New Developments of Absorbing Boundary Algorithm for the Transmission Line Matrix (TLM) Method

by

QIYANG LI

A thesis presented to
the School of Graduate Studies and Research
of the University of Ottawa
in partial fulfilment of the requirement
for the Degree of

MASTER OF APPLIED SCIENCE

in

Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
University of Ottawa

ISBN 0-315-82538-3

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis. I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Qiyang Li

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Qiyang Li

Abstract

New developments of Absorbing Boundary (AB) Algorithms for the Transmission Line Matrix (TLM) Method have been presented in this thesis.

A new approach has been proposed to obtain the optimized Dissipation Absorbing Boundary Condition (ABC). In this approach, optimization is performed directly on the structure being studied while other approaches do the optimization indirectly, e.g., using a transmission line to simulate a waveguide and then perform the optimization on that transmission line. Therefore, a more accurate arrangement of the extra lossy region, in other words, Dissipation ABC, can be obtained without involving excessive computation space. Good absorption has been achieved with an extra lossy stub much shorter than other existing approaches. This approach has been implemented in 2D and 3D TLM. It shows its great advantage especially in 3D because it is unconditionally stable while many other approaches such as extrapolation approach, are unstable.

Great efforts have also been made in implementing Liao's ABC in 2D and 3D TLM. This ABC has better absorption than the above approach when the same extra computation space is used. However, this approach is unstable when its order is high. How to choose a proper order has been discussed in this thesis.

Since Liao's ABC has good absorption and instability, while the Dissipation ABC has good stability and relatively worse absorption, we can properly combine them together to achieve good stability and acceptable absorption. This combination approach has been carefully tested in this thesis and found feasible. Choosing a proper loss tangent for the Dissipation ABC has been discussed. This approach has been implemented in 2D and 3D TLM.

The last part of this thesis is devoted to demonstrating all the above new ABCs for TLM by solving a practical E-plane filter. Comparison has been made among them and also with experimental results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

This thesis is devoted to the study of the Absorbing Boundary (AB) algorithms for the Transmission Line Matrix (TLM) Method, a time domain numerical method that is very physical and can be easily applied to solve the most general kind of electromagnetic problems.

After J. C. Maxwell established the interdependence of electricity and magnetism in the form of a group of equations, namely Maxwell's equations, in 1870's, solving electromagnetic problems amounted to finding the solutions of Maxwell's equations that satisfy specific boundary conditions. For a long time the only approach to solve electromagnetic problems was analysis. Solutions were formulated in closed-form whenever possible. However, the application of analytical approaches is quite limited. They can only be applied to problems with regular structures, ideal boundary conditions, linear materials, etc.. However, the microwave, electromagnetic and optical techniques are more complex, involving irregular shapes, lossy media, nonlinear characteristic, etc.. To meet such requirements, numerical approaches must be used.

However, numerical approches have only been feasible since the advent of computers. Large scale computers have made it possible to develop numerical methods requiring huge storage, such as 3D time domain approches. The basic idea of most numerical methods is to discretize the mathematical equations governing the real problem to yield a system of

linear equations, and then solve them. However, there exist exceptions. Some approaches are based directly on physical principles instead of mathematical equations. They are easy to study because of their real physical meaning. TLM is one of such approaches.

Normally, numerical approaches can be roughly grouped into two categories, which are frequency domain and time domain approaches. The difference between them is that the former solve time-harmonic Maxwell's equations while the latter solve time-dependent Maxwell's equations. Every method has its advantages and limitations. Thus, it is important to choose the proper method which can achieve accurate results while consuming a minimum of computer time and memory when solving a particular problem.

Frequency domain numerical methods have been widely employed to find the solutions of deterministic, eigenvalue, eddy current problems and so on. Normally, analytical preprocessing is needed in these methods. Thus, the computer time and memory can be greatly reduced. Sometimes, even a quasi-closed-form solution can be obtained provided that the structure under study is simple. This is very convenient for engineering design. Also, if only the frequency domain response is required, these methods can achieve results more accurately than time domain approaches because no Fourier Transform is needed.

However, since frequency domain methods operate in the frequency domain and are based on the principle of superposition, they cannot be used as easily to solve time-depedent and nonlinear problems. In such cases, time domain numerical methods must be employed. Moreover, the frequency domain approaches can get solutions only at one frequency point at a time. To find the solution over a wide frequency range, the same computation procedure must be repeated for each frequency point. In such a case, time domain methods are of comparable or better efficiency since they can use impulse excitation to find the time domain response at certain observation points, and yield through the Fast Fourier Transform(FFT) the frequency response over a wide frequency range. This is particularly useful in solving problems such as filters, impedance transformers, etc. The TLM method used in this thesis is such a time domain method.

