables one to enhance a network's synchronization properties [49]. More recently, several authors have developed algorithms for optimizing the synchronization of phase oscillator networks [30, 17, 16, 28, 52, 86]. The questions considered in this subset of the coupled phase oscillator literature are related to the issues addressed in the present work.

2.2 Problem Formulation

Let $H(N, e)$ be a connected, simple graph with N nodes and e edges. Each edge corresponds to an inductor that physically connects two nodes. Each node corresponds to a capacitor and resistor, wired in parallel, that physically connect the node to a common ground. Let $f \leq N$ be the number of nodes that are driven by prescribed sources. Since the voltage at the prescribed source is known, we do not model it using a node. The connection between the source and the node that it drives is modeled by a half-edge, also known as a dangling edge since one end is connected to a driven node and the other end does not connect to any node. We let $H(N, e, f)$ denote the graph together with the f half-edges.

The capacitance and conductance (inverse resistance) at node j are C_j and G_j, respectively. We let $V_j(t)$ denote the voltage from node j to ground at time t. The inductance of edge k is L_k, while the current through edge k at time t is $I_k(t)$. The exact dimensions for each component of H, along with the currents and voltages, are tabulated in Table 2.2.

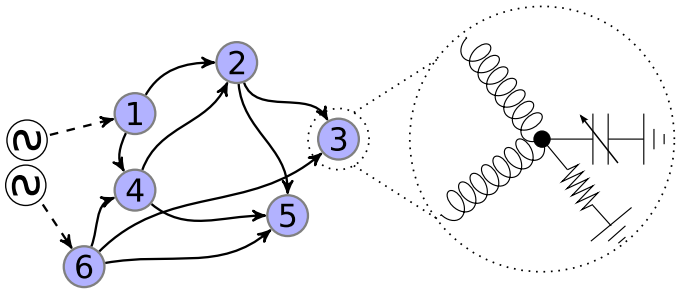
In order to write down Kirchhoff's laws, we must choose an orientation of the edges. The orientation of an edge records the direction of positive current flow through the edge. If we solve the problem with opposite orientations, the only difference we will notice is that the currents will pick up a factor of -1. Consequently, the orientation we choose does not affect the solution in any material way. In what follows, we will choose a random orientation of the edges.

In Figure 2.4, we show an example graph corresponding to $H(6, 9, 2)$. The edges are oriented randomly. The inputs are connected at nodes 1 and 6 through two inductors. These

Table 2.2: Summary of the notation used in this chapter

Notation	Significance	Size
C	Capacitance at node	$N \times 1$
L	Inductance of edge	$(e+f) \times 1$
G	Conductance at node	$N \times 1$
V	Voltage at node	$N \times 1$
I	Current through edge	$(e+f) \times 1$
W	Input forcing	$N \times 1$
B	Signed incidence matrix	$N \times (e+f)$

Figure 2.4: An example of a nonlinear electrical network.
In the graph on the left, the numbered circles are nodes, the solid arrows are edges, and the dashed arrows are half-edges. Orientation of the arrows indicates the direction of positive current flow. Each node corresponds to a voltage-dependent capacitor to ground, wired in parallel with a resistor to ground, as depicted in the zoomed-in schematic for node 3*. Each edge corresponds to an inductor. Each half-edge connects one prescribed voltage source to one given node. In the present work, all methods that are developed are valid for connected graphs with at least one half-edge. Note that the circuits in Figures 2.1 and 2.3 can all be represented using this graph formalism.*



input nodes correspond to half-edges in H. On the right we view node 3 in detail. Each of the two edges connected to this node correspond to an inductor. A capacitor with capacitance C_3 and a resistor with conductance G_3 connect node 3 to ground.

To arrange Kirchhoff's laws compactly, we use the $N \times (e+f)$ incidence matrix of $H(N, e, f)$, denoted by B. Let j be an edge connecting the nodes i' and i. If j is oriented such that positive current starts at node i' and flows to node i, we write $j = (i', i)$. If j is a half-edge attached to node i, we write $j = (\emptyset, i)$, leaving the first slot empty and orienting the half-edge so it always points toward the forced node. The entries of the incidence matrix B are

$$B_{i,j} = \begin{cases} 1 & \text{if } j = (i', i) \text{ for some node } i' \text{ or } i' = \emptyset \\ -1 & \text{if } j = (i, i') \text{ for some node } i' \\ 0 & \text{otherwise.} \end{cases}$$

This chapter will only consider single frequency time-harmonic forcing of the form $ae^{i\omega t} + \overline{a}e^{-i\omega t}$ where $a \in \mathbb{C}^N$. Let P be an $N \times (e+f)$ matrix with entries $P_{i,j} = 1$ if node i is connected to an input edge j and 0 otherwise. Using the projection matrix P we define the forcing

$$W(t) = P^{\mathrm{T}}(ae^{i\omega t} + \overline{a}e^{-i\omega t}). \tag{2.1}$$

Using the notation summarized in Table 2.2, Kirchhoff's laws for the nonlinear circuit on the graph $H(N, e, f)$ can now be written compactly as

$$L\frac{\mathrm{d}I}{\mathrm{d}t} = -B^{\mathrm{T}}V + W \tag{2.2}$$

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = BI - GV. \tag{2.3}$$

Here $L(\mathrm{d}I/\mathrm{d}t)$, $C(\mathrm{d}V/\mathrm{d}t)$, and GV are examples of component-wise multiplication of vectors. For $a, b \in \mathbb{C}^m$, we define $c = ab \in \mathbb{C}^m$ by $c_j = a_j b_j$ for $1 \leq j \leq m$. Note that in this case, we can also write $c = \mathrm{diag}(a)b$. Here $\mathrm{diag}(a)$ is the $m \times m$ matrix that contains the vector a along its diagonal ($[\mathrm{diag}(a)]_{ii} = a_i$) and is zero elsewhere.

The formulation (2.2 and 2.3) generalizes previous formulations [13, 12] where the capacitors were constant and the systems considered were linear.

By differentiating (2.3) and inserting it into (2.2), we obtain a second-order system for the voltages:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[C\frac{\mathrm{d}V}{\mathrm{d}t}\right] + G\frac{\mathrm{d}V}{\mathrm{d}t} + \Delta V = V_{\mathrm{in}}. \tag{2.4}$$

Here

$$V_{\mathrm{in}}(t) = B[\mathrm{diag}(L)]^{-1}W(t) \tag{2.5}$$

$$\Delta = B[\mathrm{diag}(L)]^{-1}B^{\mathrm{T}}. \tag{2.6}$$

Note that Δ is the weighted Laplacian for the network with edge weights given by reciprocal inductance.

We assume that the capacitance at node i depends on the voltage at node i:

$$C_i(V_i) = C^0(1 - \varepsilon V_i), \tag{2.7}$$

where $C^0 \in \mathbb{R}$ is a constant. Note that this choice of capacitance function means that (2.4) features a quadratic nonlinearity.

We can then formulate the *frequency response problem* for the nonlinear electrical network: given the amplitude vector a and frequency ω for the forcing function (2.1), determine the steady-state solution $V(t)$ of (2.4).

2.3 Algorithms for the forward problem

We now discuss two algorithms to solve the frequency response problem defined in the preceding Section. The two approaches are independent of each other. Correctness of the algorithms is proved numerically in detail in Section 2.5.1.

2.3.1 Perturbative Algorithm

The first algorithm is based on a perturbative expansion in powers of ε. We use dots to denote differentiation with respect to time. Substituting the capacitance function (2.7) in (2.4) and rearranging, we obtain

$$C^0\ddot{V} + G\dot{V} + \Delta V = V_{\mathrm{in}} + \frac{\varepsilon C^0}{2}\frac{\mathrm{d}^2}{\mathrm{d}t^2}\left[V^2\right]. \tag{2.8}$$

We expand

$$V(t) = V_0(t) + \varepsilon V_1(t) + \varepsilon^2 V_2(t) + \cdots \tag{2.9}$$

Inserting (2.9) into (2.8), we obtain equations for each order of ε. At zeroth order, we obtain

$$C^0 \ddot{V}_0 + G\dot{V}_0 + \Delta V_0 = V_{\text{in}}. \tag{2.10}$$

For $k \geq 1$, the k-th order equation is

$$C^0 \ddot{V}_k + G\dot{V}_k + \Delta V_k = C^0 \frac{\mathrm{d}}{\mathrm{d}t}\left[\sum_{m=0}^{k-1} V_{k-1-m}\dot{V}_m\right]. \tag{2.11}$$

We now solve (2.10-2.11). Let us introduce the Fourier transform in time,

$$\widehat{\psi}(\alpha) = \int_{-\infty}^{\infty} e^{-i\alpha t}\psi(t)\,\mathrm{d}t, \tag{2.12}$$

with inverse Fourier transform

$$\psi(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\alpha t}\widehat{\psi}(\alpha)\,\mathrm{d}\alpha. \tag{2.13}$$

Note that with these definitions,

$$\widehat{\dot{\psi}}(\alpha) = i\alpha\widehat{\psi}(\alpha).$$

This implies that the Fourier transforms of both sides of (2.10-2.11) can be summarized by writing

$$\mathcal{L}(\alpha)\widehat{V_k}(\alpha) = \begin{cases} \widehat{V_{\text{in}}} & k = 0 \\ -(\alpha^2/2)C^0\widehat{\phi_k}(\alpha) & k \geq 1, \end{cases} \tag{2.14}$$

where $\mathcal{L}(\alpha)$ is the linear operator

$$\mathcal{L}(\alpha) = -\alpha^2\,\mathrm{diag}(C^0) + i\alpha\,\mathrm{diag}(G) + \Delta, \tag{2.15}$$

and

$$\phi_k(t) = \sum_{m=0}^{k-1} V_{k-1-m}V_m. \tag{2.16}$$

By (2.5) and (2.1), we see that

$$\widehat{V_{\text{in}}} = 2\pi B[\mathrm{diag}(L)]^{-1}P^{\mathrm{T}}\left(a\delta(\alpha-\omega) + \overline{a}\delta(\alpha+\omega)\right), \tag{2.17}$$

where δ is the Dirac delta. Then the $k = 0$ branch of (2.14) yields

$$\widehat{V}_0(\alpha) = 2\pi\left[a_{0,1}(\alpha)\delta(\alpha-\omega) + a_{0,-1}(\alpha)\delta(\alpha+\omega)\right] \tag{2.18a}$$

$$a_{0,1}(\alpha) = [\mathcal{L}(\alpha)]^{-1}B[\mathrm{diag}(L)]^{-1}P^{\mathrm{T}}a \tag{2.18b}$$

$$a_{0,-1}(\alpha) = [\mathcal{L}(\alpha)]^{-1}B[\mathrm{diag}(L)]^{-1}P^{\mathrm{T}}\overline{a}. \tag{2.18c}$$

Using the inverse Fourier transform, we have

$$\begin{aligned} V_0(t) &= a_{0,1}(\omega)e^{i\omega t} + a_{0,-1}(-\omega)e^{-i\omega t} \\ &= a_{0,1}(\omega)e^{i\omega t} + \text{c.c.}, \end{aligned}$$

where "c.c." stands for the complex conjugate of the previous terms. Here we have used the property that $\mathcal{L}(-\alpha) = \overline{\mathcal{L}(\alpha)}$.

Once we have computed $V_0(t)$, we can insert it into (2.16) to compute $\phi_1(t)$. We will find that $\phi_1(t)$ is a linear combination of $e^{-2i\omega t}$, $e^{0i\omega t}$, and $e^{2i\omega t}$. Using this fact in the $k = 1$ branch of (2.14), we can solve for $\widehat{V}_1(\alpha)$ and then apply the inverse Fourier transform to compute $V_1(t)$. We will find that $V_1(t)$ contains the same modes as $\phi_1(t)$.

The above shows how we get the perturbative solution algorithm started. Now let us move to the more general case where we seek $V_k(t)$ for any $k \geq 1$. Assume that we have already computed $V_j(t)$ for $0 \leq j \leq k-1$, and that the solution takes the following form:

$$V_{2m}(t) = \sum_{\ell=0}^{m} a_{2m,2\ell+1} e^{(2\ell+1)i\omega t} + \text{c.c.} \tag{2.19a}$$

$$V_{2m+1}(t) = \sum_{\ell=0}^{m+1} a_{2m+1,2\ell} e^{(2\ell)i\omega t} + \text{c.c.} \tag{2.19b}$$

In words, V_{2m} contains odd modes $1, 3, \ldots, 2m+1$, and V_{2m+1} contains even modes $0, 2, \ldots, 2m+2$. Here we assume that $0 \leq 2m < 2m+1 \leq k-1$, and that the $a_{i,j} \in \mathbb{C}^N$ coefficients are known.

In order to solve for $V_k(t)$, we use the $k \geq 1$ branch of (2.14), which requires us to compute (2.16). We have two cases, when k is odd and when k is even. In both cases, it is a simple (if tedious) algebraic exercise to show that $\phi_k(t)$ yields:

- when k is odd, a sum of even Fourier modes $-(k+1), \ldots, -2, 0, 2, \ldots, (k+1)$, and
- when k is even, a sum of odd Fourier modes $-(k+1), \ldots, -3, -1, 1, 3, \ldots, (k+1)$.

In both cases, it is clear that using (2.14) to solve for $\widehat{V}_k(\alpha)$ results in a sum of Dirac delta's. Applying the inverse Fourier transform yields $V_k(t)$, which will be a sum of Fourier modes. One can check that $V_k(t)$ will have precisely the form (2.19a) or (2.19b) depending on whether k is even or odd, respectively.

The algorithm is then clear. Starting with (2.19), we apply component-wise multiplication to particular pairs of the $a_{i,j}$ vectors in order to compute the coefficients of the Fourier modes of $\phi_k(t)$ defined in (2.16). Next, we combine the step of solving for $\widehat{V}_k(\alpha)$ using the $k \geq 1$ branch of (2.14) together with the step of computing the inverse Fourier transform. After component-wise multiplication of the Fourier coefficients of ϕ_k by $-(\alpha^2/2)C^0$, we multiply each coefficient on the left by $[\mathcal{L}(\alpha)]^{-1}$ with α set to match the frequency of the corresponding Fourier mode. Dividing these coefficients by 2π yields the Fourier coefficients of $V_k(t)$, as desired.

While we have presented the algorithm in an intuitive way, the statements made above can be made rigorous, and a convergence theory for the perturbative expansion (2.9) can be established. This is the subject of future work.

There are a few brief remarks to make about the algorithm presented above:

- As described above, we consider only those networks that contain resistance at all nodes, i.e., $G_i > 0$ for all nodes i. Such an assumption is not only physically realistic; it also guarantees that for all $\alpha \in \mathbb{R}$, the matrix $\mathcal{L}(\alpha)$ is invertible. The invertibility for the $\alpha = 0$ case is a consequence of Corollary 1.

- In this work, we are interested in the weakly nonlinear regime where $\varepsilon\|a\|$ is sufficiently small such that the perturbative method converges as shown in prior work [10]. As the nondimensional constant $\varepsilon\|a\|$ is increased beyond the breakdown point of the perturbative method, direct numerical solutions of the equations of motion reveal subharmonic oscillations, and eventually, chaotic oscillations.
- The fact that the Fourier transform yields the steady-state solution has been explained in earlier work [10]. By fixing an arbitrary set of initial conditions and using the Laplace transform to derive the full solution, one can show that after the decay of transients, the part of the solution that remains is precisely what we obtain using the Fourier transform. This also explains why it was not necessary for us to specify initial conditions for (2.4) —the initial conditions only influence the decaying transient part of the solution.

2.3.2 Iterative Algorithm

The perturbative method developed above shows us that the solution $V(t)$ is a sum of harmonics where the fundamental frequency is given by the input frequency ω. This implies that the steady-state solution $V(t)$ is periodic with period $T = 2\pi/\omega$ since all other Fourier coefficients will be zero This observation leads us to ask whether it is possible to directly solve for the Fourier coefficients of $V(t)$ without first expanding in powers of ε. In this section, we develop a fixed point iteration scheme that accomplishes this task.

First, we integrate both sides of (2.8) from $t = 0$ to $t = T$ to derive

$$\Delta \int_0^T V(t)\,\mathrm{d}t = 0. \tag{2.20}$$

We show below that as long as the network contains at least one half-edge, Δ is invertible. Hence (2.20) implies

$$\int_0^T V(t)\,\mathrm{d}t = 0. \tag{2.21}$$

This means there is no zero/DC mode present in $V(t)$, motivating the Fourier series expansion

$$V(t) = \sum_{k=1}^{\infty} \alpha_k e^{ik\omega t} + \text{c.c.} \tag{2.22}$$

In order to compute the solution, we truncate at $k = M$, leading to an approximation $\underline{V} \approx V$:

$$\underline{V}(t) = \sum_{k=1}^{M} \alpha_k e^{ik\omega t} + \text{c.c.} \tag{2.23}$$

Using orthogonality we derive

$$\alpha_k = \frac{1}{T}\int_0^T e^{-ik\omega t}\underline{V}(t)\,\mathrm{d}t.$$

Using the T-periodicity of V and integration by parts, we have

$$\frac{1}{T}\int_0^T e^{-ik\omega t}\dot{\underline{V}}(t)\,\mathrm{d}t = ik\omega\alpha_k.$$

To simplify notation, we combine (2.1) and (2.5) and write $V_{\text{in}} = we^{i\omega t} + \text{c.c.}$ where

$$w = B[\text{diag}(L)]^{-1}P^{\text{T}}a. \tag{2.24}$$

Now let $\delta_{m,n}$ denote the Kronecker delta function which equals 1 if $m = n$, and 0 otherwise. We multiply both sides of (2.8) by $e^{-ik\omega t}$, integrate from $t = 0$ to $t = T$, and finally divide by T to obtain

$$\mathcal{L}(k\omega)\alpha_k = w\delta_{k,1} + \overline{w}\delta_{k,-1} + \frac{\varepsilon C^0(ik\omega)^2}{2}\widetilde{\underline{V}^2}_k, \tag{2.25}$$

where $\mathcal{L}$ was defined in (2.15) and

$$\widetilde{\underline{V}^2}_k = \frac{1}{T}\int_0^T e^{-ik\omega t}\underline{V}^2\,\mathrm{d}t. \tag{2.26}$$

Because the form of the nonlinearity is simple, we can insert (2.23) into (2.26) and derive

$$\widetilde{\underline{V}^2}_k = \sum_{\ell=-M}^{M}\alpha_\ell\alpha_{k-\ell}, \tag{2.27}$$

with the understanding that $\alpha_0 = 0$, $\alpha_{-j} = \overline{\alpha_j}$ for $j > 0$, and $\alpha_j = 0$ for $|j| > M$. We insert (2.27) into (2.25) and obtain

$$\mathcal{L}(k\omega)\alpha_k = w\delta_{k,1} + \overline{w}\delta_{k,-1} - \frac{\varepsilon C^0 k^2\omega^2}{2}\sum_{\ell=-M}^{M}\alpha_\ell\alpha_{k-\ell}.$$

We convert this into an iterative scheme in a natural way. Let $\alpha_k^{(j)}$ denote the j-th iterate, and assume that α_k terms appearing on the left-hand side are at iteration $j+1$, while those appearing on the right-hand side are at iteration j. Let $\mathbf{A}^{(j)}$ denote the $N \times M$ complex matrix whose k-th column is $\alpha_k^{(j)}$. Then the scheme is

$$\mathbf{A}^{(j+1)} = F^M(\mathbf{A}^{(j)}) \tag{2.28}$$

where the k-th column of the matrix $F^M(\mathbf{A}^{(j)})$ is

$$F_k^M(\mathbf{A}^{(j)}) = [\mathcal{L}(k\omega)]^{-1}\left(w\delta_{k,1} - \frac{\varepsilon C^0 k^2\omega^2}{2}\sum_{\ell=-M}^{M}\alpha_\ell^{(j)}\alpha_{k-\ell}^{(j)}\right). \tag{2.29}$$

Here we assume $1 \le k \le M$, which is also why we have deleted the second Kronecker delta from the right-hand side.

Starting at $\mathbf{A}^{(0)}$, we iterate forward using (2.28), stopping the computation when $\|F(\mathbf{A}^{(j)}) - \mathbf{A}^{(j)}\|$ is below a specified tolerance. Note that in our implementation of F, we precompute and store the LU factorization for the M matrices $\{\mathcal{L}(k\omega)\}_{k=1}^M$, since this part of the computation of the right-hand side of (2.29) does not change from one iteration to the next.

Again, we have derived the algorithm but have not proven its convergence. Instead, we will demonstrate empirically that the algorithm converges using several numerical tests.

2.4 Inverse Problem

In this section, we consider the inverse problem of finding a set of inductances such that Δ, the Laplacian defined by (2.6), achieves a desired spectrum. Before describing an algorithm to solve this inverse problem, we review basic spectral properties of Δ.

Lemma 1 *Assume all inductances are positive. Then Δ as defined in (2.6) is symmetric positive semidefinite, and all its eigenvalues must be nonnegative.*

Proof. Let $[\mathrm{diag}(L)]^{-1/2}$ be the diagonal matrix whose (i,i)-th element on the diagonal is $L_i^{-1/2}$, for $1 \le i \le e$. Since $L_i > 0$, the matrix $\Delta^{1/2} = B[\mathrm{diag}(L)]^{-1/2}$ is real. Then $\Delta = \Delta^{1/2}\left(\Delta^{1/2}\right)^{\mathrm{T}}$, and for any $v \in \mathbb{R}^N$, we have $v^{\mathrm{T}}\Delta v = \left[\left(\Delta^{1/2}\right)^{\mathrm{T}} v\right]^{\mathrm{T}}\left(\Delta^{1/2}\right)^{\mathrm{T}} v \ge 0$. □

Let $\{\lambda_i\}_{i=1}^N$ denote the spectrum of Δ, with eigenvalues arranged in nondecreasing order: $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_N$. The above argument shows that $\lambda_1 \ge 0$. We can be more precise about this: if there are no half-edges, then $\lambda_1 = 0$, while the presence of at least one half-edge causes $\lambda_1 > 0$.

