Modeling and Analysis of Transmission Line Networks

Sasan H. Ardalan

Abstract

A data structure and algorithm for representing, and *simultaneously* solving for all nodes within complex transmission line systems is presented. The method is based on representing the network as a recursive tree structure and solving for the voltage, current, and impedance at each node using recursive programming techniques. First, all frequency dependent parameters within the tree structure are updated, then in a post-order traversing of the tree, the impedances at each node are computed followed by a pre-order traversing of the tree to compute node voltages and currents. The technique is useful in networks with many branches and mixed transmission line characteristics. Applications include the modeling and simulation of pulse propagation in distribution line carrier networks, local area networks, and the digital subscriber loop with bridged taps, and plane wave propagation . This paper expands on the original paper [1] and includes simulations in support of the algorithm.

http://trantopcalc.sourceforge.net/

Table of Contents

I Introduction

In many digital communication systems, complex transmission line networks are encountered which contain mixed transmission lines with different characteristics and many branches. Examples include the distribution line carrier network, the digital subscriber loop with bridge taps, and certain local area network configurations. The transmission line network usually introduces severe magnitude and phase distortion resulting in the degradation of bit error rate performance in digital transmission systems.

In this paper we present a program for computer modeling and simulation of complex transmission line networks. The network is represented in the computer by a recursive binary tree data structure. Using recursive programming techniques, the node voltage current, and impedance at each node within the tree structure is computed. In this manner the frequency response of the network, from the source node to the receiving node is computed. The impulse response or the pulse response of the network is then calculated from the frequency response using the Fast Fourier Transforms.

Computer programs for modeling transmission line networks have been written using ABCD parameters [2]. In this paper a technique in which the frequency response at all nodes within the network are obtained simultaneously is presented. The technique is also suitable for the computer aided design and modeling of digital communication systems, with complex transmission line networks.

II Transmission Line Networks

Consider the basic problem of simulating pulse transmission through a loaded transmission line. Assuming that the pulse is band-limited with a cutoff frequency of f_c , we can obtain the pulse response by computing the inverse FFT of the complex multiplication of the frequency response of the pulse and the transmission line network. Therefore, as a first step in calculating the frequency response of the network, we analyze the network response to a single sinusoid of frequency f_0. Consider the loaded transmission line connected to the generator E_g through a source impedance Z_s as shown in Fig. 1 [3].

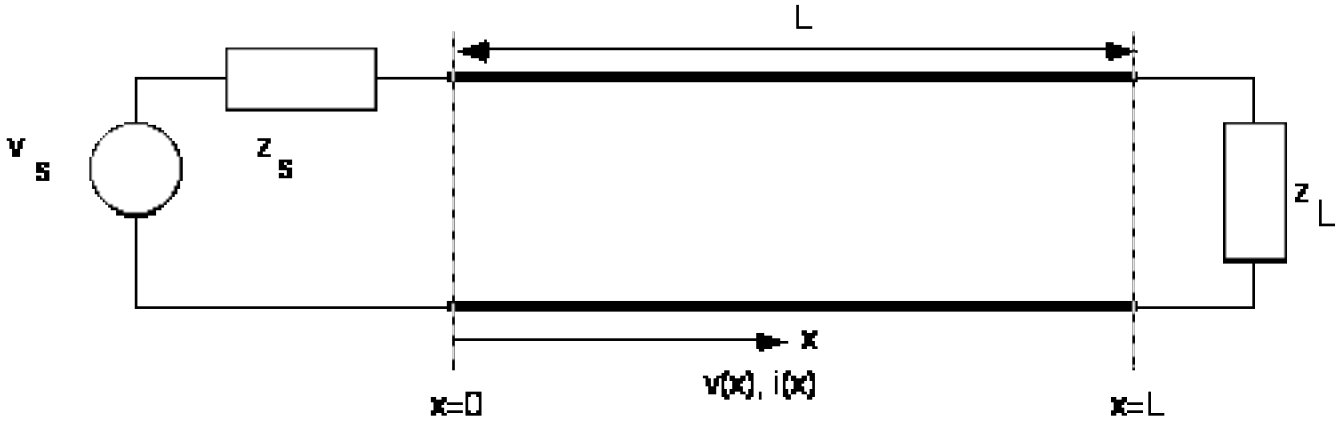


Figure 1. Generator connected to loaded transmission line

The voltage and current at any point on the transmission line can be obtained from the following expressions:

$$v(x) = \frac{v_s z_0}{z_0 + z_s} e^{-\gamma x} \frac{1 + \Gamma_L e^{-2\gamma(L-x)}}{1 - \Gamma_s \Gamma_L e^{-2\gamma L}} \tag{1}$$

$$i(x) = \frac{v_s}{z_0 + z_s} e^{-\gamma x} \frac{1 - \Gamma_L e^{-2\gamma(L-x)}}{1 - \Gamma_s \Gamma_L e^{-2\gamma L}} \tag{2}$$

In the above expressions

$$\gamma = \sqrt{(r + j\omega l)(g + j\omega c)} \tag{3}$$

is the propagation constant and

$$z_0 = \sqrt{\frac{r + j\omega l}{g + j\omega c}} \tag{4}$$

is the characteristic impedance of the transmission line. The expressions for the source and load reflection coefficients are,

$$\Gamma_L = \frac{z_L - z_0}{z_L + z_0} \tag{5}$$

$$\Gamma_s = \frac{z_s - z_0}{z_s + z_0} \tag{6}$$

The expression for *v(x)* includes the superposition of all waves reflecting from the source and load mismatches. This can be seen by a Taylor series expansion of (1)

$$v(x) = \frac{v_s z_0}{z_0 + z_s} [\, e^{-\gamma x} + \Gamma_L e^{-\gamma(L-x)} + \Gamma_L \Gamma_s e^{-\gamma(2L+x)} + \Gamma_L^2 \Gamma_s e^{-\gamma(3L-x)} + \Gamma_L^2 \Gamma_s^2 e^{-\gamma(3L+x)} + \dots \,] \tag{7}$$

To obtain the shape of the pulse at the load, we evaluate *v(L)* at frequencies from *f=0* to *f=fc* in discrete steps where f_c is the cutoff frequency of the band-limited pulse. The number of points must be a power of 2 such that the inverse FFT may be used to obtain the sampled pulse response at the load.

Consider now the case where the boundary voltage and current are known on a section of transmission line. See Fig. 2. Evaluate *v(0)* in (4) and then compute.

$$\frac{v(x)}{v(0)} = e^{-\gamma x}\frac{1 + \Gamma_L e^{-2\gamma(L-x)}}{1 + \Gamma_L e^{-2\gamma L}} \tag{8}$$

Also

$$\frac{i(x)}{i(0)} = e^{-\gamma x}\frac{1 - \Gamma_L e^{-2\gamma(L-x)}}{1 - \Gamma_L e^{-2\gamma L}} \tag{9}$$

Thus using (8) and (9) the voltage and current can be evaluated at any point on the transmission line given the boundary voltage and current.