The TLM method was developed by P. B. Johns and his co-workers in the early 70's. The basic principle of this method is Huygens's principle of wave propagation which will be explained fully in Chapter 2. According to Huygens, any wave front can be considered as consisting of many infinitesimal sources. The wavelets generated by such sources form a new wave front. This concept can be proved to be identical to Maxwell's equations and is easily implemented on a computer. No analytical pre-processing is needed for this method.

To handle closed boundary problems, the present numerical approaches are sufficient. Some of them have even been developed into commercial software, such as the Electromagnetic Wave Simulator, developed by Hoefer and So [10] using TLM. However, microwave and electromagnetic techniques are not confined only to problems with closed boundaries. Many applications are open boundary problems, such as antenna problems, scattering problems, and so on. In these cases, the infinite space must be simulated. One way to do this is to make the computation region large enough so that there is no reflection from the boundary within the computation time. This will be very expensive in terms of computer time and memory.

An alternative is the absorbing boundary (AB) algorithm. An absorbing boundary is an artificial boundary which absorbs waves incident upon it with negligible reflection within the frequency range of interest. It can be placed very close to the excitation points so that the computation region is greatly reduced.

Modern microwave and optical techniques often operate over wide frequency ranges. Thus a wideband absorbing boundary is of great importance. Furthermore, high absorbing performance is also required especially in time domain approaches because frequency domain results obtained by Fourier Transform are sensitive to errors in the time domain response. Thirdly, a good absorbing boundary can handle waves coming from all directions.

Absorbing boundary conditions (ABCs) can be realised in a variety of ways, which can be classified as follows:

1. Infinite extent of the computational domain is simulated by terminating the

computation before reflections reach the area of interest(Benchmark approach).

2. The propagation space is made progressively lossy and thus absorbs all outgoing waves (Dissipation approach).
3. The field at the absorbing boundaries is predicted by sampling the space and time evolution of the fields before the boundary (Extrapolation approach).
4. The field incident on the boundary is convolved with the impulse response of open space (Discrete Green's Function or Johns Matrix approach).

In this thesis, a Liao's ABC has been studied in particular. It has good absorption over a wide frequency range.

Liao's ABC is a kind of extrapolation algorithm. It is potentially unstable, especially in the 3D case. Dissipation AB Algorithm has the advantage that it is stable, but they are less accurate. If they can be combined, one may get an absorbing boundary with both high accuracy and stability. This approach has been studied and realized in this thesis.

1.2 The organization of this thesis

Chapter 2 contains the basis of the TLM Method. Equivalence between voltage and current on transmission lines, and electric and magnetic field in Maxwell's equations is demonstrated. Series and shunt nodes in 2D-TLM, and distributed nodes in 3D-TLM are introduced. Scattering of Dirac impulses, introduction of boundary condition, simulation of lossy materials and calculation of fields are also briefly described.

Chapter 3 is devoted to introducing the recent developments of AB algorithms in time domain numerical methods. Benchmark, Dissipation, Extrapolation and Discrete Green's Function approach have been chosen as typical examples. These methods are compared, and special emphasis is placed on Extrapolation approaches because of their efficiency and good absorption.

Chapter 4 contains the first part of the new contribution of this thesis, namely the

implementation of Direct Optimized Dissipation ABC. Numerical results and discussions are given.

In Chapter 5, Liao's ABC has been carefully studied and implemented in 2D TLM. It has been tested in two waveguide structures. Good absorption has been observed.

Chapter 6 introduced a new concept which combines the Dissipation ABC with the extrapolation ABC to achieve good stability and acceptable absorption. How to choose a proper loss tangent has been discussed in this chapter.

Chapter 7 extends these approaches to the 3D-TLM mesh with distributed nodes. In the 3D case, instability is more serious than in 2D. Numerical results and discussions regarding this problem are given in this chapter.

In Chapter 8, the ABCs implemented in this thesis are applied to solve a specific problem in order to demonstrate their effectiveness. Results are compared with those published in the literature.

Chapter 9 contains the conclusion of this thesis. Possible future work is also pointed out.

1.3 Original contributions in this thesis

The original contributions in this thesis falls into three parts which are the implementation of the Direct Optimized Dissipation ABC approach, Liao's ABC approach and combination of Dissipation ABC and Liao's ABC approach in 2D and 3D TLM. They all have good absorption characteristic over a wide range of incident angles. The latter two ABCs are efficient in terms of computer time and memory. They have been tested in air-filled waveguide, dielectric-loaded waveguide, and parallel metal plate waveguide with infinitive width, etc.. Good results have been achieved. Instability has been found to be much more serious in 3D-TLM than in 2D. Several ways to suppress the instability have been introduced and tested. Finally, these ABCs have been employed in the analysis of an E-plane filter. Results have been compared to the published data.

