IMPEDANCE MATCHING OPTIMIZATION BASED ON MATLAB

A Thesis in
Electrical Engineering

by
Kaiming Li

© 2013 Kaiming Li

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2013
The thesis of Kaiming Li was reviewed and approved* by the following:

James K. Breakall  
Professor of Electrical Engineering  
Thesis Advisor

Shizhuo Yin  
Professor of Electrical Engineering

Kultegin Aydin  
Professor of Electrical Engineering  
Department Head

*Signatures are on file in the Graduate School
ABSTRACT

In the design of RF radiators, impedance matching has been costing a significant amount of time and energy. Approaches to solve impedance matching problems mainly rely on computer aided numerical optimizations nowadays. An approach based on the MATLAB Global Optimization Toolbox™ is proposed in this thesis. This approach combines brute-force techniques and the Real Frequency Technique, which are the two main components of modern impedance matching methods.

The proposed approach allows the designer to choose a desired circuit topology, and use actual measurement impedance data from a candidate 80-meter high-frequency (HF) dipole antenna to get the optimum matching network. After comparing the performance of different algorithms at a single frequency, the simulated annealing algorithm is adopted as the solver in the Global Optimization Toolbox™. Four combinations of inductor and capacitor (L only; C only; LC in series and LC in parallel) are filled in topology blank boxes, and then the optimizer compares all possibilities and presents the best matching network result.

The Global Optimization Toolbox™ from MATLAB is the main software tool used in this thesis. The results are presented respectively for an L network, a T network, a Pi network, and 5-element and 6-element network topologies. The bandwidth can be increased from 3.4% to at least 14.4% after the optimization, with a VSWR of 2:1.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENTS</td>
<td>x</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>Introduction to Impedance Matching</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Impedance Matching Theory</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Bode-Fano Limit</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Previous Approaches to Impedance Matching Problems</td>
<td>7</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Analytic Approaches</td>
<td>9</td>
</tr>
<tr>
<td>1.4.2</td>
<td>Computer Aided Design Approach</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>The Approach of Impedance Matching Optimization Using MATLAB</td>
<td>12</td>
</tr>
<tr>
<td>1.6</td>
<td>Thesis Organization</td>
<td>13</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Global Optimization Toolbox™ in MATLAB</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>Global Optimization Toolbox™</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Global Optimization Solvers in Toolbox</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Options Setting in Global Optimization Toolbox</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Simulated Annealing Options</td>
<td>21</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Genetic Algorithm Options</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Defining an Optimization Problem from Network Matching</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Writing the Objective Function</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Writing Constraints</td>
<td>25</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Setting Options and Passing Parameters</td>
<td>26</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Basic Impedance Matching Network Topologies</td>
<td>27</td>
</tr>
<tr>
<td>3.1</td>
<td>L Network</td>
<td>28</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Use L network to match two resistive impedances</td>
<td>29</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Use L network to match complex impedances</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>T Network</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Pi Network</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Networks with More Elements</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Topology Used in this Thesis</td>
<td>37</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Solver Selection for Impedance Matching Problem</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Impedance Matching Calculator</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Verifying Matching Results Using Smith Chart</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Solve for Optimized Matching Network at Single Frequency</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison Between Solvers</td>
<td>48</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>MATLAB Modeling, Simulation, and Results</td>
<td>52</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1-1: A load connected to a transmission line through a matching network..................2
Figure 1-2: Circuits with simple reactive loads. ..............................................................4
Figure 1-3: Ideal power transmission over desired frequency band. .................................7
Figure 1-4: Darlington Representation of the load impedance. ........................................9
Figure 1-5: Four basic lumped element circuit to fill in topology positions.....................12
Figure 2-1: The interface of the Global Optimization Toolbox when solver is Genetic
Algorithm..................................................................................................................16
Figure 2-2: An L network where a capacitor is in shunt with the load, and then in series
with an inductor.......................................................................................................23
Figure 3-1: An ideal transformer.......................................................................................27
Figure 3-2: L network topology for Rsource<Rload. .........................................................29
Figure 3-3: L network topology for Rsource>Rload. .........................................................29
Figure 3-4: T network topology.......................................................................................31
Figure 3-5: T network as two back-to-back L networks with a virtual resistance in
between.....................................................................................................................32
Figure 3-6: Pi network topology.......................................................................................33
Figure 3-7: Pi network as two back-to-back L networks with a virtual resistance in
between.....................................................................................................................34
Figure 3-8: Cascaded L networks. ..................................................................................35
Figure 3-9: Basic impedance matching problem. ............................................................38
Figure 3-10: Topologies used to start with. .....................................................................38
Figure 4-1: Impedance matching calculator from EEWeb.com........................................40
Figure 4-2: Impedance matching calculator from Changpuak.ch......................................41
Figure 4-3: Impedance Matching Network Designer by John Wetherell. ......................43
Figure 4-4: Designer results for matching 50 Ohms source to 100 Ohms load at 300MHz...

Figure 4-5: Verifying L network results in the Smith Chart............................................

Figure 4-6: L network with a capacitor in shunt with the load, and then in series with an
inductor.........................................................................................................................

Figure 4-7: L network matching process showed in Smith Chart.................................

Figure 5-1: Pi Topology................................................................................................

Figure 5-2: All Networks in Pi Topology.................................................................

Figure 5-3: All Networks in L Topology.................................................................

Figure 5-4: All Networks in T Topology.................................................................

Figure 5-5: VSWR over 3.5-3.85 MHz before adding the matching network..........

Figure 5-6: VSWR over 3.5-3.85 MHz with $C_1=3706pF, C_2=6626pF, L=745.3nH$.....

Figure 5-7: VSWR over 3.5-3.85 MHz with start point [10 5 15] using patternsearch....

Figure 5-8: VSWR over 3.5-3.85 MHz with start point [500 500 500] using
patternsearch............................................................................................................

Figure 5-9: VSWR over 3.5-3.85 MHz using ga (Both starting points)....................

Figure 5-10: VSWR over 3.5-3.85 MHz with start point [10 5 15] using simulannealbnd....

Figure 5-11: VSWR over 3.5-3.85 MHz with start point [500 500 500] using
simulannealbnd.................................................................

Figure 5-12: VSWR over 3.5-3.85 MHz with 100 loops using patternsearch............

Figure 5-13: VSWR over 3.5-3.85 MHz with 5000 loops using patternsearch............

Figure 5-14: VSWR over 3.5-3.85 MHz with 100 loops using simulannealbnd............

Figure 5-15: VSWR over 3.5-3.85 MHz with 5000 loops using simulannealbnd............

Figure 5-16: VSWR over 3.4-4 MHz with 50 loops using simulannealbnd.................

Figure 5-17: VSWR over 3.4-4 MHz with 100 loops using simulannealbnd.................

Figure 5-18: VSWR over 3.4-4 MHz with 500 loops using simulannealbnd.................

Figure 5-19: VSWR over 3.4-4 MHz with 1000 loops using simulannealbnd.............
Figure 5-20: VSWR over 3.4-4 MHz with 5000 loops using *simulannealbnd*.........................69

Figure 5-21: VSWR over 3.5-3.85 MHz using Direct Search..................................................72

Figure 5-22: VSWR over 3.4-4 MHz using Direct Search......................................................73
LIST OF TABLES

Table 2-1: Solver Characteristics in Global Optimization Toolbox. ........................................19

Table 2-2: Common Options in Global Optimization Toolbox. ..............................................20

Table 2-3: Types of Constraints in Global Optimization Toolbox. . ......................................25

Table 2-4: Capacitance, Inductance and Corresponding Reactance Values at 3.7MHz. ............25

Table 3-1: An 80-meter dipole antenna impedance file before matching (Part). .....................37

Table 4-1: Optimizer Results to Compare Between ga and simulannealbnd. ........................50

Table 5-1: L and C Value Results Given By Optimizers. ....................................................61

Table 5-2: L and C Value Results Given By Loop Optimizers. .............................................64

Table 5-3: L, C Values and Bandwidths Results by simulannealbnd in loops. ......................67
ACKNOWLEDGEMENTS

First I would like to thank my family, for supporting me though my time as a student, and for respecting my opinion for choosing my own study path.

I want to thank Prof. James K. Breakall for providing me the opportunity to study at the Pennsylvania State University as his graduate student, and for being an insightful mentor both in life and research. As a professional in antenna engineering, he uses his solid and deep knowledge to help me build up and dig further my interest. I really appreciate him guiding me on my research, and providing his technical advice on the projects I work on. I also want to thank him for being generous with his time and energy, and for encouraging me to continue exploring in antenna area.

I thank Prof. Shizhuo Yin for agreeing to be a member on my thesis committee, and providing me with feedback and encouragement. I was fortunate to take his course EE 522 in Spring 2012, and I appreciate his valuable opinions on my further researching life.

I thank Prof. Victor Pasko for helping me build a solid background of electromagnetic theory in his course EE 531 in Fall 2011. I also want to thank him for supporting during my whole graduate student time as the Graduate Program Coordinator.

Thank you to Sherrydawn Jackson, for being a patient Graduate Program Assistant, and for solving my entire academic affair related questions and confusions at any time.

I would also like to thank Jing Jiang, for working on the same topic as this thesis of mine with another numerical method in FEKO. Her work makes much academic contribution and helps me with the background.
Chapter 1

Introduction

1.1 Background

In the 1920s, experimental electrical engineering has experienced its first important period. Both DC and AC power were generated and successfully delivered. Lots of fundamental concepts were presented and explained, for example, in [1]-[3]. Interest of RF power amplifiers was greatly aroused. There was also heated discussion in techniques that could extract a circuit topology, given a transfer function and the driving point impedance functions.

In [4], both the T network and the Pi network were adopted as matching networks at the output side of a RF amplifier. Such circuits, as mentioned in the paper, could couple the power from the output of the amplifier to the load, which is an antenna. Some give their understanding of impedance matching concepts from a mathematical perspective [5]. There was also specific discussion about using antenna impedance, which, in most cases, is complex [6]. These are the beginnings of impedance matching theory.
1.2 Impedance Matching Theory

In antenna engineering, to maximize the signal power transfer or minimize the reflection from the load are prime goals. Power passes through a network from a source to a load generally through a sequence of two-port networks. Maximum power transfer is obtained when the Thevenin equivalent impedance of a source and load are matched. Specifically, the load should be the complex conjugate of the impedance that the load sees looking back toward the source. At radio and microwave frequencies, where lots of antennas work, generally complex impedances need to be matched.

Figure 1-1: A load connected to a transmission line through a matching network

An impedance matching problem is illustrated in Figure 1-1. In this figure, \( Z_0 \) is the characteristic impedance of the transmission line, or more generally, the feed. \( Z_{load} \) is the load impedance, and \( Z_{in} \) stands for the input impedance looking towards the load. The matching network is regarded ideal, so no power is lost. Another advantage of impedance matching is the SNR (Signal-to-Noise Ratio) of the whole system can be improved. Moreover, when the real part of \( Z_{load} \) is nonzero, a matching network between \( Z_{load} \) and \( Z_0 \) can always be found.

Impedance matching at a single frequency is fairly easy. The found networks usually work well over a 5% or less bandwidth. However, matching a load over a wider band of frequencies is desired more often. Design becomes much more difficult as desired bandwidth
increases. Some matching networks require an adjustment part to match variable load impedance as frequency changes.

According to [7], depending on the application, an impedance matching network may consist of

(a) Lumped elements only. These are the smallest networks, but have the most stringent limit on frequency of operation. The relatively high resistive losses of inductors limit their performance.

(b) Distributed elements (micro strip-line elements or whatever the appropriate transmission medium might be) only. These have excellent performance, but their size restricts their use to above a few Gigahertz.

(c) A combination of lumped and distributed elements, primarily small sections of transmission lines with capacitors. The line lengths are smaller than with the distributed elements only, but higher performance than lumped elements on their own.

(d) Ad hoc solutions (suggested by input impedance behavior and features of various components).

There are generally two approaches in designing matching networks. One is to develop design equations and synthesis of desired results. The other one is to choose a circuit topology and then use a circuit optimizer to arrive at circuit values that yield the desired characteristics. In this thesis, the second approach is adopted.
1.3 Bode-Fano Limit

In the 1930s-1950s, Bode and Fano published their works about the theoretical limits of bandwidth that can be achieved [8]-[10]. When Bode addressed this limit considering a simple load in a single matching problem, communication engineers were provided with a frequency band bound for a given load. This is a tool to compare the performance of a designed impedance matching network with a certain bandwidth. This set up the basis for broadband impedance matching. Later, Fano turned this impedance matching problem into a filter design problem and worked with more general loads. Then the Bode-Fano limits, or the Bode-Fano criteria were addressed.