Lemma 2 *Let $\mathcal{H} = H(N,e)$ be a connected graph with N nodes, e edges, and zero half-edges. For a particular orientation of the graph, let $\mathcal{B}$ denote the signed incidence matrix. Then* $\mathrm{rank}(\mathcal{B}) = N-1$.

Proof. Let r be any integer from 1 to $N-1$. Consider any subset S of r vertices of the graph. Take the sum of the rows of the incidence matrix corresponding to the elements of S. This sum cannot be zero; if it were, there would be no path connecting S to the complement S^c and the resulting graph would not be connected. Hence the sum of these rows must contain a nonzero entry. As the same would be true if we considered linear combinations of the rows corresponding to S, we conclude that any subset of at most $N-1$ rows must be linearly independent. At the same time, if we take the sum of all the rows we get a zero row, because each column contains precisely one $+1$ and one -1. □

Lemma 3 *Let $\mathcal{H}' = H(N,e,f)$ be a connected graph with N nodes, e edges, and $f > 0$ half-edges. For a particular orientation of the graph, let $\mathcal{B}'$ denote the signed incidence matrix. Then* $\mathrm{rank}(\mathcal{B}') = N$.

Proof. Without loss of generality, we can assume that the $N \times (e+f)$ incidence matrix $\mathcal{B}'$ is organized such that the first e columns correspond to full edges, while columns $e+1, \ldots, e+f$ correspond to half-edges. Now choose any j such that $1 \le j \le f$, and examine column $e+j$ of $\mathcal{B}$. Let k be the unique row in which this column contains ± 1. Since row k of $\mathcal{B}$ is the only row that contains an entry in column $e+j$, row k is linearly independent from the other $N-1$ rows of $\mathcal{B}$. By Lemma 2, the submatrix of $\mathcal{B}$ consisting of all rows other than row k has rank $N-1$. Including row k increases the rank by one, yielding a rank N matrix. □

Lemma 4 *For a connected graph $\mathcal{H}' = H(N,e,f)$ with N nodes, e edges, and $f > 0$ half-edges, let Δ be the edge-weighted graph Laplacian defined in (2.6). Assume all inductances are positive. Then* $\mathrm{rank}(\Delta) = N$.

Proof. The $(e+f)\times(e+f)$ diagonal matrix $[\mathrm{diag}(L)]^{-1}$ has rank $e+f > N$. Let B be the signed incidence matrix for a particular orientation of $\mathcal{H}'$. By Lemma 3, $\mathrm{rank}(B) = N$, implying $\mathrm{rank}(B[\mathrm{diag}(L)]^{-1/2}) = N$, which implies $\mathrm{rank}(\Delta) = \mathrm{rank}\left[(B[\mathrm{diag}(L)]^{-1/2})(B[\mathrm{diag}(L)]^{-1/2})^{\mathrm{T}}\right] = N$. □

Corollary 1 *Let $\mathcal{H}'$, Δ and the inductances satisfy the hypotheses of Lemma 4. Then Δ is symmetric positive definite and all eigenvalues of Δ are positive, i.e., $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$.*

Proof. Combine Lemmas 1 and 4. □

We now describe an algorithm that quantifies how we must change the vector of inductances L in order to make Δ have a desired set of eigenvalues. In what follows, we assume we work with a system that satisfies the hypotheses of Corollary 1.

For $n \leq N$, let $\lambda^* = (\lambda_1^*, \ldots, \lambda_n^*)^{\mathrm{T}}$ denote a vector of desired eigenvalues satisfying

$$0 < \lambda_1^* \leq \lambda_2^* \leq \cdots \leq \lambda_n^*.$$

We treat the vector of inductances L as a variable, and let $\lambda(L)$ denote the sorted vector of eigenvalues of the graph Laplacian Δ defined in (2.6). Since Δ is symmetric, it possesses an orthonormal basis of eigenvectors. We assume that $v_j(L)$ is the normalized eigenvector corresponding to $\lambda_j(L)$.

Now let $\mathcal{F} : \mathbb{R}^{e+f} \to \mathbb{R}^n$ be the function

$$\mathcal{F}(L) = \begin{bmatrix} \lambda_1(L) \\ \lambda_2(L) \\ \vdots \\ \lambda_n(L) \end{bmatrix} - \begin{bmatrix} \lambda_1^* \\ \lambda_2^* \\ \vdots \\ \lambda_n^* \end{bmatrix}. \tag{2.30}$$

We now apply a version of Newton's method to find a zero of this function. To use Newton's method we will need to compute the Jacobian $J(\mathcal{F}(L))$. Let primes denote differentiation with respect to L_k. To form the Jacobian we need to find

$$\lambda_j' := \frac{\partial}{\partial L_k}\lambda_j(L).$$

We proceed by implicit differentiation, starting from the eigenvector equation

$$\Delta v_j(L) = B[\mathrm{diag}(L)]^{-1}B^{\mathrm{T}}v_j(L) = \lambda_j(L)v_j(L).$$

Differentiating both sides with respect to L_k, and omitting the dependence on L, we obtain

$$B([\mathrm{diag}(L)]^{-1})'B^{\mathrm{T}}v_j + B[\mathrm{diag}(L)]^{-1}B^{\mathrm{T}}v_j' = \lambda_j'v_j + \lambda_j v_j'. \tag{2.31}$$

Since Δ is symmetric,

$$v_j^{\mathrm{T}}B[\mathrm{diag}(L)]^{-1}B^{\mathrm{T}} = (B[\mathrm{diag}(L)]^{-1}B^{\mathrm{T}}v_j)^{\mathrm{T}} = (\lambda_j v_j)^{\mathrm{T}} = v_j^{\mathrm{T}}\lambda_j. \tag{2.32}$$

Multiplying (2.31) on the left by v_j^{T} and using (2.32) together with $(v_j)^{\mathrm{T}}v_j = 1$, we obtain

$$v_j^{\mathrm{T}}B([\mathrm{diag}(L)]^{-1})'B^{\mathrm{T}}v_j = \lambda_j', \tag{2.33}$$

where

$$([\mathrm{diag}(L)]^{-1})' := \frac{\partial}{\partial L_k}([\mathrm{diag}(L)]^{-1}) = \left[0, 0, \cdots, -\frac{1}{L_k^2}, \cdots, 0\right]^{\mathrm{T}}.$$

Using λ_j' we can compute the entries of the Jacobian matrix and the corresponding Newton's method with pseudoinverse becomes

$$L^{(i+1)} = L^{(i)} - [J(\mathcal{F}(L))]^{\dagger}\,\mathcal{F}(L) \tag{2.34}$$

where † denotes the Moore-Penrose pseudoinverse.

Using (2.34) as shown might produce inductances such that the ratio of the largest to smallest inductance is too large. In order to avoid these large variations, we constrain $\epsilon^{-1} \leq L_i \leq \epsilon$. We incorporate these constraints using an active set approach, replacing $\mathcal{F}$ by the function $\mathcal{G}^i : \mathbb{R}^{e+f+m} \to \mathbb{R}^{n+m}$, where i denotes the iteration number and m denotes the number of constraints violated by $L^{(i)}$. Let $\mathcal{Q}^{\pm}$ denote the functions

$$\mathcal{Q}^{+}(x) = \begin{cases} \frac{1}{2}x^2 & x > 0 \\ 0 & x \leq 0, \end{cases} \qquad \text{and} \qquad \mathcal{Q}^{-}(x) = \begin{cases} 0 & x \geq 0 \\ \frac{1}{2}x^2 & x < 0. \end{cases} \tag{2.35}$$

For every constraint p violated from below, we set $\mathcal{G}_p^i(L) = \mathcal{Q}^{-}(L_p - \epsilon^{-1})$. For every constraint q violated from above, we set $\mathcal{G}_q^i(L) = \mathcal{Q}^{+}(L_q - \epsilon)$. Since the $\mathcal{Q}^{\pm}$ functions are continuously differentiable, it is easy to compute the Jacobian $J(\mathcal{G}^i(L))$ and then apply the algorithm

$$L^{(i+1)} = L^{(i)} - \left[J(\mathcal{G}^i(L))\right]^{\dagger}(\mathcal{G}^i(L)). \tag{2.36}$$

Algorithm (2.36) can be used to alter all the eigenvalues of the system if $n = N$ and $\lambda^* \in \mathbb{R}^N$. Alternatively, one can set $n = 2$ and only request the two smallest eigenvalues to be changed to λ_1^* and λ_2^*, respectively.

In the next section we show that altering the lowest eigenvalue λ_1 is enough to cause higher energy transfer to the higher modes. To show, we will use (2.36) to change λ_1 to some desired value, keeping λ_2 constant. We note that since we do not constrain $\lambda_3, \cdots, \lambda_N$, they can change as a result of altering L, but $\lambda_2 \leq \lambda_j$ for $j \in \{3, \ldots, N\}$ will be maintained.

For all applications of this inverse problem algorithm described in the next section, we use (2.36) with the initial conditions $L^{(0)} = [1, 1, \ldots, 1]^{\mathrm{T}}$ and the constraint violation parameter $\epsilon = 10^3$.

2.4.1 Gap Tuning: Methodology

How does the steady-state voltage in the nonlinear circuit change as a function of the gap between the first two eigenvalues of the graph Laplacian Δ? In this section, we address this question by combining the perturbative/iterative algorithms with the inverse problem algorithm. We describe numerical experiments designed to test the effect of closing the graph Laplacian's first eigenvalue gap on the circuit's ability to (a) transfer more energy to higher harmonics, and (b) generate higher-amplitude output signals.

We conduct our numerical experiments on three types of random graphs, all generated using the NetworkX package [43]:

- Barabási-Albert (BA), a preferential attachment model with one parameter, m, the number of edges to draw between each new node and existing nodes [8]. We set $m = 3$ in our experiments.

- Watts-Strogatz (WS), a small world model with two parameters, k, the number of nearest neighbor nodes to which each node is initially connected, and p, the probability of rewiring each edge [84]. In our experiments, we set $k = 5$ and $p = 0.3$.
- Erdös-Rényi (ER), a classical model in which edges are drawn independently with uniform probability p [33]. In our experiments, we set $p = 0.25$.

When we produce realizations of any of these graphs, we accept only those graphs that are connected.

Suppose we have used one of these three models to generate a connected, random graph with N nodes. To make this a concrete circuit problem, we set $C_i^0 = 1$ for all nodes i, and $L_j = 1$ for all edges j. We fix the nonlinearity parameter $\varepsilon = 0.5$. We select $\lfloor N/10 \rfloor$ nodes uniformly at random, and attach half-edges to these nodes with inductance $L_j = 1$. For each node i, we set the conductance $G_i = 0.15$ for the BA and WS graphs, and $G_i = 0.5$ for the ER graphs. This selection will be explained in more detail below.

With these parameters set, we have enough information to compute the graph Laplacian Δ defined by (2.6). As we did before, let $\lambda_1, \ldots, \lambda_N$ denote the eigenvalues of Δ sorted in increasing order. We set the forcing frequency $\omega = \sqrt{\lambda_2}$. Since this value is a resonant frequency of the linear, undamped system we expect it lies close to a resonance for the nonlinear, damped system. The type of forcing we consider is $A \sin \omega t$, a special case of (2.1) with $a = A/(2i)$.

With this setup, we use both the perturbative method and the iterative method to compute the steady-state solution $V(t)$. For the perturbative method, we solve up to order 9, and for the iterative method, we solve using 20 modes. This means that the iterative scheme captures ten modes—11ω through 20ω—that are not captured by the perturbative scheme. We compare the two solutions as a check for whether the number of modes we have considered is sufficient. In all tests, we find that there is no significant difference between the solutions, implying that the first 10 harmonics—ω through 10ω—are sufficient to resolve the solution.

Since $\varepsilon = 0.5$, the capacitance model (2.7) is valid only for $V_i < 2$. For all computed solutions, we check that the maximum voltage across all nodes in one cycle satisfies this constraint.

One quantity of interest in our simulations is the amount of energy in the higher harmonics. Let Ψ be an $N \times M$ complex matrix such that the j-th column of Ψ contains the Fourier coefficients of the $+j\omega$ mode over all N nodes. Here j goes from 1 to M, the maximum mode to which the solution is computed. We then define

$$\kappa_{\text{pre}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\left\| (\Psi_{n2}, \Psi_{n3}, \ldots, \Psi_{nM})^{\mathrm{T}} \right\|_2}{\left\| (\Psi_{n1}, \Psi_{n2}, \ldots, \Psi_{nM})^{\mathrm{T}} \right\|_2}, \tag{2.37}$$

the fraction of energy in modes $+2\omega$ and higher, averaged over all nodes. We also compute

$$\mathrm{V}_{\text{pre}} = \max_{1 \le i \le N} \max_{0 \le t \le T} |V_i(t)|, \tag{2.38}$$

the maximum magnitude voltage produced anywhere in the circuit during one period. For both κ and V, the subscript "pre" denotes that these quantities have been computed before we change L to manipulate the eigenvalues of Δ.

Table 2.3: Eigenvalue gaps for random graphs.
For each of three types of random graphs, we vary the number of nodes N and record the first eigenvalue gap $\lambda_2 - \lambda_1$. The displayed results have been averaged over 200 realizations.

	$N = 25$	$N = 75$	$N = 125$	$N = 175$
Barabási-Albert (BA)	0.6408	0.3561	0.3155	0.2850
Watts-Strogantz (WS)	1.4180	1.3255	1.2970	1.2758
Erdös-Rényi (ER)	1.5469	8.6936	17.7061	26.5297

Having computed κ_{pre}, we now study how this fraction changes when we reduce the gap between the first two eigenvalues of Δ. For a fixed $\delta \in (0, 1)$, we set $\lambda_1^* = \delta\lambda_2$ and $\lambda_2^* = \lambda_2$, and then apply the inverse problem algorithm.

Using (2.36), we solve for a vector of inductances L^* such that the first two eigenvalues of Δ are given by λ_1^* and λ_2^*. When we iterate forward using (2.36), if we find that $||\mathcal{G}^i(L)||_2 \geq 10^{-12}$ after 200 iterations, we generate a new random graph and restart the experiment. In our current work we rely on the Newton-step and do not try to find the optimal step length which can cause certain graphs configuration to lead to slow convergence. The 200 iteration limit is employed strictly to ensure fast simulations. In our simulations, we have not found any graph configuration unable to solve the inverse problem if we let the solver run for more steps.

We recompute the graph Laplacian Δ using the new vector inductances L^*, and again apply the perturbative and iterative algorithms to solve for the steady-state solution $V(t)$. Using this solution, we compute the energy in the higher harmonics using the right-hand side of (2.37), now labeling this average fraction as κ_{post}. We also compute the right-hand side of (2.38) and label this quantity as V_{post}.

Let us now describe how we choose the particular values of the conductance G_i and the eigenvalue fraction δ. In Table 2.3, we tabulate $\lambda_2 - \lambda_1$, the gap between the second and first eigenvalue for each of the three types of random graphs described above. The eigenvalue gaps we present are averaged over 200 simulations for each of four graph sizes: $N \in \{25, 75, 125, 175\}$.

We observe that the eigenvalue gaps for the BA and WS graphs do not change appreciably as a function of N, while for ER graphs, the gaps grow quickly as a function of N. Our choice of δ is guided by these results. For BA and WS graphs, we choose $\delta \in [0.5, 0.99]$. For ER graphs, we choose $\delta \in [0.25, 0.75]$.

When we solve for the steady-state voltages on these three types of graphs, we also notice a difference. For ER graphs, the maximum voltage grows quickly as a function of N, while for BA and WS graphs, the same phenomenon does not occur. To counteract the large growth of maximum voltages for large graph sizes, we set the conductance G_i to the larger value of 0.5 for ER graphs, causing more energy to dissipate through resistors. For BA and WS graphs, we set G_i to 0.15.

2.5 Results and Discussion

2.5.1 Comparison of Steady-State Algorithms

In this section, we compare steady-state solutions computed by numerical integration against the solutions computed using the perturbative and iterative methods derived earlier.

For the tests described in this section, the domain is a 20×20 square lattice with $N = 400$ nodes. Nodes along the left and bottom boundaries of the lattice are driven by input forcing. The input provided is $0.03\sin(\omega t)$ with $\omega = 1$. For the capacitance model (2.7), we set $C^0 = 1$ and $\varepsilon = 0.5$. For each edge j, we set $L_j = 1$. The conductance G_i is set to 0.01 at all points except for the top-right corner, where it is set to 1.0.

To compare the results of the perturbative and iterative methods against the numerical integrator, we will need to obtain the steady-state solution using the numerical integrator. To do this, we start at $t = 0$ and numerically integrate the first-order system (2.2-2.3) forward in time for 1500 cycles. The ODE solver uses the Dormand-Prince (*dopri5*) method with relative and absolute tolerances equal to 10^{-10} and 10^{-12}, respectively. For the parameters given above, this number of cycles is sufficient so that, from one cycle to the next, the change in the solution is on the order of the relative tolerance of the numerical integrator. Hence we take the solution over the last cycle to be the steady-state solution.

The ODE solvers we tried are not state-of-the-art solvers and there exist other solvers which might be able to solve the current problem both accurately and faster than the *dopri5* solver. Since these approaches do not form the thesis of our work, we do not pursue any such possibilities. As a result, our work does not provide an exhaustive discussion of possible approaches.

As a preliminary check, we directly compare the three steady-state solutions. We treat the solution obtained from numerical time integration as a reference solution $z^{\text{ref}}(t)$. Let $z^{(i)}(t)$ denote either the perturbative or iterative solution after i iterations—for the perturbative method, the iteration count is defined as the largest mode number present in the solution. Let T be the period of the steady-state solution, and for an integer $\tau > 0$, consider the equispaced discretization of the interval $[0, T]$ given by $\{t_k = kT/\tau\}_{k=1}^{k=\tau}$. For each iteration i, we evaluate both the the perturbative/iterative and reference solution on this equispaced grid with $\tau = 64$ points, and we compute the error

$$\mathrm{E}(i) = \left[\frac{1}{\tau} \sum_{j=1}^{N} \sum_{k=1}^{\tau} \left(z_j^{(i)}(t_k) - z_j^{\text{ref}}(t_k) \right)^2 \right]^{1/2} . \tag{2.39}$$

In Figure 2.5 we have plotted $\log_{10} \mathrm{E}(i)$ as a function of the iteration i. While both methods initially tend towards the reference solution, we see from Figure 2.5 that the error does not drop below 10^{-9}. In the following subsections, we provide evidence that the reference solution is less accurate than the solutions computed using the perturbative/iterative methods. This explains why the error in Figure 2.5 does not go to zero, i.e., why the perturbative/iterative methods will not converge to the solution produced by time integration.

Our first tests concern the Fourier coefficients of the computed solutions. In what follows, we use $\mathbf{A}$ to denote the vector of Fourier series coefficients associated with a steady-state solution computed using any of the three methods discussed above.

Figure 2.5: Error between perturbative/iterative solutions and reference solution. *The reference solution has been computed via numerical time integration. We plot the log of the error as a function of the number of iterations. As shown in Figures 2.6 and 2.7 together with Tables 2.4 and 2.5, the perturbative/iterative solutions are more accurate than the reference solution. This explains why, in the above plot, the perturbative and iterative solutions do not converge to the reference solution.*

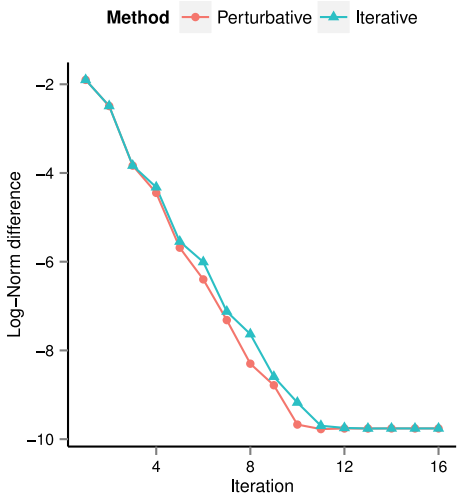


Table 2.4: Comparison of the three solutions using the fixed point error metric (2.40).
The numerical time stepping solutions as noted below is further away from the fixed point of the system as compared to the other methods.

Scheme	$\max_{1\leq k\leq M} \left\|\alpha_k - F_k^M(\mathbf{A})\right\|_\infty$
Numerical	2.035×10^{-11}
Perturbative	3.4321×10^{-16}
Iterative	2.7144×10^{-16}

Fixed Point Error

Suppose that $V(t)$ is an exact T-periodic steady-state solution of (2.4). If we were to expand this solution in a Fourier series as in (2.22), the resulting (infinite) coefficient vector $\mathbf{A}$ would satisfy $\mathbf{A} = F_k^\infty(\mathbf{A})$ for all k, with F as in (2.29).

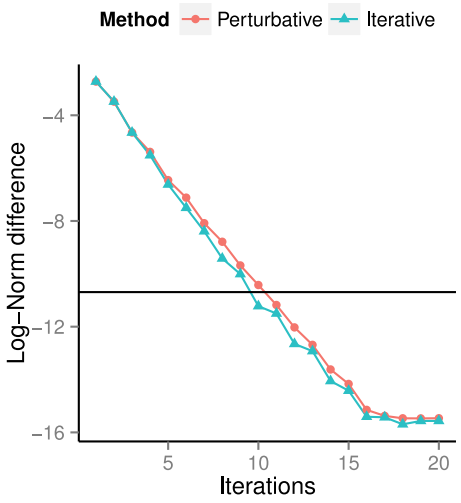
In both the perturbative and iterative methods, what we seek is a finite-mode truncation of the exact solution. For the iterative method we fix $M = 20$ so that the highest mode has frequency 20ω. In the perturbative method we solve up to order 19, which implies that the highest mode in the solution has frequency 20ω.