Chapter 2

Review of Transmission Line Matrix (TLM) Method

2.1 Introduction

The TLM formalism has been developed by P. B. Johns in a number of original papers in the 1970's [1, 2, 13, 14, 15, 16]. His main idea is to discretize the space into rectangular elements in 2D and cubic elements in 3D. The corners of such elements are called "nodes". Nodes are connected to their neighbouring nodes by transmission lines along the sides of the above mentioned elements. Now, waves are considered travelling only along transmission lines and scattering at "nodes". The length of each side of the element is Δl. If the wave propagates at velocity c, and the time needed by an impulse to travel from one node to the next is defined as Δt, we can relate Δt and Δl by,

$$\Delta t = \Delta l / c \tag{2.1}$$

2.2 The 2D-TLM Method

2.2.1 Equivalence between Maxwell's Equations and 2D-TLM method

Fig. 2.1(a) shows the layout of a 2D-TLM mesh. In 2D-TLM, transmission lines can be interconnected in two ways, and thus can form two types of nodes, which are shunt and series nodes (Fig. 2.1(b),(c)). To solve 2D electromagnetic problems, either the shunt node

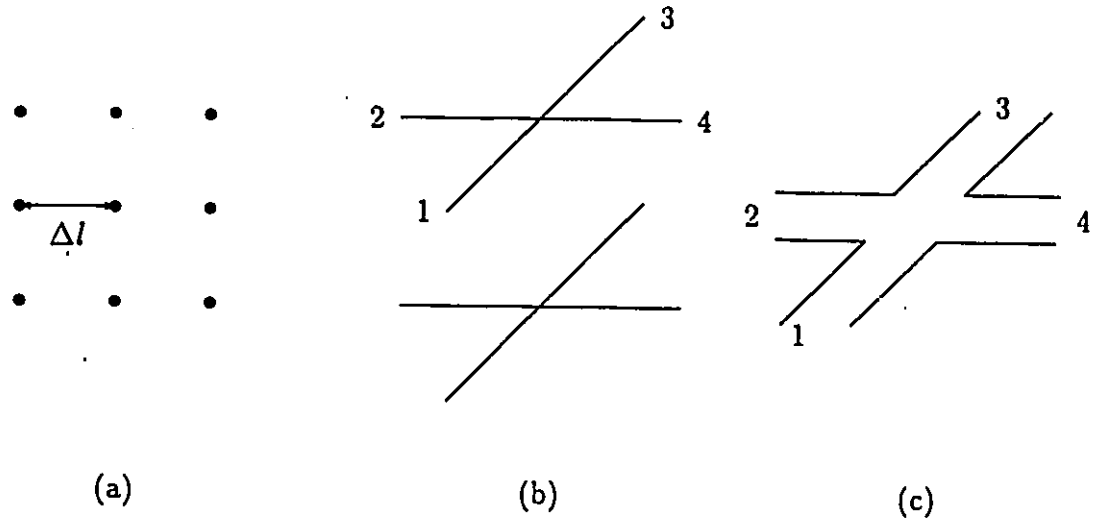


Figure 2.1: (a) A TLM mesh. (b)Shunt node. (c)Series node.

or the series node alone is sufficient. Therefore, only the shunt node has been used in this thesis when solving 2D problems.

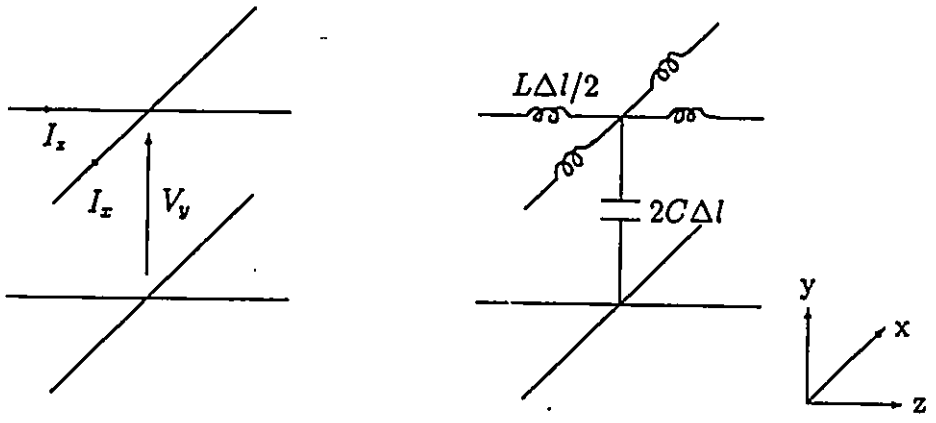
It has been proved in [12, 13] that a direct equivalence can be established between the voltages and currents on the transmission lines, and the electric and magnetic fields of Maxwell's equations. Fig. 2.2 shows a shunt node and its equivalent lumped element model. The equivalences can be given as[12, 13],

$$E_y \equiv V_y \quad H_z \equiv I_x \quad H_x \equiv -I_z \quad \mu \equiv L \quad \varepsilon \equiv 2C \tag{2.2}$$

Care must be taken when simulating wave velocity since the equivalence $\varepsilon \equiv 2C$, instead of $\varepsilon \equiv C$, exists. A TLM mesh constructed by transmission lines with $\mu_r = \varepsilon_r = 1$ does not simulate a medium with $\varepsilon_r = 1$, but $\varepsilon_r = 2$ [12].