Figure 1-2: Circuits with simple reactive loads.
(a) Parallel RC load. (b) Parallel RL load. (c) Series RL load. (d) Series RC load.
The Bode-Fano limits for the four simple load circuits shown in Figure 1-2 are:

(a) Parallel RC load: \[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi}{R(\omega_0 C)} \]  

(b) Parallel RL load: \[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi(\omega_0 L)}{R} \]  

(c) Series RL load: \[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi R}{(\omega_0 L)} \]  

(d) Series RC load: \[ BW \frac{1}{\Gamma_{avg}} \leq \pi R(\omega_0 C) \]  

Here, \( \Gamma \) is the reflection coefficient looking into the matching network. \( \Gamma_{avg} \) is the average absolute value of \( \Gamma(\omega) \). \( \frac{BW}{\omega_0} \) is the fractional bandwidth of the matching network. To state this simply, the Bode-Fano limits can be written in terms of reactance and susceptance:

\[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi G}{B} \]  

\[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi R}{X} \]  

where G is the load conductance, B is the load susceptance, and X is the load reactance. This can be written in terms of the load quality factor Q as follows:

\[ BW \frac{1}{\Gamma_{avg}} \leq \frac{\pi}{Q} \]  

It can be seen from the above expressions that the more reactive energy that is stored in a load, the narrower the bandwidth of a match. The higher the Q is, the narrower the bandwidth of the match for the same average in-band reflection coefficient. Accordingly, it will be much harder
to design the matching network to achieve a specified matching bandwidth. Only when the load is purely resistive can a match over all frequencies be found.

With lumped elements to construct the matching network, L and C elements can satisfy the match over a finite bandwidth. With more L and C elements, the bandwidth can be relatively increased. In this thesis, the MATLAB approach allows the designer to choose the topology of the matching network. For each element position, there can be two L or C elements at most. All together there would be 12 elements. This results in a good match over fairly broad bandwidth.
1.4 Previous Approaches To Antenna Impedance Matching Problems

The classical broadband matching theory deals with the proper design of the lossless matching networks between the prescribed terminations. As discussed before, the matching network is needed to provide the maximum power transfer for the received signal to the end-user over the desired frequency band. Single frequency matching has been considered in the classical literature when designing matching networks.

The power transfer capability of the matching network can be best measured with the transducer power gain $T$, which is defined as the ratio of power delivered to the load $P_L$ to the available power $P_A$ of the generator. Here we are interested in $T$ over a wide band, which is:

$$T(\omega) = \frac{P_L(\omega)}{P_A(\omega)}$$

(1-8)

Under ideal circumstances, we require the flat transducer power gain characteristic in the operation band, at a same gain level with sharp rectangular roll-off, as shown in Figure 1-3. However, in reality we can only achieve ideal power transfer at one frequency. A good matching network design gives a relatively high and flat gain characteristic over the desired frequency band.

[Figure 1-3: Ideal power transmission over desired frequency band]
In [11], three design methodologies are introduced to solve the impedance matching problems:

(a) Single Matching. At a single frequency, a system is designed to make sure both input impedance of the receiver and output impedance of the transmitter are approximately equal to 50 Ω. In this case, the prime task is to construct a lossless two-port between the generator and the load.

(b) Double Matching. As the desired bandwidth increases, the input impedance of the receiver and the output impedance of the transmitter can no longer equal to 50 Ω at all time. Both source impedance and load impedance might be complex. The design of matching network must consider both complex terminations for optimum power transfer.

(c) Filter of Insertion Loss Problem. The classic filter design problem can be regarded as an impedance matching problem with a resistive source and load. The key is to limit the power transfer over an operational band. In this case, a lossless network must be constructed in between the resistive source and a resistive load. Far-end filter elements can also be considered as part of the source and load networks.

To solve broadband impedance matching problems, there are mainly two approach categories: Analytic Approach and Computer Aided Design Approach.
1.4.1 Analytic Approach

Gain-Bandwidth Theory is a classical procedure to solve impedance matching problems. A famous theory is broadly adopted in this approach: The Darlington Theorem.

![Darlington Representation of the load impedance](image)

Figure 1-4: Darlington Representation of the load impedance

Figure 1-4 shows a lossless two port, named the Darlington Equivalent. In the Darlington Theorem, any positive real impedance or admittance functions, or corresponding bounded real reflection coefficients can be synthesized as a lossless two port terminated into a unit resistance. The analytic approach to single matching problems was first addressed by Fano, based on Bode’s limitations and the Darlington Equivalent. Later, Youla developed the concept of complex normalization, providing an excellent solution to single matching problems.

Yarman and Carlin [12] accomplished the complete analytic solution to the double matching problem. This theorem enables designers to fully describe the doubly matched system in terms of real normalized scattering parameters after replacing the source and load with corresponding Darlington equivalents.

The progress in the analytic approach requires the designer to approximate the load with an equivalent circuit model. Moreover, this approach works well only for simple load types. As the wireless revolution is sweeping across the globe today, more complex load types and circuit configurations need to be considered in impedance matching. The analytic approach can no
longer satisfy the growing demand. In this thesis, no deeper discussion of the analytic approach will be presented.

1.4.2 Computer Aided Design Approach

On one hand, in impedance matching problems, a large amount of optimization, complex number calculation and comparing cannot be avoided. On the other hand, the analytic approach is limited in solving impedance matching situations with complex loads. Thus, the Computer Aided Design Approach is becoming more popular.

Computer Aided Design Approaches are accomplished by means of numerical optimizations, which work on the transducer power gain in the desired frequency band. In practical systems, mainly two categories of Computer Aided Design Approaches are studied: Brute-force techniques and the Real Frequency technique.

**Brute-force Technique**

In the Brute-force technique, the designer selects the circuit topology for the matching network, and then determines the element values by means of optimization. This technique deals with nonlinear optimization, and can result in wideband matching networks for antennas in most cases. The Brute-force technique can give satisfactory solutions for most narrow bandwidth impedance matching problems.
The Real Frequency Technique directly works on the generation of realizable network functions. The numerical set-up of this technique optimization is always stable and convergent. This is a practical and easy to use new technique, which has aroused great interest among researchers in recent years.

The Carlin’s Real Frequency Technique [13] is a numerical technique gain-bandwidth optimization. Unlike the Analytic Approach, it does not need any approximation of equivalent load impedance circuits. This Real Frequency Technique relies on the load’s actual measurement data. According to a comprehensive review by Newman [14], the Real Frequency Technique requires non-unique operations with rational polynomial approximations, and further extraction of equalizer parameters using the Darlington procedure. Moreover, a transformer is required to match the obtained equalizer to the fixed generator resistance of 50 \( \Omega \).

The MATLAB-based approach addressed in this thesis has similarities with the Brute-force technique and Real Frequency Technique respectively. This approach allows the designer to select the topology and then determine the element values. Also, it does not need circuit approximation, but uses actual measurement impedance data from an antenna to get the optimized matching network. The approach in this thesis can be regarded to have a combination of the advantages of both of the above techniques.
1.5 The Approach of Impedance Matching Optimization Using MATLAB

The Global Optimization Toolbox™ in MATLAB is the main numerical tool used in this thesis. This MATLAB approach allows the designer to select from several basic topologies, including the L network, the T network, the Pi network and networks with 5 or 6 element positions. Then the algorithm fills each element position with one of the following four basic lumped element circuits:

![Four basic lumped element circuit](image)

Figure 1-5: Four basic lumped element circuit to fill in topology positions

The optimization solver is set up with a start point containing the initial guess of all elements, and an objective function calculating the maximum of the VSWR over the desired frequency band. With each filled topology, the minimum of the objective function is found and stored. After completing all filling possibilities, the minimums are compared to find the smallest one, which is presented with all element values and basic circuit configurations. This result would be the best matching network under the selected topology.

As described above, the load impedance used to deduce the objective function comes from actual dipole antenna simulation data, which would save much progressing time and gives
more reliable results than by using an equivalent circuit. Also, the designer can balance between the space available and the desired bandwidth by controlling the number of topologies.

1.6 Thesis Organization

The thesis is organized as follows:

In Chapter 2 the Global Optimization Toolbox in MATLAB is introduced. The main part is the detailed explanation about each solver and other options in the toolbox and their application conditions. Then the impedance matching problem is defined according to the toolbox’s input and output. Chapter 3 gives three kinds of basic impedance matching network topologies, and introduces briefly more-elements topologies.

Chapter 4 and 5 make up the core of the thesis. Chapter 4 introduces assistant tools used in this thesis: Impedance Matching Calculator and Smith Chart. After detailed comparison between several potential solvers, Chapter 4 gives the final solver used. Chapter 5 describes the actual modeling using this solver. The optimization is performed and results are presented.

Chapter 6 concludes the whole thesis and gives suggestions for future research directions.
Chapter 2

Global Optimization Toolbox\textsuperscript{TM} in MATLAB

In this thesis, the numerical tool used is the Global Optimization Toolbox\textsuperscript{TM} in MATLAB. Over a broad interested frequency band, an antenna would present different complex impedances at different frequencies. According to earlier work of Jing Jiang on this topic with FEKO [15], one single optimization process might take several hours to finish on a computer with an Intel Pentium 4 Processor. It takes such a long time because at each frequency point, FEKO would simulate the antenna as well as the matching circuit and then perform the optimization. Maxwell’s equations are involved at every frequency point for the electric field and magnetic field intensities. In this thesis, the input is the complex impedance values of the same dipole at different frequency points. The task for MATLAB optimization is to find the minimum of a function at each impedance value. The electromagnetic simulation in FEKO is turned into a minimum finding problem in MATLAB. Thus much time is saved and more simulation times are feasible. It only takes several minutes even for the six-element topology to get the optimization result on a computer with Intel Core i5 processor.
2.1 Global Optimization Toolbox™

According to MathWorks [16], the Optimization Toolbox™ provides widely used algorithms for standard and large-scale optimization. These algorithms solve constrained and unconstrained continuous and discrete problems. The toolbox’s optimization software includes functions for linear programming, quadratic programming, binary integer programming, nonlinear optimization, nonlinear least squares, systems of nonlinear equations, and multi-objective optimization. You can use them to find optimal solutions, perform tradeoff analyses, balance multiple design alternatives, and incorporate optimization methods into algorithms and models.

The key features of the toolbox are as follows:

(a) Interactive tools for defining and solving optimization problems and monitoring solution progress.

(b) Global search and multi-start solvers for finding single or multiple global optima.

(c) Genetic algorithm solver that supports linear, nonlinear, and bound constraints.

(d) Multi-objective genetic algorithm with Pareto-front identification, including linear and bound constraints.

(e) Pattern search solver that supports linear, nonlinear, and bound constraints.

(f) Simulated annealing tools that implement a random search method, with options for defining the annealing process, temperature schedule, and acceptance criteria.

(g) Parallel computing support in multi-start, genetic algorithm, and pattern search solver.

(h) Custom data type support in genetic algorithm, multi-objective genetic algorithm, and simulated annealing solvers.

Here this section will briefly introduce how easy it is to use the Optimization Toolbox. Figure 2-1 shows the interface of the Optimization Toolbox. It is a clear and simple tool for
common optimization problems. After defining a certain optimization problem, the user can select a solver accordingly. Selecting a solver is the most important step when dealing with an optimization problem, which will be explained in detail in 2.2. It should be indicated that the interface is slightly different when different solvers are selected. The objective function needs to be provided to find the maximum or minimum, or for curve fitting purposes. Then, the start point or variables can be set. Below is the constraint setting. Optimization options and their default values can be set at the right column. After running the program, the final results are clearly presented, and the times of iterations are shown.

![The interface of the Optimization Toolbox when solver is Genetic Algorithm](image.png)

Figure 2-1: The interface of the Optimization Toolbox when solver is Genetic Algorithm

The toolbox makes it much quicker and easier to solve optimization problems. It could help define and modify problems quickly, use the correct syntax for optimization functions, import and export from the MATLAB workspace, and generate code containing the configuration
for a solver and options. One can also change parameters of an optimization during the execution of certain Global Optimization Toolbox functions.

2.2 Global Optimization Solvers in Toolbox

Optimization can be used in the process of finding the point that minimizes a function. A local minimum of a function is a point where the function value is smaller than or equal to the values at nearby points, but possibly greater than at a distant point. A global minimum is a point where the function value is smaller than or equal to the values at all other feasible points. There is an Optimization Toolbox™ and a Global Optimization Toolbox™ among all the toolboxes in MATLAB. Generally, the Optimization Toolbox solvers find a local optimum. In this thesis, we need to find the global minimum of a nonlinear function. Global Optimization Toolbox solvers are designed to search through more than one basin of attraction. The solvers search in different ways:

(a) GlobalSearch and MultiStart generate a number of starting points. They then use a local solver to find the optima in the basins of attraction of the starting points.

(b) ga (Genetic Algorithm) uses a set of starting points and iteratively generates better points from the population. As long as the initial population covers several basins, ga can examine several basins.

(c) simulannealbnd (Simulated Annealing) performs a random search. Generally, simulannealbnd accepts a point if it is better than the previous point. simulannealbnd occasionally accepts a worse point, in order to reach a different basin.

(d) patternsearch looks at a number of neighboring points before accepting one of them. If some neighboring points belong to different basins, patternsearch in essence looks in a number of basins at once.
There is a sixth solver in the Global Optimization Toolbox named `gamultiobj`, which is not used to find a minimum. Therefore no more discussion about this solver is necessary.