Combining the ideas of the last two paragraphs, it is natural to use

$$\mathcal{E}_M(\mathbf{A}) = \max_{1\leq k\leq M} \|\alpha_k - F_k^M(\mathbf{A})\|_\infty \tag{2.40}$$

as an error metric for the M-mode truncation of the exact solution. In Table 2.4, we record (2.40) for solutions computed using the perturbative, iterative, and numerical integration

Figure 2.6: Log of the fixed point error (2.40) of the perturbative/iterative solutions after i iterations.
Up to iteration 16*, both curves are close to linear with slopes of* -2.1407 *(perturbative) and* -2.2326 *(iterative), indicating exponential convergence of both methods. Note that only* 10 *iterations are required to reach error values which correspond to that of the numerical integrator's solution.*



methods. Note that the iterative and perturbative methods directly provide us with the Fourier coefficients necessary for this calculation. We compute the Fourier coefficients of the numerical integrator's solution using the FFT. Table 2.4, shows that the perturbative and iterative solutions are about five orders of magnitude closer to being fixed points of F_k^M than the solution obtained from numerical integration.

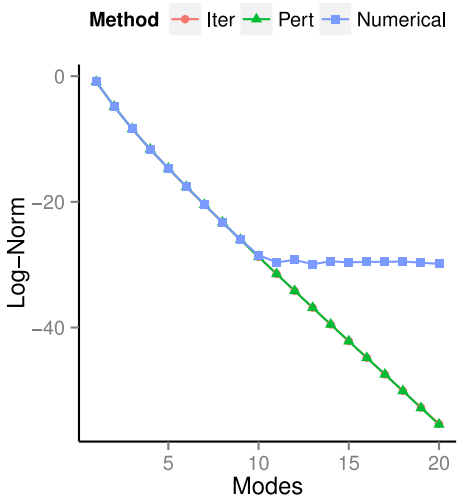
For the perturbative and iterative methods, let us examine how the fixed point error (2.40) decreases as a function of iteration count. In Figure 2.6, we plot $\log \mathcal{E}_M(\mathbf{A}^{(i)})$ versus the iteration number i. Here $\mathbf{A}^{(i)}$ is the vector of Fourier coefficients for the solution computed after only i iterations. The plot shows that, for both the perturbative and iterative methods, approximately 10 iterations are required to match the fixed point error of the solution computed using time integration. The error of this latter solution, taken from Table 2.4, is represented on Figure 2.6 by a horizontal black line.

Figure 2.6 also shows that the perturbative and iterative methods converge exponentially in the number of iterations. From iteration 1 until iteration 16, fitting lines of best fit to the perturbative and iterative error curves results in slopes of -2.1407 and -2.2326, respectively. For both methods, this can be approximated by writing $\mathcal{E}_M(\mathbf{A}^{(i)}) \sim e^{-2i}$. After 16 iterations, the error has approached machine epsilon, and both curves level off before reaching the final values shown in Table 2.4.

Decay Rate

Suppose we write the first-order system (2.2 –2.3) in the form $\dot{\mathbf{z}} = \mathbf{F}(\mathbf{z}, t)$, with $\mathbf{z} = (I, V)^{\mathrm{T}}$. Then on the open set $\mathcal{D} = \{I \in \mathbb{R}^{e+f}, V \in \mathbb{R}^N, t \in \mathbb{R} \mid |V_i| < \varepsilon^{-1},\ 1 \le i \le N\}$, the vector field $\mathbf{F} : \mathcal{D} \to \mathbb{R}^{e+f+N}$ is C^∞, i.e., $\mathbf{F}$ is j times continuously differentiable for any integer

Figure 2.7: Decay of Fourier coefficients.
We plot $\log\|\alpha_k\|$ *versus* k *to illustrate the decay of Fourier coefficients for the three methods. The iterative and perturbative curves coincide and are nearly linear with slope* -2.8004*; the exponential decay of these Fourier coefficients is consistent with theory. The time integrator's Fourier coefficients do not decay after mode* 10*, violating the theoretical decay rate.*



$j \geq 1$. By the standard existence/uniqueness theorem for ordinary differential equations, it follows that wherever the solution $\mathbf{z}(t) = (I(t), V(t))^{\mathrm{T}}$ exists, it must also be C^∞ in t.

The above observation enables us to test the decay of the Fourier coefficients of all three solutions. For if the steady-state solution $V(t)$ is C^∞ in t, then the Fourier series coefficients of V must satisfy the following decay property:

$$\text{For all } \ell \geq 1, \text{ there exists } C_\ell \text{ such that } \|\alpha_k\| \leq C_\ell k^{-\ell}. \tag{2.41}$$

To examine the decay of the Fourier coefficients for the three computed solutions, we plot $\log\|\alpha_k\|$ versus k in Figure 2.7. For the perturbative and iterative solutions, the curves on the plot coincide and are nearly linear with slope -2.8004. This implies that $\|\alpha_k\| \sim e^{-2.8k}$, which is sufficient to satisfy the theoretical decay rate given by (2.41).

The Fourier coefficients obtained from the numerical integrator's solution, on the other hand, do not decay at all beyond mode 10. This violates the theoretical decay rate (2.41) even for $\ell = 1$.

Energy Conservation

Next we test the energy conservation properties of the three computed solutions. We proceed to derive an energy balance equation. Because our capacitors are voltage-dependent, the charge Q and voltage V are related via $\mathrm{d}Q = C(V)\mathrm{d}V$, which implies

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = C(V)\frac{\mathrm{d}V}{\mathrm{d}t}.$$

Using this in (2.3) together with (2.2), we obtain

$$I^{\mathrm{T}}L\frac{\mathrm{d}I}{\mathrm{d}t} + V^{\mathrm{T}}\frac{\mathrm{d}Q}{\mathrm{d}t} = I^{\mathrm{T}}W - V^{\mathrm{T}}GV. \tag{2.42}$$

Table 2.5: Comparison of the three solutions' preservation of the energy balance (2.44).
The perturbative and iterative method shows machine precision energy balance while the numerical time stepping method is about 6 orders of magnitude lower in terms of energy balance preservation over 1 cycle.

Scheme	$\left\lvert \int_0^T I^{\mathrm{T}} W \mathrm{d}t - \int_0^T V^{\mathrm{T}} G V \mathrm{d}t \right\rvert$
Numerical	5.7841×10^{-12}
Perturbative	2.1684×10^{-18}
Iterative	1.2576×10^{-18}

Let $\mathbf{M}(t)$ be the total energy stored in the magnetic fields of all inductors at time t. Then

$$\frac{\mathrm{d}\mathbf{M}}{\mathrm{d}t} := \frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{1}{2}\sum_{i=1}^{e+f} L_i I_i^2\right] = \sum_{i=1}^{e+f} I_i L_i \frac{\mathrm{d}I_i}{\mathrm{d}t} = I^{\mathrm{T}} L \frac{\mathrm{d}I}{\mathrm{d}t},$$

the first term on the left-hand side of (2.42). Let $\mathbf{E}(t)$ be the total energy stored in the electric fields of all capacitors at time t. Then

$$\frac{\mathrm{d}\mathbf{E}}{\mathrm{d}t} := \frac{\mathrm{d}}{\mathrm{d}t}\left[\sum_{i=1}^{N} \int_0^{Q_i(t)} V_i(q)\mathrm{d}q\right] = \sum_{i=1}^{N} V_i \frac{\mathrm{d}Q_i}{\mathrm{d}t} = V^{\mathrm{T}} \frac{\mathrm{d}Q}{\mathrm{d}t},$$

the second term on the left hand side of (2.42). Hence (2.42) reads:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{M}(t) + \mathbf{E}(t)\right) = I^{\mathrm{T}} W - V^{\mathrm{T}} G V. \tag{2.43}$$

If the system has reached a T-periodic steady state, then $I(t)$, $V(t)$, $\mathbf{M}(t)$, and $\mathbf{E}(t)$ will all be T-periodic. Therefore, integrating (2.43) in t from $t = 0$ to $t = T$, we find that the left-hand side vanishes. The remaining terms yield the following energy balance equation:

$$\int_0^T I^{\mathrm{T}} W \mathrm{d}t = \int_0^T V^{\mathrm{T}} G V \mathrm{d}t. \tag{2.44}$$

The left-hand side is the energy pumped into the system over one cycle, while the right-hand side denotes the energy dissipated through resistors, again over one cycle.

Table 2.5 shows the absolute difference between the left- and right-hand sides of (2.44), computed for each of the three methods. We find that for the perturbative and iterative methods' energy balance errors are below machine epsilon. The numerical integrator yields an error approximately five orders of magnitude larger than that of the two other methods.

Computational Time

The results presented thus far indicate that whether we measure error using the fixed point error (2.40) or the violation of the energy balance (2.44), the solution obtained via numerical integration has errors that are approximately five orders of magnitude larger than that of the perturbative/iterative methods. The actual values of the errors committed by the numerical integrator in Tables 2.4 and 2.5, as well as the final error values for the curves in Figure 2.5,

Table 2.6: Timing results for three frequency response algorithms.
For the numerical method, Time I records the time required to integrate forward by 1500 *cycles and obtain a solution with fixed point error* $\approx 2 \times 10^{-11}$ *as in Table 2.4. For the perturbative/iterative methods, Time I records the time required to compute* 10 *iterations, resulting in a fixed point error comparable to that of the numerical method—see the crossing of the curves with the black horizontal line in Figure 2.6. For the perturbative/iterative methods, we also record under Time II the time required to achieve the* $O(10^{-16})$ *errors as in Table 2.4. All times are averaged over* 10 *runs.*

Scheme	Time I (to achieve comparable error)	Time II (to achieve $O(10^{-16})$ error)
Numerical	483.2126 s	N/A
Perturbative	1.71024 s	7.29115 s
Iterative	0.68374 s	0.89012 s

are close to the numerical integration relative and absolute tolerances of 10^{-10} and 10^{-12}, respectively. We hypothesize that, if computational time were not an issue, we could run the numerical integrator with smaller tolerances and obtain steady-state solutions that more closely match, in the same error metrics described above, the perturbative and iterative solutions.

As we now proceed to show, computational time is a major issue for the time integration method. In Table 2.6, we record the time required to compute steady-state solutions using the three methods. We see from the Time I column that to achieve the error of $\approx 2 \times 10^{-11}$ in Table 2.4, the numerical integrator requires over 483 seconds. We know from Figure 2.6 that the perturbative/iterative methods require 10 iterations to achieve approximately the same error as the time integrator; the remaining entries in the Time I column show that both the perturbative and iterative algorithms compute such a solution hundreds of times faster than the time integrator.

The Time II column in Table 2.6 records how long it takes the perturbative/iterative algorithms to achieve the errors recorded in Table 2.4. Observe that even if we run the perturbative/iterative algorithms all the way to full convergence, they are much faster than time integration. In this case, the time integrator is 542 (respectively, 66) times slower than the iterative (respectively, perturbative) algorithm.

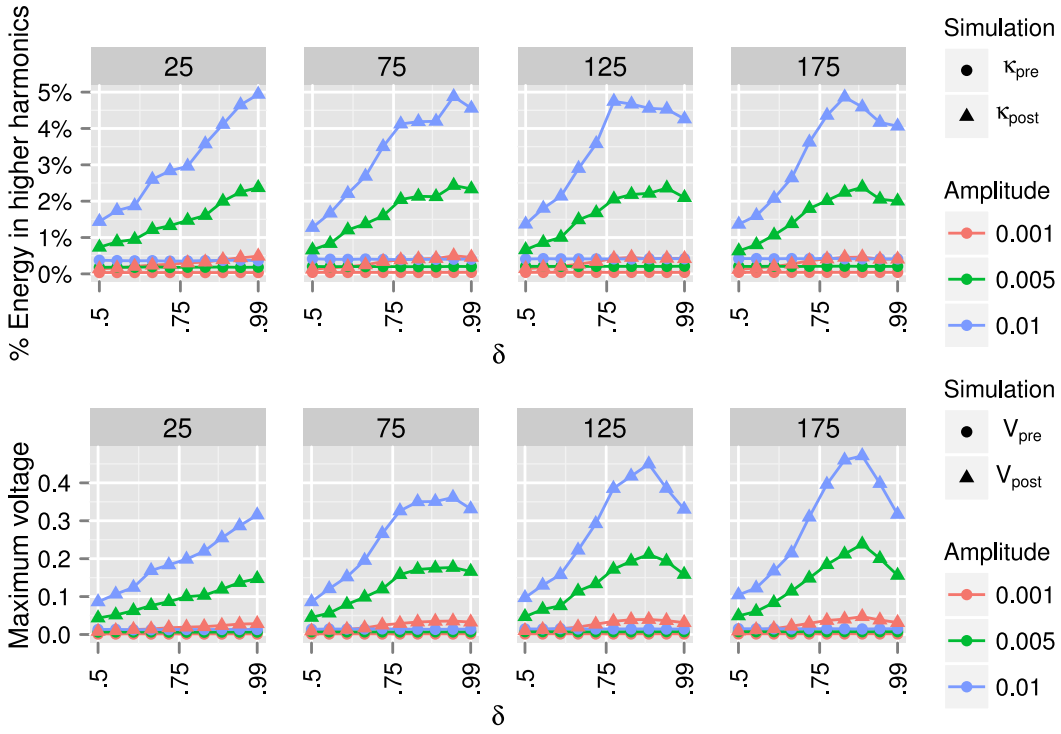
Note that the perturbative and iterative algorithms were implemented in Python using the Numpy/Scipy packages. The *dopri5* implementation used for numerical time integration is the implementation provided by the *scipy.integrate.ode* module. All times reported are average times across 10 runs on the same machine.

2.5.2 Gap Tuning

For each $N \in \{25, 75, 125, 175\}$, and for each of 10 values δ chosen in an equispaced fashion from the intervals given above, we compute 100 runs of the complete procedure described above—see Gap Tuning: Methodology Section 2.4.1. For each such run, we compute pre/post values of κ and V for three values of the input forcing amplitude, which we take to be the same at all input nodes k: $A_k \in \{0.001, 0.005, 0.01\}$. These results for κ and V, averaged over the 100 runs, are plotted in Figures 2.8, 2.9, and 2.10.

Figure 2.8 shows the results for Barabási-Albert (BA) graphs. By shrinking the gap between the first two eigenvalues, the percentage of energy transferred to higher harmonics

Figure 2.8: Barabási-Albert random graph results.
From left to right, we present results for Barabási-Albert random graphs with $N = 25$, 75, 125, and 175 nodes. For each graph, we use algorithm (2.36) to modify the inductances L such that the ratio of the smallest to the second smallest eigenvalue is δ. We use pre *and* post *to denote, respectively, the graphs before and after algorithm (2.36) is applied.*
By shrinking the gap between the first two eigenvalues, the energy transferred to higher harmonics (2.37) can be increased from approximately $\kappa_{pre} \approx 0.5\%$ to $\kappa_{post} \approx 5\%$ (for all graph sizes), and the maximum voltage (2.38) can be increased from $\mathrm{V}_{pre} < 0.05$ volts to $\mathrm{V}_{post} \in [0.3, 0.5]$ volts (depending on the graph size). We also note that for larger graphs, choosing $\delta = 1$ (i.e., no gap between the first two eigenvalues) does not yield optimal behavior.

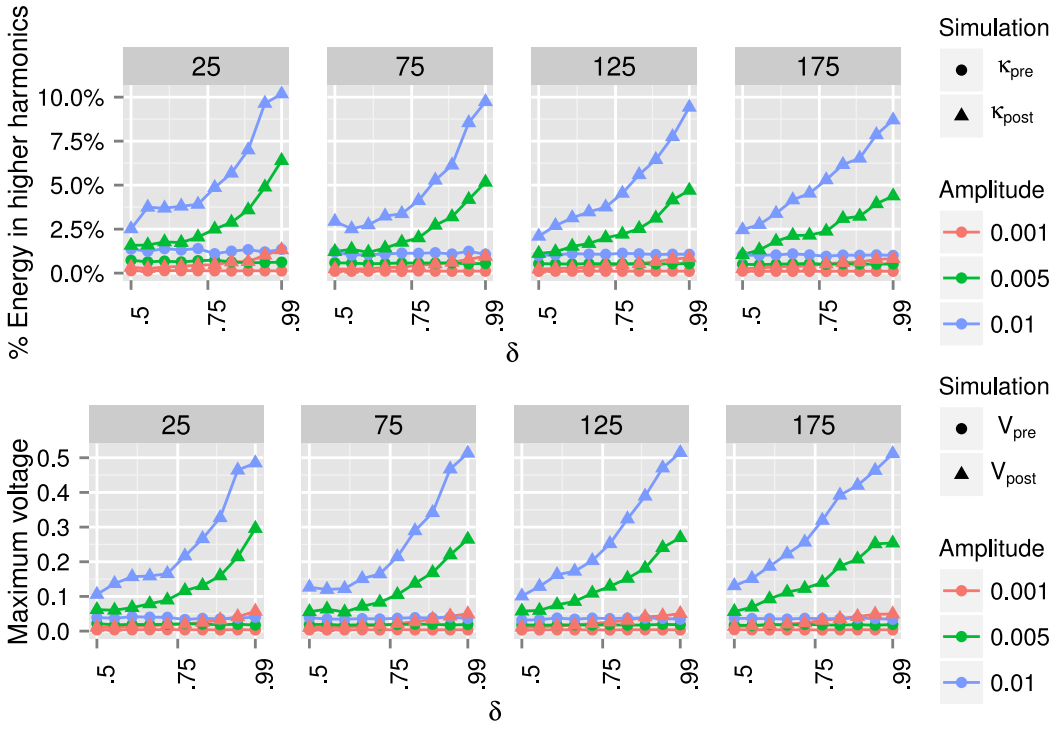


(2.37) can be increased by approximately one order of magnitude, for all graph sizes, while the maximum magnitude voltage (2.38) can be increased by a factor of 6 to 20, depending on the graph size. Note that for larger graphs, choosing $\delta = 1$, i.e., forcing the first two eigenvalues to coincide, does not yield optimal energy transfer to higher harmonics.

The results in Figure 2.9 for Watts-Strogatz (WS) graphs are similar to those for BA graphs. We again find that by shrinking the gap between the first two eigenvalues, the energy transferred to higher harmonics (2.37) can be increased. However, the values of κ_{post} for Watts-Strogatz graphs are about twice as large as the values of κ_{post} for Barabási-Albert graphs in Figure 2.8. For all graph sizes, tuning the eigenvalue gap can increase the percentage of energy transferred to higher harmonics by a factor of up to 8, while the maximum magnitude voltage can be increased by approximately one order of magnitude.

In Figure 2.10, we present the results for Erdös-Rényi graphs. The results again support the finding that by shrinking the gap between the first two eigenvalues, the circuit can transfer more energy to higher harmonics and boost the peak magnitude of output signals.

Figure 2.9: Watts-Strogatz random graph results.
From left to right, we present results for Watts-Strogatz random graphs with $N = 25$, 75, 125, *and* 175 *nodes. For each graph, we use algorithm (2.36) to modify the inductances* L *such that the ratio of the smallest to the second smallest eigenvalue is* δ. *We use* pre *and* post *to denote, respectively, the graphs before and after algorithm (2.36) is applied.*
By shrinking the gap between the first two eigenvalues, the energy transferred to higher harmonics (2.37) can be increased from $\kappa_{pre} \approx 1\%$ *to* $\kappa_{post} \geq 8.75\%$ *(for all graph sizes), and the maximum voltage (2.38) can be increased from* $\mathrm{V}_{pre} \approx 0.05$ *volts to* $\mathrm{V}_{post} \approx 0.5$ *volts (for all graph sizes). The values of* κ_{post} *for Watts-Strogatz graphs are about twice as large as the values of* κ_{post} *for Barabási-Albert graphs in Figure 2.8.*



Specifically, we see that the energy transferred to higher harmonics (2.37) can be increased to $\kappa_{\text{post}} \in [1, 8]\%$, and the maximum voltage (2.38) can be increased to $\text{V}_{\text{post}} \in [0.1, 0.8]$.

The results for Erdös-Rényi graphs are much more strongly dependent on the number of nodes N than the results shown in Figures 2.8 or 2.9. Note that the peak voltages for the $N = 175$ graphs with forcing amplitude 0.01 are the largest voltages for any graphs considered in this chapter. We can increase the peak voltages for smaller graphs by choosing a smaller value of the conductance than $G_i = 0.5$ (for all nodes i), the value used to compute the results in Figure 2.10.

For all three types of graphs, both *pre* and *post* values of κ and V increase as a function of the input forcing amplitude.

Code

All code necessary to reproduce the above results have been posted as a public repository on GitHub, accessible at the following URL:

`https://github.com/GarnetVaz/Nonlinear-electrical-oscillators`

We use Python together with the `numpy`, `scipy`, `matplotlib`, and `networkx` modules for all numerical computing. The code that generates Figures 2.8,2.9, and 2.10 is set to utilize 10 processors using the open-source `multiprocessing` module. For plotting, we use R together with the `ggplot2`, `plyr`, and `reshape` packages. All languages, packages and modules used are open source.

Assuming all packages and modules have been correctly installed, one can reproduce all results by running the Python codes `numerical_comparison.py` and `graphmulti.py`. The latter code may require several hours to run. The Python codes will generate figures using R; the R codes we provide need not be run independently.

Further details on how to run the codes, including the specific versions of required packages and modules, are given in the `README.md` file at the URL given above.

The code that we provide can easily be modified to run simulations not described here. For example, one can compare the perturbative/iterative algorithms against numerical integration using graphs other than the 20×20 grid graph used above. One can also explore gap tuning results for random graphs with different parameters than the ones we have chosen.