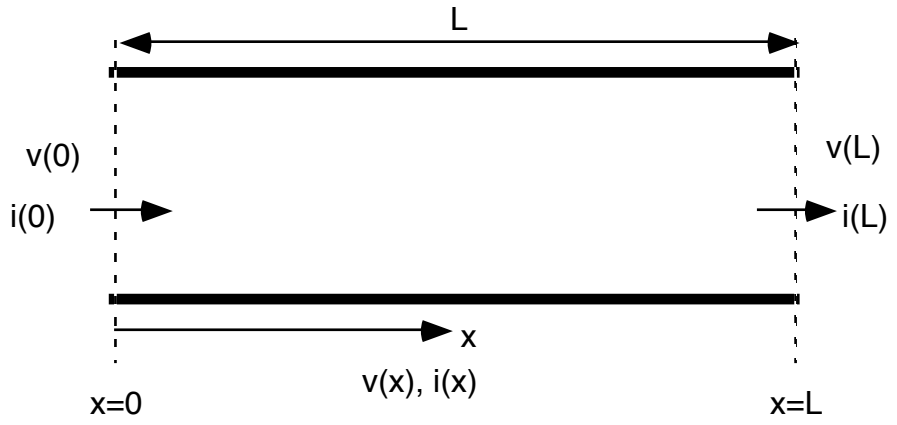


Figure 2. Section of transmission line with boundary voltages and currents

With the above preliminaries, we will examine the simple network in Fig. 3 and present a methodology for its solution. In Fig. 3, the nodes have been labeled n_1 through n_5. To solve this network, that is to obtain the voltage and current at each node and at any location within the network, consider equation (1). This equation suggests that if the impedance at node n_1 was known then the voltage and current at node n_1 can be calculated from the generator and source impedance. Thus the first step is to obtain the impedance at n_1. This impedance is seen to consist of the parallel combination of the impedance looking into n_5 and n_2 from n_1.

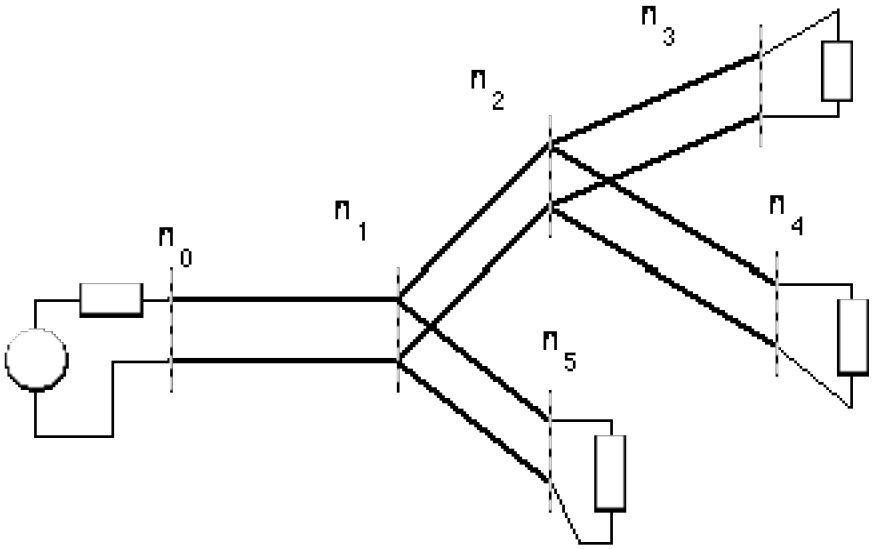


Figure 3. Example transmission line network

These impedances can be obtained by noting that (Fig. 4),

$$Z_{in}(x) = \frac{1 + \Gamma_L e^{-2\gamma(L-x)}}{1 - \Gamma_L e^{-2\gamma(L-x)}} Z_0 \tag{10}$$

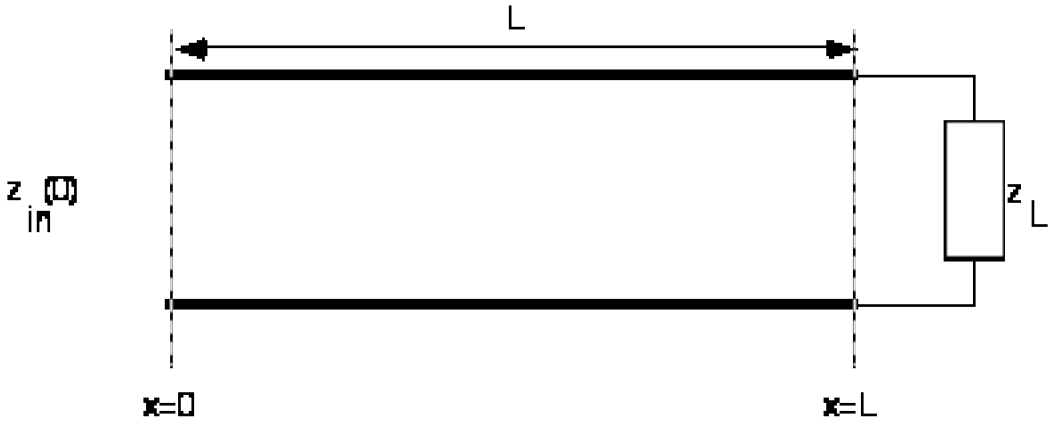


Figure 4. Input impedance of a loaded transmission line.

Thus, the first step is to calculate the impedances looking into n_3 and n_4 from n_2. The parallel combination forms the impedance at n_2. The impedance at n_1 is thus calculated by the parallel combination of the impedances looking into n_2 and n_5.

Therefore, the following methodology is suggested for solving the network. In the first pass, starting from the three loaded end nodes, the impedances are calculated and the parallel combination of these impedances at the parent node forms the parent node impedance. Working backward in this manner, the impedance at the root node (n_1 in the example) is calculated. Using (1) the voltage and current at the root node n_1 is calculated. Using (8) and (9) and the boundary voltages and currents, calculated at the parent node, the voltage

and current at each node in the network can be calculated. Note that the current at each node is split into two currents flowing into each node.

In the next section, a computer program will be introduced which uses recursion and recursive data structures available in C to solve complex transmission line networks.

III Recursive Programming and Data Structures

To introduce the algorithm for solving a complex transmission line network, we first consider the case where the network is limited to the binary tree structure shown in Fig. 5. In the figure, the generator is connected to the root of the tree through a source impedance Z_s. The tree consists of nodes which are either parents or leaves. A leaf is a node which is terminated on a load. For example, n_3, n_4, n_6, n_7, n_9, n_{11}, and n_{12}. Parent nodes have two branches. A left branch and a right branch. Nodes n_1,n_2, n_5, n_8, and n_{10} are parent nodes.