Selecting a solver is based on problem characteristics and on the type of solution you want. Table 2-1 illustrates the solver characteristics in the Global Optimization Toolbox. As the interested problem is definitely nonlinear and nonsmooth, we could choose `simulannealbnd` as the solver. On one hand, when looking for the global optimum, we could sacrifice a little time to get the optimized result. Referring to the Convergence column, it is obvious that the solver `simulannealbnd` satisfies this need perfectly. On the other hand, `simulannealbnd` cannot run in parallel, which will slow down the solving process more. Moreover, the user has to supply the solver with start points manually. In order to be complete, in Chapter 4, `ga` and `simulannealbnd` will be compared before making the final selection.
Table 2-1: Solver Characteristics in Global Optimization Toolbox

<table>
<thead>
<tr>
<th>Solver</th>
<th>Convergence</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalSearch</td>
<td>Fast convergence to local optima for smooth problems.</td>
<td>Deterministic iterates; Gradient-based; Automatic stochastic start points;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removes many start points heuristically</td>
</tr>
<tr>
<td>MultiStart</td>
<td>Fast convergence to local optima for smooth problems.</td>
<td>Deterministic iterates; Can run in parallel; Gradient-based;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stochastic or deterministic start points, or combination of both;</td>
</tr>
<tr>
<td>patternsearch</td>
<td>Proven convergence to local optimum, slower than gradient-based solvers.</td>
<td>Deterministic iterates; Can run in parallel; No gradients;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User-supplied start point</td>
</tr>
<tr>
<td>ga</td>
<td>No convergence proof.</td>
<td>Stochastic iterates; Can run in parallel; Population-based; No gradients;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Automatic start population, or user-supplied population, or combination of both</td>
</tr>
<tr>
<td>simulannealbnd</td>
<td>Proven to converge to global optimum for bounded problems with very slow</td>
<td>Stochastic iterates; No gradients; User-supplied start point; Only bound</td>
</tr>
<tr>
<td></td>
<td>cooling schedule.</td>
<td>constraints</td>
</tr>
</tbody>
</table>
2.3 Options Setting in Global Optimization Toolbox™

The options structure contains options used in the optimization routines. If, on the first call to an optimization routine, the options structure is not provided, or is empty, a set of default options is generated. Some of the default options values are calculated using factors based on problem size. Some options are dependent on the specific solver algorithms.

Control options can be changed using the `optimset` function. An options structure is created and some minimization parameters are set. Then options can be passed as an input to the optimization objective function. Table 2-2 shows some common options used in optimization.

Table 2-2: Common Options in Global Optimization Toolbox

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>A flag indicating whether intermediate steps appear on the screen.</td>
<td>‘notify’; ‘iter’; ‘off’; ‘final’</td>
<td>‘notify’</td>
</tr>
<tr>
<td>FunValCheck</td>
<td>Check whether objective function and constraints values are valid.</td>
<td>‘on’; ‘off’</td>
<td>‘off’</td>
</tr>
<tr>
<td>MaxFunEvals</td>
<td>The maximum number of function evaluations allowed.</td>
<td></td>
<td>500</td>
</tr>
<tr>
<td>MaxIter</td>
<td>Maximum number of iterations allowed.</td>
<td></td>
<td>500</td>
</tr>
<tr>
<td>OutputFcn</td>
<td>Display information on the iterations of the solver.</td>
<td>[] (none)</td>
<td></td>
</tr>
<tr>
<td>PlotFcns</td>
<td>Plot information on the iterations of the solver.</td>
<td>[] (none)</td>
<td></td>
</tr>
<tr>
<td>TolFun</td>
<td>The termination tolerance for the function value.</td>
<td>1.e-4</td>
<td></td>
</tr>
<tr>
<td>TolX</td>
<td>The termination tolerance for x.</td>
<td>1.e-4</td>
<td></td>
</tr>
</tbody>
</table>

With both `simulannealbnd` and `ga` solvers, we could set all the above options with `optimset`. Besides, these two solvers have special options respectively.
2.3.1 Simulated Annealing Options

The temperature parameter used in simulated annealing controls the overall search results. The temperature for each dimension is used to limit the extent of search in that dimension. The toolbox lets you specify initial temperature as well as ways to update temperature during the solution process. Control options can be set using the `saoptimset` function. The two temperature-related options are the `InitialTemperature` and the `TemperatureFcn`. We can also use `ReannealInterval` to control the reannealing process. `InitialTemperature` can be set to a vector of length less than the number of variables (dimension); the solver expands the vector to the remaining dimensions by taking the last element of the initial temperature vector.

The default temperature function used by `simulannealbnd` is called `temperatureexp`. In the `temperatureexp` schedule, the temperature at any given step is 0.95 times the temperature at the previous step. This causes the temperature to go down slowly at first but ultimately get cooler faster than other schemes. If another scheme is desired, e.g. Boltzmann schedule or “Fast” schedule annealing, then `temperatureboltz` or `temperaturefast` can be used, respectively.

Reannaling is a part of annealing process. After a certain number of new points are accepted, the temperature is raised to a higher value in hope to restart the search and move out of a local minimum. Performing reannealing too soon may not help the solver identify a minimum, so a relatively high interval is a good choice. The interval at which reannealing happens can be set using the `ReannealInterval` option.
2.3.2 Genetic Algorithm Options

The default initial population is created using a uniform random number generator. Default values for the population size and the range of the initial population are used to create the initial population. Using the Genetic Algorithm Solver, we could pass PopulationSize and PopInitRange to gaoptimset.

The default population size used by ga is 20. This may not be sufficient for problems with a large number of variables; a smaller population size may be sufficient for smaller problems.

The initial population is generated using a uniform random number generator in a default range of [0, 1]. This creates an initial population where all the points are in the range 0 to 1. The initial range can be set by changing the PopInitRange option using gaoptimset. The range must be a matrix with two rows. To specify a different initial range for each variable, the range must be specified as a matrix with two rows and 'numberOfVariables' columns.
2.4 Defining an Optimization Problem from Network Matching

In this approach of impedance matching using MATLAB, the designer is allowed to choose a topology, and then all possible networks in this topology are listed. For each network, the optimized lumped-element values are plugged in to get the bandwidth, and the network with the widest bandwidth is stored. Here each network needs one optimization.

In a fixed-topology impedance matching problem, the goal is to find the lumped-element values to make the bandwidth of a certain antenna widest. The bandwidth is determined by the range where the VSWR (Voltage Standing Wave Ratio) is below 2:1. Therefore, the maximum of the VSWR over the interested bandwidth needs to be minimized. Here we get our objective function for this optimization problem: the expression of the VSWR.

As in Figure 2-2, we will take the simplest L network as an example.

Figure 2-2: An L network where a capacitor is in shunt with the load, and then in series with an inductor
2.4.1 Writing the Objective Function

The input impedance $Z_{in}$ looking from the source side is

$$Z_{in} = Z_L + Z_C / |Z_{load}|$$

(2-1)

With $Z_L = j\omega L$ and $Z_C = \frac{1}{j\omega C}$, we get

$$Z_{in} = j\omega L + \frac{1}{j\omega C} \cdot \frac{Z_{load}}{1 + j\omega C Z_{load}}$$

(2-2)

$$Z_{in} = j\omega L + \frac{Z_{load}}{1 + j\omega C Z_{load}}$$

(2-3)

The reflection coefficient $\Gamma$ is

$$\Gamma = \frac{Z_{in} - Z_{source}}{Z_{in} + Z_{source}}$$

(2-4)

$$\Gamma = \frac{j\omega L + \frac{Z_{load}}{1 + j\omega C Z_{load}} - Z_{source}}{j\omega L + \frac{Z_{load}}{1 + j\omega C Z_{load}} + Z_{source}}$$

(2-5)

$$\Gamma = \frac{j\omega L(1 + j\omega C Z_{load}) + Z_{load} - Z_{source}(1 + j\omega C Z_{load})}{j\omega L(1 + j\omega C Z_{load}) + Z_{load} + Z_{source}(1 + j\omega C Z_{load})}$$

(2-6)

The Voltage Standing Wave Ratio is

$$\text{VSWR} = \frac{V_{max}}{V_{min}} = \frac{1 + |\Gamma|}{1 - |\Gamma|}$$

(2-7)

Thus the objective function is max (VSWR).
2.4.2 Writing Constraints

There are generally four types of constraints in the toolbox, as showed in Table 2-3. In impedance matching design using lumped-elements, the constraints are relatively simple. We choose lower and upper bounds for individual elements.

Table 2-3: Types of Constraints in Global Optimization Toolbox

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bound Constraints</td>
<td>$x \geq l$ and $x \leq b$</td>
</tr>
<tr>
<td>Linear Inequality Constraints</td>
<td>$A \cdot x \leq b$</td>
</tr>
<tr>
<td>Linear Equality Constraints</td>
<td>$Aeq \cdot x = beq$</td>
</tr>
<tr>
<td>Nonlinear Constraints</td>
<td>$c(x) \leq 0$ and $ceq(x) = 0$</td>
</tr>
</tbody>
</table>

Considering common values for inductors and capacitors are quite small (in $10^{-12}$ to $10^{-9}$ order of magnitude), we use values of capacitive reactance $X_c$ and inductive reactance $X_L$ instead of values of capacitance C and inductance L. For example, at the frequency of 3.7MHz, C, L and their corresponding reactance are shown in Table 2-4. Considering typical values of L and C used in an impedance matching network in microwave frequency band, a reactance range from 1 Ohm to 1000 Ohms could cover most possible values of L and C.

Table 2-4: Capacitance, Inductance and Corresponding Reactance Values at 3.7MHz

<table>
<thead>
<tr>
<th>C/pF</th>
<th>43010</th>
<th>430.1</th>
<th>43.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>L/nH</td>
<td>43.01</td>
<td>4301</td>
<td>43010</td>
</tr>
<tr>
<td>X/Ohms</td>
<td>1</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>
2.4.3 Setting Options and Passing Parameters

We are interested in options that affect optimization results directly. Two groups of the options shown in Table 2-2 need to be studied: tolerance and stopping criteria. The options about tolerance include TolFun and TolX, both of whose default values are set at 1e-4. The options about stopping criteria include MaxFunEvals and MaxIter, with default values 500. Since input X and function value VSWR are both in range greater than 1, 1e-4 could meet the accuracy demand perfectly. However, we need to increase stopping criteria to larger values, like 2000, to avoid stopping the program before the optimized \( L \) and \( C \) values are found. For everything else, the default values are adopted.

The Objective Function Equation 2-8 has three variables: \( \omega \), \( L \) and \( C \). The load impedance \( Z_{load} \) is determined by the frequency, therefore can be seen as one variable with \( \omega \). What needs to be optimized in the Objective Function are \( L \) and \( C \) values, instead of all variables. In this situation, an anonymous handle is needed to pass the objective function to the optimization solvers. A function handle is used to call functions indirectly. Putting \( L \) and \( C \) in one vector \( x \), we can write the handle as:

\[
handle = @(x)Objective\_Function(x,omega)
\]

where omega stands for the variable we do not want to optimize.
Chapter 3

Basic Impedance Matching Network Topologies

When impedance matching is performed between two resistive levels, the network could be simple. An ideal transformer shown in Figure 3-1 is the most straightforward way to match two resistances. The constitutive relations of this ideal transformer are:

\[ V_1 = nV_2 \]  

and \[ I_1 = -\frac{1}{n} I_2 \]

We could get the matching relations between the resistances:

\[ R_{in} = \frac{V_1}{I_1} = -n^2 \frac{V_2}{I_2} = n^2 R_L \]

where \( n \) is the winding ratio of the transformer.

Figure 3-1: An Ideal Transformer

In real life, most impedance matching problems are about complex loads, like transistors, mixers, and antennas. Lumped elements are used to transform one complex impedance to another.
The reactive elements—capacitors and inductors—have different performances when connected in different ways: in shunt or in series.

Considering the real part of the impedances, when connected in series, a reactive element can transform a smaller resistance up to a larger value. When connected in shunt, a larger resistance is transformed to a smaller one. Adding a reactive element in shunt will resonate the imaginary part, and one in series will cancel the imaginary part. All these conclusions could be deduced from basic circuit laws, therefore the steps are omitted here.

Overall, the majority of impedance matching designs fall into some basic topologies. We will discuss some common matching networks regarding their topologies in this chapter.

3.1 L Network

From the above theory, we could use one reactive element to achieve the real part value, and use another element to deal with the imaginary part resulting from the first element. One of the two elements should be in shunt with the load, and the other one should be in series with it. Therefore, we could match any two resistive values by using only two reactive elements, one in shunt and the other in series. This is an L network. In all L networks, one of the elements should be a capacitor, and the other should be an inductor. The reason will be presented below.
3.1.1 Use L network to match two resistive impedances

Figure 3-2: L network topology for $R_{\text{source}} < R_{\text{load}}$

Figure 3-3: L network topology for $R_{\text{source}} > R_{\text{load}}$

Figure 3-2 and 3-3 show two types of L network topologies. The load impedance is $Z_{\text{load}} = R_{\text{load}} + jX_{\text{load}}$, and the source impedance is $Z_{\text{source}} = R_{\text{source}} + jX_{\text{source}}$. We have design equations for matching two resistive impedances.