2.6 Conclusion

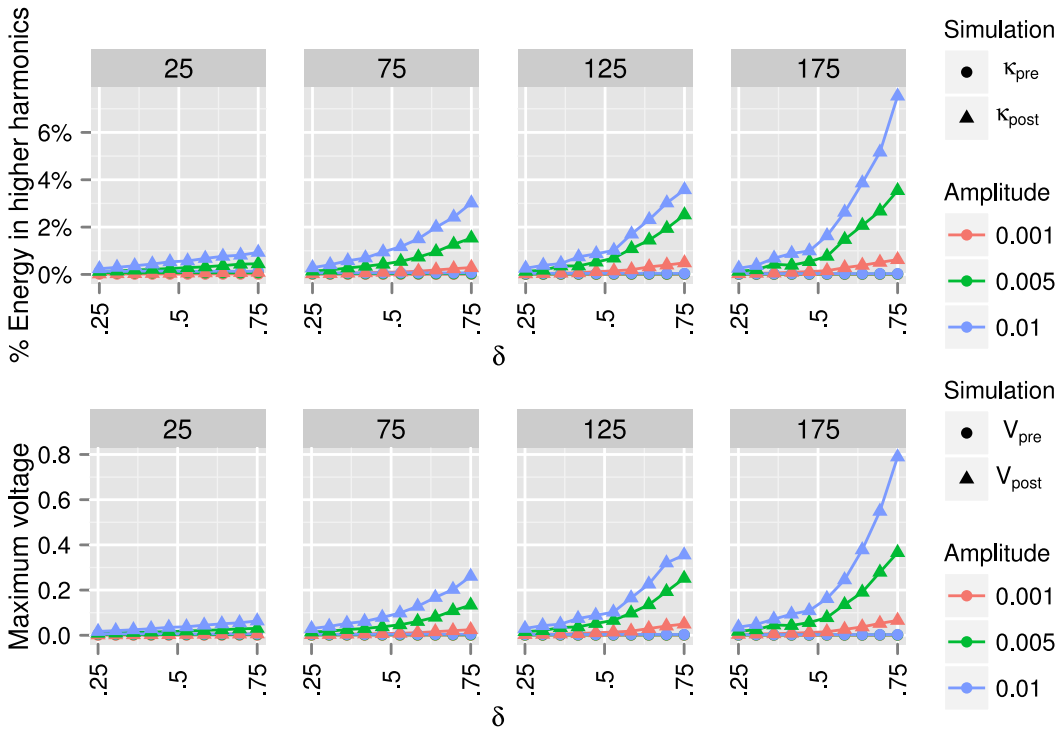
For nonlinear electrical circuits on arbitrary connected graphs, we have developed two numerical methods to compute the steady-state voltage. Using both absolute metrics and relative comparisons with a solution obtained via direct numerical integration, we validated the new algorithms. The results show that for the same error tolerance, both the perturbative and iterative methods are orders of magnitude faster than the solution obtained by time stepping. Moreover, these methods are able to capture the behavior in high Fourier modes and converge to machine precision in a fixed point error metric.

In the following chapter, we show how we can apply the steady-state algorithms developed above to solve Maxwell's equations in nonlinear electromagnetic media [14]. This application makes use of the correspondence between the nonlinear electrical network and a finite volume discretization of Maxwell's equations on an unstructured mesh.

Figure 2.10: Erdös-Rényi random graph results.
From left to right, we present results for Erdös-Rényi random graphs with $N = 25$, 75, 125, *and* 175 *nodes. For each graph, we use algorithm (2.36) to modify the inductances* L *such that the ratio of the smallest to the second smallest eigenvalue is* δ. *We use* pre *and* post *to denote, respectively, the graphs before and after algorithm (2.36) is applied.*

By shrinking the gap between the first two eigenvalues, the energy transferred to higher harmonics (2.37) can be increased to $\kappa_{post} \in [1, 8]\%$ *(depending on the graph size), and the maximum voltage (2.38) can be increased to* $\mathrm{V}_{post} \in [0.1, 0.8]$ *(depending on the graph size).*

The results for Erdös-Rényi graphs are much more strongly dependent on the number of nodes N *than the results shown in Figures 2.8 or 2.9. Note that the peak voltages for the* $N = 175$ *graphs with forcing amplitude* 0.01 *are the largest voltages for any graphs considered in this chapter. We can increase the peak voltages for smaller graphs by choosing a smaller value of the conductance than* $G_i = 0.5$ *(for all nodes* i*), the value used to compute the results in this figure.*



In order to enhance the nonlinearity-driven features of these circuits, we developed a Newton-like algorithm that alters the eigenvalues of a network's graph Laplacian. The algorithm leaves the topology of the network untouched, changing only the inductances, i.e., the edge weights. By applying the Newton-like algorithm to three types of random graphs, we showed that reducing the gap between the graph Laplacian's first two eigenvalues leads to enhanced nonlinear behavior. Comparing pre- and post-optimized circuits, it is evident that optimizing the eigenvalue gap significantly increases (i) energy transfer from the fundamental driving frequency to higher harmonics, and (ii) the maximum magnitude output voltage.

In both the perturbative and iterative algorithms, the only way in which the network's structure influences its frequency response is through the graph Laplacian matrix. In our experiments, we have tuned this graph Laplacian's first eigenvalue gap by holding the topology of the graph constant and altering the edge weights, i.e., inductances. What if we had instead tuned the gap by holding the edge weights constant and altering the topology of the graph? This would amount to altering the incidence matrix B instead of the inductance vector L. So long as both types of alterations result in the same graph Laplacian Δ, our results indicate that the nonlinear electrical network's functionality should be enhanced significantly. This, of course, leads to the question of whether it is possible to algorithmically alter the network topology to achieve a particular graph Laplacian matrix, an interesting avenue for further work.

Chapter 3

FVFD Method for Nonlinear Maxwell's Equations

3.1 Introduction

In Chapter 2 we devised two algorithms for the solution of nonlinear RLC circuits where the underlying topology of the graphs is random. In this chapter, we show how these algorithms can be applied to solve for the steady-state solutions for nonlinear inhomogeneous Maxwell's Equations on arbitrary domains when subjected to harmonic forcing.

Maxwell's equations for the (H_1,H_2,E) polarized mode in a planar inhomogeneous medium with permittivity $\epsilon(x, y, E)$ and permeability $\mu(x, y)$ are given by

$$\partial_t(\mu\Lambda) = -\nabla E, \qquad \Lambda = (-H_2, H_1) \tag{3.1a}$$

$$\partial_t(\epsilon E) = -\text{ div } \Lambda. \tag{3.1b}$$

We consider an arbitrary domain Ω and use $\mathrm{d}\Omega$ to denote its boundary. We split the boundary $\mathrm{d}\Omega$ into two disjoint regions $\mathrm{d}\Omega_f$ and $\mathrm{d}\Omega_b$. The boundary $\mathrm{d}\Omega_f$ corresponds to the boundary where time-harmonic forcing is applied and $\mathrm{d}\Omega_b$ corresponds to the part of the boundary where appropriate boundary conditions are applied. Throughout this chapter we consider a real forcing applied on $\mathrm{d}\Omega_f$ which has a period T and can is represented by a general form

$$E(x, y, t)\big|_{\mathrm{d}\Omega_f} = f(x, y, t)\big|_{\mathrm{d}\Omega_f} + c.c. \tag{3.1c}$$

where "c.c." reprents the complex conjugate of f. The forcing function f is real and is also T–periodic i.e. $f(x, y, t) = f(x, y, t + T)$. For the boundary condition on $\mathrm{d}\Omega_b$ we use the Leontovich boundary conditions given by

$$\Lambda \cdot \hat{\mathbf{n}} = \sigma(x, y)E(x, y, t), \tag{3.1d}$$

where $\hat{\mathbf{n}}$ represents the unit outward normal to $\mathrm{d}\Omega_b$.

Setting $\mu(x, y, E) = \mu_0$ and $\epsilon(x, y, E) = \epsilon_0$ results in linear Maxwell's Equations. This chapter considers the case when $\mu(x, y, E) = \mu(x, y)$ but $\epsilon(x, y, E)$ still depends on E. The specific form of dependence of the permittivity involves a linear dependence on the electric field as given by

$$\epsilon(x, y, E) = \epsilon_0(1 - bE). \tag{3.2}$$

Here ϵ_0 is a constant which is dependent on space but not on the electric field E. For notational convenience we do not display ϵ_0 as a function of space. The specific choice of nonlinearity might seem very restrictive but we mention that the methods developed can be easily extended to other nonlinearities as noted in the conclusion.

Our first goal is to show that the finite volume method obtained by discretizing the considered Maxwell's equations (3.1) along with (3.2) lead to a circuit problem consisting of voltage-dependent capacitors, inductors and resistors. The formulations naturally obey Kirchoff's laws, thereby ensuring energy conservation for the system as a whole.

The equivalent circuit formulation for the case when the permittivity was *independent* of the electric field but had spatial variation has been developed in previous work [13]. The current work extends it to the case of polynomial nonlinearity in the permittivity of the material.

Numerical methods for Maxwell's Equations have a long history spanning over five decades. In our current work we derive equivalent circuits whose continuum limit yields Maxwell's Equations. There exists a large volume of study of such circuit formulations in the literature starting from work as early as the 1940's [55]. Recently, whenever a new numerical method is developed for the solution of Maxwell's Equations the equivalent circuit formulation is studied later. For example, the equivalent circuit formulations for (i) the finite difference time domain method [42], (ii) the finite element time domain method [41], (iii) finite volume Fourier domain method [13], (iv) Transmission Line Matrix method [26] and (v) Method of Moments [34] have been studied. Our current work continues on the use of equivalent circuits for nonlinear Maxwell's Equations.

3.2 Finite Volume Discretization of Maxwell's Equations

To derive equivalent circuit based relations for the system defined in (3.1), we discretize the computational domain with a triangulation. The derivation does not rely on a specific discretization and the method can be used with quadrilateral, hexagonal or hybrid meshes. An interior cell i along with its adjoining triangles is shown in Figure 3.1. For each cell i we use b_i to denote its barycenter. For the three edges of the triangle i we denote the outgoing normal vector by $\hat{n}_{i,p}$ where p is the cell sharing an edge with i. Similarly, we use $\hat{m}_{i,p}$ to denote a unit normal in the direction of the vector joining b_i to b_p. In order to relate the two sets of normal vectors we define a counter-clockwise rotation matrix $R_{i,p}$ given by

$$R_{i,p} = \begin{bmatrix} \cos\theta_{i,p} & -\sin\theta_{i,p} \\ \sin\theta_{i,p} & \cos\theta_{i,p} \end{bmatrix}.$$

For each set of vectors $\hat{n}_{i,p}, \hat{m}_{i,p}$ there exists a $\theta_{i,p}$ such that the relation

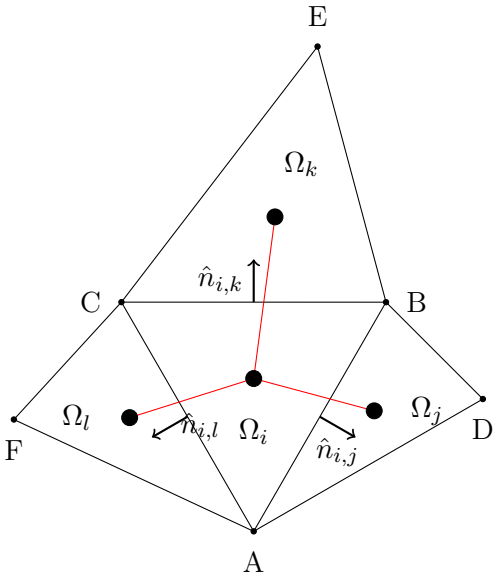
$$\hat{n}_{i,p} = R_{i,p}\hat{m}_{i,p}$$

holds true. We define the voltage in cell i having area Ω_i as

$$V_i[E] := \frac{\eta}{|\Omega_i|} \int_{\Omega_i} E \, \mathrm{d}\Omega_i \tag{3.3}$$

where $\eta > 0$ is the characteristic length scale in the out-of-place direction. The corresponding

Figure 3.1: Cell diagram for an interior triangle i along with the neighboring triangles.
$\hat{n}$ represents the unit normal vector to an edge of the triangulation. Ω represents the volume of the cells. The red lines connect the barycenters of neighboring triangles.



charge in cell i is defined as

$$Q_i[E] := \int_{\Omega_i} \epsilon(x, y, E)E \ \mathrm{d}\Omega_i = \int_{\Omega_i} \epsilon_0 \left(E - bE^2\right) \ \mathrm{d}\Omega_i \tag{3.4}$$

where we have substituted the permittivity from (3.2) above. We are treating V_i and Q_i as functionals: in both cases, the input is a function $E(x, y, t)$ and the output is a function of t. This highlights the dependence of V_i and Q_i on E, and enables us to analyze the change in these functionals with respect to E.

To compute functional derivatives, let ϕ be a test function in Ω_i. Then

$$\begin{aligned}
\left\langle \frac{\delta V_i}{\delta E}, \phi \right\rangle &= \left[\frac{d}{d\gamma} V_i[E + \gamma\phi] \right]_{\gamma=0} \\
&= \left[\frac{d}{d\gamma} \frac{\eta}{|\Omega_i|} \int_{\Omega_i} (E + \gamma\phi) \ \mathrm{d}\Omega_i \right]_{\gamma=0} \\
&= \frac{\eta}{|\Omega_i|} \int_{\Omega_i} \phi \ \mathrm{d}\Omega_i,
\end{aligned}$$

This implies that

$$\frac{\delta V_i}{\delta E} = \frac{\eta}{|\Omega_i|}, \tag{3.5}$$

a constant. Proceeding in a similar fashion for the charge Q_i we have

$$\begin{aligned}\left\langle \frac{\delta Q_i}{\delta E}, \phi \right\rangle &= \left[\left. \frac{d}{d\gamma} \right|_{\gamma=0} Q_i[E + \gamma\phi] \right]_{\gamma=0} \\ &= \left[\left. \frac{d}{d\gamma} \right|_{\gamma=0} \int_{\Omega_i} \epsilon_0 \left(E + \gamma\phi - b(E+\gamma\phi)^2 \right) \ \mathrm{d}\Omega_i \right]_{\gamma=0} \\ &= \int_{\Omega_i} \epsilon_0 (1 - 2bE)\phi \ \mathrm{d}\Omega_i.\end{aligned}$$

This now implies that

$$\frac{\delta Q_i}{\delta E} = \epsilon_0(1 - 2bE). \tag{3.6}$$

Using (3.5) and (3.6) we can derive the relation

$$\delta Q_i = \epsilon_0(1 - 2bE)\delta E = \frac{|\Omega_i|}{\eta}\epsilon_0(1 - 2bE)\delta V_i.$$

Let the capacitance of cell i be denoted by C_i. We define C_i by

$$C_i := \frac{\delta Q_i}{\delta V_i} = \frac{|\Omega_i|}{\eta}\epsilon_0(1 - 2bE). \tag{3.7}$$

Replacing ϵ_0 and E by their spatial averages, we have the approximation

$$\begin{aligned}C_i &\approx \frac{|\Omega_i|}{\eta} \left(\frac{1}{|\Omega_i|} \int_{\Omega_i} \epsilon_0 \right) \left(1 - 2b \frac{1}{|\Omega_i|} \int_{\Omega_i} E \, d\Omega_i \right) \\ &= \frac{|\Omega_i|}{\eta} \overline{\epsilon_0} \left(1 - \frac{2}{\eta} b V_i \right). \end{aligned} \tag{3.8}$$

We also have the approximation

$$\begin{aligned}Q_i &\approx \overline{\epsilon_0} \int_{\Omega_i} E - bE^2 \, d\Omega_i \\ &= \overline{\epsilon_0} \left(\frac{|\Omega_i|}{\eta} V_i - b \int_{\Omega_i} E^2 \, d\Omega_i \right) \\ &\approx \overline{\epsilon_0} \left(\frac{|\Omega_i|}{\eta} V_i - b\overline{E} \int_{\Omega_i} E \, d\Omega_i \right) \\ &= \overline{\epsilon_0} \left(\frac{|\Omega_i|}{\eta} V_i - \frac{b}{\eta} V_i \frac{|\Omega_i|}{\eta} V_i \right) \\ &= \frac{|\Omega_i|}{\eta} \overline{\epsilon_0} \left(V_i - \frac{1}{\eta} b V_i^2 \right). \end{aligned} \tag{3.9}$$

One can check that (3.8) is the derivative of (3.9) with respect to V_i, i.e., if we let C_i and Q_i denote the approximations derived in (3.8-3.9), then

$$dQ_i = C_i(V_i)dV_i,$$

which implies that

$$\frac{dQ_i}{dt} = C_i(V_i)\frac{dV_i}{dt}. \tag{3.10}$$

Consider now the time derivative of the charge, $\dot{Q}_i$. Using the definition of Q_i from (3.4) and using (3.1b) we can derive

$$\dot{Q}_i = C_i\dot{V}_i = \int_{\Omega_i} \partial_t(\epsilon E)\ \mathrm{d}\Omega_i = -\int_{\Omega_i} \mathrm{div}\ \Lambda\ \mathrm{d}\Omega_i = -\oint_{\partial\Omega_i} \Lambda^{\mathrm{T}} \cdot \hat{\mathbf{n}}\ \mathrm{d}l. \tag{3.11}$$

Let $\gamma(p)$ represent the edge of the triangle that separates cells i and p. Using the definition of the rotation matrix $R_{i,p}$ we introduced earlier, (3.11) will lead to

$$\begin{aligned} C_i\dot{V}_i &= \sum_{p\in\{k,j,l\}} -\oint_{\gamma(p)} \Lambda^T \hat{n}_{i,p}\ \mathrm{d}l \\ &= \sum_{p\in\{k,j,l\}} -\oint_{\gamma(p)} \Lambda^T R_{i,p}\hat{m}_{i,p}\ \mathrm{d}l \\ &= \sum_{p\in\{k,j,l\}} -\oint_{\gamma(p)} \left[R_{i,p}^T\Lambda\right]^T \hat{m}_{i,p}\ \mathrm{d}l. \end{aligned} \tag{3.12}$$

We define the currents flowing from cell i to cell p by $I_{i,p}$ and relate them to the magnetic field with

$$I_{i,p} := -\oint_{\gamma(p)} \Lambda^{\mathrm{T}} \cdot \hat{m}_{i,p}\mathrm{d}l. \tag{3.13}$$

Using this definition in (3.12) we can close the relationship between the time derivative of charge and current by

$$\begin{aligned} C_i\dot{V}_i &= -\sum_{p\in\{k,j,l\}} \oint_{\gamma(p)} \left[(I + O(\theta_{i,p}))\Lambda\right]^T \hat{m}_{i,p}\ \mathrm{d}l \\ &= -\sum_{p\in\{k,j,l\}} I_{i,p} + O(\theta_{i,p}) \end{aligned} \tag{3.14}$$

If we assume that in the limit of zero mesh spacing, the mesh asymptotically approaches a mesh of equilateral triangles, then in this same limit, $\theta_{i,p} \to 0$ and the above calculation becomes exact. In other words, an asymptotically equilateral mesh implies that the normal $\hat{n}_{i,p}$ will asymptotically coincide with the unit mesh vector $\hat{m}_{i,p}$. For the special cases of meshes made entirely of equilateral triangles or rectangular meshes, $\theta_{i,p} = 0$ and the relation holds exactly.

To obtain a similar relation for the rate of change of current, we can take the time derivative of the current $I_{i,p}$ and using (3.1a) we obtain

$$\dot{I}_{i,p} = \oint\limits_{\gamma(p)} -\frac{1}{\mu}(\nabla E)^T \hat{m}_{i,p} \, \mathrm{d}l.$$

We approximate μ by its average value on the segment $\gamma(p)$, i.e.,

$$\mu \approx \langle \mu \rangle := \frac{1}{|\gamma(p)|} \oint\limits_{\gamma(p)} \mu \, \mathrm{d}l.$$

Then

$$\dot{I}_{i,p} \approx -\frac{1}{\langle \mu \rangle} \oint\limits_{\gamma(p)} (\nabla E)^T \hat{m}_{i,p} \, \mathrm{d}l.$$

Let $\gamma(p)^*$ denote the midpoint of $\gamma(p)$. We use the midpoint rule to approximate the line integral and obtain

$$\dot{I}_{i,p} \approx -\frac{|\gamma(p)|}{\langle \mu \rangle} \Big[(\nabla E)^T \hat{m}_{i,p} \Big]_{\gamma(p)^*}.$$

The quantity in square brackets is the directional derivative of E in the $\hat{m}_{m,n}$ direction, which we approximate using a difference quotient:

$$\dot{I}_{i,p} \approx -\frac{|\gamma(p)|}{\langle \mu \rangle} \frac{E_p - E_i}{|\vec{m}_{i,p}|}.$$

Here E_m and E_n are the electric fields evaluated at the barycenters of cells m and n, respectively. We approximate these quantities by their cell averages, i.e., for $k \in \{i, p\}$,

$$E_k \approx \frac{1}{|\Omega_k|} \int\limits_{\Omega_k} E \, \mathrm{d}\mathbf{x} \approx \frac{V_k}{\eta}.$$

Using this approximation, we have

$$\dot{I}_{i,p} \approx \frac{|\gamma(p)|}{\langle \mu \rangle} \frac{V_i - V_p}{\eta |\vec{m}_{i,p}|}.$$

If we define the inductance of the inductors connecting cell i to cell p by

$$L_\gamma = L_{i,p} = \frac{\eta |\vec{m}_{i,p}|}{|\gamma(p)|} \langle \mu \rangle = \frac{\eta |\vec{m}_{i,p}|}{|\gamma(p)|} \frac{1}{|\gamma(p)|} \oint\limits_{\gamma(p)} \mu \, \mathrm{d}l, \tag{3.15}$$

we can relate the rate of change of current with the voltages as

$$L_\gamma \dot{I}_{m,n} \approx V_m - V_n. \tag{3.16}$$

3.2.1 Boundary conditions & forcing terms

Suppose the cell i shares an edge γ_f with the forced boundary $\mathrm{d}\Omega_f$. In order to handle the forcing we introduce a ghost cell i' which is a mirror reflection of the cell i such that they share the edge γ_f. We now compute the voltage at the cell i' by

$$V_{i'} \approx \frac{\eta}{|\Omega_{i'}|} \int\limits_{\Omega_{i'}} E \ \mathrm{d}\Omega_i \approx \frac{\eta}{|\gamma_f|} \int\limits_{\gamma_f} E \mathrm{d}l \approx \frac{\eta}{|\gamma_f|} \int\limits_{\gamma_f} f(x,y,t) \mathrm{d}l \tag{3.17}$$

The ghost cells introduce new currents which can now be used to produce the forcing as defined in (3.1c). The line integral can be handled with any choice of quadrature along γ_f.