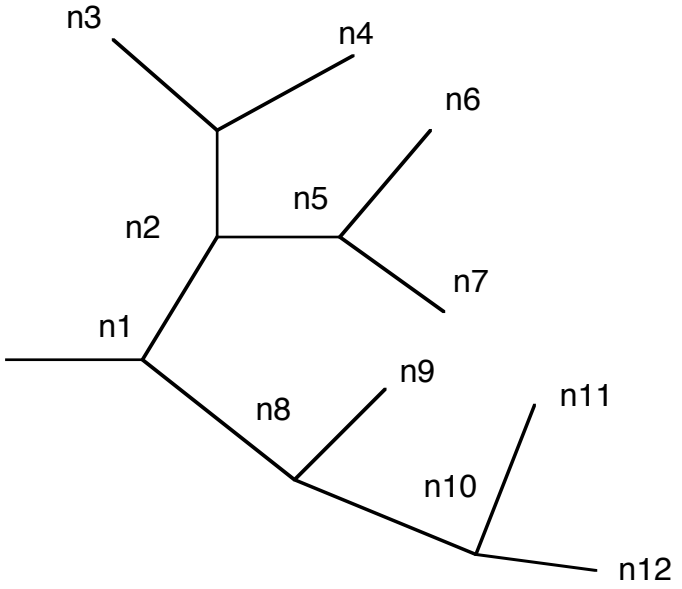


Figure 5. Transmission line network as a tree structure

In general each branch represents a transmission line with different characteristics and lengths. Each section of transmission line is associated with the node on which it terminates. Thus the section of transmission line from the generator to the root node n_1 is described in the data structure pointed to by n_1. This concept is described below.

Each node has an associated data structure which occupies memory locations. A pointer can be defined which points to the data structure in memory. As nodes are added to the tree, memory is dynamically allocated for the data structure and a pointer is defined. Thus, for the nodes of the network in Fig. 5 the following data structure can be defined in C.

```
struct  node. {
       struct node *left;
       struct node *right;
       struct node *parent;
       char name [16];
       float r,l,c,g;
       float length
       complex Z_Left;
       complex Z_Right;
       complex Z_L;
       complex  node_voltage;
       complex  left_current;
```

```
        complex  right_current;
        complex input_current;
        complex  Z0;
        complex gamma;.
}
```

Within the data structure definition are three pointers to data structures of the same type. Thus the data structure is recursive. Two pointers point to the left and right nodes while the third pointer points to the parent node. Three cases are immediately evident. If the node is a leaf then the left and right node pointers are NULL. Otherwise, they will point to the left and right child nodes attached to the node. If the node is the root node, then the pointer to the parent will be NULL.

The other data types within the structure represent data necessary to describe the node. These can be classified into two groups. One group defines the name of the node and the characteristics of the transmission line (e.g., r,l,c,g and Z_0 and g). The other group represents data which are calculated and depend on the network. These include the voltage at the node, the current flowing into the right and left nodes, and the impedances looking into the nodes.

It is very convenient to access data in a data structure using pointers to data structures. For example, to assign the variable *Z* the value of the characteristic impedance at the node pointed to by *n*p we write,

$Z = np \text{ -> } Z_0$

To access the characteristic impedance of the left child node of the node pointed to by *np* we write,

$Z = np \text{ -> } left \text{ -> } Z_0$

At this stage, we will describe recursive functions which are used to compute the impedances, voltage and currents at each node. First we introduce two methods for traversing tree data structures.

IV Traversing Trees

There are general methods for traversing trees [4]. We will apply two of these methods to solve the tree network.

4.1 Postorder Listing

Postorder traversing of trees is illustrated in Fig. 6. This method is useful in the first pass needed prior to solving for the voltages and currents of the network. As pointed out earlier the impedances at each node must be computed. Thus in Fig. 6, the impedance at 3 is the parallel combination of the impedances of the loaded transmission lines 1 and 2. Similarly, the impedance at 6 is computed from 4 and 5. Once the impedance of 3 and 6 are computed, the impedance at 7 can be calculated and so on. Careful study of the figure will show that the numbering schemes corresponds to the order in which the impedance calculations must be carried out. This order of traversing the tree is termed *postorder* listing. The method is summarized below [4]:

(1) If a tree is composed of only a single node, the post order listing consists of just that single node.
(2) If a tree consists of more than one node, the *postorder* listing consists of the *postorder* listing of each subtree, in left-to-right order, followed by the root.

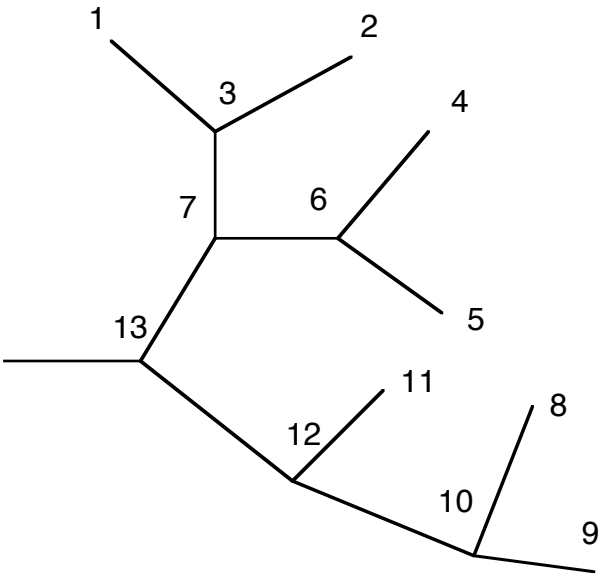


Figure 6. Post-order traversing of tree for impedance calculations

4.2 Preorder Listing

This method is used in the calculation of the voltages and currents at each node once the impedances have been determined. *Preorder* listing is illustrated in Fig. 7. Thus, once the boundary voltage at node 1 is known, the voltage at node 2 can be computed (since the impedance at 2 is also known from the first *postorder* traversing in computing the impedance). From 2, the voltage at 3 and 6 can be computed and so on. The *preorder* listing method is summarized below [3]:
(1) If a tree is composed of a single node, the *preorder* listing consists of just that single node.
(2) If a tree consists of more than one node, the *preorder* listing consists of the root, followed by the *preorder* listing of each sub tree in left-to-right order.