For networks where $R_{\text{source}} < R_{\text{load}}$:

$$|Q_1| = |Q_2| = \sqrt{\frac{R_{\text{load}}}{R_{\text{source}}} - 1}$$  \hspace{1cm} (3-4)
Here \( Q_1 = \frac{X_1}{R_{source}} \), and \( Q_2 = \frac{R_{load}}{X_2} \).

For networks where \( R_{source} > R_{load} \):

\[
|Q_1| = |Q_2| = \sqrt{\frac{R_{source}}{R_{load}}} - 1
\]

(3-5)

Here \( Q_1 = \frac{R_{source}}{X_1} \), and \( Q_2 = \frac{X_2}{R_{load}} \).

In an L matching network, there is always one inductor and one capacitor. In other words, \( Q_1 = -Q_2 \). Also when \( R_{source} \) and \( R_{load} \) are given, \( Q \) is fixed. The designer does not have a choice of circuit \( Q \).

3.1.2 Use L network to match complex impedances

Absorption and Resonance are two basic approaches when it comes to complex load values. Through absorption, the network could eliminate any stray reactance by careful placement of each matching element. Through resonance, at the interested frequency, the stray can be resonated by an equal and opposite reactance. Most impedance matching designs are certain combinations of resonance and absorption. However, if the stray reactance value in the network is larger than the element value calculated, then absorption cannot be used. Instead, resonance and absorption must be used together.
3.2 T Network

Introducing a third element into the L matching network provides the designer with more freedom to adjust $Q$. We have two topologies using three elements, T and Pi. In specific designing, the topology chosen depends on the realization constraints and the reactive parts of the source and load impedances. Absorption or resonance could be selected to cancel out the reactive part of those impedances.

With given source and load impedances, the circuit $Q$ is the minimum available when using an L network to match the impedances, which is $Q = \sqrt{\frac{R_{\text{source}}}{R_{\text{load}}}} - 1$. Now adding one more element, the $Q$ will be more controllable, and so will be the bandwidth. Generally three-element topologies are used for narrowband applications. However, we could change them a little in order to get larger bandwidth.

When two L networks are placed back-to-back, with their parallel legs connected, a T network is constructed, as shown in Figure 3-4.

![Figure 3-4: T Network topology](image)
A T network can be thought of as two L networks with a virtual resistance $R_v$ between the parallel legs of them, as shown in Figure 3-5. $R_v$ should be selected greater than the maximum of $R_{source}$ and $R_{load}$, and the relationship between $R_v$ and $Q$ is

$$Q = \sqrt{\frac{R_v}{\min(R_{source}, R_{load})}} - 1$$

(3-6)

In order to get larger bandwidth, we could choose a small $Q$ value. $R_v$ could then be calculated. Then the design approach used in L network could be adopted twice, for the left part and right part of T network respectively. In this way, we can get four final design plans.

![Figure 3-5: T network as two back-to-back L networks with a virtual resistance in between](image)
3.3 Pi Network

When two L networks are placed back-to-back, with their series legs connected, a Pi network is constructed, as shown in Figure 3-6.

![Figure 3-6: Pi network topology](image)

We could think of the Pi network as two L networks with a virtual resistance $R_v$ placed in the middle, as shown in Figure 3-7. The designing of a Pi network can be done accordingly: first design the left L network, then design the right L network. Select a $R_v$ smaller than the minimum of $R_{source}$ and $R_{load}$, because it is connected to the series leg of both L networks.

The circuit Q is $Q = \sqrt{\frac{\max(R_{source}, R_{load})}{R_v}} - 1$. Therefore, we could control the bandwidth by choosing the value of $R_v$. With a larger $R_v$ value, the bandwidth could be expanded. In other words, we could choose a relatively small $Q$ to get a large available bandwidth. The above equation can be used to get the value of $R_v$. 
In both T network design and Pi network design using the L network approach, multiple matching designs are possible. The specific final choice of the possible realizations depend on application factors. For example, the application may require passing or blocking DC current, or to eliminate stray reactances.

### 3.4 Networks with More Elements

If the load contains a nonzero reactance part, it will store some energy. Thus the bandwidth would be limited to a certain extent. If resonance and absorption are used in impedance matching, we could easily get a narrow bandwidth solution. Sufficient bandwidth is often the ultimate goal in impedance matching; therefore, we need to minimize the total stored energy. According to textbooks, “Roughly the total energy stored will be proportional to the sum of the magnitudes of the reactances in the circuit”[16]. We can expand the bandwidth by incorporating the load reactance into the matching network. Thus the choice of an appropriate matching network is crucial. There is an observation that for a complex load modeled as a series resistance and reactance, the energy stored is proportional to the reactance value and the energy delivered to the load is proportional to the resistance [17].

We could conclude from above that although more elements could make sure all stray reactance are absorbed or resonated; too many matching elements cannot guarantee a large
bandwidth. Therefore, we only cover L network, T network and Pi network topologies in this thesis. If in the future, any designs are based on this impedance matching approach based on MATLAB, the transition would be simple. If not, the maximum number of elements per topology block being 2, a Pi network could contain 6 elements at most. With these many elements, we could say the balance point of bandwidth and matching has already been passed.

Cascading simple networks like L networks or T networks could generate topologies with more elements. But two or more adjacent elements with the same signs of reactance could be replaced by one element. For example, a 3nH inductor and a 44nH inductor in series can be replaced by a 47nH inductor; a 1pF capacitor and a 4pF capacitor in parallel can be replaced by a 5pF capacitor. Therefore, when constructing cascaded networks, these situations should be avoided. Obviously, the back-to-back configuration design should be passed. Two or more series-connected L networks can be used for broadband impedance matching, as shown in Figure 3-8.

![Cascaded L networks](image)

Figure 3-8: Cascaded L networks

Designing cascaded L networks still need virtual resistors. Every $R_v$ should be chosen between $R_{source}$ and $R_L$. For example, when we design a network with two cascaded L networks, we can get $Q$'s for them:

$$Q_1 = \sqrt{R_v} \sqrt{\frac{1}{\min(R_{source}, R_{load})}} - 1$$

(3-7)
and \[ Q_2 = \sqrt{\frac{\max(R_{\text{source}}, R_{\text{load}})}{R_V}} - 1 \] (3-8)

When \( Q \) is minimum, the bandwidth could get its maximum. In this case, the minimum of \( Q \) is

\[ Q_1 = Q_2 = \sqrt{\frac{R_V}{\min(R_{\text{source}}, R_{\text{load}})}} - 1 = \sqrt{\frac{\max(R_{\text{source}}, R_{\text{load}})}{R_V}} - 1 \] (3-9)

\( R_V = \sqrt{R_{\text{load}}R_{\text{source}}} \) gives the widest frequency band. The rest of this design could refer to T or Pi network design steps.

When more than two L networks are cascaded, even wider bandwidth could be obtained. Similarly, we have

\[ R_{\text{source}} < R_{V1} < R_{V2} \ldots < R_{Vn-1} < R_{\text{load}} \] (3-10)

To get the optimum bandwidth,

\[ \frac{R_V}{R_{\text{source}}} = \frac{R_{V2}}{R_{V1}} = \frac{R_{V3}}{R_{V2}} = \ldots = \frac{R_{\text{load}}}{R_{Vn-1}} \] (3-11)

\( Q \) is given by

\[ Q_1 = Q_2 = \sqrt{\frac{R_{V1}}{R_{\text{source}}}} - 1 = \sqrt{\frac{R_{V2}}{R_{V1}}} - 1 = \ldots = \sqrt{\frac{R_{\text{load}}}{R_{Vn-1}}} - 1 \] (3-12)
3.5 Topology Used in this Thesis

The approach presented in this thesis allows the designer to choose from the following topologies: two types of L networks, T network and Pi network. We take the source impedance as 50 Ohms, which is a typical characteristic impedance of transmission lines. The load impedance is imported from a text file, with actual impedance data from an 80-meter half-wavelength dipole antenna. The frequency band given is from 3.40 MHz to 4.00 MHz, every 25 kHz. Table 3-1 gives part of the data sheet. The whole table is attached in Appendix A.

Table 3-1: An 80-meter dipole antenna impedance file before matching (Part)

<table>
<thead>
<tr>
<th>Frequency/MHz</th>
<th>Real(Z)/Ohms</th>
<th>Imag(Z)/Ohms</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.45</td>
<td>82.386</td>
<td>-58.658</td>
</tr>
<tr>
<td>3.50</td>
<td>85.532</td>
<td>-29.406</td>
</tr>
<tr>
<td>3.55</td>
<td>88.736</td>
<td>-0.204</td>
</tr>
<tr>
<td>3.60</td>
<td>92.002</td>
<td>28.981</td>
</tr>
<tr>
<td>3.65</td>
<td>95.331</td>
<td>58.180</td>
</tr>
<tr>
<td>3.70</td>
<td>98.728</td>
<td>87.429</td>
</tr>
<tr>
<td>3.75</td>
<td>102.194</td>
<td>116.762</td>
</tr>
<tr>
<td>3.80</td>
<td>105.736</td>
<td>146.215</td>
</tr>
<tr>
<td>3.85</td>
<td>109.358</td>
<td>175.823</td>
</tr>
<tr>
<td>3.90</td>
<td>113.067</td>
<td>205.625</td>
</tr>
</tbody>
</table>

Figure 3-9 indicates the impedance matching problem, and Figure 3-10 shows all the topologies the designer could choose from.
Then for each element block in the topology, the algorithm fills four kinds of basic circuits in it, as shown in Figure 1-5. For the L network for example, there are $4^2 = 16$ possible configurations. Among these there are networks where all elements are inductors, and those where all elements are capacitors, which cannot match the source and load successfully. However, the algorithm is designed to pick out the network with the widest bandwidth. The “wrong” networks could be ruled out automatically then.
Chapter 4

Solver Selection for Impedance Matching Problem

The core of this approach to solve impedance matching problems is the optimization solver, which is chosen from the Global Optimization Toolbox. In this chapter, we will discuss all global optimization algorithms provided in the Toolbox and their characteristics, by running impedance matching programs at a single frequency. We will also introduce two common impedance matching approaches, online impedance matching calculators and Smith Chart [18], to verify and compare with the results we get from the optimizers. The results are evaluated and help choose the solver that works the best.
4.1 Impedance Matching Calculator

Many impedance matching calculators can be found online with Google for example. Most of these calculators require the following parameters: Source Impedance, Load Impedance, Frequency and Desired Q. Some ask the designer to choose from L, T or Pi networks, and then fill in the parameters, like the calculator shown in Figure 4-1 [19]; others ask the parameters once and give all possible matching networks, like the ones shown in Figure 4-2 [20].

![Impedance matching calculator from Changpuak.ch](image)

Figure 4-1: Impedance matching calculator from Changpuak.ch
Figure 4-2: Impedance matching calculator from EEWeb.com
It is obvious that all these impedance matching calculators are based on the design steps discussed in Chapter 3, requiring a desired Q value. However, there is a common weakness in all of them: the matching network can only guarantee to work at the frequency given. No broadband impedance matching could be achieved using these calculators. Moreover, in some calculators, the network circuits are very limited in type.

In conclusion, online impedance matching calculators can only be used to get an approximate concept of the network design. They cannot perform broadband impedance matching; therefore designers today need full related knowledge in industrial designing. This is a waste of both time and an intelligence resource. However, we could use some online calculator for proving a given design at a certain frequency, because it strictly follows all the reliable basic design steps.

In this thesis, the following Impedance Matching Network Designer [21] is used for assistance purposes. This calculator is very accurate and straightforward, shown in Figure 4-3. More importantly, it gives the results for 16 networks with one calculation, including 4 L networks, 2 Pi networks, 2 T networks and 8 4-element networks. This calculator works as a great reference in the initial steps of the optimization approach addressed in this thesis, like selecting optimization solvers.
Impedance Matching Network Designer

(Now with 16 networks!)

Source Resistance: 50 Source Reactance: 0
Load Resistance: 100 Load Reactance: 0
Desired Q: 3 Frequency: 3.5e6

Please send comments and questions to John Wetherell at wetherel@eecs.berkeley.edu

HIGHPASS Hi-Low MATCHING NETWORK

LOWPASS Hi-Low MATCHING NETWORK

HIGHPASS Low-Hi MATCHING NETWORK

LOWPASS Low-Hi MATCHING NETWORK

Figure 4-3: Impedance Matching Network Designer by John Wetherell
4.2 Verifying Matching Results Using Smith Chart

The Smith Chart is a straightforward way in matching network design at a single frequency. It is very intuitive because everything is visualized, and the design can be done graphically. However, when our goal is to expand the frequency band through the network, Smith Chart will only serve as an assistant. In this thesis, the Smith Chart is used to verify the results from the impedance matching calculators, and sometimes is also used to test the MATLAB result. All these tests are only performed at single frequency.

The verification method will be illustrated in the following example. The source impedance is 50 Ohms, which is also regarded as the characteristic impedance. To make the process easier, we take the load impedance as 100 Ohms. The frequency is set at 300 MHz. We want to find a proper matching L network.

We put all knowns into the Impedance Matching Network Designer discussed in section 4.1. Figure 4-4 shows all L network results. Because the given load impedance 100 Ohms is greater than the source impedance 50 Ohms, there are no matching networks given for the first two- “Hi-Low Matching Network”. We will test the result of the Lowpass Low-Hi Matching Network here.