If the cell i shares an edge γ_b with the boundary $\mathrm{d}\Omega_b$ then we need to impose the conditions as in (3.1d). We have from (3.11)

$$\begin{aligned} C_k \dot{V}_k &= - \int\limits_{\gamma_b} \Lambda \cdot \hat{\mathbf{n}} \ \mathrm{d}l - \int\limits_{\partial\Omega_k \setminus \gamma_b} \Lambda^{\mathrm{T}} \cdot \hat{\mathbf{n}} \ \mathrm{d}l \\ &= - \int\limits_{\gamma_b} \sigma E \, \mathrm{d}l - \int\limits_{\partial\Omega_k \setminus \gamma_b} \Lambda^{\mathrm{T}} \cdot \hat{\mathbf{n}} \, \mathrm{d}l \end{aligned} \tag{3.18}$$

The second line integral can be evaluated as before and we focus on the first integral. We have

$$\begin{aligned} \int\limits_{\gamma_b} \sigma E \mathrm{d}l &\approx \left[\frac{1}{|\gamma_b|} \int\limits_{\gamma_b} E \mathrm{d}l \right] \left[\int\limits_{\gamma_b} \sigma \mathrm{d}l \right] \\ &\approx \left[\frac{\eta}{|\Omega_k|} \int\limits_{\Omega_k} E \ \mathrm{d}\Omega_i \right] \left[\frac{1}{\eta} \int\limits_{\gamma_b} \sigma \mathrm{d}l \right] \\ &\approx V_k G_k \end{aligned} \tag{3.19}$$

where we define the conductance on the boundary cell by G_k given by

$$G_k := \frac{1}{\eta} \int\limits_{\gamma} \sigma \ \mathrm{d}l \tag{3.20}$$

Note that if $\sigma = 0$ then (3.1d) and (3.1) imply that $\nabla \cdot E = 0$, a perfectly insulating boundary condition. If $\sigma = \infty$ then 3.1d yields $E = 0$ on the boundary which represents a perfectly conducting boundary condition. In this chapter, we choose σ to approximate outgoing boundary conditions which are obtained as follows. Dotting (3.1a) with $\hat{\mathbf{n}}$ and using (3.1d) we obtain

$$\partial_t E + \frac{1}{\epsilon\sigma} \nabla E \cdot \hat{\mathbf{n}} = 0$$

At each edge where the outgoing boundary condition is to be applied the value of $\sigma(x,y)$ we choose to implement (3.1d) is

$$\sigma = \sqrt{\frac{\epsilon_0(x,y)}{\mu(x,y)}}.$$

3.3 Assembly and Solution

In the previous section we derived relations between the E, H fields and V, I respectively. We have also derived relations to physically discrete notions of capacitance, inductors and resistors and used the dual graph of the triangulation as the graph connecting the various components. This formulation is exactly the same as the one described in the previous chapter for random graphs.

In short, we may have already solved some nonlinear Maxwell Equation system already. While random graphs need not be planar, they may not correspond to an actual finite volume discretization. The dual graph of the interior of a triangulation will be a planar 3-regular graph. This is the big difference between the work done in contrast to the previous chapter. We can think of the current work as a specific case of a general problem solved in the previous chapter. Once we have considered a discretization of the domain and computed the inductance, capacitance and conductance values and know the dual graph there is no difference in the solution technique. The derivation of the discrete system follows very closely as described in Section (2.2) with some small changes to suit the application in mind.

The second-order discrete system is shown again below for reference:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[C\frac{\mathrm{d}V}{\mathrm{d}t}\right] + G\frac{\mathrm{d}V}{\mathrm{d}t} + \Delta V = V_{\text{in}}. \tag{3.21}$$

where

$$V_{\text{in}}(t) = B[\text{diag}(L)]^{-1}W(t) \tag{3.22}$$

$$\Delta = B[\text{diag}(L)]^{-1}B^{\mathrm{T}}. \tag{3.23}$$

From (3.8) the capacitance function can be written as

$$C_i(V_i) = C^0(1 - b'V_i), \tag{3.24}$$

where $C_i^0 := \frac{|\Omega_i|}{\eta}(\overline{\epsilon_0})_i \in \mathbb{R}$ is a constant for each cell i and $b' = 2/\eta$. This choice of capacitance function means that (3.21) features a quadratic nonlinearity.

Unlike in the previous chapter, we allow the generalize the forcing function to any smooth function which can be approximated using a Fourier series.

$$f(x, y, t) = \sum_{k=-\infty}^{\infty} a_k(x, y)e^{ik\omega t} + \text{c.c.} \tag{3.25}$$

In order to implement this input numerically we truncate the series to contain only L modes to obtain

$$f(x, y, t) \approx \sum_{k=-L}^{L} a_k(x, y)e^{ik\omega t} + \text{c.c.} \tag{3.26}$$

In order to implement this time-harmonic forcing we collect all the Fourier coefficients into k distinct vectors $A_k \in \mathbb{C}^N$ for $k \in \{-L \cdots L\}$. Here, $A_k[i]$ is the local coefficient of the Fourier series over the cell i as obtained from (3.17).

In order to solve the system we can use either the perturbative approach as described in Section (2.3.1) or the iterative method described in Section (2.3.2).

The difference between the iterative and perturbative method is minimal with respect to the solution accuracy as was demonstrated in Section (2.4.1). From an implementation standpoint the iterative solver is easier to implement, since it does not need us to keep track of the current order and incrementally build solutions. This helps ease setup and makes the code easier to maintain so we only implement the iterative method.

The simulation results described in Section (2.4.1) consisted of very small problems of size smaller than 200 nodes. Since our current application lies in solving Maxwell's Equations, we can no longer use such small problem sizes. As a result we use the PETSc framework to solve medium-scale problems. The implementation details are described in Appendix A. We first begin by explaining the details of the simulations conducted.

3.4 Simulations

In this section we explain the numerical simulations conducted and discuss the convergence of the numerical scheme that was developed. In past work we have solved (3.1) for the case $b = 0$ using structured rectangular meshes [10]. The derivation in Section (3.2) extends the work to general meshes. The numerical results we discuss are split into linear (L) and nonlinear (N) cases. We further split the problem into two tests for homogeneous (LH,NH) and inhomogeneous (LI, NI) medium cases. In all cases, the domain Ω consists of a square $[0, 1] \times [0, 1]$ and we set $\eta = 1$. The forcing is applied to the left boundary and the general form is

$$f(0, y) = e^{-a(y-0.5)^2} e^{i\omega t} + \text{c.c.} \tag{3.27}$$

where a is a constant equal to 150 in all simulations. This input is a special form of (3.26) where we have only one k value which equals ω. For the linear cases there are no higher harmonics generated. Even with a single input frequency the nonlinear case generates higher harmonics through the product $\alpha_l \alpha_{k-l}$ as seen in (2.29).

For the homogeneous test cases (LH,NH), we set $\mu = 1$ and $\epsilon_0 = 9$ throughout the domain. For the inhomogeneous case (LI,NI) we consider a medium modeled after a photonic crystal device [50]. This medium consists of a circular array of inclusions with a linear defect. The permittivity is $\epsilon_0 = 9$ outside the circular inclusions and $\epsilon_0 = 1$ inside. The permeability of the medium is left unchanged at $\mu = 1$. The circular inclusions have a radius of $1/40$ and are separated from each other with a distance of $1/10$. The linear defect is obtained by removing the inclusions along the $y = 0.5$ axis.

Mesh Generation. For all decompositions of the domain Ω we use triangles. The algorithm does not constrain us to use any particular discretization method and we could have used quadrilaterals, hexagons or a combinations of any of then. Since the mesh distribution cannot be done for heterogeneous meshes we use triangulations, which are more flexible than quadrilateral meshes. Each test case is split further into two cases consisting either entirely of equilateral triangles (E) or by unstructured triangles (U). The use of unstructured triangles provides great flexibility in adapting the mesh to curved material inhomogeneities. Note that the use of unstructured mesh leads to (3.14) not being satisfied exactly. We compare the solutions in each case for two mesh types. The unstructured meshes are generated so as to force small triangulations near regions of high contrasting material properties for the inhomogeneous cases (LIU, NIU). For the unstructured homogeneous case (LHU, NHU) we

build meshes that contain smaller triangles around $y = 0.5$, the axis along which the forcing propagates. All unstructured meshes we generated used DistMesh [69]. The meshes range from size 1K-500K cells for the equilateral case and 1K-300K for the unstructured case.

Choice of parameters. Once the material properties are decided we need to select the strength of nonlinearity b' in (3.24) and fundamental frequency ω in (3.27). For the simulations each instance of the problem is solved for $b' \in \{0.005, 0.01\}$ for $\omega \in \{0.2, 0.4, 0.6, 0.8\} * (2\pi)$. We use α to denote the unscaled values of ω corresponding to $\alpha = \{0.2, 0.4, 0.6, 0.8\}$. In all computations we solve for the first 10 Fourier coefficients $\{\omega, 2\omega, \ldots, 10\omega\}$.

Error computation. The case $b = 0$ leads to the linear Maxwell equations and the exact solution for this case with homogeneous medium was used to compute the error in past work [10]. We use the same exact solution here to compute the error for the linear homogeneous test case. For all other cases we do not have a closed-form solution. In order to derive the convergence order of the method, we use the solution obtained on the finest grid as the reference solution corresponding to a particular choice of material properties along with b and ω. Since successive meshes are not obtained from refining coarse meshes we cannot directly approximate the error of a coarse mesh with respect to the fine mesh. One problem we face if we try to recursively refine a mesh is the degradation of mesh quality which is essential for our method. Let S_f represent the solution obtained from the fine mesh and S_c represent a coarse mesh solution. Note that for both cases S_f and S_c are composed of Fourier coefficients for 10 modes. The number of Fourier coefficients for each mode of the fine solution is greater than the coarse solution since the finer mesh has more triangles. The procedure we use to compute the error is as follows. Let R correspond to a uniform square mesh over Ω. Let $R_{l,m}$ correspond to the mid-points of each square in R where the indices l, m are used to enumerate the rectangles in the x, y directions respectively. We interpolate S_f onto the grid R and perform a similar interpolation for S_c. The interpolated solutions on R are denoted by $\tilde{S}_f$ and $\tilde{S}_c$ respectively where both solutions now contain 10 modes at the same points. Let $\tilde{S}_*(p, k)$ represent the Fourier coefficient of the solution at point $p \in R$ corresponding to mode k. If R is decomposed into squares of sides δ_x then the error between the fine and coarse solution within one single square of side δ_x can now be computed as

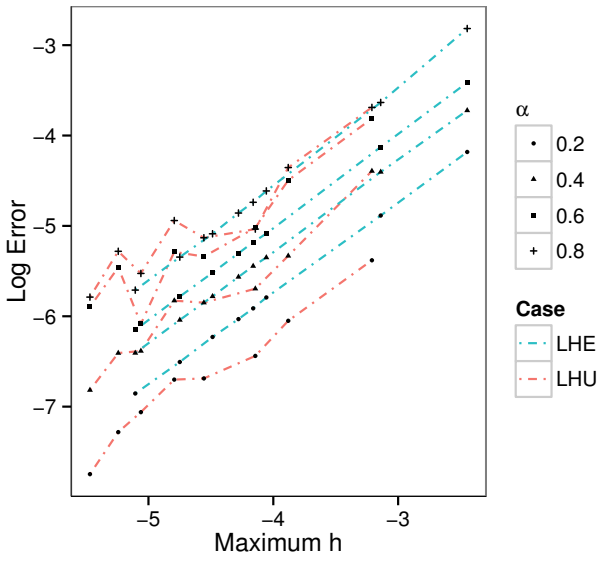
$$E = \|(\tilde{S}_f - \tilde{S}_c)\delta_x\|. \tag{3.28}$$

By summing over the entire region and over all the harmonics we obtain the full error.

3.4.1 Software

For all computations, we use the PETSc library [6]. The PETSc library allows us to solve the problems in a distributed computing environment. The mesh is distributed across all processors and computations of the parameters C_0, L, G and V_{in} are done locally on each processor. Rather than compute the incidence matrix B, we instead form the matrix Δ directly since we do not need the incidence matrix while solving the second order system. For the iterative procedure in (2.29), we need to solve k linear systems until convergence. For the results in this paper, we use a direct solver to obtain a LU decomposition of each matrix $\mathcal{L}(k\omega)$. We use the `SuperLU_DIST` package [60], to obtain this decomposition. A more detailed explanation of the implementation along with further run-time details provided in the Appendix A.

Figure 3.2: Convergence plot for the linear simulations.
The x-axis corresponds to the logarithm of the maximum edge length among all triangles used. The error is computed with respect to the exact solution. The red lines correspond to LHU while the blue lines correspond to the LHE test cases. The convergence for the equilateral triangulations is monotonic while for unstructured meshes it is not.



3.4.2 Convergence results

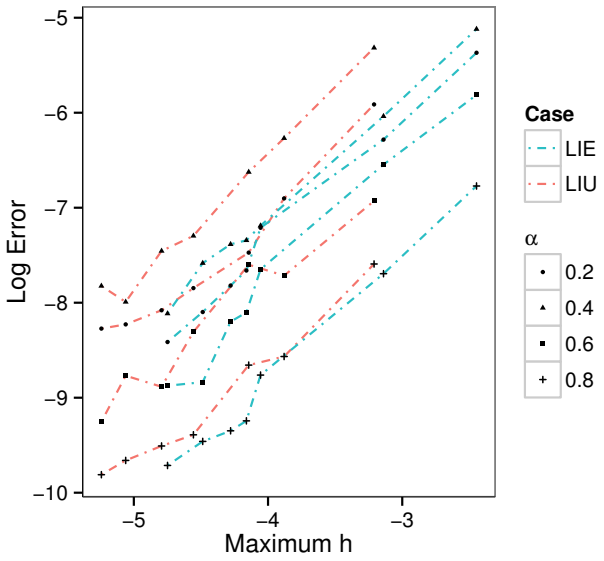
We now discuss the convergence of the test problems using the setup described before. For each set of tests we provide a convergence plot along with the slope indicating the rate of convergence. For the convergence plots we have plotted the error computed using (3.28) as a function of the maximum edge length among all the triangles used in the mesh. For each test case the results are shown for various values of α. We discuss each case separately below.

Linear homogeneous case. The linear homogeneous case is the only one for which we have an exact solution available. This exact solution at the centroids of the triangles is used as a reference solution. These solutions were obtained in previous work [13]. There are two test cases (LHE and LHU) which fall under this criterion since the discretization does not play any role in the exact solution. The log-log error plot for these two cases for different values of α are shown in Figure 3.2. The x-axis corresponds to the logarithm of the maximum edge length of the triangles and the y-axis plots the logarithm of the error obtained for each case. The red lines correspond to convergence plots for unstructured meshes and blue lines correspond to convergence obtained for equilateral triangles. We also provide the corresponding rate of convergence in Table 3.1. The drop in the rate of convergence for unstructured meshes can be attributed to the error introduced in Equation 3.14. The equilateral triangulation does not have this problem due to the equation being satisfied exactly and hence converges to first order exactly. This highlights the importance of generating high quality meshes. An alternative is to use only regular meshes such as the equilateral triangulation mesh or a square mesh, thereby simplifying the convergence process considerably.

Table 3.1: Order of convergence for linear homogeneous problem.
The error computation for the linear homogeneous cases is done with respect to the exact solution. The LHE case corresponds to the use of equilateral triangulation while the LHU uses unstructured triangulations.

α	LHE	LHU
0.2	1.0054	0.8320
0.4	1.0113	0.8454
0.6	1.0298	0.7995
0.8	1.0877	0.7511

Figure 3.3: Order of convergence for linear inhomogeneous problem.
Log-Log error plot for linear inhomogeneous test cases. The red lines correspond to LIU while the blue lines correspond to the LIE test cases. The error computations are done with respect to the solutions obtained from the finest mesh.



Linear inhomogeneous case. For the inhomogeneous test cases we do not have an exact solution. In order to compute the order of convergence, we use the finest mesh solution as an approximation to the exact solution and compute the errors as mentioned in Section 3.4. The log-log error plots are shown in Figure 3.3. The corresponding rate of convergence is tabulated in Table 3.2. From the Table 3.2 we see that even with the use of unstructured meshes we obtain a first order rate of convergence.

Nonlinear test cases For the nonlinear cases, we solve for the first 11 harmonic frequencies $\{\omega, 2\omega, \ldots, 10\omega\}$. The strength of the nonlinearity b' in (3.24) is set to either 0.005 or 0.01 to test for effects of the parameter on the method. The convergence plots for $b' = 0.005$ is shown in Figures 3.4. Figure 3.5 shows the corresponding plot for $b' = 0.01$. In each figure we plot the error on the y-axis as a function of the maximum edge length over all triangles in the triangulation for both the equilateral triangulation case as well as the unstructured mesh case. From both these figures we see that the error drops as we reduce the maximum edge

Table 3.2: Order of convergence for linear homogeneous problem.
The error computation for the linear inhomogeneous cases is done with respect to the solution obtained from the finest mesh. The LIE case corresponds to the linear inhomogeneous solutions based on equilateral meshes while the LIU case uses unstructured meshes. The order of convergence is dependent on the underlying mesh and the methodology used to obtain the error.

α	LIE	LIU
0.2	1.3206	1.1764
0.4	1.2573	1.3190
0.6	1.3953	1.1273
0.8	1.3204	1.0944

length over all triangles. In Table 3.5, we tabulate the order of convergence for all the test cases obtained by fitting a line of best fit for convergence plots. The order of convergence for all cases is close to 1 as expected. The results show that having an equilateral triangulation gives consistent results while the convergence of unstructured meshes is affected by the triangulation with respect to monotonicity.

For the nonlinear case the fixed point algorithm will be used and we would like to ensure that the number of iterations required to obtain convergence does not grow excessively as we increase the size of the problem. In order to verify this we also provide the number of iterations required for the iterative solver to converge. The convergence is tested by comparing two consecutive solutions and we declare convergence when the difference is below 10^{-10}. Table 3.3 shows the number of iterations required for the fixed point solver to converge for the equilateral triangulations and the corresponding Table 3.4 shows the iterations for the unstructured meshes. From the Tables, we see that the number of iterations required increases slightly as the forcing frequency increases. The number of iterations required for the solver to converge is independent of the size of the problem. The maximum number of iterations for the homogeneous problem is 14 while the inhomogeneous problem takes less than 13 iterations to reach convergence.

3.5 Conclusion

We have developed a new numerical method for the solution of nonlinear Maxwell's equations that can handle material inhomogeneity and is independent of the underlying mesh, i.e., can handle triangular, quadrilateral or hybrid conforming meshes. The conversion of the time-domain problem to a Fourier-domain problem enabled us to obtain the steady-state solution of the system without needing to step forward in time. While we only demonstrated results with a single forcing frequency α, the algorithm enables solutions for any harmonic forcing by breaking it into constituent frequencies. This further allows us to use the method for triangular/sawtooth forcing functions. While the algorithm is built only for a quadratic nonlinearity, the iterative procedure we used does not have this constraint. In (2.29), the product $\alpha_l \alpha_{n-l}$ is a convenient product we obtained for the convolution due to our choice of nonlinearity. If we instead choose any other bounded nonlinearity, while it may not be possible to obtain such a simple product, one can always use a corresponding quantity in the time-domain and then convert it back into the Fourier domain. This makes the method greatly flexible in its ability to handle complex nonlinear functional dependence of the

Table 3.3: Number of iterations for fixed point solver to converge for equilateral triangulation.

We see that the number of iterations to convergence is almost constant as a function of mesh size. An increase in frequency causes the number of iterations to increase as does the magnitude of the nonlinearity b′. The convergence of the solver is checked by comparing two consecutive solutions and ensuring that the difference is smaller than 10^{-10}. The maximum number of iterations is less than 14 in all cases.

type	b'	α	1K	3K	23K	28K	36K	54K	92K	187K
homogeneous	0.005	0.2	5	5	5	5	5	5	5	5
		0.4	6	6	6	6	6	6	6	6
		0.6	6	7	6	6	6	6	6	6
		0.8	8	7	7	7	7	7	7	7
	0.01	0.2	5	5	5	5	5	5	5	5
		0.4	7	7	7	7	7	7	7	7
		0.6	8	9	8	8	8	8	8	8
		0.8	9	10	9	10	10	10	10	10
inhomogeneous	0.005	0.2	4	5	5	5	5	5	5	5
		0.4	6	6	6	6	6	6	6	6
		0.6	7	7	8	8	8	8	8	7
		0.8	8	8	8	9	9	8	8	8
	0.01	0.2	5	5	5	5	5	5	5	5
		0.4	7	7	8	7	7	7	7	7
		0.6	9	9	10	10	10	10	10	10
		0.8	9	14	12	13	12	12	11	11

permittivity on the electric field.

To the best of our knowledge, we do not know of any other method available which can solve for the steady-state solution while providing scalability at the same time. The current work is a first step in this direction. Given the feasibility of our approach there are many avenues for further research in order to extend our work.

If unstructured mesh are not required then by using either equilateral triangulations or rectangular meshes our algorithm can yield results without the introduction of errors in (3.12). On the other hand if the use of unstructured meshes is necessary for a particular application then a good quality mesh generator becomes a crucial part of the method. Most mesh generators for partial differential equations are built with the finite element procedure in mind. For such methods, as long as the minimum angle is above a certain threshold the mesh is considered acceptable. For our case, we would really want the meshes to be as close to the optimal meshes. For the triangular mesh case, by optimal we mean meshes such that the normal from a cell center to its neighbor should lie in the same direction as the line joining the centroids of the triangles. Rather than rely solely on mesh generation software an alternate approach is to use a mesh optimization software like Mesquite [21]. A simple Laplacian smoothing of the mesh with local geometry already built in may be sufficient. The Mesquite package can be used with the Message Passing Interface thereby allowing much larger meshes to be used. Also, the package is flexible enough to provide a

Table 3.4: Number of iterations for fixed point solver to converge for unstructured meshes.