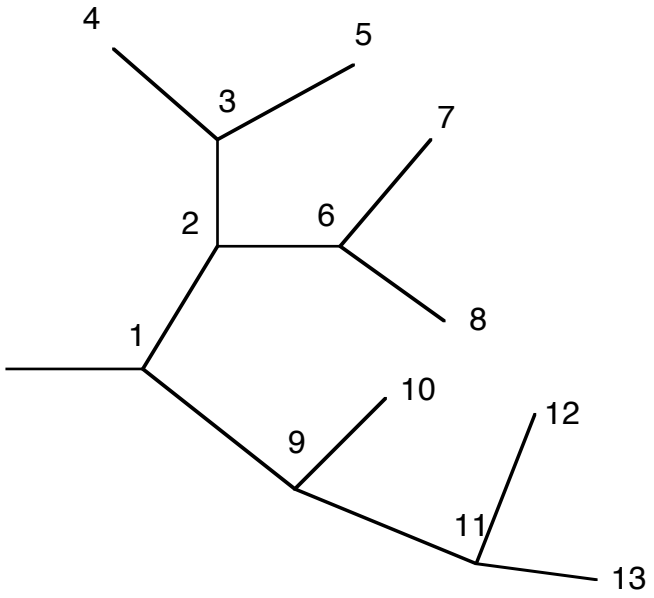


Figure 7 Pre-order traversing of tree for current and voltage calculations

V Recursive Calculation of Network Impedances

The following C code presents a recursive function that calculates the impedance at each node using a post order traversing of the tree structure

```
(1) Complex calculate_impedance (root)
       struct node * root;.
     {
(2)    if ((root-> left == NULL) && (root -> right == NULL))  {
       /*
        * we are at a leaf
        *  calculate impedance a distance root ->length away from theload, root ->Z_L
        *  calculate reflection coefficient at load
        */
              rcl = calc_rcl (root-> Z_L ,root -> Z_0);
              return (line_impedance (rcl, root -> Z_0,
                            root -> gamma,root -> length);
       }
(3)    root -> left_impedance = calculate_impedance (root -> left);
       root -> right_impedance = calculate_impedance (root -> right);
       /*
         *  calculate parallel combination of left and right
         *  impedance
         */
(4)    root ->Z_L  = Z_impedance_parallel (root -> left_impedance,
                                   root-> right_impedance);
       rcl = calc_rcl (root ->Z_L, root -> Z_0 );
(5)    return ((line_impedance(rcl, root ->Z_0 ,  root -> gamma,
                     root->length));
       }
```

Function explanation:.

(1) The function argument, *root*, is a pointer to a node within the tree. The function returns the impedance "looking into a node," and it is complex.
.
(2) The function checks to see if the node is a leaf. If it is, then it calculates and returns the impedance looking into the leaf node. This is one of the terminating conditions of the recursive function. In other words, once a leaf node is reached, the function returns. The function *line_impedance ()* calculates the impedance based on equation (10)

(3) If the node is not a leaf, then the function recursively calls itself by passing the left branch node pointer. After returning the impedance looking into the left node, the function is recursively called to calculate the right branch node impedance

(4) Once the left and right impedances at the node are known, the parallel combination is computed. This produces the total node impedance.

(5) At this point the impedance looking into the node is computed and returned by the function. This is also another terminating condition.

After this function is called, the impedances at all nodes are stored in the data structures pointed to by each node within the network.

VI Recursive Calculation of Node Currents and Voltage.

After the function *calculate_impedance (root)* is called, each node contains the terminating or left and right branch impedance. The next recursive function, calculates the total current flowing into each node including the left and right branch currents. The function makes use of equation (9) to compute the node current *I(L)* based on knowledge of the boundary current *I(0)*. The function performs a pre-order traversing of the tree network.

```
(1) calc_current (root, i_input)
     struct node * root;
     complex i_input;
     {
(2)            complex line_current();
       /*
         *  evaluates boundary current based on Eq. (9)
         */
        complex calc_left_current ();
        complex calc_right_current ();
       /*
         *  calculate the total node current based on equation (9)
         *  i_input is the boundary current corresponding to I(0) in
         *  the equation
         */
(3)    root -> input_current = line_current (root, i_input, root ->Z_L ,
                              root -> length);
(4)    if (root -> left ! = (struct node *) NULL)    {
              root -> left_current = calc_left_current (root -> left_impedance,
              root -> right_impedance, root -> input_current);
              calc_current (root -> left, root -> left_current);
       }
(5)    if (root -> right ! = (struct node  *) NULL) {
              root -> right_current = calc_right_current (root -> left_impulse,
              root -> right_impulse, root _> input_current);
              calc_current (root ->right , root-> right_current);
              }
```

Explanation:

(1) The *root* argument is a pointer to the current node, *i_input* is the complex boundary current corresponding to *I(0)* in equation (9). That is, it is the total current flowing into the transmission line associated with the node.

(2) The function *line_current ()* computes the current *I(L)* in equation (9). The functions *calc_left_current ()* and *calc_right_current (),* calculate the currents flowing into the left and right branches.

(3)This operation calculates *I(L)* and stores it in the data structure of the node. Note that since this is a recursive procedure for traversing all the nodes of the tree, this operation will occur for each node.

(4) If the left branch is not a termination, then recursively call the function by moving to the node attached to the current node, *root*. However, first compute the boundary current *I(0)* flowing into the node, in this case *left_current*.

(5) The same as in step 4 but for the right branch.

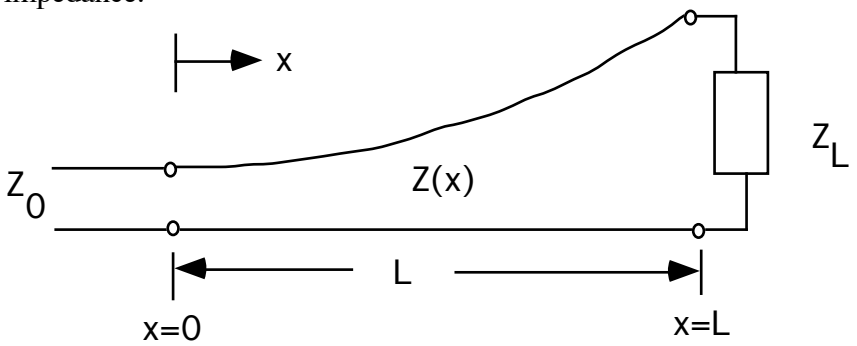
After this procedure is called, all node data structures will contain the total current, and the left and right currents. Note that when the procedure is first called, *i_input* is the total current flowing from the generator into the network. It is computed based on equation (2).

VII Computer Simulation Results

7.1 Accuracy of Modeling Continuous Taper with Cascaded Sections

The purpose of this experiment is to determine whether the results produced by *TranTopCalc* for a transmission line with a tapered characteristic impedance converge to the true solution as the number of sections is increased. In addition, it is desired to determine the accuracy to be expected in a given simulation as a function of the number of sections used in *TranTopCalc*. The following was done jointly with Gary Ybarra at Duke University while he was at NC State.

Consider the transmission line below with an exponentially tapered characteristic impedance.



In our simulation we used a source impedance of 100Ω and a load impedance of 500Ω. Also, the total length is L=10m. We used sections of two wire cable with geometry below.