2.2.2 Scattering of Dirac impulses at shunt nodes

Employing the transmission line theory, the scattering of a Dirac impulse at shunt nodes can be easily obtained. In [12] and [13], this scattering feature has been given in the form



(a) (b)

Figure 2.2: (a)Shunt node. (b)Equivalent lumped element model.

of a scattering matrix,

$$ {}_{k+1}\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix}^r = \frac{1}{2}\begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \cdot {}_{k}\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{pmatrix}^i \tag{2.3} $$

where $V_n (n = 1, 2, 3, 4)$ is the voltage on branch n, subscript k represents response at time step $k\Delta t$, r and i represent reflection and incidence, respectively.

Since one node is connected to its neighbouring nodes by a transmission line, any scattering impulse generated from it becomes an incident impulse at its neighbouring nodes. Thus,

$$ \begin{aligned} {}_{k+1}V_1^i(z,x) &= {}_{k+1}V_3^r(z,x-1) \\ {}_{k+1}V_2^i(z,x) &= {}_{k+1}V_4^r(z-1,x) \\ {}_{k+1}V_3^i(z,x) &= {}_{k+1}V_1^r(z,x+1) \\ {}_{k+1}V_4^i(z,x) &= {}_{k+1}V_2^r(z+1,x) \end{aligned} \tag{2.4} $$

where (z, x) is the space coordinate of the node.

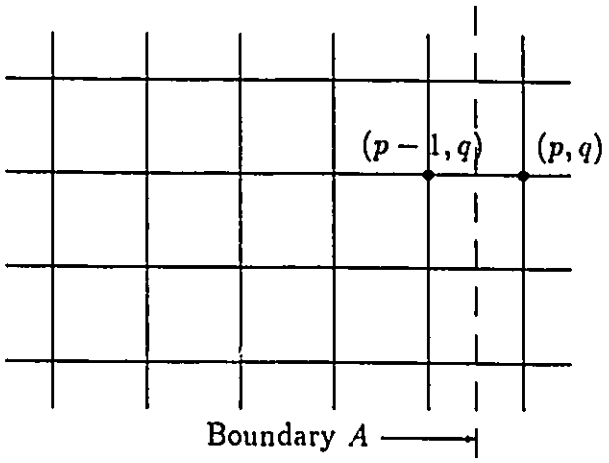


Figure 2.3: Placing a boundary in the TLM mesh.

2.2.3 Boundary conditions

Due to the limitation of computer memory, we need to confine the computation region by boundary conditions. We can easily implement these boundary conditions by employing the concept of reflection coefficient. For example, in Fig. 2.3, boundary "A" can be taken into account by the equation,

$$ {}_kV_4^i(p-1,q) = {}_kV_2^r(p,q) = \rho\ {}_kV_4^r(p-1,q) \tag{2.5} $$

where

$$ \rho = \frac{\frac{Z_c}{\sqrt{2}} - Z_0}{\frac{Z_c}{\sqrt{2}} + Z_0} \tag{2.6} $$

is the boundary reflection coefficient. $Z_0 = \sqrt{\mu_0/\epsilon_0}$ is the characteristic impedance of the mesh line. Z_c is the surface impedance of the boundary.

For open space, the concept of absorbing boundary condition (ABC) must be introduced. It will be fully described in Chapter 3.

2.2.4 Simulation of lossy material

To simulate lossy inhomogeneous materials, a permittivity stub and a loss stub connected to the node can be used [12]. A permittivity stub is an open-ended stub with a length of $\Delta l/2$ and normalized characteristic admittance of y_0 while a loss stub is a matched stub with normalized characteristic admittance of g_0. Following the same procedure of finding the scattering matrix for the normal node with four branches, the scattering matrix for nodes with permittivity and loss stubs can be easily obtained[12, 13],

$$ {}_{k+1}\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix}^r = \frac{1}{y}\begin{pmatrix} -(y-2) & 2 & 2 & 2 & 2y_0 \\ 2 & -(y-2) & 2 & 2 & 2y_0 \\ 2 & 2 & -(y-2) & 2 & 2y_0 \\ 2 & 2 & 2 & -(y-2) & 2y_0 \\ 2 & 2 & 2 & 2 & 2y_0 - y \end{pmatrix} \cdot {}_{k}\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix}^i \qquad (2.7) $$

where $y = 4 + y_0 + g_0$.