The number of iterations for convergence to within 10^{-10} is independent of the size of the mesh. For higher values of forcing frequency the number of iterations needed increases. There is no noticeable difference between the homogeneous and inhomogeneous test cases. As we increase the strength of the nonlinearity the number of iterations needed for the fixed-point solver to converge increases.

type	b'	α	3K	12K	21K	48K	76K	135K	196K	306K
homogeneous	0.005	0.2	5	5	5	5	5	5	5	5
		0.4	6	6	6	6	6	6	6	6
		0.6	7	7	7	6	7	6	6	6
		0.8	8	8	8	7	7	7	7	7
	0.01	0.2	5	5	5	5	5	5	5	5
		0.4	7	7	7	7	7	7	7	7
		0.6	10	9	8	8	8	8	8	8
		0.8	13	12	11	10	10	10	9	9
inhomogeneous	0.005	0.2	5	5	5	5	5	5	5	5
		0.4	6	6	6	6	6	6	6	6
		0.6	7	8	8	7	8	7	7	8
		0.8	8	9	9	9	9	8	8	8
	0.01	0.2	5	5	5	5	5	5	5	5
		0.4	7	8	8	8	7	7	7	7
		0.6	10	11	11	9	10	10	10	10
		0.8	12	12	13	12	12	12	12	12

specific optimization function with which to alter the mesh. Because we know the criterion our meshes optimally should satisfy, this approach might work very well.

The graph Laplacian which is the dual graph of the triangulation contains only 3 non-zeros for each row regardless of the mesh size. This is an extremely sparse matrix and one could use iterative schemes to solve the fixed point (2.29). The direct solvers are much faster for the problem sizes that we considered here. In the future if we desire to solve problems of much larger sizes it might be beneficial to consider iterative solvers. The linear systems we need to solve are of the form

$$A[k] = (2\pi ik)^2 C_0 + (2\pi ik) G_i + \Delta.$$

The matrix Δ is a symmetric positive definite matrix. The diagonal component involving the capacitance and the conductance change depending on the harmonic we are interested in solving. PETSc allows one to define a shell matrix which could be very beneficial in this case. Since iterative solvers like GMRES would only require one to compute matrix-vector products all matrices can reuse the Laplacian matrix Δ without creating further copies. This formulation could help scale the problem much further depending on our needs. Our current simulations use a direct solver which enables us to solve for a large number of harmonics. When using an iterative solver care must be taken to ensure that the solver uses finer tolerances.

Figure 3.4: Convergence plot for nonlinear homogeneous test cases with $b' = 0.005$. *Log-Log error plot for nonlinear inhomogeneous test cases with $b' = 0.005$. The red lines correspond to NHU while the blue lines correspond to the NHE test cases.*

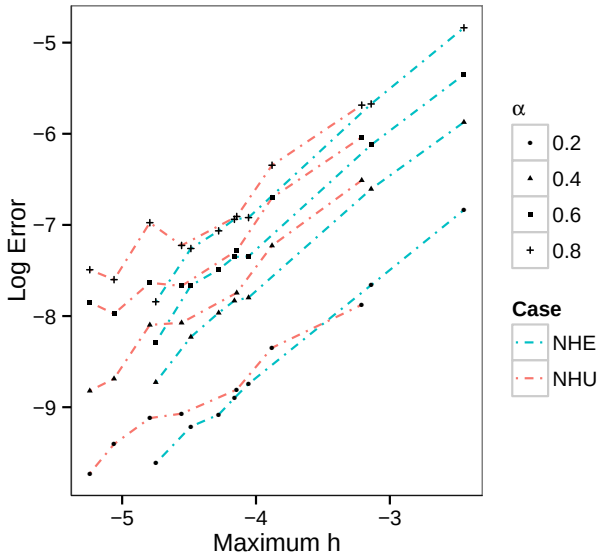


Table 3.5: Convergence orders for nonlinear homogeneous case (NHE and NHU) with $b' = 0.005$ and $b' = 0.01$.

	$b' = 0.005$		$b' = 0.01$	
α	NHE	NHU	NHE	NHU
0.2	1.198078	0.868255	1.198082	0.868295
0.4	1.207757	1.119995	1.207824	1.119862
0.6	1.222071	0.947510	1.222142	0.947382
0.8	1.258230	0.902042	1.258885	0.902255

The system of matrices we need to solve are complex symmetric and not Hermitian. There are only some algorithms known to efficiently solve systems of such type [7], [40]. Currently, however, no large-scale implementations exist. Our current implementation uses the complex matrix directly. Further experiments could be conducted by converting the complex system into an equivalent real system. Such transformations change the eigenvalue distribution and hence might be hard to solve, but merit further exploration.

Figure 3.5: Convergence plot for nonlinear homogeneous test cases with $b' = 0.01$. *The red lines correspond to NHU while the blue lines correspond to the NHE test cases.*

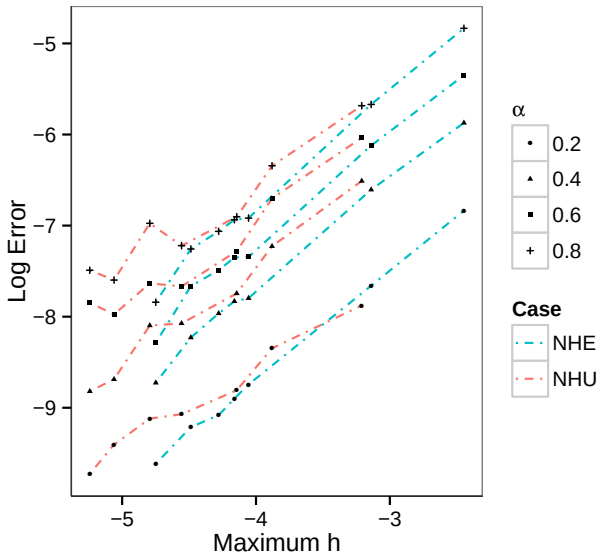


Table 3.6: Convergence orders for nonlinear inhomogeneous case (NIE and NIU) with $b' = 0.005$ and $b' = 0.01$.

	$b = 0.005$		$b = 0.01$	
α	NIE	NIU	NIE	NIU
0.2	1.3250	1.1028	1.3250	1.1028
0.4	1.3611	1.1136	1.3612	1.1137
0.6	1.3446	1.1404	1.3450	1.1404
0.8	1.3521	1.1612	1.3515	1.1617

Figure 3.6: Log-Log error plot for nonlinear inhomogeneous test cases with $b' = 0.005$.

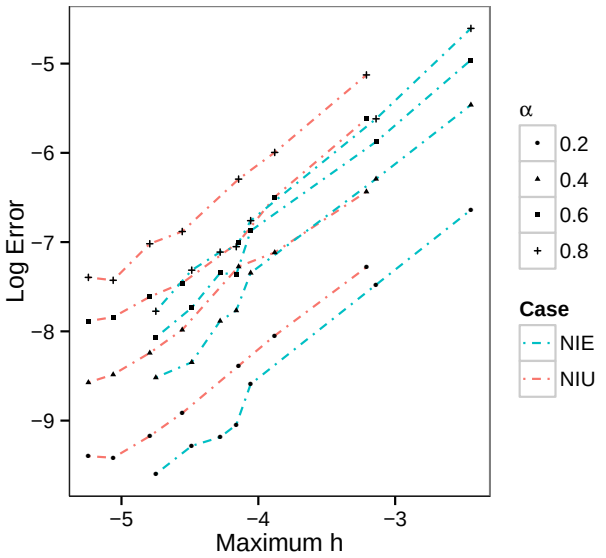
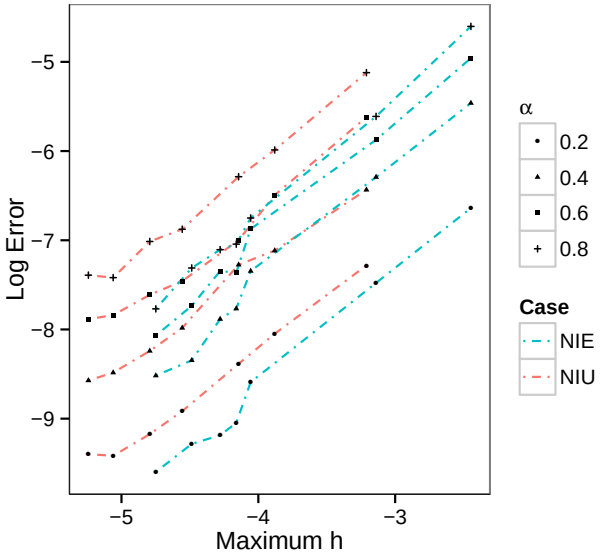


Figure 3.7: Log-Log error plot for nonlinear inhomogeneous test cases with $b' = 0.01$.



Chapter 4

Quantile Regression Tree

4.1 Introduction

Decision trees are one of the most widely used methods in data mining. Trees enjoy various advantages, among which are interpretability, ability to handle different types of predictors, and ability to handle missing data. Fitting the tree to data is typically performed using recursive partitioning algorithms, which can be efficient and scalable. Tree methods in modern use go back to the works of Breiman et al. [20] and Quinlan [71], which have spawned numerous variations that attempt to improve the predictive power of the model and/or the efficiency of the fitting algorithm.

When tree models are applied to regression problems, the most widely used splitting criterion employs an ordinary least squares (OLS) loss function. Trees built with this OLS loss may lead to unsatisfactory performance when, for example, the response variable contains large outliers. To understand why, suppose we have a number of instances at a node that we are about to split, and that some of these instances contain a few large outliers in the response variable. Assume there is no split that causes all of the outliers to be separated from the rest of the instances. Then, whatever split we choose will lead to one or more outliers at a child node. If these children are leaf (i.e., terminal) nodes, then the OLS tree says the predicted value should be the *mean* of the response variable for the instances associated with the leaf. It is this mean that will be sensitive to the presence of outliers.

To generate trees that are more robust to outliers in the response variable, Breiman et al. [20, Chap. 8] suggested a splitting criterion based on least absolute deviation (LAD). In this approach, when we arrive at a leaf node, the predicted value will be the *median* of the response variable for the instances associated with the leaf. Consistency of LAD trees has been proven [24].

In this chapter, we generalize Breiman's framework to arrive at *quantile trees*. The splitting criterion uses a tilted absolute value loss function (4.2) that, in a natural way, allows us to develop a model for the τ-th quantile of the response variable. The LAD *median tree* is included as a special case. There are three main reasons for exploring quantile trees. First, we hypothesize that quantile trees should share with LAD trees the property of robustness with respect to outliers. They also give us a natural way to deal with skewed errors. Second, linear quantile regression has proven useful in many applications, especially when one is interested in features of the conditional density $f(Y|\mathbf{X})$ that go beyond the conditional expectation $E[Y|\mathbf{X}]$. Here Y is the response variable and $\mathbf{X}$ contains the predictors. Quantile

trees should enable us to probe similar features of the conditional density without making parametric assumptions on the form of the model. Finally, we believe one of the main reasons that both LAD and quantile trees have not been more widely adopted is that the associated splitting criteria are viewed as computationally inefficient when compared to OLS trees. It is of interest to try to design a more efficient, scalable tree algorithm that uses the loss function (4.2).

Let us briefly discuss prior work where trees have been employed to provide nonparametric estimates of conditional quantiles. In [25], a polynomial is fitted to the instances in each terminal node. In this scheme, the node splitting criterion involves p-values from linear quantile regression. This yields a piecewise polynomial model for the conditional quantile function.

In [62], a forest of OLS trees is modified in one way: the prediction at a leaf node is not just the mean or median of the response values—instead, we compute the empirical probability mass function (PMF) of response values. If there are n trees in the forest, then a new covariate x yields n leaf PMF's, which are averaged and used to compute a single empirical CDF. Numerically inverting this CDF yields the quantile estimate. This approach is called "quantile regression forests."

Next, [68] uses both the OLS loss and the tilted absolute value loss used here. However, the estimated conditional quantile at a leaf node is calculated by adding an offset to the conditional expectation. This approach only works when making strong distributional assumptions on the model errors; in this case, it is assumed that they are i.i.d. homoscedastic normal.

Moving beyond LAD and quantiles, [39] has shown that splitting criteria based on robust M-estimators may lead to more robust trees.

Our work differs from all prior work discussed above in one or both of the following ways: (1) our splitting criterion is based on a tilted absolute value loss, and (2) our predicted value at a leaf node is the empirical quantile of the response values at the leaf. Using this approach, our main contribution is an algorithm which attains the same complexity as OLS trees.

4.2 Preliminaries

4.2.1 Decision trees

Decision trees are non-parametric models built for supervised learning problems. In supervised learning the objective is to understand and build a model for a desired response variable Y. The usual setup involves a set of predictor variables X where X is an $N \times p$ matrix containing all the inputs that were observed. Each row of the matrix corresponds to a single sample point. The number of samples is denoted by N and there are p predictors. For row i we observe the output Y_i. Table 4.1 provides a small sample of the white wine data set. The complete data set consists of 11 predictor variables and 1 response variable, and there are 4198 sample points. We only show 6 of the predictor variables and 4 sample points. The predictor variables shown include the *fixed acidity, volatile acidity, density, pH and sulphates*. The response variable is the quality of the wine. The quality provided in the Table is determined by wine tasters and the values lie in the range [0,10].

Decision trees use a simplifying heuristic to construct the model. The decision tree

Table 4.1: Sample data from white wine data set.
For clarity we only show 6 predictor variables and the response variable in this case is called quality. The aim is to predict the quality of the wine for an unknown case given the predictor variables.

fixed acidity	volatile acid	density	pH	sulphates	alcohol	quality
7.0	0.27	1.00	3.00	0.45	8.8	6
6.3	0.30	0.99	3.30	0.49	9.5	6
8.1	0.28	0.99	3.26	0.44	10.1	6
7.2	0.23	0.99	3.19	0.40	9.9	6

obtained for the white wine data set is shown in Figure 4.1. Starting from the entire data set we recursively partition the data set. At the root node of the tree which includes the entire data set we ask the question: If we had to split the data into two parts based on the value of a single predictor variable which one would we use and at what value would we create the split? If we know how to find such a predictor variable and the value to create our split we can then continue recursively until needed. While this heuristic of selecting a single predictor variable at each node might seem very loose, decision trees in fact produce highly interpretable and accurate prediction models. The exact algorithm used to create each split for both the OLS trees and the quantile trees is described in the next Section.

For the decision tree shown in Figure 4.1 at the root node the variable chosen by the algorithm is *alcohol* and the split value is 11. This means that we split the data set into two mutually exclusive parts. The left branch contains all the sample points in the data set which have alcohol content less than or equal to 11, while the right branch will contain the rest of the sample points. Next, from the sample points which had an alcohol content of less than 11, we split the points based on the volatile acidity of the samples. If the volatile acidity is greater than 0.25 then we move such samples to the right and carry on recursively. However, if the volatile acidity of the samples is less than or equal to 0.25 then we do not split the samples further. Instead we use the mean or median value of the quality among all samples which fall under this case as the model response value. In the sample decision tree shown we would predict the quality of the wine to be 5.4.

4.2.2 Decision tree algorithm

Given p predictors $X_1, \ldots, X_p$ and the response variable Y, tree models minimize the node deviance

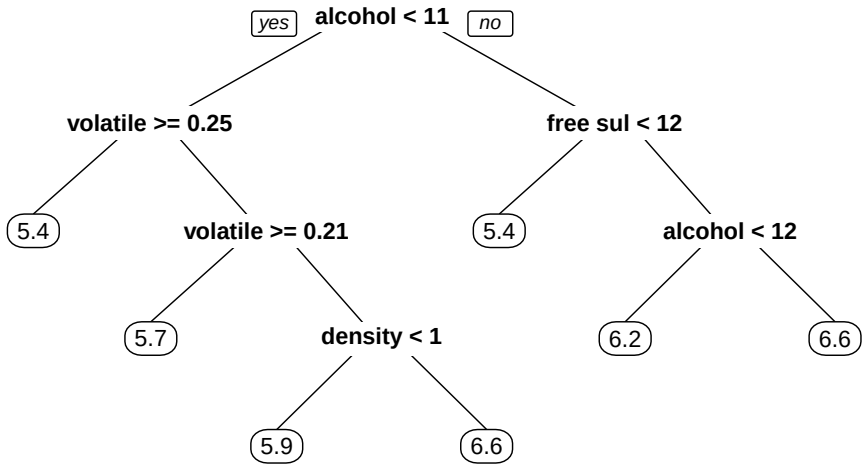
$$S_{\text{node}} = \sum_{i \in \text{node}} \rho_{\text{model}}(y_i - \theta_{\text{model}}), \tag{4.1}$$

where ρ_{model} is the loss function and θ_{model} is the associated M-estimator. The M-estimator for a squared loss criterion becomes the mean of the sample while the absolute deviation criterion leads to the median as the associated M-estimator. We define

$$\rho_{\text{OLS}}(x) = x^2,$$

$$\rho_{\text{QT}}(x) = \begin{cases} (\tau - 1)x & x < 0 \\ \tau x & x \geq 0. \end{cases} \tag{4.2}$$

Figure 4.1: Decision tree obtained for the white wine data set.
The root node splits the data using the amount of alcohol content. If the alcohol content is less than 11% then we move to the left branch. Otherwise we move to the right branch. Once we have decided on which one of the branches to take we repeat the procedure until we reach the leaf nodes. The leaf nodes do not split the data set further but rather provide an estimate for the quality of the wine.



The OLS model is the traditional deviance based splitting model and QT is a quantile based model which is the focus of this chapter.

Let $\{i_1, \ldots, i_\nu\} = \{i : i \in \text{node}\}$ and let $\mathbf{y} = (y_{i_1}, \ldots, y_{i_\nu})$, the vector of response values associated with the node i. The optimal M-estimator θ_{model} associated with the two models defined in (4.2) is given by

$$\begin{aligned}\theta_{\text{model}} &= \underset{y^*}{\operatorname{argmin}} \sum_{i \in \text{node}} \rho_{\text{model}}(y_i - y^*) \\ &= \begin{cases} \mu(\mathbf{y}) & \text{model} = \text{OLS} \\ F_{\mathbf{y}}^{-1}(\tau) & \text{model} = \text{QT}, \end{cases}\end{aligned}$$

where $\mu(\mathbf{y})$ is the mean of $\mathbf{y}$ and $F_{\mathbf{y}}^{-1}(\tau)$ is the empirical τ-th quantile of $\mathbf{y}$.

Once the model and the loss function ρ have been chosen, the tree can be built using

a recursive algorithm. Fix a threshold $\delta > 0$ and positive integers n_m and n_c. The value of n_m decides the minimum number of sample points in $\mathbf{y}_i$ required to attempt a split at node i. The variable n_c is the minimum number of sample points required at any node. The constant δ is a user defined value to stop splitting nodes unless they provide sufficient decrease. Initially, let all instances be contained in a single node. For this single node obtain S_{root} from (4.1). Then the recursive splitting algorithm can be defined as follows.

1. If $|\text{node}| \leq n_m$, then label node as a *leaf*. Go to step 6.

2. Calculate the node deviance S_{node} using (4.1). When we split on a fixed predictor X_i, let l and r denote the sets of instances that branch to the left and right, respectively. Let L and R denote the collection of all distinct sets l and r that are obtained by considering all binary splits over all predictors X_i. Set $\phi(l, r) = S_{\text{node}} - (S_l + S_r)$ and calculate
$$(l^*, r^*) = \underset{l \in L, r \in R}{\text{argmax}}\ \phi(l, r), \qquad \Delta_{\text{node}} = \phi(l^*, r^*).$$

3. If $\Delta_{\text{node}} \leq \delta S_{root}$, then label the node as a *leaf*.

4. If $|l^*| \leq n_c$ or $|r^*| \leq n_c$, then label the node as a *leaf*.

5. If this node has not been labeled as a *leaf*, then delete this node and split it by labeling l^* as a node and r^* as a node.

6. If all nodes are labeled as leaves, then STOP. Else, move to the next node. Go to Step 1.

The condition in Step 3 avoids the cost of building deep trees which will be pruned, thereby decreasing the cost of building the tree.

The most time-consuming step in the tree building procedure above is Step 2. To compute the values of $(l*, r*)$ we need to consider every possible split and the most efficient way is to first sort the values. If the node has n elements this operation takes $O(n \log n)$ time. OLS trees can then compute $(l*, r*)$ in linear time using an online algorithm by updating the values of the mean and $\phi(l, r)$ across all possible splits. In order to achieve the same time complexity for the QT model we now present our algorithm. Note that maximizing $\phi(l, r)$ is equivalent to minimizing $S_l + S_r$ and we will use this to choose the best split.

4.3 Qtree algorithm

The recursive partitioning algorithm defined in the previous section differs between the two models defined in (4.2) only at Step 2. In order to explain our algorithm we focus only on finding the optimal split at one node for one predictor variable. Given such an algorithm we can then obtain the best predictor variable among all the predictor variables by simple comparison.

The starting point are two vectors X_0 and Y_0 which hold the values corresponding to a predictor and response at any particular node of the tree. We first find the sort order denoted by *ord* for the vector X_0 and obtain two vectors $X = X_0[ord]$ and $Y = Y_0[ord]$. In order to break the problem of computing S_l and S_r into an online algorithm we will need to compute the τ'th quantile in an online fashion. What we mean by an online algorithm

here is as follows: suppose we have already computed the value of S_l for a certain number of points. Now we add a new point into the left child. We would like to update S_l without having to sort all the values again and then computing the value of S_l from scratch.

In general, computing the quantile in an online manner is considered expensive since it requires at least a partial sort of the data. However, in the present case, if we are able to obtain the new quantile for each insertion in logarithmic time then the cost incurred overall is the same order as sorting which we incur even while using OLS trees. We use this already incurred cost to update the quantile without worsening algorithmic complexity.

The update of quantiles should be done with regards to efficiency, requiring the use of certain well-studied data structures. The following two requirements on a data structure will be essential to our algorithm:

- Insertion/deletion time of a value in the structure should be at most $O(\log n)$.
- Finding the maximum/minimum should be at most $O(\log n)$.

Any data structure that permits the above bounds will work for our algorithm. The current work will demonstrate it using a heap based priority queue. Priority queues built with heaps are a very well-studied data structure offer $O(1)$ access to the priority element and $O(\log n)$ time for insertion and delete operations. The priority element depends on our choice of ordering and we will use both min and max heaps.