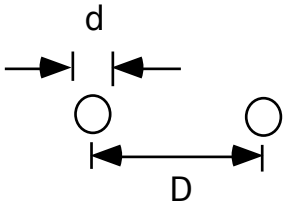


Figure 8

The characteristic impedance is,

$$Z_0 = 120 \ln(\frac{2D}{d})$$

For the continuous exponential taper

$$\ln Z = \frac{x}{L} \ln Z_L$$

(We sincerely apologize for using the symbol "L" for Load, inductance, and Length, hoping the context will provide identity).

The exact differential equation relating the reflection coefficient to location along the line is (Riccati equation, see R.E. Collin, *Foundations for Microwave Engineering,* page 252)

$$\frac{d\Gamma}{dx} = j2\beta\Gamma - \frac{1}{2}(1 - \Gamma^2)\frac{d \ln Z_L}{dx}$$

The exact solution to this differential equation for the input reflection coefficient for the case of an exponential taper is

$$\Gamma_i = \frac{A \sin(BL/2)}{B \cos(BL/2) + j 2\beta \sin(BL/2)}$$

$$A = \frac{\ln Z_L}{L}, \quad B = \sqrt{4\beta^2 - A^2}$$

In order to test our program *TranTopCalc*, we cascaded sections of two wire transmission lines. *TranTopCalc* requires values of inductance L and capacitance C.

$$L = \frac{Z_0}{v_c} z_L^{\frac{x}{L}} \qquad C = \frac{1}{v_c Z_0} z_L^{-\frac{x}{L}}$$

where v_c is the velocity of light in free space.

Figure 9 shows a comparison of the results from *TranTopCalc* and the exact solution to the Riccati equation. The best comparison is in the Figure 10 which is a "ZOOMED" view of the figure. A simulation was performed with 500 sections and the result could not be distinguished from the exact solution. The conclusion is that *TranTopCalc* produces the exact solution asymptotically as the number of sections is increased. The reason for this asymptotic exactness is that *TranTopCalc* includes the effects of **all** reflections, not a simple approximation where second and higher order reflections are neglected.

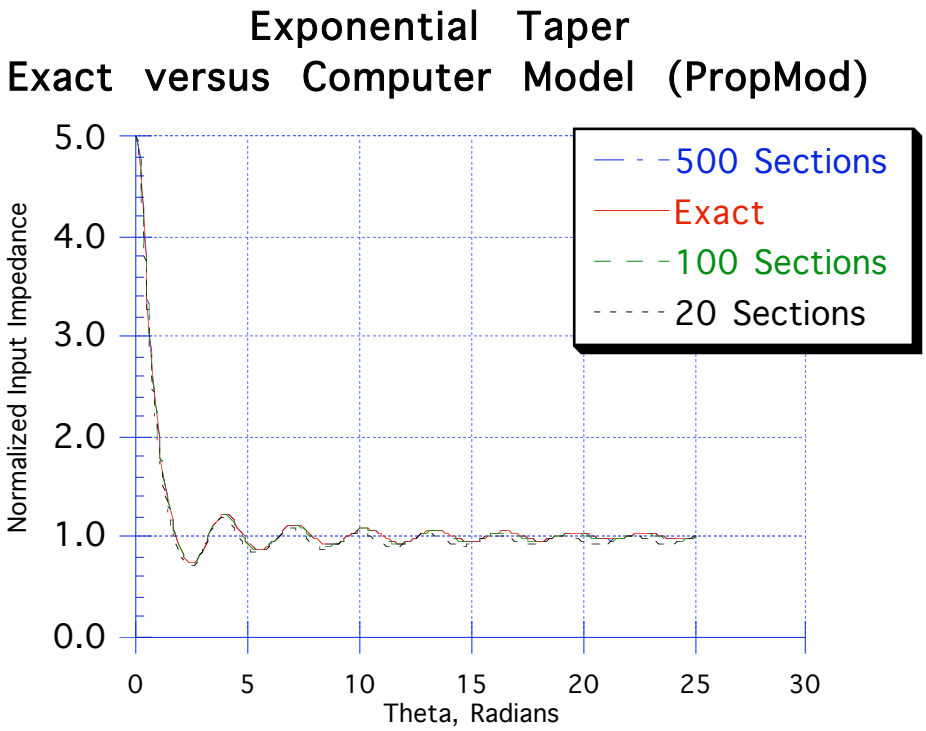


Figure 9.

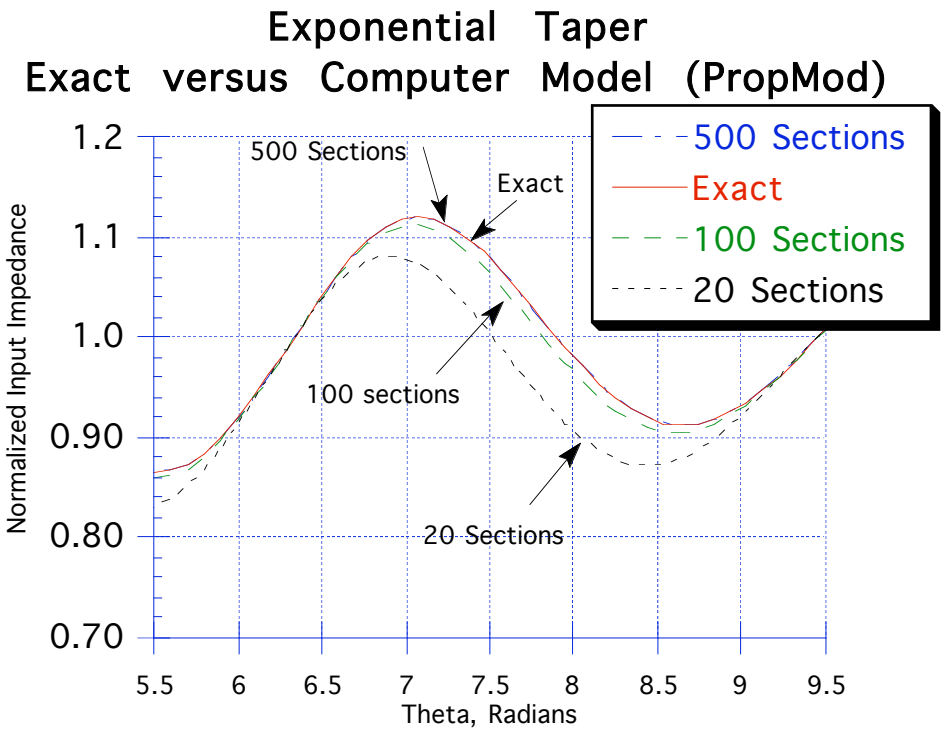


Figure 10.

7.2 Double Stub Tuner Modeling and Simulation

The following double stub tuner is described in [5]. The double stub tuner matches the load to 50 Ohms. In modeling and simulating this network, we will use the coaxial transmission line. The transmission line parameters are for a 50 Ohm characteristic impedance. The network is matched at 10MHz. The wavelength is 20m.