2.2.5 Computing the fields

After we have established the above knowledge, we can compute the time domain response of an excitation at any observation point and time. For example, if the voltage at a shunt node simulates the electric field, the field components can be given by,

$$ \begin{aligned} {}_kE_y &\equiv {}_kV_y &&= 2(\sum_{m=1}^{4} {}_kV_m^i + {}_kV_5^i y_0)/y \\ -{}_kH_x &\equiv {}_kI_z &&= ({}_kV_2^i - {}_kV_4^i)/Z_0 \\ {}_kH_z &\equiv {}_kI_x &&= ({}_kV_1^i - {}_kV_3^i)/Z_0 \end{aligned} \qquad (2.8) $$

If the excitation is a Dirac impulse, the frequency domain response over a wide frequency range can be obtained by performing a Fourier Transform of the time domain response.

2.3 The 3D-TLM Method

By combining the shunt node and the series node one can construct a 3D-TLM mesh. Fig. 2.4 shows a complete 3D-TLM "node" which is really a cell of $\Delta l/2$ size. We can see

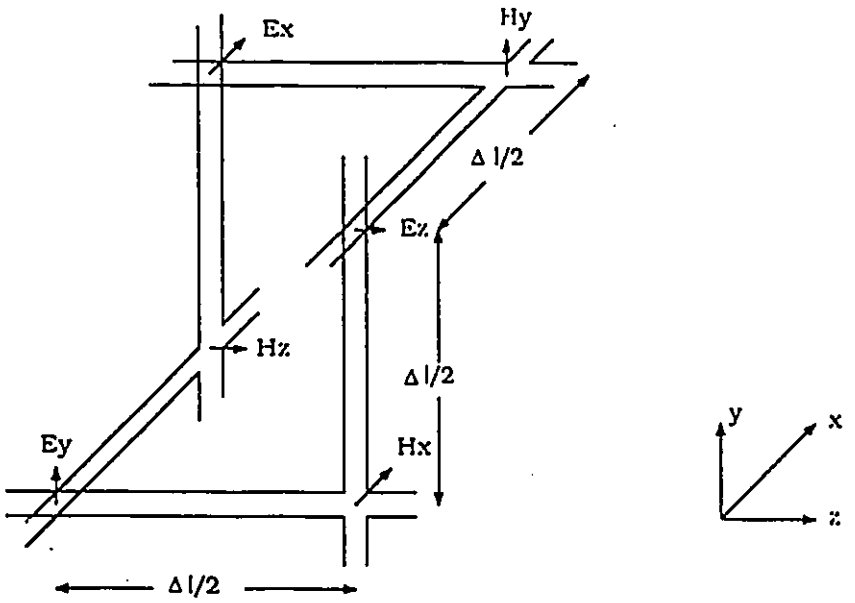


Figure 2.4: 3D-TLM distributed node.

the "node" is made up of three shunt and three series nodes connected together. The six field components are simulated by the currents and voltages at the six corners of this cube-like "node". For this reason, the node is called "distributed node".

Chapter 3

Development of Absorbing Boundary Algorithms

3.1 Introduction

As mentioned in Chapter 1, an Absorbing Boundary, henceforth called AB, is an artificial boundary to simulate infinite free space. Therefore, large computer memory and time can be saved if such a boundary is implemented properly. In this chapter, a brief description of the major existing AB Algorithms will be presented.

To evaluate an AB algorithm, some standards must be set. Small reflection is of great importance for a good AB. Otherwise, large errors in the frequency domain will occur since the Fourier Transform is very sensitive to errors in time domain results. A practical AB has normally a return loss smaller than $3\%(-30dB)$. Secondly, stability within the computation time is needed. Thirdly, a good AB should be able to handle waves within a wide range of incident angles. This is extremely important when analyzing non-TEM waves. Furthermore, an AB should not consume excessive computer time and memory. Finally, since the AB is used in the time domain, it should function well over a wide frequency range.

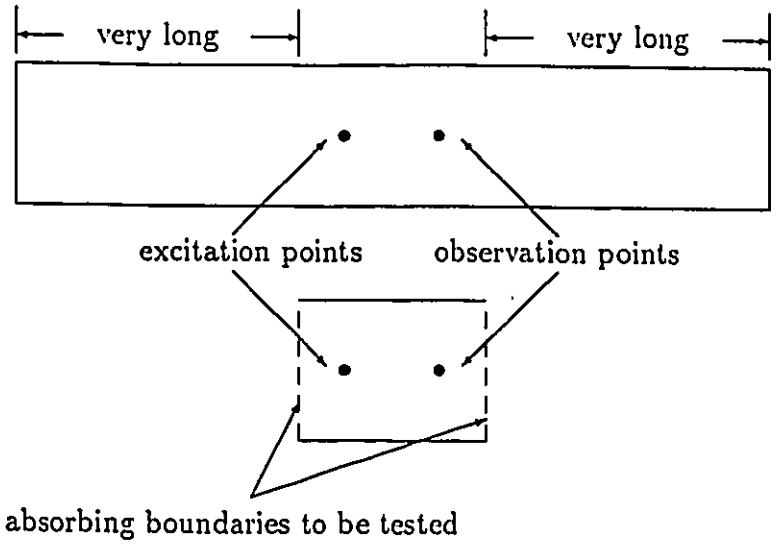


Figure 3.1: The Benchmark Approach is used normally to test other AB algorithms.