We use two heaps denoted by H_{low} and H_{high}, which correspond to max and min heaps, respectively. The max heap H_{low} allows access to the maximum element of the heap (denoted by $H_{low}[top]$) in $O(1)$ time and the min heap H_{high} allows access to the minimum element of the heap (denoted by $H_{high}[top]$) in $O(1)$ time. We denote by N_P the first P points in Y. We can easily obtain the τ'th quantile if we place $N_P^- = \lceil (N_P - 1)\tau \rceil$ points in H_{low} and the remaining $N_P^+ = N_P - N_p^-$ points in H_{high}. The τ'th quantile can now be computed as

$$q_P = H_{low}[top] + [H_{high}[top] - H_{low}[top]]\,(\tau(N_P - 1) - (\lceil (N_P - 1)\tau \rceil - 1)). \tag{4.3}$$

The value of the quantile q_P is a function of both the set of points P and the desired quantile τ, but the choice of quantile is fixed at the start of the algorithm and does not change. For notational clarity refer to q as a function of P only. Define $R = \{P \cup s\}$ where $s = Y[N+1]$ is the new point we wish to insert. In order to update the quantile when we insert s, we first check if $s \leq H_{low}$. If this is true we insert s into H_{low}, else we insert it into H_{high}. Once inserted we need to ensure that there are exactly N_r^- points in H_{low} and N_r^+ points in H_{high} such that the τ'th quantile can still be obtained using (4.3). If this condition is violated, we need to pop the top value from H_{low} and insert it into H_{high}, or vice versa, to ensure the condition holds. This provides us with an online algorithm for updating the quantile when we insert one value at a time. Using (4.3) we can now obtain the updated quantile q_R. In this manner, starting with a single point $Y[1]$ inserted into H_{low} we can keep updating the quantile until we have inserted all the points.

We now describe how we can update $\rho(x)$ when we insert a new point using the previous value. We denote by P^-, P^+ the set of points in H_{low}, H_{high} respectively. Equation (4.2) can be written as

$$QAD_{q_P,P} = \tau \sum_{P^+} (y_i - q_P) + (\tau - 1) \sum_{P^-} (y_i - q_P). \tag{4.4}$$

where $QAD_{q_P,P}$ corresponds to the general expression $\rho(x)$. We can now simplify this expression to obtain

$$\begin{aligned} QAD_{q_p,P} &= \tau \sum_{P^+}(y_i - q_p) + (\tau - 1)\sum_{P^-}(y_i - q_p) \\ &= \tau \sum_{P^+} y_i - N_p^+ \tau q_p + (\tau - 1)\sum_{P^-} y_i - N_p^-(\tau - 1)q_p \\ &= \tau \sum_{P^+} y_i + (\tau - 1)\sum_{P^-} y_i - q_p \left[N_p^-(\tau - 1) + N_p^+ \tau\right] \end{aligned} \tag{4.5}$$

Suppose we have already computed $QAD_{q_p,P}$ and we now wish to update the value when we insert the data point s as before with $R = \{P \cup s\}$. Consider first the case where the point is added to the left heap H_{low}. We have two possibilities to consider to obtain $QAD_{q_R,R}$.

- Case 1: Addition of the point s does not cause any points to be moved from H_{low} to H_{high} in the update of the quantile. In this case we obtain

$$\begin{aligned} QAD_{q_R,R} &= \tau \sum_{R^+}(y_i - q_R) + (\tau - 1)\sum_{R^-}(y_i - q_R) \\ &= \tau \sum_{P^+}(y_i - q_R) + (\tau - 1)\sum_{P^-}(y_i - q_R) + (\tau - 1)(s - q_R) \\ &= \tau \sum_{P^+} y_i + (\tau - 1)\sum_{P^-} y_i - q_R \left[N_p^-(\tau - 1) + N_p^+ \tau\right] + (\tau - 1)(s - q_R) \\ &= QAD_{q_P,P} + (q_P - q_R)\left[N_p^-(\tau - 1) + N_p^+ \tau\right] + (\tau - 1)(s - q_R). \end{aligned} \tag{4.6}$$

- Case 2: When we add the point s to H_{low} we need to re-balance the heaps by moving $H_{low}[top]$ to the heap H_{high} in order to obtain the new quantile. Let l denote the value of $H_{low}[top]$. Again we can derive a similar update equation as in (4.6):

$$\begin{aligned} QAD_{q_R,R} &= \tau \sum_{R^+}(y_i - q_R) + (\tau - 1)\sum_{R^-}(y_i - q_R) \\ &= \tau \sum_{P^+}(y_i - q_R) + \tau(l - q_R) \\ &\quad + (\tau - 1)\sum_{P^-}(y_i - q_R) - (\tau - 1)(l - q_R) + (\tau - 1)(s - q_R) \\ &= \tau \sum_{P^+} y_i + (\tau - 1)\sum_{P^-} y_i \\ &\quad - q_R \left[N_p^-(\tau - 1) + N_p^+ \tau\right] + l - q_R + (\tau - 1)(s - q_R) \\ &= QAD_{q_P,P} + (q_P - q_R)\left[N_p^-(\tau - 1) + N_p^+ \tau\right] \\ &\quad + (l - q_R) + (\tau - 1)(s - q_R) \end{aligned} \tag{4.7}$$

Note that Case 2 is a generalization of Case 1.

If the point s to be added is greater than $H_{low}[top]$ then we insert it into H_{high}. Following a similar procedure we can obtain the following update equation for similar cases:

- Case 1: No movement of points from H_{high} to H_{low} when updating quantile.

$$QAD_{q_R,R} = QAD_{q_P,P} + (q_P - q_R)\left[(\tau-1)N_p^- + \tau N_p^+\right] + \tau(s - q_R) \tag{4.8}$$

- Case 2: Move $H_{high}[top]$ to H_{low} when updating quantile.

$$QAD_{q_R,R} = QAD_{q_P,P} + (q_P - q_R)\left[(\tau-1)N_p^- + \tau N_p^+\right] + (q_R - l) + \tau(s - q_R) \tag{4.9}$$

These four cases, covering all possible cases, can be represented compactly using a single update equation as follows. Let $\mathcal{I} = 1$ if a value has to be moved from the left heap to the right or vice versa and 0 otherwise. Let $\mathcal{J} = 1$ if we move a value from the left heap to the right and $\mathcal{J} = -1$ if we move a value from the right heap to the left. Let $\mathcal{K} = 1$ if the value was inserted into the left heap and 0 otherwise.

$$\begin{aligned} QAD_{q_R,R} = QAD_{q_P,P} + (q_P - q_R)\left[(\tau-1)N_p^- + \tau N_p^+\right] \\ + \mathcal{I}\mathcal{J}(l - q_R) + \mathcal{K}(\tau-1)(s - q_R) + (1-\mathcal{K})\tau(s - q_R) \end{aligned} \tag{4.10}$$

Equation (4.10) provides us with a streaming update equation to update the value of the loss obtained from the left child as we move one point at a time.

So far we have only discussed the case of adding points sequentially into the heaps. To find (l^*, r^*) we need to remove points sequentially from the right split starting with all the points. Deletion of floating-point values from heaps can run into trouble but we can avoid that problem simply by inserting points in reverse order.

We demonstrate the evaluation of $\rho(x)$ for a simple node with 5 values below. Let y_a, y_b, y_c, y_d, y_e be a list of the response variables sorted based on the order of X_i. By sequentially inserting these values we obtain the following results of $\rho(x)$.

Table 4.2: Iteration progress in forward and reverse to obtain values of $\rho(x)$ for all combinations of (l, r). The left split is used to compute S_l and the right split is used to compute S_r.

Iteration	Left split	Iteration	Right split	Full result
1	–	6	$\{y_a, y_b, y_c, y_d, y_e\}$	$\rho(\emptyset, \{a, b, c, d, e\})$
2	$\{y_a\}$	5	$\{y_b, y_c, y_d, y_e\}$	$\rho(\{a\}, \{b, c, d, e\})$
3	$\{y_a, y_b\}$	4	$\{y_c, y_d, y_e\}$	$\rho(\{a, b\}, \{c, d, e\})$
4	$\{y_a, y_b, y_c\}$	3	$\{y_d, y_e\}$	$\rho(\{a, b, c\}, \{d, e\})$
5	$\{y_a, y_b, y_c, y_d\}$	2	$\{y_e\}$	$\rho(\{a, b, c, d\}, \{e\})$
6	$\{y_a, y_b, y_c, y_d, y_e\}$	1	–	$\rho(\{a, b, c, d, e\}, \emptyset)$

Once we obtain the values of $\rho(x)$ for all possible split points we can easily obtain the best possible split point to obtain (l^*, r^*) using a linear scan.

The pseudocode for computing the left child is described in Algorithm 1. The input Y is the response vector, sorted based on the sort order of the predictor vector. QAD is initialized to zero for computing the left child. After each insertion we need to rebalance the heaps and obtain the new quantile values. Once rebalanced we can compute the new quantile value and obtain the next value of the QAD using the previous one using the update (4.10). To start we set $QAD[0]$ and $QAD[1]$ equal to zero. In order to compute the complete QAD vector the algorithm is very similar but we insert points in the reverse order and add the changes to the QAD vector. At the end a linear scan identifies the best possible split.

Algorithm 1 Streaming update for quantile loss for left child

procedure LEFT_QAD(Y, QAD, τ)
  initialize H_{low}, H_{high}
  set $QAD[1] = QAD[2] = 0.0$
  insert $Y[1]$ into H_{low}
  set $q_P = Y[1]$
  for i:=2.. (N+1) **do**
    if $Y[i] \leq H_{low}[top]$ **then**
      insert $Y[i]$ into H_{low}
    else
      insert $Y[i]$ into H_{high}
    end if
    RebalanceHeaps(H_{low}, H_{high}, τ)
    q_R = FindQuantile(H_{low}, H_{high}, τ)
    $QAD[i] = QAD[i-1]$+ update using (4.10)
    set $q_P = q_R$
  end for
end procedure

procedure REBALANCEHEAPS(H_{low}, H_{high}, τ)
  $nl \leftarrow size(H_{low})$
  $nr \leftarrow size(H_{high})$
  $test \leftarrow \lceil((nl + nr - 1)\tau)$
  if $nl > test$ **then**
    pop $H_{high}[top]$ and insert into H_{low}
  else if $nl < test$ **then**
    pop $H_{low}[top]$ and insert into H_{high}
  end if
end procedure

procedure FINDQUANTILE(H_{low}, H_{high}, τ)
  $nn \leftarrow size(H_{low}) + size(H_{high})$
  $test \leftarrow \lceil((nn - 1)\tau)\rceil$
  $q \leftarrow H_{low}[top] + (H_{high}[top] - H_{low}[top])(\tau(nn - 1) - (test - 1))$
end procedure

Note on algorithmic complexity: For each predictor variable we need to first find the sort order for the points in the node. This operation takes $O(N \log N)$ where N is the number of points. The worst case complexity for the `LEFT_QAD` algorithm involves N insertions, N deletions for rebalances and N re-insertions. This provides an upper bound of $O(N \log N)$ for both the left and right children, the same order as sorting. Since OLS trees also need to sort, our algorithm maintains the same order of complexity as OLS trees.

4.4 Computational results

In this section we provide both timing and accuracy results obtained for our qtree method against traditional OLS trees. For comparison we use both the *tree* package [75] and the *rpart* package [81] from R [73]. Both these packages build trees using the OLS criterion. Our implementation is provided as a package in R and uses the *RcppArmadillo* package [32]. We first discuss the scalability of our algorithm.

4.4.1 Scalability

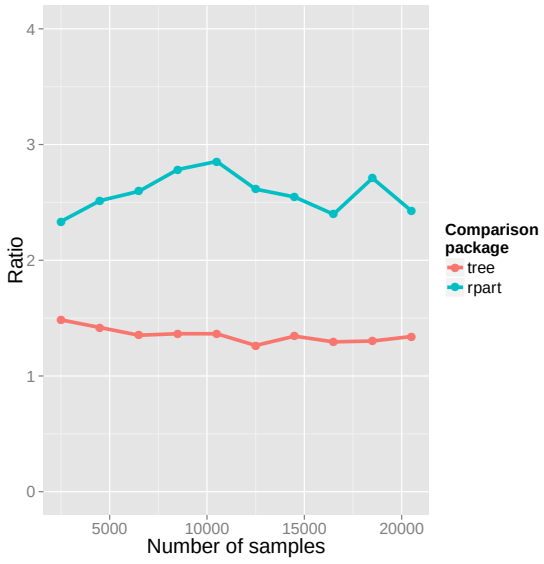
In the preceding section we showed that the worst case asymptotic complexity of the qtree algorithm is the same as OLS trees. In order to demonstrate this with practical examples we consider two data sets for the tests. The first is the California housing data set [66]. This data set has 20,640 samples and 8 predictor variables. The second is the CT slices data set available at UCI machine learning repository [36]. This data set has 53,500 samples and 384 predictor variables. For both tests we set the following common parameters. When considering if a node has to be split there should be at least 20 sample points in the node. The minimum number of samples in any split cannot be smaller than 7. The value of Δ_{node} which is the amount of reduction offered by introducing the split cannot be smaller than 0.01. These are the default settings for *rpart* package. The package *rpart* can perform 10-fold cross-validation internally but we turn it off for this study. We also request rpart to not search for surrogate variables which cuts the time required to build the trees.

For the California housing data set we consider sub-samples of the data without replacement of increasing sizes $\{1500, 2500, \ldots, 19500\}$. For the CT slices data set the sample sizes are chosen to be $\{5000, 10000, \ldots, 50000\}$. We use all the predictor variables in both cases. For each sample size we perform 100 runs of the simulation and present the averaged results.

In Figure 4.2 we plot the ratio of time taken by our algorithm against the tree and rpart packages for the California housing data set. The ratio of time taken against both packages is less than 3 and does not show an increase in the time as a function of the number of data points. In Figure 4.3 we show a similar plot for the CT slices data set. We notice again that the ratio of time is less than 5 and does not grow as a function of the number of data points. Both these results confirm that the algorithmic complexity of our algorithm is the same as that of OLS trees. We also provide the exact run times in Table 4.3 performed on a i7 laptop with 4Gb memory.

The tree package is written with the aim of simplicity while the rpart package is the standard optimized version for growing OLS trees. Since almost all the work is done in deciding the split we cannot expect the running time of our algorithm to be less than a factor of 2 than that of the rpart package. This is because our algorithm requires two passes; once

Figure 4.2: Ratio of time taken by our algorithm to the tree and rpart packages for the California housing data set.



for computing the left QAD and then to update it to obtain the complete QAD when doing the right split. The OLS trees on the other hand just need a single pass over the data and can compute the best split with $O(1)$ memory.

4.4.2 Model accuracy

Our proposed algorithm can handle any quantile value $0 < q < 1$ which allows us to build non-parametric models that extend linear quantile regression [53]. A complete analysis of the abilities and scope with regards to the linear quantile regression case will be done elsewhere. In this chapter we contrast the use of LAD trees which correspond to $\tau = 0.5$ against traditional OLS trees, keeping in mind the underlying objective function.

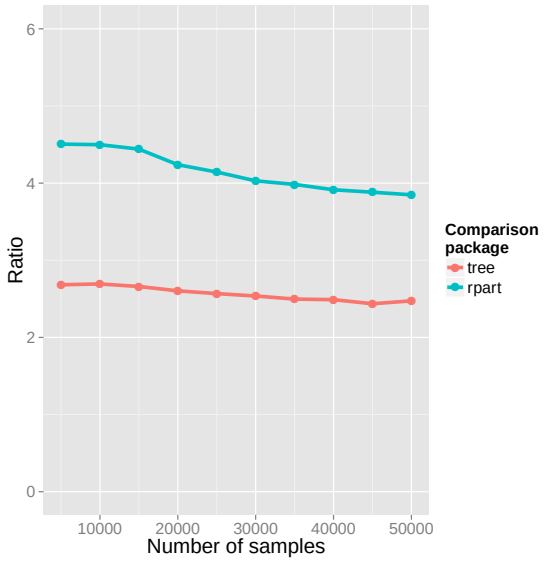
The objective function optimized by recursive partitioning methods using the OLS criterion is the squared error loss also known as deviance. LAD trees in contrast will optimize the absolute deviation function. The aim of these methods is to provide a fast greedy heuristic to minimize the following two objective functions:

- Mean Squared Error (MSE): $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- Mean Absolute Deviation (MAD): $\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$

where we define the true and predicted values by y_i and $\hat{y}_i$ respectively over the tested samples. Since the functions which are optimized are different we will obtain different models in each case. For prediction purposes it will be beneficial if we decide on which one of these loss functions is appropriate for our analysis.

We build models using the qtree algorithm and the rpart method and estimate these two error functions using a standard 10-fold cross-validation procedure. The internal cross-validation procedure of rpart is turned off and we use the same folds for both the methods.

Figure 4.3: Ratio of time taken by our algorithm to the tree and rpart packages for the CT slices data set.



The data sets used to perform the tests include:

- Wine: The wine data set is available at the UCI machine learning repository [27]. It consists of two data sets both with 11 predictors. The red wine data set has 1599 samples and the white wine data set has 4898 samples. The response is the quality of wine which is real valued and lies between 1 and 10.
- Crime: The Communities and Crime data set from the UCI machine learning repository [74]. The complete data set consists of 1994 samples with 128 instances. Since there are missing values in some columns we eliminate those columns reducing the number of predictors to 96. The output is measure of crime rate which lies between 0 and 1.
- CA housing: This is the same data set used in the timing analysis. The response variable is the log of the median house prices.

Our algorithm follows closely to rpart package with regards to most of the control parameters and hence we use the same settings whenever possible. Minimum split size is set to 20 and minimum samples in a leaf node is set to 7. In order to prevent pruning both methods use the notion of a minimum required reduction in error in order to grow a node. We use the same default value of 0.01 for this parameter. Note that while the same numeric value is being used, the rpart routine will use the squared value while we consider the absolute value. In order to avoid this inconsistency we report the size of the trees grown in both cases. All results are averaged over 100 runs where each run does a 10 fold cross-validation.

The results are tabulated in Table 4.4. LAD trees perform better for every data set if our criterion for minimization is the mean absolute deviation. OLS trees perform better for every data set if the criterion is minimization of mean squared error. These results show that our objective decides the use of either methods. From the table we also notice that LAD trees are able to achieve lower MAD errors using trees of smaller sizes. For both the wine

Table 4.3: Average of raw times taken by all three algorithms for the housing and CT slices data set.
The ratio is non-increasing as sample sizes increase and is within a small constant factor of OLS trees for various sample sizes of the data.

Housing				CT Slices			
Size	qtree	tree	rpart	Size	qtree	tree	rpart
1.5K	0.0202	0.0136	0.0086	5K	2.5418	0.9476	0.5641
3.5K	0.0383	0.0270	0.0152	10K	5.1707	1.9200	1.1495
5.5K	0.0603	0.0446	0.0232	15K	7.8585	2.9546	1.7697
7.5K	0.0814	0.0596	0.0292	20K	10.6167	4.0779	2.5048
9.5K	0.1016	0.0744	0.0356	25K	13.4323	5.2316	3.2416
11.5K	0.1276	0.1009	0.0488	30K	16.2734	6.4152	4.0386
13.5K	0.1518	0.1129	0.0596	35K	19.1917	7.6866	4.8185
15.5K	0.1734	0.1339	0.0723	40K	22.1166	8.8849	5.6510
17.5K	0.2035	0.1563	0.0751	45K	25.0712	10.2914	6.4583
19.5K	0.2294	0.1710	0.0945	50K	28.0457	11.3384	7.2880

Table 4.4: Accuracy results for MAD and MSE for qtree and rpart methods.
The size column indicates the size of the tree grown including the leaf nodes. On every data set LAD trees obtain a smaller error if we consider the MAD loss function and similarly OLS trees perform better on every data set if the loss function is MSE.

	LAD trees			OLS trees		
	MAD	MSE	Size	MAD	MSE	Size
Red wine	0.4843	0.5670	7.68	0.5304	0.4619	19.91
White wine	0.5275	0.6553	9.00	0.6041	0.5817	11.84
Crime	0.1022	0.0340	22.39	0.1070	0.0308	34.54
Housing	0.2808	0.1382	24.03	0.2857	0.1370	27.11

data sets LAD trees provide a significantly lower error rate as compared to the OLS trees. For the same two data sets the OLS trees provide much lower error rates for the MSE error. However, the difference between both trees is small for both MAD and MSE for the Crime and Housing data sets. This shows that the choice of loss function is also dependent on the data set.

4.5 Conclusion

We have derived a general algorithm for an absolute deviation loss function that works for any quantile value $0 < \tau < 1$. Performing numerical simulations we showed that the asymptotic complexity of our algorithm is similar to and within a small constant factor of that of OLS based trees with a small constant factor multiple. This is a significant reduction in computational complexity over a naive implentation.

Our current work can be extended to the following use cases:

1. Random Forests: Random forests [19] are built using sub-samples of data by restricting the choice of predictors at each node to a subset of the predictors. Their have a higher

predictive ability than a single tree since they trade off bias to achieve a lower variance. Since they are built with repeated calls to a tree building algorithm, our current work easily fits into the framework of random forests.

2. Boosting: Boosting [37] is a technique of building a strong learner from a collection of weak base learners. The most widely-used base learners are decision stumps/ trees. Since their introduction, they have achieved much success in solving a variety of supervised learning problems. One of the strongest features of boosting is their ability to resist over fitting. The first variant of boosting introduced was able to deal with classification problems; since then there have been many of variations of boosting that can handle regression problems as well. A specific case we mention is the GBM procedure [38]. Quoting from [38], the authors mention the problem with directly minimizing the MAD error and state
"Squared error loss is much more rapidly updated than mean-absolute-deviation when searching for splits during the tree building process"
Our algorithm eliminates this problem and allows one to directly minimize the MAD in their algorithm.