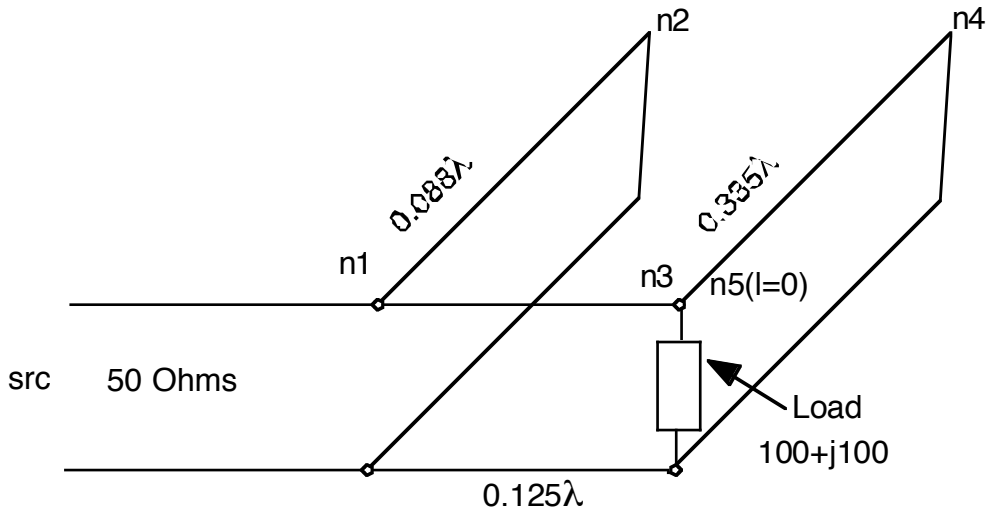


Figure 11

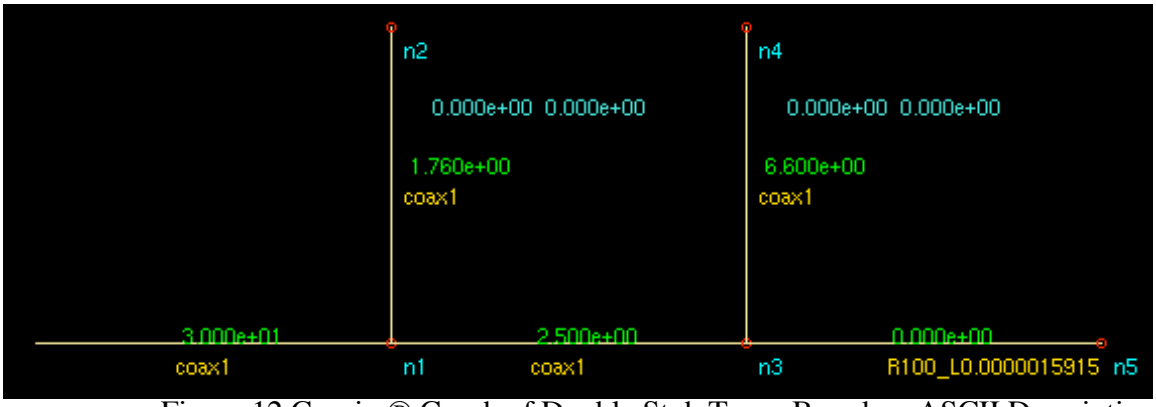


Figure 12 Capsim® Graph of Double Stub Tuner Based on ASCII Description

The network is described in *TranTopCalc* using the following topology (stored in an ASCII file):

```
n1 n2 n3
n3 n4 n5
n2
n4
n5
end
n1 coax1 30
n2 coax1 1.76 0 0
n3 coax1 2.5
n4 coax1 6.6 0 0
n5 R100_L0.0000015915 0 open
```

Not e that the lengths are in meters based on the 20m wavelength. The load at 10MHz, 100+j100 is modeled using lumped circuit parameters in node n5 : R100_L0.0000015915.

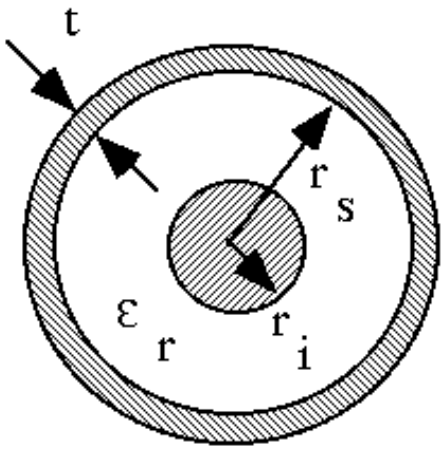


Figure 13 Coax with Four Parameters

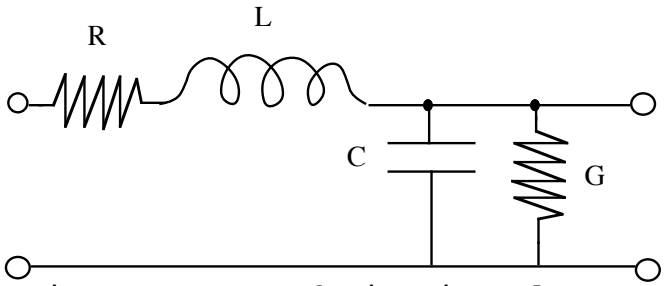


Figure 14 Lumped Circuit Element

```
#19m 0.053437723 5.530183e-7 5.157361e-11 1.031472e-9
#22m 0.10749682  5.405909e-7 5.157361e-11 1.031472e-9
#24m 0.17025507  5.903004e-7 5.157361e-11 1.031472e-9
#26m 0.27340231  6.213688e-7 5.157361e-11 1.031472e-9
coax1       0.001    0.0035   2.25  0.001
balsh1   0.001  0.0035 2.25 0.002
wireabg1 0.001 0.1 2.25 0.0
paral1   0.001 0.01 2.25 0.0
```

Figure 15 Contents of the File *trans_types.dat* with Transmission Line Type Parameters

The performance of the double stub tuner is shown in Figures 16-18. The VSWR is plotted against the frequency (+- 10% of 10MHz).

This result was obtained using *TranTopCalc* with integrated TCL which makes all calculations of the network including setting frequency and node parameters available through TCL commands. The Appendix lists the TCL script used to calculate the variation of VSWR versus length.

Finally, Figure 19 shows the impulse response of the double stub tuner. The sampling rate is 1GHz.