3.2 The Benchmark Approach

The idea of this approach is to place the boundaries so far away that the reflection from them will not reach the observation points within the required computation time. It is obvious that the result from it is correct and accurate for the given discretization. Since the boundaries are far away, the cost of computer time and memory is high. Thus, this is not a practical approach. Normally, it is only employed to generate benchmark results to test other AB algorithms.

3.3 The Discrete Green's Function or Johns Matrix Approach

For electromagnetic waves, the Green's Function is given by the solution of the following equation satisfying the required boundary conditions:

$$(\nabla^2 - \mu\varepsilon\frac{\partial^2}{\partial t^2})G(r,t;r',t') = \delta(r-r')\delta(t-t') \tag{3.1}$$

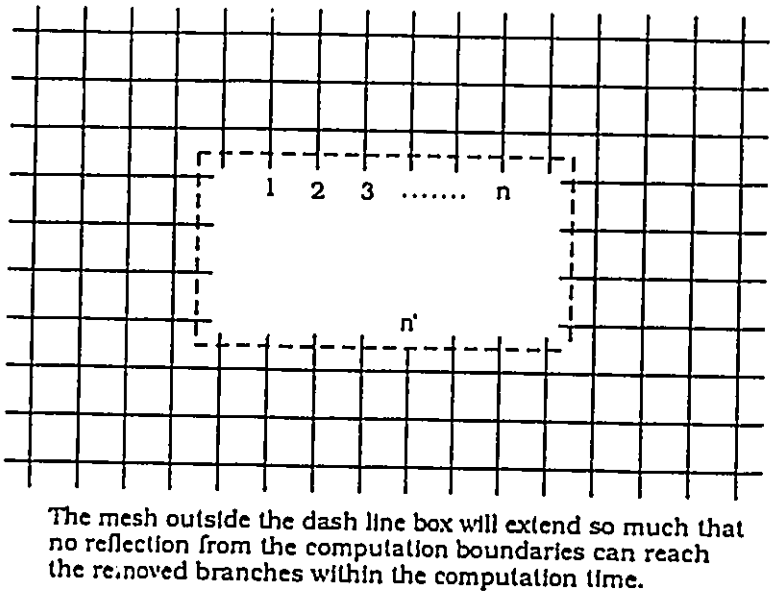


Figure 3.2: Illustration of numerical results of Green's Function.

Its physical meaning is the field response at point r at time t to a Dirac impulse at point r' at time t'. A closed-form analytical solution is normally hard to achieve if the boundaries are irregular, or the material is inhomogeneous. However, it is fairly easy to obtain a numerical result.

Suppose we are going to study the region inside the dash line box in Fig. 3.2. First of all, we find the Green's function of the outer region. We terminate the branches across the dash line box(they are sometimes called "removed branches") by their characteristic impedance, such that $\Gamma_i = 0$, where Γ_i is the instantaneous impulse reflection coefficient. We inject a Dirac impulse at branch n' at time t'. At time step t, we will see the response at branch n. This is one term of the discrete time-domain Green's function. The total Green's function or John's matrix is obtained by exciting each of the removed branches and computing the response at all removed branches, including the excited one. This procedure is described in detail in [11].

In classical electromagnetic theory, if we have the Green's function $G(r,t;r',t')$ and the source function $S(r',t')$ for a given problem, the resulting field at point r at time t can be

obtained analytically by the integral

$$\Phi(r,t) = \int_{V'} \int_{t'} G(r,t;r',t') S(r',t') dV' dt' \tag{3.2}$$

where V' is source volumn. The numerical form of the above integration can be given as,

$$\Phi(n,t) = \sum_{n'=1}^{N} \sum_{t'=1}^{t} G(n,t;n',t') S(n',t') \tag{3.3}$$

where N is the number of removed branches.

We can see from the above that, after the Green's function of a given outer region has been obtained, this region can be easily connected to the inner region by a numerical convolution. Even if the inner region has undergone some changes, we can still use the same Green's function for the outer region. Under certain condition this can save much computer time and memory [7].