3. Quantile Regression: While our current work shows the application of our algorithm only to LAD trees the algorithm can build trees with any quantile of interest. This allows us to obtain a distribution of the response variable rather than just a point estimate. This is a very powerful technique used in linear quantile regression [53]. Linear quantile regression is an extension of linear regression that allows us to interpret the underlying model dependencies by providing a distribution of the response. However, the underlying assumption in these models is that the dependence of the predictors is linear. This is a very severe restriction when trying to infer the underlying model. By generalizing to the non-parametric case, we can immediately handle unknown nonlinear relationships between the predictor and response variables.

Appendix A

FVFD implementation

We describe here the implementation details for the finite volume Fourier domain solver described in Chapter 3. The solver is written in PETSc [6], which enables easy implementation for large scale solvers for partial differential equations. The main steps involved in the code are as follows.

A.0.1 Mesh Generation

The mesh is generated by an external library. We used two approaches for mesh generation.

- Triangle : The Triangle library [77] is written in C and constructs Delaunay triangulations.
- DistMesh : DistMesh [69] is a Matlab based triangulation library which solves a spring-mass system to obtain a high-quality mesh.

The Triangle library can be called from PETSc directly, but the complete API is not exposed and hence we generate meshes externally and then load them.

The meshes in both cases are obtained in vertex and triangles format. The Triangle library is very fast and can generate meshes of size up to a few million within a minute on a laptop. However, the meshes obtained can only be constrained to have a minimum angle of 33 degrees among all the triangles. This is a severe restriction for our case and hence we do not use it to build unstructured meshes. The DistMesh library can produce much better quality meshes achieving a minimum angle of approximately 45 degrees. The time taken to generate these meshes, however, is very large and it can take up to a few days to generate a mesh of size 1 million. In order to prevent mesh generation from being time-consuming we reuse the meshes whenever possible.

A.0.2 Loading the mesh into PETSc

The meshes generated from either of the two libraries can be directly loaded in to PETSc without requiring any pre-processing with the help of the `DMPlexCreateFromCellList` function. This function creates the data structure which holds the mesh on a single processor. `DMPlexDistribute` is used to distribute the mesh across all available processes. Since the method is a finite volume method, we choose an overlap of 1 across processors which gives us access to the neighboring cells. We have a choice of using either the `Chaco` [46] or `Metis` [51]

libraries to control the distribution of meshes. The load balancing of both methods is comparable and hence we decided to use `Chaco`.

Once the distributed mesh is available across processors, we need to compute the various parameters for our algorithm. These include L, G, C_0 and Δ. Because of the distributed nature, the vectors L, G and C need to be computed locally. We first compute the number of local cells on each processor and compute C_0 and G locally. Similarly, depending on the specific forcing applied, we also build the right hand side forcing term V_{in}. The forcing is computed through a line integral and we use the GNU libraries *qng* integration method. We now need to compute the inductances and the graph incidence matrix.

A.0.3 Computing the dual of the mesh

Due to the distributed nature it becomes hard to compute the graph incidence matrix directly as the number of edges is not easily known. The second-order formulation of the problem does not require use to compute either the inductance L or the graph incidence matrix as long as we can obtain the weighted graph Laplacian Δ.

Consider the following simple discretization of the domain shown in Figure A.1. The

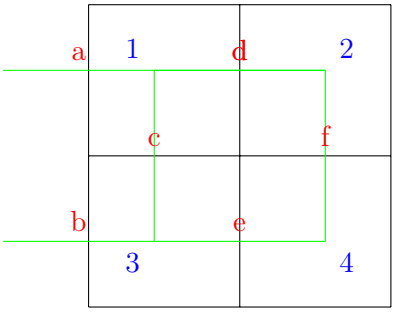


Figure A.1: Mesh and dual

incidence matrix for this mesh is given by:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix} \end{array}.$$

Note that we do not require that the inputs be placed at the ends. They can appear in any order, i.e., the labelling of the nodes does not affect the computation of Δ. This is very helpful since we do not need to worry about the distribution of the mesh across processors.

Suppose the matrix L^{-1} is as shown:

$$\begin{bmatrix} L_a & 0 & 0 & 0 & 0 & 0 \\ 0 & L_b & 0 & 0 & 0 & 0 \\ 0 & 0 & L_c & 0 & 0 & 0 \\ 0 & 0 & 0 & L_d & 0 & 0 \\ 0 & 0 & 0 & 0 & L_e & 0 \\ 0 & 0 & 0 & 0 & 0 & L_f \end{bmatrix}.$$

Then we find that the product $B * L^{-1} * B^t$ is given by:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} L_a + L_c + L_d & -L_d & -L_c & 0 \\ -L_d & L_d + L_f & 0 & L_f \\ -L_c & 0 & L_b + L_c + L_e & -L_e \\ 0 & -L_f & -L_c & L_f + L_e \end{pmatrix} \end{array}.$$

We see from this that the computation of the graph Laplacian can be done row-by-row. The diagonal part involves all inductances which are connected to the cell. The off-diagonal part is always negative, and its value is given by the inductance separating the two cells. In the case when the cell is an input cell, once the inductance is computed there is no difference in the two cases. Since the graph Laplacian involves only the cells, it also provides a natural way to obtain the number of rows on each processor: this equals the number of C_0 components on the same processor. This approach leads to the following algorithm for the computation of Δ.

A.0.4 Algorithm for computing Δ

The algorithm requires a few things to be available before it can be run. The requirements are:

- `indIs`: Contains the index set obtained from `DMPlexGetCellNumbering`. This is a global-to-local numbering that PETSc provides automatically.
- `cellVec`: Contains the centroid information for the cell. This can be computed by looping over the cells and simultaneously obtaining C_0 and G.
- `delta`: Matrix initialized to place the values of the entries to be inserted. The number of rows allocated on each processor is equal to the number of cells on the particular processor.

  The procedure for filling the matrix is provided in Algorithm 0.

A.0.5 Linear systems

Once we have assembled the vectors C_0 and G and computed the matrix Δ, we can generate the linear systems needed to compute the solutions. We choose to solve all systems up to order 10. This gives us 10 linear systems of the form

$$A(k) = C_0(-k^2(2\pi)^2) - 2\pi kiG + \Delta.$$

Since the vectors and matrices are laid out in distributed memory in a consistent manner, formation of the matrices just involves shifting the diagonal of Δ by a different amount each time.

PETSc provides a common interface for solving linear systems, either using direct solvers or iterative solvers compactly. The following four lines of code express this for use with *any* solver (either direct or iterative and with/without preconditioning) for the linear problem.

```
procedure computeDelta(..)
    for i ← cStart, cEnd do
        if indIs[i] ≤ 0 then
            continue
        end if
        numVals ⟵ 0
        rowId ⟵ i
        cInd ⟵ [i, −1, −1, −1]
        val ⟵ [0, 0, 0, 0]
        centOrig ⟵ cellInfo.centroid
        E = [e1, e2, e3] ⟵ Support(i)
        for j ← 0, 2 do
            xj := {cm, cn} ⟵ Cone(E[j])
            if |xj| == 1 then
                if ejvertical then
                    if ejleft boundary thenn
                        Lj ⟵ 2 ∗ computeL()
                        val[0]+ = Lj
                    else
                        G[i] ⟵ ComputeG()
                    end if
                else
                    G[i] ⟵ ComputeG()
                end if
            else
                neighCell = xj[0] == i ? xj[1] : xj[0]
                centNext ⟵ cellInfo− > centroid[indIs[neighCell]]
                Lj ⟵ computeL()
                if indIs[neighCell] > 0 then
                    cInd[numVals] = indIs[neighCell]
                else
                    cInd[numVals] = −(indIs[neighCell] + 1)
                end if
                val[numVals + +] = −Lj
            end if
        end for
        for k ← 1, numVals − 1 do
            vals[0]+ = −vals[k]
        end for
        MatSetValues(delta, 1, rowId, numVals, cInd, vals, INSERT_VALUES)
    end for
    MatAssemblyBegin(delta, MAT_FINAL_ASSEMBLY)
    MatAssemblyEnd(delta, MAT_FINAL_ASSEMBLY)
end procedure
```

```
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
ierr = KSPSetOperators(ksp,delta0,delta0,SAME_PRECONDITIONER);CHKERRQ(ierr);
ierr = KSPSetFromOptions(ksp);CHKERRQ(ierr);
ierr = KSPSolve(ksp,model->Vin,sol[0]);CHKERRQ(ierr);
ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
```

We first create a distributed solver whose span is the entire range of processors through `PETSC_COMM_WORLD`. We then set the matrix on which we plan to run the solver on the next line. The flag `SAME_PRECONDITIONER` is used to signify that the matrix will not change from one iteration to another and hence if we build a pre-conditioner then the same pre-conditioner can be reused.

The next line sets all command line options for our solver. This is crucial and lets us change the type of solver without any modification to the code and without need to recompile. We finally solve the system, save our solution and clean up.

A.0.6 Post-processing

The solution obtained through the PETSc code is written for final post-processing into files along with the centroid information. Here, it is important to consider the representation of the points. We do not use parallel I/O. Writing a parallel vector involves sending all components to processor 0 and then writing to file. We maintain a mapping for the relation between the cell centroids and the solution computed so we can plot it directly without any need to permute the solution.

Post-processing is done by importing the solutions into Python through the API provided for loading binary files produced through PETSc. Once the solution and the centroids are loaded, we can then produce plots of the solution and verify the order of accuracy of the solution.

The entire code is available on a GitHub repository at

https://github.com/GarnetVaz/NME.

The code requires PETSc (version $\geq$ 3.4) to be installed and it can support both the homogeneous or inhomogeneous problem through a command line flag. The forcing and boundary conditions are hard-coded and requires the user to make changes appropriately for use in other cases. Instructions to run the code are provided in the source code.

Bibliography

[1] Ehsan Afshari, Harish S. Bhat, A Hajimiri, and Jerold E. Marsden. Extremely wideband signal shaping using one-and two-dimensional nonuniform nonlinear transmission lines. *Journal of Applied Physics*, 99(5):054901, 2006.

[2] Ehsan Afshari, Harish S. Bhat, and Ali Hajimiri. Ultrafast analog fourier transform using 2-d lc lattice. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(8):2332–2343, 2008.

[3] Ehsan Afshari, Harish S. Bhat, Ali Hajimiri, and Jerrold E. Marsden. Extremely wideband signal shaping using one- and two-dimensional nonuniform nonlinear transmission lines. *Journal of Applied Physics*, 99:054901, 2005.

[4] Ehsan Afshari and Ali Hajimiri. Nonlinear transmission lines for pulse shaping in silicon. *IEEE Journal of Solid-State Circuits*, 40(3):744–752, 2005.

[5] Alex Arenas, Albert Díaz-Guilera, Jurgen Kurths, Yamir Moreno, and Changsong Zhou. Synchronization in complex networks. *Physics Reports*, 469:93–153, 2008.

[6] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2013. http://www.mcs.anl.gov/petsc.

[7] Ilan Bar-On and Victor Ryaboy. Fast diagonalization of large and dense complex symmetric matrices, with applications to quantum reaction dynamics. *SIAM Journal on Scientific Computing*, 18(5):1412–1435, 1997.

[8] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[9] Harish S. Bhat and Ehsan Afshari. Nonlinear constructive interference in electrical lattices. *Physical Review E*, 77:066602, 2008.

[10] Harish S. Bhat, Wooram Lee, Georgios N. Lilis, and Ehsan Afshari. Steady-state perturbative theory for nonlinear circuits. *Journal of Physics A: Mathematical and Theoretical*, 43:205101, 2010.

[11] Harish S. Bhat and B. Osting. The zone boundary mode in periodic nonlinear electrical lattices. *Physica D*, 238:1216–1228, 2009.

[12] Harish S. Bhat and Braxton Osting. 2-D Inductor-capacitor lattice synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30:1483–1492, 2011.

[13] Harish S. Bhat and Braxton Osting. Kirchhoff's laws as a finite volume method for the planar Maxwell equations. *IEEE Transactions on Antennas and Propagation*, 59:3772–3779, 2011.

[14] Harish S. Bhat and Garnet J. Vaz. Finite volume method for planar Maxwell equations in nonlinear media. In preparation, 2013.

[15] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424:175–308, 2006.

[16] Markus Brede. Construction principles for highly synchronizable sparse directed networks. *Physics Letters A*, 372:5305–5308, 2008.

[17] Markus Brede. Locals vs. global synchronization in networks of non-identical Kuramoto oscillators. *The European Physical Journal B*, 62:87–94, 2008.

[18] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[19] L. Breiman. Random forests. *Machine Learning*, 45:5–32, October 2001.

[20] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[21] Michael L. Brewer, Lori Freitag Diachin, Patrick M. Knupp, Thomas Leurent, and Darryl J. Melander. The mesquite mesh quality improvement toolkit. In *IMR*, 2003.

[22] M. Case, E. Carman, R. Yu, M. J. W. Rodwell, and M. Kamegawa. Picosecond duration, large amplitude impulse generation using electrical soliton effects. *Applied Physics Letters*, 60(24):3019–3021, 1992.

[23] M. Case, M. Kamegawa, R. Yu, M. J. W. Rodwell, and J. Franklin. Impulse compression using soliton effects in a monolithic GaAs circuit. *Applied Physics Letters*, 58(2):173–175, 1991.

[24] P. Chaudhuri. Asymptotic consistency of median regression trees. *Journal of Statistical Planning and Inference*, 91:229–238, 2000.

[25] P. Chaudhuri and W.-Y. Loh. Nonparametric estimation of conditional quantiles using quantile regression trees. *Bernoulli*, 8(5):561–576, 2002.

[26] Christos Christopoulos. The transmission-line modeling (TLM) method in electromagnetics. *Synthesis Lectures on Computational Electromagnetics*, 1(1):1–132, 2005.

[27] Paulo Cortez, Juliana Teixeira, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Using data mining for wine quality assessment. In *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 66–79. Springer-Verlag, 2009.

[28] Majid Dadashi, Iman Barjasteh, and Mahdi Jalili. Rewiring dynamical networks with prescribed degree distribution for enhancing synchronizability. *Chaos*, 20:043119–043119–6, 2010.

[29] B. DiDonna and T. Lubensky. Nonaffine correlations in random elastic media. *Physical Review E*, 72:066619, 2005.

[30] Luca Donetti, Pablo I. Hurtado, and Miguel A. Muñoz. Entangled networks, synchronization, and optimal network topology. *Physical Review Letters*, 95:188701, 2005.

[31] Freeman J. Dyson. The dynamics of a disordered linear chain. *Physical Review*, 92:1331–1338, 1953.

[32] Dirk Eddelbuettel and Conrad Sanderson. Rcpparmadillo: Accelerating R with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014.

[33] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[34] Leopold B. Felsen, Mauro Mongiardo, and Peter Russer. *Electromagnetic field computation by network methods*. Springer, 2009.

[35] E. Fermi, J. Pasta, and S. Ulam. Studies of non linear problems. Technical Report LA-1940, Los Alamos National Laboratory, 1955.

[36] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[37] Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[38] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[39] G. Galimberti, M. Pillati, and G. Soffritti. Notes on the robustness of regression trees against skewed and contaminated errors. In *New Perspectives in Statistical Modeling and Data Analysis*, pages 255–263. Springer-Verlag, 2011.

[40] Wilfried N Gansterer, Hannes Schabauer, Christoph Pacher, and Norman Finger. Tridiagonalizing complex symmetric matrices in waveguide simulations. In *Computational Science–ICCS 2008*, pages 945–954. Springer, 2008.

[41] K. Guillouard, M.F. Wong, Fouad V. Hanna, and J. Citerne. A new global time-domain electromagnetic simulator of microwave circuits including lumped elements based on finite-element method. *Microwave Theory and Techniques, IEEE Transactions on*, 47(10):2045–2049, 1999.

[42] Wojciech K. Gwarek. Analysis of an arbitrarily-shaped planar circuit a time-domain approach. *IEEE Transactions on Microwave Theory and Techniques*, 33(10):1067–1072, 1985.

[43] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[44] Ruonan Han and Ehsan Afshari. A 260GHz broadband source with 1.1mW continuous-wave radiated power and EIRP of 15.7dBm in 65nm CMOS. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 138–139, 2013.

[45] Ruonan Han and Ehsan Afshari. A high-power broadband passive terahertz frequency doubler in CMOS. *IEEE Transactions on Microwave Theory and Techniques*, 61:1150–1160, 2013.

[46] Bruce Hendrickson and Robert Leland. The chaco user's guide: Version 2.0. Technical report, Technical Report SAND95-2344, Sandia National Laboratories, 1995.

[47] R. Hirota and K. Suzuki. Studies on lattice solitons by using electrical networks. *Journal of the Physical Society of Japan*, 28:1366–1367, 1970.

[48] Ryogo Hirota and Kimio Suzuki. Studies on lattice solitons by using electrical networks. *Journal of the Physical Society of Japan*, 28(5):1366–1367, 1970.

[49] Mahdi Jalili, Ali Rad, and Martin Hasler. Enhancing synchronizability of weighted dynamical networks using betweenness centrality. *Physical Review E*, 78:016105, 2008.

[50] J.D. Joannopoulos, S.G. Johnson, J.N. Winn, and R.D. Meade. *Photonic Crystals: Molding the Flow of Light (Second Edition)*. Princeton University Press, 2011.

[51] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[52] David Kelly and Georg A. Gottwald. On the topology of synchrony optimized networks of a Kuramoto-model with non-identical oscillators. *Chaos*, 21:025110, 2011.

[53] R. Koenker. *Quantile Regression*. Cambridge University Press, 2005.

[54] Hiroshi Kori and Alexander Mikhailov. Strong effects of network architecture in the entrainment of coupled oscillator systems. *Physical Review E*, 74:066115, 2006.

[55] Gabriel Kron. Equivalent circuit of the field equations of maxwell-I. *Proceedings of the IRE*, 32(5):289–299, 1944.

[56] R. Landauer. Shock waves in nonlinear transmission lines and their effect on parametric amplification. *IBM Journal of Research and Development*, 4(4):391–401, 1960.

[57] Rolf Landauer. Shock waves in nonlinear transmission lines and their effect on parametric amplification. *IBM Journal of Research and Development*, 4(4):391–401, 1960.

[58] Wooram Lee, Muhammad Adnan, Omeed Momeni, and Ehsan Afshari. A nonlinear lattice for high-amplitude picosecond pulse generation in CMOS. *IEEE Transactions on Microwave Theory and Techniques*, 60:370–380, 2012.

[59] Wooram Lee and Ehsan Afshari. A CMOS noise-squeezing amplifier. *IEEE Transactions on Microwave Theory and Techniques*, 60:329–339, 2012.

[60] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.

[61] Georgios N. Lilis, Jihyuk Park, Wooram E. Lee, Guansheng Li, Harish S. Bhat, and Ehsan Afshari. Harmonic generation using nonlinear LC lattices. *IEEE Transactions on Microwave Theory and Techniques*, 58:1713–1723, 2010.

[62] N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999, 2006.

[63] Omeed Momeni and Ehsan Afshari. A broadband mm-wave and terahertz traveling-wave frequency multiplier on CMOS. *IEEE Journal of Solid-State Circuits*, 46:2966–2976, 2011.

[64] Cécile Monthus and Thomas Garel. Random elastic networks: a strong disorder renormalization approach. *Journal of Physics A: Mathematical and Theoretical*, 44:085001, 2011.

[65] L. A. Ostrovskii, V. V. Papko, and Yu. A. Stepanyants. Solitons and nonlinear resonance in two-dimensional lattices. *Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki*, 78:831–841, 1980. [Sov. Phys. JETP 51, 417 (1980)].

[66] R. K. Pace and R. Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[67] Kwangho Park, Liang Huang, and Ying-Cheng Lai. Desynchronization waves in small-world networks. *Physical Review E*, 75:026211, 2007.

[68] C. Perlich, S. Rosset, R. D. Lawrence, and B. Zadrozny. High-quantile modeling for customer wallet estimation and other applications. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 977–985, 2007.

[69] P. Persson and G. Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2004.

[70] Martin Pototschnig, Jens Niegemann, Lasha Tkeshelashvili, and Kurt Busch. Time-Domain Simulations of the Nonlinear Maxwell Equations Using Operator-Exponential Methods. *IEEE Transactions on Antennas and Propagation*, 57(2):475–483, 2009.

[71] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[72] Ross J. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[73] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[74] Michael Redmond and Alok Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141(3):660–678, 2002.

[75] Brian Ripley. *tree: Classification and regression trees*, 2014. R package version 1.0-35.

[76] Masayuki Sato, S. Yasui, M. Kimura, T. Hikihara, and A. Sievers. Management of localized energy in discrete nonlinear transmission lines. *EPL (Europhysics Letters)*, 80(3):30002, 2007.

[77] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996.

[78] Yu. A. Stepanyants. Experimental investigation of cylindrically diverging solitons in an electric lattice. *Wave Motion*, 3:335–341, 1981.

[79] Yu. A. Stepanyants. Experimental study of "Cerenkov" radiation from solitons in two-dimensional LC-lattices. *Radiophysics and Quantum Electronics*, 26(7):601–607, 1983.

[80] Allen Taflove. Application of the finite-difference time-domain method to sinusoidal steady-state electromagnetic-penetration problems. *Electromagnetic Compatibility, IEEE Transactions on*, (3):191–202, 1980.

[81] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2014. R package version 4.1-6.

[82] Brian P. Tighe. Dynamic critical response in damped random spring networks. *Physical Review Letters*, 109:168303, 2012.

[83] Yuichi Togashi and Alexander S. Mikhailov. Nonlinear relaxation dynamics in elastic networks and design principles of molecular machines. *Proceedings of the National Academy of Sciences*, 104:8697–8702, 2007.

[84] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.

[85] M. Wyart, H. Liang, A. Kabla, and L. Mahadevan. Elasticity of floppy and stiff random networks. *Physical Review Letters*, 101:215501, 2008.

[86] Tatsuo Yanagita and Alexander S. Mikhailov. Design of oscillator networks with enhanced synchronization tolerance against noise. *Physical Review E*, 85:056206, 2012.

[87] Kane S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations. *IEEE Transactions on Antennas and Propagation*, 14(3):302–307, 1966.