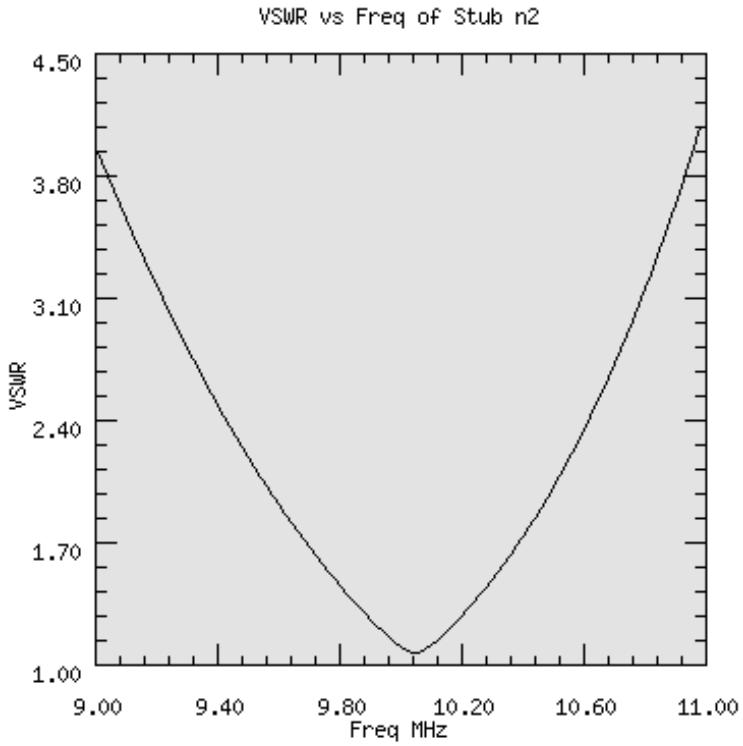


Figure 16

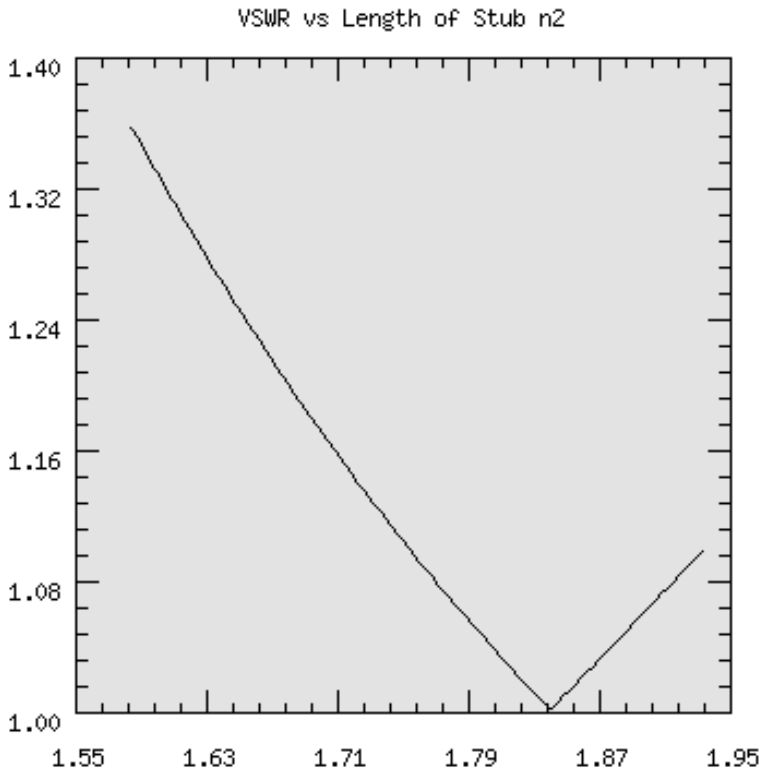
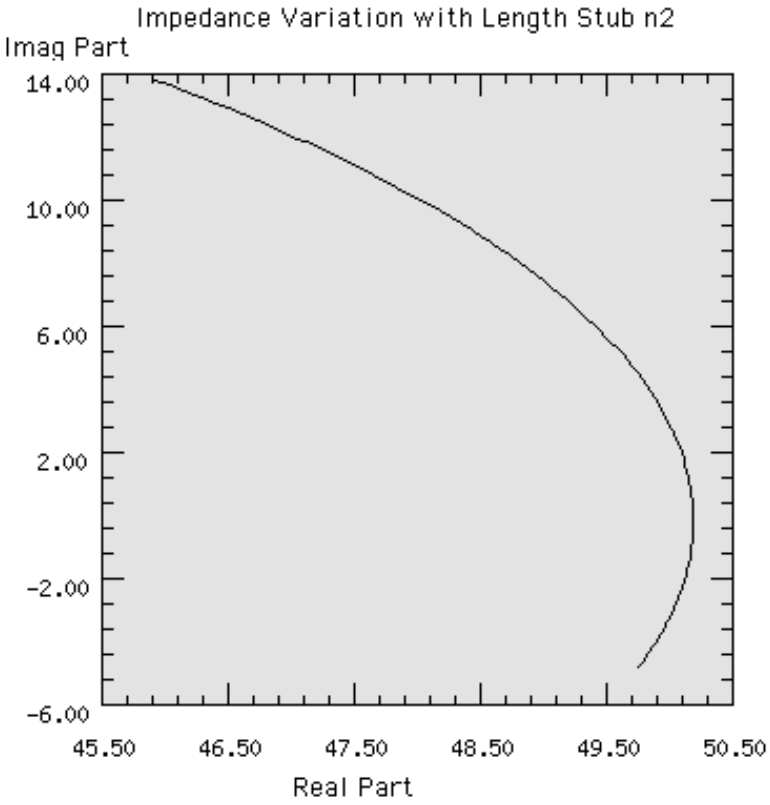

VSWR vs Length of Stub n2
1.40
1.32
1.24
1.16
1.08
1.00
1.55
1.63
1.71
1.79
1.87
1.95

Figure 17


Impedance Variation with Length Stub n2
Imag Part
14.00
10.00
6.00
2.00
-2.00
-6.00
45.50
46.50
47.50
48.50
49.50
50.50
Real Part
Figure 18