Figure 4-4: Designer results for matching 50 Ohms source to 100 Ohms load at 300 MHz
In the Smith Chart, the following parameters are set up:

Load impedance = 100 Ohms
Reference impedance = 50 Ohms
Frequency = 300 MHz

Figure 4-5 shows the Smith Chart result. We start from the load side and mark the load impedance as DP1. Adding a shunt capacitor of 5.305pF makes the point go to DP2. Then a series inductor of 26.525nH brings it back to the center, which is 50 Ohms. Since DP3 is where the source impedance is, this matching network design is correct.

The Smith Chart is a convenient tool to verify the matching network results. All network designs in this thesis presented have been verified by plotting in the Smith Chart.

Figure 4-5: Verifying L network results in the Smith Chart
4.3 Solve for Optimized Matching Network at Single Frequency

Now that we have all the tools, we could start programming to solve for matching networks. To begin with, we want to find the optimized values for the following L network in Figure 4-6:

![Image](image_url)

Figure 4-6: L network with a capacitor in shunt with the load, and then in series with an inductor

The given numbers are as follows: source impedance $Z_{\text{source}} = 50$ Ohm, load impedance $Z_{\text{load}} = 100$ Ohm. The objective function is VSWR, which we want to minimize. The equations are given as follows:

$$Z_{\text{in}} = Z_L + \frac{Z_C}{Z_{\text{load}}} \quad (4-1)$$

$$\Gamma = \frac{Z_{\text{in}} - Z_{\text{source}}}{Z_{\text{in}} + Z_{\text{source}}} \quad (4-2)$$

$$\text{VSWR} = \frac{V_{\text{max}}}{V_{\text{min}}} = \frac{1 + |\Gamma|}{1 - |\Gamma|} \quad (4-3)$$

The lower and upper bounds are set as: $\text{lb} = [1 \ 1]$, $\text{ub} = [1000 \ 1000]$ Ohms. We use the optimizer to find the best values of L reactance and C reactance. The MATLAB code can be found in Appendix F.
Take Solver *simulannealbnd* as an example, we could get VSWR=1.0068 when L = 2163nH, C = 428.6pF. When checking the result with the Impedance Matching Network Designer, it gives L = 2151nH, C = 430.1pF, which is fairly close with the MATLAB result. Figure 4-7 shows the matching result in Smith Chart. The matching path is exactly the same with that in Figure 4-6. The reason why L and C values are different is the frequency difference.

Figure 4-7: L network matching process showed in Smith Chart
4.4 Comparison between Solvers

A solver needs to be selected from the list in Chapter 2.2. We can compare their characteristics, and find one according to the impedance matching problem situation. The following are some main characteristics [16]:

(a) Convergence: Solvers can fail to converge to any solution when started far from a local minimum. When started near a local minimum, gradient-based solvers converge to a local minimum quickly for smooth problems. patternsearch provably converges for a wide range of problems, but the convergence is slower than gradient-based solvers. Both ga and simulannealbnd can fail to converge in a reasonable amount of time for some problems, although they are often effective.

(b) Iterates: Solvers iterate to find solutions. The steps in the iteration are iterates. Some solvers have deterministic iterates. Others use random numbers and have stochastic iterates. We could get the iterates from output structure of every algorithm.

(c) Gradients: Some solvers use estimated or user-supplied derivatives in calculating the iterates. Other solvers do not use or estimate derivatives, but use only objective and constraint function values.

(d) Start points: Most solvers require you to provide a starting point for the optimization. One reason they require a start point is to obtain the dimension of the decision variables. ga does not require any starting points, because it takes the dimension of the decision variables as an input. ga can generate its start population automatically.

As the impedance matching problem is not about pattern search, we will compare ga, patternsearch and simulannealbnd, and try to find the best solver for this problem.

First, unlike simulannealbnd and patternsearch, ga does not require start points. Solver ga is better in this case, but inputting start points by hand is not too much work, either. Second,
we can evaluate iteration and gradients by the times it cost to find the optimum, which can be presented after running the optimizer. Comparing the three algorithms, we can find all of them require only a few seconds to run at most, and the iteration times are several thousand. From this perspective, all algorithms will do fine. Finally, we could evaluate the convergence of the two algorithms by running them multiple times, and compare the results.

At 3.7 MHz, we are matching a 100 Ohms load to a 50 Ohms source. An L network with a capacitor in parallel with the load is adopted as the matching network. Lower bounds and upper bounds for the lumped-elements reactance values are 1 Ohm and 1000 Ohms. Basic start settings for the two algorithms are as follows. For `simulannealbnd` and `patternsearch`, the start point is set at the middle: [500 500] Ohms; for `ga`, the number of variables is set as 2. The related MATLAB codes can be found in Appendix B.

Here I run each of them 10 times, to find the L and C values to get the minimum VSWR of L network. The results are presented in Table 4-1. We could compare the results with the Impedance Matching Network Designer, which is L = 2151nH, C = 430.1pF.
Table 4-1: Optimizer Results to Compare Between \textit{ga} and \textit{simulannealbnd}

<table>
<thead>
<tr>
<th>No.</th>
<th>\textit{patternsearch}</th>
<th>\textit{simulannealbnd}</th>
<th>\textit{ga}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L/nH</td>
<td>C/pF</td>
<td>VSWR</td>
</tr>
<tr>
<td>1</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>2151</td>
<td>430.1</td>
<td>1.00</td>
</tr>
<tr>
<td>Iterates</td>
<td>42</td>
<td>2695</td>
<td>1040</td>
</tr>
<tr>
<td>VSWR</td>
<td>1.00</td>
<td>1.367</td>
<td>2.98</td>
</tr>
</tbody>
</table>

From the above table, we can see with \textit{patternsearch}, we could get right values every time, with perfect matched VSWR=1.00. With \textit{simulannealbnd}, the VSWR stays below 2, which is relatively stable. About 50% of the time we could get correct results with \textit{simulannealbnd}. But with \textit{ga}, the VSWR jumps from 1.29 to more than 7, with an average of 2.98. If we use the L and C optimized by \textit{ga}, there is a good chance that we will get a bad-performance matching network.

From the iterates perspective, there is no doubt \textit{patternsearch} works the best with only 42 iterations, much less than the other two. Although it took \textit{ga} 1040 iterations to get the result, \textit{simulannealbnd} can get to a much more accurate one with about twice the time. We can say that
*patternsearch* is the best choice in this problem, and *simulannealbnd* has a much better convergence characteristic than *ga* does. No matter we care more about accuracy or time here, we will select *patternsearch* as the solver. We can also use *ga* or *simulannealbnd* if we run the algorithm many times, compare the results and pick the best result.
Chapter 5

MATLAB Modeling, Simulation, and Results

In this chapter, the procedure of the main program and the optimizer program will be explained. Then, we enhanced this method with adding loops into the program. The theory and modeling of the basic direct search approach is described and compared with the Optimization Toolbox approach. All results will be presented and analyzed. After comparing, we will see the pros and cons of each method, and the designer can choose from them accordingly.

5.1 Procedure and Modeling

From the former chapters we can get a rough idea of how the program works. Here is a conclusion of the main program procedure:

(a) Start the program by importing the impedance file of an antenna including frequencies and corresponding impedances.

(b) Ask the designer to choose from the following topologies: L network, T network, and Pi network.

(c) Run all the optimizers according to selected topology.

(d) Compare the VSWR output of all the optimizers, and choose the best result to present to the designer.

The main program MATLAB code is attached in Appendix C.
5.1.1 Step-by-step Procedure of the optimizer

When the topology is chosen, every possible network in this topology will run through the optimizer outputting its optimized VSWR, L and C values. For example, if Pi topology in Figure 5-1 is chosen, we will have the following networks shown in Figure 5-2:

![Diagram](image)

**Figure 5-1: Pi Topology**

Similarly, we could get 8 networks for L topology, and 8 networks for T topology, as shown in Figures 5-3 and 5-4. Further study can be performed by filling in the topology blocks with more complex lumped element combinations, like a capacitor and an inductor in parallel, or a capacitor and an inductor in series. Accordingly, there would be more variety of network structures for one selected topology, which will not be further discussed in this thesis.
Figure 5-3: All Networks in L Topology

Figure 5-4: All Networks in T Topology

For a Pi1 network, a corresponding optimizer is run and the optimized values for VSWR₁, L₁, C₁ and bandwidth BW₁ are stored; then go through the same process with Pi2 network, Pi3 network, etc. Here we take the Pi3 network as an example and explain the Pi3 optimizer procedure:

(a) Import the antenna impedance file.

(b) Give the start point when using `patternsearch` or `simulannealbnd`; give number of variables when using `ga` as the solver. The starting point is in Ohms, which stands for the value of the reactance of the lumped element.

(c) Set lower bound and upper bounds for each element.
(d) Calculate L and C start values based on the center frequency. Get the maximum of VSWR over all frequencies, and then the chosen solver is used to minimize this maximum.

(e) Calculate L and C values from the reactance values result of step (d).

(f) Get the maximum of VSWR over all frequencies based on (e) results.

(g) Store the maximum of VSWR, L, and C values for later comparing.

Now we have a set of maximum VSWR values stored, together with their bandwidth, L and C values, and their network structure. Their VSWR maximums could be used to represent their matching performance now. Then, we compare all VSWR maximums, and find the minimum of them. This network is chosen as the final best design in this topology. Then the output is displayed, with the L and C values, the network structure, and the VSWR-frequency plot.

MATLAB codes for above program and related functions of the Pi3 network are attached in Appendix C. The solver patternsearch is used in this program.

5.1.2 Basic Blocks needed

From Section 5.1.1 we can see the procedure for all network structures are similar, except for the VSWR functions and the number of lumped elements. Therefore, codes are needed for the VSWR of each network structure. First, \( Z_{in} \) is calculated using basic circuit laws; then we get \( \Gamma \) from \( Z_{in} \) and \( Z_{source} \). VSWR can then be obtained from \( \Gamma \).

Take Pi3 network as an example. Referring to Figure 5-1 and 5-2 (3), we have

\[
Z_{in} = \frac{-jX_1}{(-jX_2 + jX_3 / Z_{load})}
\]  

(5-1)
\[
\Gamma = \frac{Z_{\text{in}} - Z_{\text{source}}}{Z_{\text{in}} + Z_{\text{source}}}
\]  
\[\text{(5-2)}\]

\[
\text{VSWR} = \frac{V_{\max}}{V_{\min}} = \frac{1 + |\Gamma|}{1 - |\Gamma|}
\]  
\[\text{(5-3)}\]

The MATLAB code for the Pi3 network is attached in Appendix C.

For networks with the same topology, only the signs of reactances are different. We are interested in the element impedances, including reactance \( X \) and resistance \( R \).

\[
Z = R + jX
\]  
\[\text{(5-4)}\]

Both capacitive reactance \( X_C \) and inductive reactance \( X_L \) contribute to the total reactance \( X \).

\[
X = X_L - X_C
\]  
\[\text{(5-5)}\]

\[
X_C = \frac{1}{\omega C} = \frac{1}{2\pi fC}
\]  
\[\text{(5-6)}\]

\[
X_L = \omega L = 2\pi fL
\]  
\[\text{(5-7)}\]

Therefore, all inductive elements have impedance \( jX \), and all capacitive elements have impedance \(-jX\). Turning one function into another only needs a corresponding sign change. For example, if we want to write the VSWR function for network Pi4, we have

\[
Z_{\text{in}} = jX_1 / (jX_2 - jX_3 / Z_{\text{load}})
\]  
\[\text{(5-4)}\]

The rest of the function is the same with that of Pi3 network. All VSWR function codes can be found in Appendix D.
5.2 Simulation and Results

The frequency band we are interested in is 3.5 to 3.85 MHz. The load is an 80 meter half-wave (total length = 127.33 feet) dipole antenna made out of #10 gauge wire at a height of 100 feet over average earth ground ($\varepsilon_r = 13, \sigma = 0.005$ S/m), whose impedance file is in Appendix A. The impedance file was generated from the antenna simulation software package GNEC [22] based on NEC-4 [22]. Impedance matching optimizers will run based on principles discussed above, and the maximum of VSWR over the frequency will be used as the criteria of bandwidth: the smaller the maximum is, the better the bandwidth we get.

5.2.1 Set-up

Import impedance file

The 80-meter dipole antenna’s impedance and frequency data is given in a text file. In the text file $A15.txt$, columns are separated by spaces. This file also includes other information, like the magnitude and angle of the impedance in polar form. We are only adopting the frequency, real part of impedance, and imaginary part of impedance in this case.

In order to import this data file into a matrix in MATLAB, a function importfile.m is created. The MATLAB code for importfile.m can be found in Appendix C. This function reads all data from the text file, and creates a $15 \times 9$ matrix $A15$ storing it. Here the frequency is in MHz, so we use the first column times $10^6$ as the frequency data.
Prepare data input

From Equation 5-6 and Equation 5-7, we can see angular frequency $\omega$ is used a lot in this program. Vector $\omega$mega is created based on $\omega = 2\pi f$.