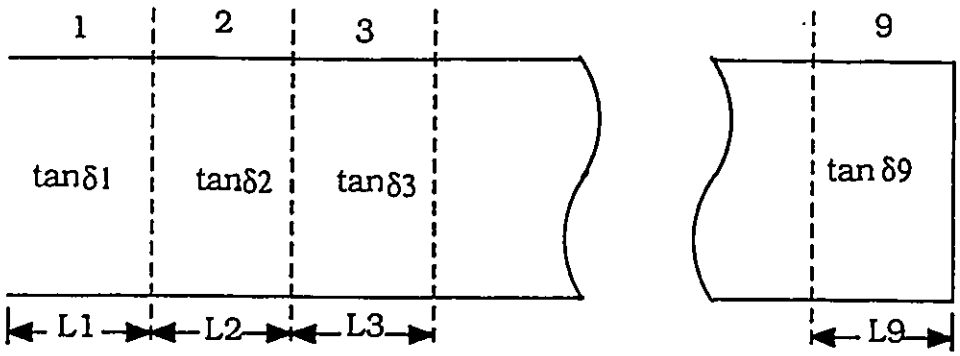
This idea can be used to generate absorbing boundaries. It yields results identical to the benchmark approach. It is also stable and can handle waves within a wide range of incident angles. The shortcoming of it is that large memory and long time are needed to calculate the Green's function and to perform the convolution if the number of removed branches is very large.

3.4 Dissipation Approach

When a wave is travelling in lossy media, part of its energy will be dissipated. Thus, it is possible to simulate the free space by making the region surrounding the computational domain progressively lossy so that it dissipates energy rather than carries it into infinity.

The implementation of this approach in TLM has been described in [7]. In that paper, a number of uniform lossy sections of waveguide have been cascaded to simulate an infinitely long waveguide. When applied to a WR28 waveguide, a return loss of less than $-32dB$ over the operating band of the WR28 waveguide has been achieved.

Using this approach, the energy of the incident waves is absorbed gradually in the lossy



(tanδ is the dielectric loss tangent of the section.)

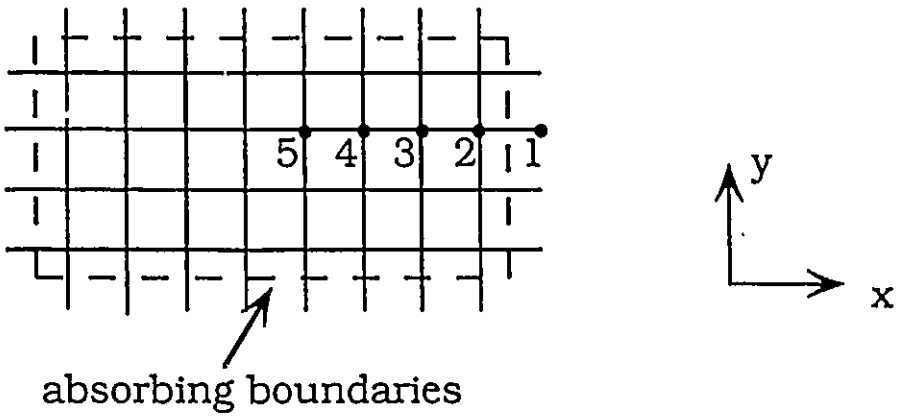
Figure 3.3: Modeling of a wide-band absorbing waveguide termination by a cascade of several lossy sections

sections. No nonphysical energy is introduced. Thus, this approach is unconditionally stable.

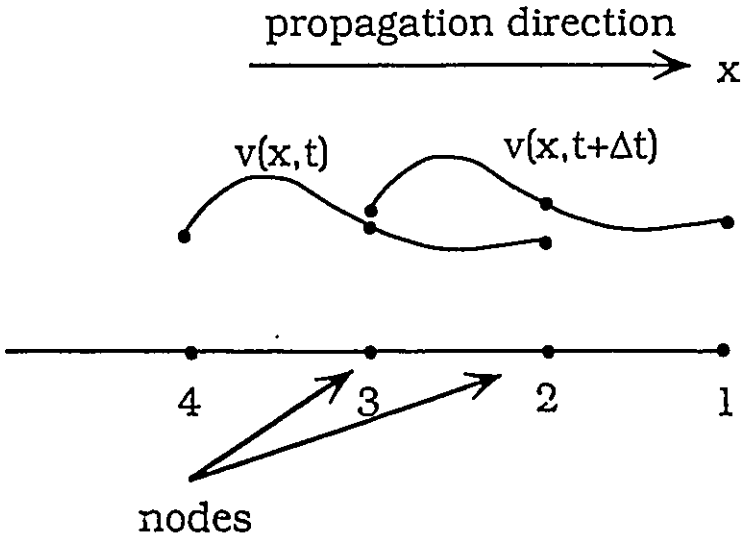
However, it is also mentioned in [7] that to obtain a good absorption, the lossy section must be several wavelengths' long. In the paper, $3.785\lambda_g$ has been chosen to get a reflection smaller than $-32dB$ over the whole operating band. That means an extra section of $180\Delta l$ has been used. Thus this approach is quite costly in computer time and memory.