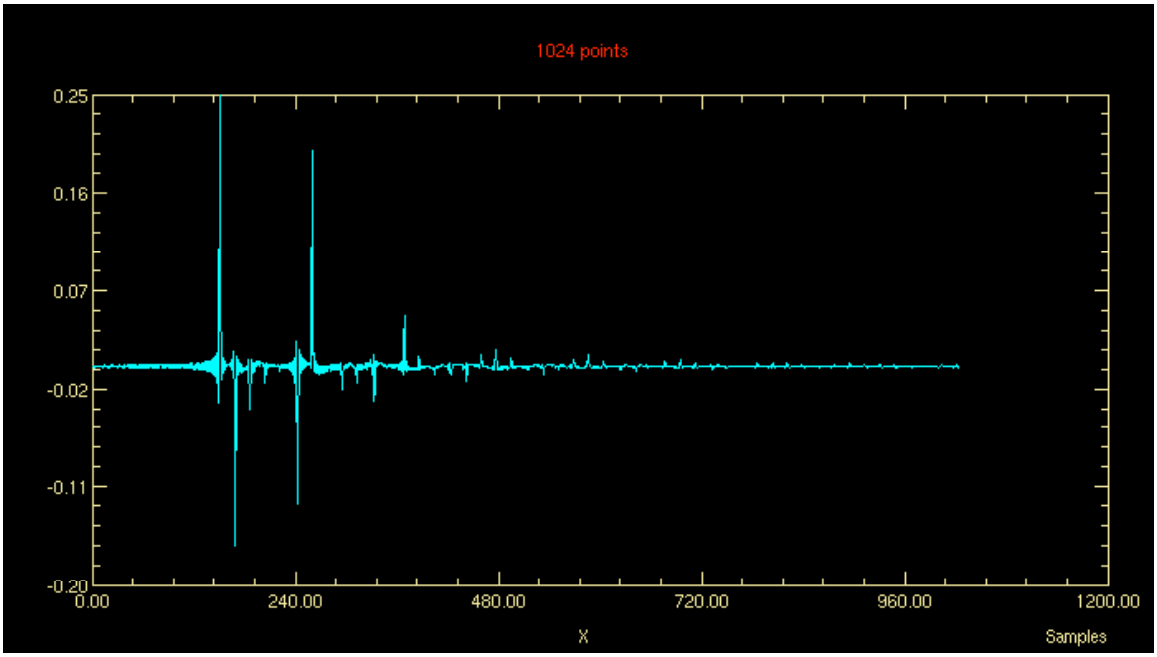


Figure 19 Impulse Response at Node n1 for Double Stub Tuner Sampling Rate 1GHz.

VIII Conclusions

In this technical report, a computer program is described which simultaneously solves for all nodes within complex networks of transmission lines. A tree data structure was introduced for representing the network in the computer. Recursive procedures were presented for traversing the tree data structure to compute the impedance, voltage and current at each node within the network. Simulation results were then presented in which the impulse response of a test network composed of transmission lines of various characteristics and lengths was computed. The impulse response was then related to the network in terms of the predicted reflections and delays.

The program efficiently solves complex transmission line networks and has applications in the area of Computer Aided Design (CAD) of digital communication networks. Specific applications include the Distribution Line Carrier Network, the digital subscriber loop, and Local Area Networks.

IX References

[1] Sasan Ardalan, Susan Alexander, Ken Shuey, *Computer Modeling and Analysis of Complex Transmission Line Networks,* Center for Communications and Signal Processing, North Carolina State University, 1987

[2] D.G. Messerschmitt, "Transmission line modeling program written in C,"*IEEE Journal on Selected Areas in Communications*,Vol. SAC-2, pp. 148-153, January 1984.

[3] Dworsky, Lawrence N, *Modern Transmission Line Theory and Applications*,John Wiley and Sons, Inc., 1979.

[4] Gerald E. Sobelman, David E. Krekelberg, *.Advanced C Techniques and Applications,* Que Corporation, Indianapolis, Indiana,, 1985.

[5] Robert Grover Brown, Robert A. Sharpe, William Lewis Hughes, Robert E. Post, *Lines, Waves, and Antennas, Second Edition,* Ronald Press Company, 1973 .

X Appendix

TranTopCalc TCL Code for Varying Length of Stub n2 in Double Stub Tuner

```
#  Transmission Line Topology Calculation and Analysis (TranTopCalc)
#  Command Tool Scripts
#    Copyright (C) 2006  Sasan H Ardalan
#
#    This script is free software; you can redistribute it and/or
#    modify it under the terms of the GNU  General Public
#    License as published by the Free Software Foundation; either
#    version 2 of the License, or (at your option) any later version.
#
#   This script is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#     General Public License for more details.
#
#    You should have received a copy of the GNU  General Public
#    License along with this library; if not, write to the Free Software
#    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
#    http://www.xcad.com
#    Raleigh, North Carolina
#

#
# This is a double stub tuner. At node n1 we have a match.
# node n2 is the second stub near the generator
# At 10MHz the network is matched.
#
# in the following script we change the length of the stub n2
# by +- 10% around the length where a match is acheived.
# 100 points are calculated.
# results are stored as follows.
# The complex input impedance is stored in a complex vector
# It is stored in a filex as complex numbers (a+bi)
# The VSWR and Length are stored in a list.
# This list is later stored in a file as two columns
#

#
# load the transmission line topology file "t.top"
#

tload t.top T
#print the topology
tpr $T
# set the complex generator and generator source impedance
cx 1.0 0.0 g
cx 50.0 0. zs
#calculate all nodes at 10MHz and store the impedances, z0, gamma,
# and current for each node
tcalc $T 1e7 $g $zs timp tcur
#print total input impedance and current (complex scalars)
vpr $timp
vpr $tcur
# get the length of the stub n2
ngetlength $T n2 n4len
#calculate +-10% and dlength
```

```
set lenBegin [ expr $n4len -0.1*$n4len]
set lenEnd [ expr $n4len + 0.1*$n4len ]
set dlen [expr ($lenEnd-$lenBegin)/100.0 ]
#create complex and real vectors to store results (much more efficient than lists)
vcxcr  zvlen 100
vcr zmag 100
# get z0 and gamma (propagation constant) to use in VSWR calculation
nz0gamma $T n1 z0 gamma
# initialize list to store results (in addition to storing in vectors)
set results [list {}]

puts "Iterating over length"

for { set i 0 } { $i<100 } {incr i} {
    set len [expr $lenBegin +$i*$dlen ]
    #change the length of the stub
    nsetlength $T n2 $len
    #recalculate the network for new length
    tcalc $T 1e7 $g $zs
    # store all the computed values for node n1 (left right impedance, current )
    # the variables are automatically created
    nvalues $T n1
    # compute total inpedance at node n1 (variables $n1_zl and
    #  $n1_zr were automatically created in the previous command)
    zpar $n1_zl $n1_zr zt
    vpr $zt
    #store the input impedance at n1 in the complex vector $zvlen (z versus length)
    vcxset $zvlen  $zt $i
    #get the magnitude of the impedance
    cxgetpolar $zt zmagval theta

    #compute reflection coefficient
    rflc $zt $z0 rflc_n1
    #compute vswr
    vswr $rflc_n1 vswr_n1
    #store in real vector
    vset $zmag $zmagval $i


    #store vswr in list with length
    lappend results [list $len $vswr_n1 ]
}

#print resulst of input impedance
vpr $zvlen 14.6f zin_vs_len.dat -twocol

#
# create a file to store the vswr vs length
#
puts "Storing results in vswr.dat"
set out [ open vswr.dat w ]

# puts $out [ llength $results ]
#puts -nonewline $out "#Length"
#puts -nonewline $out "\t"
#puts    $out "MagZin"

foreach value $results {
   puts  -nonewline $out [lindex $value 0]
   puts  -nonewline $out "\t"
   puts   $out [lindex $value  1]
}
close $out
```