From A15.txt, we can get the real part impedance vector $\text{real}$, and the imaginary part vector $\text{imag}$. The load impedance is also an important input, so we have vector $z_{\text{load}} = \text{real} + i\times \text{imag}$.

VSWR before impedance matching

In order to show the impedance matching network’s effect on VSWR, Figure 5-5 shows the VSWR plot over frequency before matching.

![VSWR plot over 3.5-3.85 MHz before adding the matching network](image)

Figure 5-5: VSWR over 3.5-3.85 MHz before adding the matching network
5.2.2 Simulation Using Optimizers

Based on the steps in Section 5.1.1, MATLAB codes are written for using solver *patternsearch, ga*, and *simulannealbnd*, respectively. The codes for the Pi3 network using *patternsearch* are attached in Appendix C. Codes for the other two solvers can be generated by switching “*patternsearch*” to “*ga*” or “*simulannealbnd*”, therefore these codes will not be displayed.

Here, we are trying to find the optimized values of L and C for the Pi3 network to get the widest possible frequency band matching of the antenna to a 50 Ohms transmission line. The frequency-impedance file of the 80-meter dipole antenna is given in Appendix A.

Before starting the simulation, we need some criteria to decide whether the result is good enough. First, bandwidth is the main goal in impedance matching. Bandwidth can be calculated from Equation 5-5.

\[
BW = \frac{f_{\text{high}} - f_{\text{low}}}{\sqrt{f_{\text{high}} \cdot f_{\text{low}}}}
\]  

(5-5)

Here, \( f_{\text{high}} \) and \( f_{\text{low}} \) refer to the high and low frequencies where the VSWR equals 2.

With Equation 5-5, we can calculate the bandwidth before matching. From Figure 5-5, we can read \( f_{\text{high}} = 3.74 \text{ MHz} \), and \( f_{\text{low}} = 3.61 \text{ MHz} \).

\[
BW = \frac{3.74 - 3.61}{\sqrt{3.74 \cdot 3.61}} = \frac{0.13}{3.6744} = 3.4\%
\]

Besides, a set of optimized L and C values for the Pi3 network are provided by Prof. James K. Breakall, based on a FORTRAN code NECOPT [23][24], GNEC, and FEKO:

\[
\begin{align*}
C_1 &= 3706 \text{ pF} \\
C_2 &= 6626 \text{ pF} \\
L &= 745.3 \text{ nH}
\end{align*}
\]  

(5-6)
To compare with MATLAB results, the values above are turned into Ohms:

\[
\begin{align*}
X_{C_1} &= 11.6 \\
X_{C_2} &= 6.5 \\
X_L &= 17.3
\end{align*}
\] (5-7)

With these start values, Figure 5-6 shows the VSWR-frequency result. It can be seen that over all of this frequency band, the VSWR is below 2, giving the largest bandwidth one can get.

Figure 5-6: VSWR over 3.5-3.85 MHz with $C_1=3706\text{pF}$, $C_2=6626\text{pF}$, $L=745.3\text{nH}$

Since optimizers can give quite different answers with different start values, two sets of results are presented: one with start values quite close to values in Equation (5-6) [10 5 15]; the other with start values in the middle of the whole searching range [500 500 500]. When using solver ga, the start point has no effect on results. The VSWR plots are listed below in Figure 5-7 to Figure 5-11. The L and C values are shown in Table 5-1.
Table 5-1: L and C Value Results Given By Optimizers

<table>
<thead>
<tr>
<th>Solver</th>
<th>patternsearch</th>
<th>ga</th>
<th>simulannealbnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start point/Ohms</td>
<td>[10 5 15]</td>
<td>[10 5 15]</td>
<td>[500 500 500]</td>
</tr>
<tr>
<td>$C_1$/pF</td>
<td>4115</td>
<td>43.31</td>
<td>69.43</td>
</tr>
<tr>
<td>$C_2$/pF</td>
<td>7874</td>
<td>523.2</td>
<td>8662</td>
</tr>
<tr>
<td>L/nH</td>
<td>682.8</td>
<td>6906</td>
<td>649.6</td>
</tr>
<tr>
<td>Max(VSWR)</td>
<td>1.34</td>
<td>2.04</td>
<td>4.18</td>
</tr>
<tr>
<td>Time used/s</td>
<td>0.51</td>
<td>1.37</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Figure 5-7: VSWR over 3.5-3.85 MHz with start point [10 5 15] using patternsearch
Figure 5-8: VSWR over 3.5-3.85 MHz with start point [500 500 500] using patternsearch

Figure 5-9: VSWR over 3.5-3.85 MHz using ga (Both starting points)
Figure 5-10: VSWR over 3.5-3.85 MHz with start point [10 5 15] using `simulannealbnd`.

Figure 5-11: VSWR over 3.5-3.85 MHz with start point [500 500 500] using `simulannealbnd`.
5.2.3 Simulation with Loops

In the simulation process, it is found that running one solver over and over again could give many different results. Meanwhile, start point selection is crucial to the result. In order to get the largest bandwidth possible in limited time, the optimizer is placed in a loop. Every time the loop is run, a random start point is generated. After running many times, all results are compared, and the one with smallest VSWR is chosen as the final result. The MATLAB codes for loop optimization of the Pi3 network are in Appendix E. The optimizer is run 100 times and 5000 times using `patternsearch` and `simulannealbnd`, respectively, to compare these two solvers for performance in loops. The L and C values are shown in Table 5-2, and VSWR plots are listed below in Figure 5-12 to Figure 5-15.

<table>
<thead>
<tr>
<th>Table 5-2: L and C Value Results Given By Loop Optimizers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solver</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Loops</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C₁/pF</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C₂/pF</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>L/nH</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Max(VSWR)</td>
</tr>
<tr>
<td>3.5-3.85 MHz</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Figure 5-12: VSWR over 3.5-3.85 MHz with 100 loops using \textit{patternsearch}

Figure 5-13: VSWR over 3.5-3.85 MHz with 5000 loops using \textit{patternsearch}
Comparing the above figures, we can see *patternsearch* with much more loops does not necessarily give better results. However, increasing loop number using *simulannealbnd* can
dramatically expand the bandwidth. Therefore, more cases are run using `simulannealbnd`, with only loop numbers changed each time. To compare the bandwidth difference, Figure 5-16 to Figure 5-20 show VSWR over 25 frequency points within the 3.4 to 4.0 MHz range. The results and optimized values are shown in Table 5-3.

Table 5-3: L, C Values and Bandwidths Results by `simulannealbnd` in loops

<table>
<thead>
<tr>
<th>Loops</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁/pF</td>
<td>839.5</td>
<td>1690</td>
<td>1828</td>
<td>3230</td>
<td>2910</td>
</tr>
<tr>
<td>C₂/pF</td>
<td>1758</td>
<td>3526</td>
<td>3111</td>
<td>5407</td>
<td>5342</td>
</tr>
<tr>
<td>L/nH</td>
<td>2596</td>
<td>1520</td>
<td>1499</td>
<td>901.5</td>
<td>960.3</td>
</tr>
<tr>
<td>Bandwidth/VSWR&lt;2</td>
<td>10.9%</td>
<td>12.8%</td>
<td>14.4%</td>
<td>14.4%</td>
<td>14.4%</td>
</tr>
<tr>
<td>Bandwidth/VSWR&lt;1.5</td>
<td>4.1%</td>
<td>6.8%</td>
<td>3.9%</td>
<td>12.0%</td>
<td>10.9%</td>
</tr>
<tr>
<td>Time/min</td>
<td>1.8</td>
<td>4.1</td>
<td>19.7</td>
<td>37.8</td>
<td>216</td>
</tr>
</tbody>
</table>

Figure 5-16: VSWR over 3.4-4 MHz with 50 loops using `simulannealbnd`
Figure 5-17: VSWR over 3.4-4 MHz with 100 loops using *simulannealbnd*

Figure 5-18: VSWR over 3.4-4 MHz with 500 loops using *simulannealbnd*
Figure 5-19: VSWR over 3.4-4 MHz with 1000 loops using `simulannealbnd`

Figure 5-20: VSWR over 3.4-4 MHz with 5000 loops using `simulannealbnd`
5.3 Result Analysis

5.3.1 Results Analysis for Optimizer with loops

Comparing Figure 5-7 - Figure 5-11 to Figure 5-12 – Figure 5-15, clearly optimization with loops works much better than single optimization. When given start points very close to the known solution, both simulannealbnd and patternsearch can find good global optimums easily. Solver ga does not work with any starting points, therefore cannot give satisfying result in this case.

However, it is impossible and meaningless to expect solutions with close starting points prepared for every network. Optimization results with random generated start points make more sense then. From this perspective, when time is not considered as an essential factor, loop optimization should be adopted for the impedance matching network design, with as many loops as possible.

Theoretically, we are looking for L and C values for the largest bandwidth; time cost could be very disappointing. Industry designers always want to find the balance between time and accuracy. It takes simulannealbnd 3.6 hours to run 5000 loops. Comparing between results of 100 loops and 5000 loops using simulannealbnd, we can see it is worthy to sacrifice several hours to get a significant bandwidth increase. This results in even better bandwidth than the values found using the FORTRAN code NECOPT and GNEC. Therefore, it is reasonable to adopt the latter one as the final design.

Comparing between solvers, we can see patternsearch converges much faster than simulannealbnd. But as the number of loops increase, patternsearch fails to give a result as optimized as simulannealbnd does. In the L network where the nonlinear objective function is smooth, however, patternsearch can get accurate optimized design results every time, with much
less time spent. In conclusion, we could adopt *patternsearch* as the solver for 2-element matching network design, and adopt *simulannealbnd* with a large number of loops for 3 and more element matching network design.

### 5.3.2 Results Analysis with Direct Search

When we discard all optimization tools, but mesh the L-C surface into grids and calculate the VSWR to find the smallest set, this is a manual process to find the optimum for the impedance matching problem. In this procedure, no advanced algorithm is used, like gradients or genetic algorithm. The calculation task is much more CPU time consuming than that using Optimization Tools. However, the global optimum is guaranteed if the L-C plane is meshed fine enough. We call this method Direct Search in this thesis, and corresponding MATLAB codes can be found in Appendix F.

The same network-matching problem is dealt with here: matching this 80-meter antenna to 50 Ohm transmission line with a Pi3 network. Because the Direct Search approach takes much more time than the optimization tools, the searching range is greatly decreased here. With the given optimized values $[11.6, 6.5, 17.3]$ Ohms, the bound is set as $[1, 20]$ for each element. This range is divided into linearly spaced 500 points. There are altogether $500^3$ sets of values. With every set of values, the VSWR is calculated over the frequency, and the maximum VSWR is stored. After calculation, all these maximums are compared, and the minimum of them is picked along with corresponding L and C values. This is taken as the final result.
The Direct Search result is shown in Figure 5-21. It costs about 6 hours to finish running this program, and the optimized values are as follows:

\[
\begin{align*}
C_1 &= 4086 \text{ pF} \\
C_2 &= 7830 \text{ pF} \\
L &= 686.4 \text{nH}
\end{align*}
\]

Figure 5-21: VSWR over 3.5-3.85 MHz using Direct Search
The VSWR over wider frequency range is shown in Figure 5-22. The bandwidth is given by Equation 5-5.

$$BW_{VSWR=2} = \frac{3.92 - 3.44}{\sqrt{3.92 \cdot 3.44}} = \frac{0.48}{3.672} = 13.1\%$$

The bandwidth with a VSWR 1.5:1 can also be given by:

$$BW_{VSWR=1.5} = \frac{3.88 - 3.48}{\sqrt{3.88 \cdot 3.48}} = \frac{0.40}{3.674} = 10.9\%$$

This result for VSWR 2:1 is not as good as the bandwidth by running `simulannealbnd` with 500 loops, which is 14.4%. The result for VSWR 1.5:1 is better than running `simulannealbnd` with 500 loops, but not as good as running `simulannealbnd` with 1000 loops, which is 12.0%.

With 15 frequency points, the Direct Search needs to calculate $500^3 \times 15$ times. Comparing to the several thousand times using optimization algorithms; this will take much longer time to run. It can be imagined that without given optimized values, we might have to run the search in a larger range $[1, 1000]$. If we want to set the step of the grid at 0.1, we need
10000^3 \times 15 \text{ calculations. Approximately, the running time is proportional to calculation times; so we can get the time needed to run the Direct Search program over [1,1000] is about 48000 hours, which is 2000 days.}

In conclusion, it is obvious that simulannealbnd with loops is the best way to find the optimized matching network with a large bandwidth, when the topology has more than 3 elements to be decided. The designer can get the largest bandwidth possible within 20 minutes when matching a Pi3 network. It is also easy to transit into networks other than the Pi3 network, by simply calling VSWR functions of the network interested in and changing the number of variables.
6.1 Thesis Contributions

This thesis proposes a new approach of wide band impedance matching network design based on the MATLAB Global Optimization Toolbox. An 80-meter dipole antenna is matched with a 50 Ohms transmission line over the frequency 3.5 MHz to 3.85 MHz. The bandwidth is increased from 3.4% to 14.4% after the optimization. Past approaches cannot perform impedance matching network quickly and accurately enough over such large bandwidth. On one hand, existing impedance matching calculators can only design matching networks at a single frequency, with a fixed load impedance. On the other hand, the approach for wide band impedance matching using FEKO or NECOPT takes too much time, for it simulates the antenna as well as the matching circuit and then performs the optimization.

The two problems above have been solved by the approach proposed in this thesis. Actual measured data of an 80-meter half-wave dipole antenna serves as the load, making the simulation results reliable. According to the program process, it is also easy to import impedance files for other loads into the program and look for their matching networks. The designer can also control the balance between optimization time and the bandwidth after matching. For the antenna discussed in the thesis, 20 minutes to 3.6 hours running time gives a bandwidth of 14.4%, and 4.1 minutes running time gives a bandwidth of 12.8%. If the designer only needs a relatively large bandwidth, say 12.8%, he or she can run 100 loops and save the time.

Other than that, several global optimization algorithms have been compared using results and examples. In conclusion, patternsearch converges fast, and can get similar results every time. For 2-element networks, patternsearch is a good way to solve the optimization problem. But for
more complex element matching networks, it cannot find global optimums even after 5000 loops. Therefore, the best solver for 3-element or more matching networks is *simulannealbnd*.

Besides, the main program described in the thesis allows the designer to get optimized impedance matching networks with very few steps. The designer only needs to provide an impedance file of the load, and choose the topology desired. This program will enable people with little knowledge of impedance matching to find the optimized network quickly.

### 6.2 Future Research Directions

Research on this topic still has a long way to go. Some future enhancement and research directions will be outlined below.

(a) Shorten the time of running the programs by optimizing the MATLAB codes.

(b) Explore the feasibility of using Genetic Algorithm for impedance matching network optimization, which seem to work poorly in any network shown in this thesis.

(c) Develop software with the idea described in this thesis, with a GUI itself, instead of basing it on MATLAB or some other third party numerical methods.
### References


22 L. B. Cebik, “Antenna Modeling Programs and Resources,” Retrieved from:
http://www.antennex.com/shack/Nov05/resources.html

23 J. K. Breakall, J. S. Young, R. J. Bauerle, A. I. McDowell and T. A. Erdley, “Numerical
Electromagnetics Code Optimization Design Software (NECOPT),” 10th Annual Review
of Process in Applied Computational Electromagnetics. Naval Postgraduate School,

(NEC),” EMC Symposium and Exhibition, Zurich, Switzerland, 1985.
# Appendix A

## Antenna Impedance File - Before Matching

### Input Impedance and VSWR

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Tag</th>
<th>Seg</th>
<th>Real(Z)</th>
<th>Imag(Z)</th>
<th>Mag(Z)</th>
<th>Phase(Z)</th>
<th>Zo</th>
<th>VSWR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.400000</td>
<td>7</td>
<td>41</td>
<td>71.433</td>
<td>-122.065</td>
<td>141.430</td>
<td>-59.664</td>
<td>50.000</td>
<td>6.14</td>
</tr>
<tr>
<td>3.425000</td>
<td>7</td>
<td>41</td>
<td>72.794</td>
<td>-111.010</td>
<td>132.749</td>
<td>-56.745</td>
<td>50.000</td>
<td>5.34</td>
</tr>
<tr>
<td>3.450000</td>
<td>7</td>
<td>41</td>
<td>74.167</td>
<td>-99.986</td>
<td>124.491</td>
<td>-53.433</td>
<td>50.000</td>
<td>4.64</td>
</tr>
<tr>
<td>3.475000</td>
<td>7</td>
<td>41</td>
<td>75.551</td>
<td>-88.991</td>
<td>116.737</td>
<td>-49.670</td>
<td>50.000</td>
<td>4.02</td>
</tr>
<tr>
<td>3.500000</td>
<td>7</td>
<td>41</td>
<td>76.947</td>
<td>-78.021</td>
<td>109.582</td>
<td>-45.397</td>
<td>50.000</td>
<td>3.48</td>
</tr>
<tr>
<td>3.525000</td>
<td>7</td>
<td>41</td>
<td>78.353</td>
<td>-67.071</td>
<td>103.139</td>
<td>-40.564</td>
<td>50.000</td>
<td>3.02</td>
</tr>
<tr>
<td>3.550000</td>
<td>7</td>
<td>41</td>
<td>79.772</td>
<td>-56.143</td>
<td>97.548</td>
<td>-35.137</td>
<td>50.000</td>
<td>2.63</td>
</tr>
<tr>
<td>3.575000</td>
<td>7</td>
<td>41</td>
<td>81.203</td>
<td>-45.231</td>
<td>92.951</td>
<td>-29.118</td>
<td>50.000</td>
<td>2.31</td>
</tr>
<tr>
<td>3.600000</td>
<td>7</td>
<td>41</td>
<td>82.647</td>
<td>-34.331</td>
<td>89.494</td>
<td>-22.558</td>
<td>50.000</td>
<td>2.06</td>
</tr>
<tr>
<td>3.625000</td>
<td>7</td>
<td>41</td>
<td>84.103</td>
<td>-23.442</td>
<td>87.309</td>
<td>-15.575</td>
<td>50.000</td>
<td>1.87</td>
</tr>
<tr>
<td>3.650000</td>
<td>7</td>
<td>41</td>
<td>85.572</td>
<td>-12.560</td>
<td>86.489</td>
<td>-8.350</td>
<td>50.000</td>
<td>1.77</td>
</tr>
<tr>
<td>3.675000</td>
<td>7</td>
<td>41</td>
<td>87.054</td>
<td>-1.681</td>
<td>87.070</td>
<td>-1.106</td>
<td>50.000</td>
<td>1.74</td>
</tr>
<tr>
<td>3.700000</td>
<td>7</td>
<td>41</td>
<td>88.549</td>
<td>9.197</td>
<td>89.025</td>
<td>5.929</td>
<td>50.000</td>
<td>1.80</td>
</tr>
<tr>
<td>3.725000</td>
<td>7</td>
<td>41</td>
<td>90.057</td>
<td>20.077</td>
<td>92.268</td>
<td>12.568</td>
<td>50.000</td>
<td>1.93</td>
</tr>
<tr>
<td>3.750000</td>
<td>7</td>
<td>41</td>
<td>91.583</td>
<td>30.963</td>
<td>96.675</td>
<td>18.680</td>
<td>50.000</td>
<td>2.11</td>
</tr>
<tr>
<td>3.775000</td>
<td>7</td>
<td>41</td>
<td>93.120</td>
<td>41.859</td>
<td>102.096</td>
<td>24.205</td>
<td>50.000</td>
<td>2.35</td>
</tr>
<tr>
<td>3.800000</td>
<td>7</td>
<td>41</td>
<td>94.763</td>
<td>52.766</td>
<td>108.385</td>
<td>29.133</td>
<td>50.000</td>
<td>2.63</td>
</tr>
<tr>
<td>3.825000</td>
<td>7</td>
<td>41</td>
<td>96.241</td>
<td>63.690</td>
<td>115.406</td>
<td>33.496</td>
<td>50.000</td>
<td>2.95</td>
</tr>
<tr>
<td>3.850000</td>
<td>7</td>
<td>41</td>
<td>97.825</td>
<td>74.631</td>
<td>123.043</td>
<td>37.340</td>
<td>50.000</td>
<td>3.30</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50.000</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>----</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.875000</td>
<td></td>
<td></td>
<td>99.427</td>
<td>85.597</td>
<td>131.196</td>
<td>40.725</td>
<td>50.000</td>
<td>3.69</td>
</tr>
<tr>
<td>3.900000</td>
<td></td>
<td></td>
<td>101.045</td>
<td>96.586</td>
<td>139.782</td>
<td>43.708</td>
<td>50.000</td>
<td>4.12</td>
</tr>
<tr>
<td>3.925000</td>
<td></td>
<td></td>
<td>102.681</td>
<td>107.605</td>
<td>148.735</td>
<td>46.341</td>
<td>50.000</td>
<td>4.58</td>
</tr>
<tr>
<td>3.950000</td>
<td></td>
<td></td>
<td>104.335</td>
<td>118.656</td>
<td>158.003</td>
<td>48.675</td>
<td>50.000</td>
<td>5.07</td>
</tr>
<tr>
<td>3.975000</td>
<td></td>
<td></td>
<td>106.010</td>
<td>129.743</td>
<td>167.545</td>
<td>50.749</td>
<td>50.000</td>
<td>5.59</td>
</tr>
<tr>
<td>4.000000</td>
<td></td>
<td></td>
<td>107.705</td>
<td>140.870</td>
<td>177.327</td>
<td>52.600</td>
<td>50.000</td>
<td>6.14</td>
</tr>
</tbody>
</table>
Appendix B

MATLAB codes for L network Impedance Matching Optimization at 3.7 MHz

MATLAB code for matching 100 Ohms to 50 Ohms at 3.7 MHz using `patternsearch`, with start value [500 500]:

```matlab
clc;
clear all

freq = 3.7e6; % frequency in Hz
zload = 100;
omega = 2*pi.*freq;

x0 = [500 500]; % setting start point
lb = [1 1];
ub = [1000 1000]; % setting lower bound and upper bound

%% get optimized values L and C
% anynomous function handle
handle = @(x)L1(x,zload);

% get the optimum Xl, and Xc
[b,VSWR,exitflag,output] = optimizeps (x0,lb,ub,handle);

% storing all values
out = output;
xl = b(1);
xc = b(2);

% calculate L (in nH) and C (in pF) values
L = 1e9 * xl/omega;
C= 1e12 * 1/(omega*xc);

Lreact = omega.*L*1e-9;
Creact = 1e12./(omega.*C);

x = [Lreact,Creact];
VSWR = L1(x,zload);
VSWR
L
C
```
MATLAB code for matching 100 Ohms to 50 Ohms at 3.7 MHz using *simulannealbnd*, with start value [500 500]:

```matlab
clc;
clear all

freq = 3.7e6; % frequency in Hz
zload = 100;
omega = 2*pi.*freq;

x0 = [500 500]; % setting start point
lb = [1 1];
ub = [1000 1000]; % setting lower bound and upper bound

%% get optimized values L and C
% anonomous function handle
handle = @(x)L1(x,zload);

% get the optimum XL, and XC
[b,VSWR,exitflag,output] = optimize (x0,lb,ub,handle);

% storing all values
out = output;
xl = b(1);
xc = b(2);

% calculate L (in nH) and C (in pF) values
L = 1e9 * xl/omega;
C= 1e12 * 1/(omega*xc);

Lreact = omega.*L*1e-9;
Creact = 1e12./(omega.*C);

x = [Lreact,Creact];
VSWR = L1(x,zload);
```
MATLAB code for matching 100 Ohms to 50 Ohms at 3.7 MHz using ga, with number of variables 2:

```matlab
clc;
clear all

freq = 3.7e6;       % frequency in Hz
zload = 100;
omega = 2*pi.*freq;

nvars = 2;          % setting number of variables
lb = [1 1];
ub = [1000 1000];   % setting lower bound and upper bound

%% get optimized values L and C
% anonomous function handle
handle = @(x)L1(x,zload);

% get the optimum Xl, and Xc
[b,VSWR,exitflag,output] = ga (handle,nvars,[],[],[],[],lb,ub);

% storing all values
xl = b(1);
xc = b(2);

% calculate L (in nH) and C (in pF) values
L = 1e9 * xl/omega;
C= 1e12 * 1/(omega*xc);

Lreact = omega.*L*1e-9;
Creact = 1e12./(omega.*C);

x = [Lreact,Creact];
VSWR = L1(x,zload);
```
MATLAB code for function L1:

```matlab
function m = L1(x,zload)
    % set source impedance
    zs = 50;
    % calculate input impedance looking from source side
    zin = 1i*x(1)-1i*x(2)*zload/(-1i*x(2)+zload);
    % calculate reflection coefficient
    gamma = (zin-zs)/(zin+zs);
    % calculate VSWR
    m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function optimizeps:

```matlab
function [x,fval,exitflag,output] = optimizeps(x0,lb,ub,handle)
    % Start with the default options
    options = saoptimset;
    % Modify options setting
    options = saoptimset(options,'AnnealingFcn', @annealingfast);
    options = saoptimset(options,'Display', 'off');
    options = saoptimset(options,'HybridInterval', 'end');
    options = saoptimset(options,'OutputFcns', { [] });
    [x,fval,exitflag,output] = ...
        patternsearch(handle,x0,[],[],[],[],lb,ub,options);
end
```

MATLAB code for function optimize:

```matlab
function [x,fval,exitflag,output] = optimize(x0,lb,ub,handle)
    % Start with the default options
    options = saoptimset;
    % Modify options setting
    options = saoptimset(options,'AnnealingFcn', @annealingfast);
    options = saoptimset(options,'Display', 'off');
    options = saoptimset(options,'HybridInterval', 'end');
    options = saoptimset(options,'OutputFcns', { [] });
    [x,fval,exitflag,output] = ...
        simulannealbnd(handle,x0,lb,ub,options);
end
```
Appendix C

MATLAB codes for Main Program and Pi3 Optimizer

MATLAB code for main program:

```matlab
%% import file and initialize
clear all
importfile('A15.txt');

freq = A15(:,1).*1e6; % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;
length = 15;
for j=1:length
    zload(j) = real(j) + 1i*imag(j);
end

%% choose topology
prompt = ['Choose one topology from the following: 1.L 2.Pi 3T '];

%% calculate all network results
result = input(prompt);
if result == 1
    [x(1,:),v(1)] = VSWR_L1(omega,A15,zload);
    [x(2,:),v(2)] = VSWR_L2(omega,A15,zload);
    [x(3,:),v(3)] = VSWR_L3(omega,A15,zload);
    [x(4,:),v(4)] = VSWR_L4(omega,A15,zload);
    [x(5,:),v(5)] = VSWR_L5(omega,A15,zload);
    [x(6,:),v(6)] = VSWR_L6(omega,A15,zload);
    [x(7,:),v(7)] = VSWR_L7(omega,A15,zload);
    [x(8,:),v(8)] = VSWR_L8(omega,A15,zload);
else if result == 2
    [x(1,:),v(1)] = VSWR_Pi1(omega,A15,zload);
    [x(2,:),v(2)] = VSWR_Pi2(omega,A15,zload);
    [x(3,:),v(3)] = VSWR_Pi3(omega,A15,zload);
    [x(4,:),v(4)] = VSWR_Pi4(omega,A15,zload);
    [x(5,:),v(5)] = VSWR_Pi5(omega,A15,zload);
    [x(6,:),v(6)] = VSWR_Pi6(omega,A15,zload);
    [x(7,:),v(7)] = VSWR_Pi7(omega,A15,zload);
    [x(8,:),v(8)] = VSWR_Pi8(omega,A15,zload);
else if result == 3
    [x(1,:),v(1)] = VSWR_T1(omega,A15,zload);
    [x(2,:),v(2)] = VSWR_T2(omega,A15,zload);
    [x(3,:),v(3)] = VSWR_T3(omega,A15,zload);
    [x(4,:),v(4)] = VSWR_T4(omega,A15,zload);
    [x(5,:),v(5)] = VSWR_T5(omega,A15,zload);
    [x(6,:),v(6)] = VSWR_T6(omega,A15,zload);
    [x(7,:),v(7)] = VSWR_T7(omega,A15,zload);
end
```

[x(8,:),v(8)] = VSWR_T8(omega,A15,zload);
else
disp('Wrong input.');
end
end
end

%% compare and display the final result
[VSWR,k] = min(v);
values = x(k,:);
VSWR
values

MATLAB code for function importfile:

function importfile(fileToRead1)
%IMPORTFILE(FILETOREAD1)
% Imports data from the specified file
% FILETOREAD1: file to read

% Import the file
rawData1 = importdata(fileToRead1);

% For some simple files (such as a CSV or JPEG files), IMPORTDATA might
% return a simple array. If so, generate a structure so that the output
% matches that from the Import Wizard.
[~,name] = fileparts(fileToRead1);
newData1.(genvarname(name)) = rawData1;

% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
    assignin('base', vars{i}, newData1.(vars{i}));
end
MATLAB code for Pi3 network using *patternsearch*:

```matlab
%% import file and initialize
clc;
clear all
importfile('A15.txt');

freq = A15(:,1).*1e6;            % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;

start = [10 5 15];            % setting start point
lb = [1 1 1];
ub = [1000 1000 1000];        % setting lower bound and upper bound
length = 15;

handle = @(x)Pi3_opt(x,A15);

% get the optimum Xc1, Xl and Xc2 at frequency f(i)
[b,VSWR,exitflag,output] = optimizeps (start,lb,ub,handle);

% storing all values
out = output;
xc1 = b(1);
xc2 = b(2);
xl = b(3);

C1 = 1e12 * 1/omega(8)*xc1;
C2 = 1e12 * 1/omega(8)*xc2;
L = 1e9 * x1/omega(8);

%% plot VSWR at all frequencies
% calculate VSWR at all frequency points
for j=1:length
    zload(j) = real(j) + 1i*imag(j);
    xc1t = 1e12./omega(j)*C1;
    xc2t = 1e12./omega(j)*C2;
    xlt = omega(j)*L*1e-9;
    x = [xc1t,xc2t,xlt];
    y(j) = Pi3(x,zload(j));
end
[Vmax,k] = max(y);
plot(freq,y);
xlabel('Frequency/Hz');
ylabel('VSWR');
title('VSWR with matching network optimized');
```
MATLAB code for function Pi3_opt:

```matlab
function m = Pi3_opt(x,A15)

% set source impedance
zs = 50;
freq = A15(:,1).*1e6; % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;

C1 = 1e12 * 1/(omega(8)*x(1));
C2 = 1e12 * 1/(omega(8)*x(2));
L = 1e9 * x(3)/omega(8);

for j=1:15
    zload(j) = real(j) + 1i*imag(j);
    xc1(j) = 1e12./(omega(j)*C1);
    xc2(j) = 1e12./(omega(j)*C2);
    xl(j) = omega(j)*L*1e-9;
    x0 = [xc1(j) xc2(j) xl(j)];
    g(j) = Pi3(x0,zload(j));
end
[Vmax,k] = max(g);

% calculate input impedance looking from source side
para = 1i*xl(k)*zload(k)/(1i*xl(k)+zload(k));
seri = para - 1i*xc2(k);
zin = -seri*1i*xc1(k)/(-1i*xc1(k)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function Pi3:

```matlab
function m = Pi3(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = 1i*x(3)*zload/(1i*x(3)+zload);
seri = para - 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function optimizeps:

```matlab
function [x,fval,exitflag,output] = optimizeps(x0,lb,ub,handle)
% Start with the default options
options = saoptimset;
% Modify options setting
options = saoptimset(options,'AnnealingFcn', @annealingfast);
options = saoptimset(options,'Display', 'off');
options = saoptimset(options,'HybridInterval', 'end');
options = saoptimset(options,'OutputFcns', { [] });
[x,fval,exitflag,output] = ...
patternsearch(handle,x0,[],[],[],[],lb,ub,options);
```
Appendix D

MATLAB codes for all networks’ VSWR functions

MATLAB code for function L1:

```matlab
function m = L1(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
zin = 1i*x(1)-1i*x(2)*zload/(-1i*x(2)+zload);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function L2:

```matlab
function m = L2(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = 1i*x(2)*zload/(1i*x(2)+zload);
zin = -1i*x(1)+para;
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function L3:

```matlab
function m = L3(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
zin = -1i*x(1)-1i*x(2)*zload/(-1i*x(2)+zload);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function L4:

```matlab
function m = L4(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
zin = 1i*x(1)+1i*x(2)*zload/(1i*x(2)+zload);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function L5:

```matlab
function m = L5(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = 1i*x(2)+zload;
zin = -1i*x(1)*seri/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function L6:

```matlab
function m = L6(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -1i*x(2)+zload;
zin = 1i*x(1)*seri/(1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function L7:

```matlab
function m = L7(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -1i*x(2)+zload;
zin = -1i*x(1)*seri/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function L8:

```matlab
function m = L8(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = 1i*x(2)+zload;
zin = 1i*x(1)*seri/(1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function Pi1:

```matlab
function m = Pi1(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = 1i*x(3)*zload/(1i*x(3)+zload);
seri = para - 1i*x(2);
zin = seri*1i*x(1)/(1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function Pi2:

```matlab
function m = Pi2(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = -1i*x(3)*zload/(-1i*x(3)+zload);
seri = para + 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function Pi3:

```matlab
function m = Pi3(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = 1i*x(3)*zload/(1i*x(3)+zload);
seri = para - 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function Pi4:

```matlab
function m = Pi4(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = -1i*x(3)*zload/(-1i*x(3)+zload);
seri = para - 1i*x(2);
zin = seri*1i*x(1)/(1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function Pi5:

```matlab
function m = Pi5(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = -1i*x(3)*zload/(-1i*x(3)+zload);
seri = para + 1i*x(2);
zin = seri*1i*x(1)/(1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function Pi6:

```matlab
function m = Pi6(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = 1i*x(3)*zload/(1i*x(3)+zload);
seri = para + 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function Pi7:

```matlab
function m = Pi7(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = -1i*x(3)*zload/(-1i*x(3)+zload);
seri = para - 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function Pi8:

```matlab
function m = Pi8(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
para = li*x(3)*zload/(li*x(3)+zload);
seri = para + li*x(2);
zin = seri*li*x(1)/(li*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T1:

```matlab
function m = T1(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -li*x(3) + zload;
para = li*x(2)*seri/(li*x(2)+seri);
zin = para - li*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T2:

```matlab
function m = T2(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = li*x(3) + zload;
para = -li*x(2)*seri/(seri-li*x(2));
zin = para + li*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function T3:

```matlab
function m = T3(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = 1i*x(3) + zload;
para = -1i*x(2)*seri/(seri-1i*x(2));
zin = para - 1i*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T4:

```matlab
function m = T4(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -1i*x(3) + zload;
para = -1i*x(2)*seri/(seri-1i*x(2));
zin = para + 1i*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T5:

```matlab
function m = T5(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = 1i*x(3) + zload;
para = 1i*x(2)*seri/(seri+1i*x(2));
zin = para - 1i*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
MATLAB code for function T6:

```matlab
function m = T6(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -1i*x(3) + zload;
para = 1i*x(2)*seri/(seri+1i*x(2));
zin = para + 1i*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T7:

```matlab
function m = T7(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = -1i*x(3) + zload;
para = 1i*x(2)*seri/(seri-1i*x(2));
zin = para - 1i*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```

MATLAB code for function T8:

```matlab
function m = T8(x,zload)
% set source impedance
zs = 50;
% calculate input impedance looking from source side
seri = 1i*x(3) + zload;
para = li*x(2)*seri/(seri+li*x(2));
zin = para + li*x(1);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
```
%% import file and initialize
clc;
clear all
importfile('A15.txt');

freq = A15(:,1).*1e6;          % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;

lb = [1 1 1]; % setting lower bound and upper bound
ub = [1000 1000 1000];
length = 15;

for i=1:5000
    start(i,:) = 1000*rand(1,3); %generating random start points
    handle = @(x)Pi3_opt(x,A15);

    % get the optimum Xc1, Xl and Xc2 at frequency f(i)
    [b,VSWR,exitflag,output] = optimize (start(i,:),lb,ub,handle);

    % storing all values
    out = output;
    xc1(i) = b(1);
    xc2(i) = b(2);
    xl(i) = b(3);

    C1(i) = 1e12 * 1/(omega(8)*xc1(i));
    C2(i) = 1e12 * 1/(omega(8)*xc2(i));
    L(i) = 1e9 * xl(i)/omega(8);

    % plot VSWR at all frequencies
    % calculate VSWR at all frequency points
    for j=1:length
        zload(j) = real(j) + 1i*imag(j);
        xcl = 1e12./(omega(j)*C1(i));
        xc2 = 1e12./(omega(j)*C2(i));
        xl = omega(j)*L(i)*1e-9;
        x = [xcl,xc2,xl];
        y(j) = Pi3(x,zload(j));
    end

    Vmax(i) = max(y);
end

[VSWRmax,k] = min(Vmax);
c1 = C1(k);
c2 = C2(k);
l = L(k);
for m=1:length
    zload(m) = real(m) + 1i*imag(m);
    xclt = 1e12./(omega(m)*C1(k));
    xc2t = 1e12./(omega(m)*C2(k));
    xlt = omega(m)*L(k)*1e-9;
    x = [xclt,xc2t,xlt];
    g(m) = Pi3(x,zload(m));
end
plot(freq,g);
xlabel('Frequency/Hz');
ylabel('VSWR');
title('VSWR with matching network optimized');
%% import file and initialize
clc;
clear all
importfile('A15.txt');

freq = A15(:,1).*1e6; % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;

num=500;

a=linspace(1,20,num);
b=linspace(1,20,num);
c=linspace(1,20,num);

tic;
for j=1:num
    xc1s=a(j);
j
    for k=1:num
        xc2s=b(k);
        for n=1:num
            xls=c(n);
            C1 = 1e12 * 1/(omega(8)*xc1s);
            C2 = 1e12 * 1/(omega(8)*xc2s);
            L = 1e9 * xls/omega(8);
            for i=1:15
                xc1 = 1e12./(omega(i)*C1);
                xc2 = 1e12./(omega(i)*C2);
                xl = omega(i)*L*1e-9;
                zload = real(i)+1i*imag(i);
                x0=[xc1,xc2,xl];
                v(i)=Pi3(x0,zload);
            end
            VSWR(j,k,n)=max(v);
        end
    end
end
[toc]=min(VSWR(:));
[J,K,N]=ind2sub(size(VSWR),ind);

XC1=a(J);
XC2=b(K);
XL=c(N);

t=toc;
C1 = 1e12 * 1/(omega(8)*XC1);
C2 = 1e12 * 1/(omega(8)*XC2);
L = 1e9 * XL/omega(8);

for i=1:15
    xc1 = 1e12./omega(i)*C1;
    xc2 = 1e12./omega(i)*C2;
    xl = omega(i)*L*1e-9;
    zload = real(i)+i*imag(i);
    x0=[xc1,xc2,xl];
    y(i)=Pi3(x0,zload);
end

plot(y);