3.5 Extrapolation Approach

Fig. 3.4 shows the basic idea of this approach. In Fig. 3.4 (a), suppose at time t, voltage magnitudes at nodes inside the absorbing boundary (node $2, 3, 4, 5$) are known. At time $t + \Delta t$, it is obvious that if we want to obtain fields inside the absorbing boundary, magnitudes at nodes immediately outside the absorbing boundary(node 1) at time t must be known. To get the exact values of these magnitudes, the computational region outside the absorbing boundary must be large enough that no reflection can reach the absorbing



(a)



(b)

Figure 3.4: Schematic diagram of the extrapolation approach, (a)nodes on two sides of the absorbing boundary, (b) magnitude at node 1 at a certain time is predicted by magnitudes at node 2, 3, 4, ... at a previous time.

boundary within the computation time. This eliminates the advantage of an absorbing boundary. An alternative is the extrapolation approach which can obtain these magnitudes approximately rather than exactly. In Fig. 3.4 (b), a wave is traveling along x direction. Two curves representing magnitudes at nodes at time t and $t+\Delta t$ are shown. It is easy to see that the magnitude at node 1 at time $t+\Delta t$ is approximately equal to the magnitude at node 2 at time t. This means that when a wave travels, the magnitude at a point at a certain time can be predicted from magnitudes at nodes prior to it along the propagation direction at a previous time. This idea forms the basis of the extrapolation approach. Different ways to predict the magnitudes outside the absorbing boundary give rise to different kinds of extrapolation approaches. This approach has the advantage of saving computer time and memory since the boundaries can be placed closely to the excitation points and no excessive computation region outside the boundary is needed. Generally speaking, the extrapolation approach has good absorption and good ability to handle waves within a wide range of incident angles. However, it has the shortcoming of amplifying numerical errors and thus is potentially unstable.

3.5.1 Higdon's Absorbing Boundary Algorithm

In [8, 9], Higdon proposed an ABC for 2D waves,

$$[\prod_{j=1}^{p}((\cos\alpha_j)\frac{\partial}{\partial t}-c\frac{\partial}{\partial x})]U=0 \tag{3.4}$$

where α_j is the incident angle, c is the velocity of the wave.

This ABC has the following properties:

1. No reflection at all occurs if waves travel out of the absorbing boundary at incident angles of $\alpha_j(j=1,\cdots,p)$ with speed c. For a wave incident at an angle θ, a general reflection coefficient can be given as,

$$-\prod_{j=1}^{p}\frac{\cos\alpha_j-\cos\theta}{\cos\alpha_j+\cos\theta} \tag{3.5}$$

2. α_j can be chosen by users according to a given problem. This can optimize the absorption of this ABC in some particular problems.

3. This ABC is claimed [9] to provide a general representation of many extrapolation ABCs, such as the ABC proposed by Engquist and Majda [5, 6], Lindman[20] ABC, etc..

However, this approach is complicated and is not very convenient for engineering purposes.

3.5.2 Radiation Boundary Operator approach

Two Radiation Boundary Operators have been discussed by Moore et al [23]. One is the Mode-Annihilating Operator, the other is One-Way Wave Equation Operator. In both cases, radiation operators are generated directly from the wave equation, and then applied to create a radiation boundary condition.

Mode-Annihilating Operator

According to Moore [23], this type of operator is based on the idea of suppressing modes of the far-field expansion of outward propagating solutions to the wave equation.

In 3D, the time domain scalar wave equation is

$$\nabla^2 U(R,\theta,\phi,t) - \frac{\partial^2}{\partial t^2} U(R,\theta,\phi,t) = 0 \tag{3.6}$$

where R, θ, ϕ are the spherical coordinates. The radiating solutions (i.e., solutions propagating in directions which are outward from the origin of a spherical coordinate system) are given in [23] as,

$$U(R,\theta,\phi,t) = \sum_{i=1}^{\infty} \frac{f_i(t-R,\theta,\phi)}{R^i} \tag{3.7}$$

where f_i is a function determined by the excitation source arrangement. This expansion satisfies the Sommerfeld radiation condition which is derived in [27] as follows:

$$\lim_{R\to\infty} R(\frac{\partial}{\partial R} + \frac{\partial}{\partial t})U = 0 \tag{3.8}$$

Thus,

$$B_1 U \equiv (\frac{\partial}{\partial R} + \frac{\partial}{\partial t})U = O(R^{-2}) \tag{3.9}$$

where B_1 is the first order Mode-Annihilating Operator, which suppresses the first mode in Eqn. 3.7.

Higher order operators have also been derived in [23]. A simple recursion relationship has been given,

$$B_n = (L + \frac{2n-1}{R})B_{n-1} \tag{3.10}$$

where,

$$L \equiv \frac{\partial}{\partial R} + \frac{\partial}{\partial t} \tag{3.11}$$

B_n suppresses the nth order mode in Eqn. 3.7 and leaves an error as follows,

$$B_n U = O(R^{-2n-1}) \tag{3.12}$$

Omitting this error gives the Mode-Annihilating ABC,

$$B_n U = 0 \tag{3.13}$$

One-Way Wave Equation operator

By factoring the wave equation, we can get an equation which permits wave propagation only in one direction. This is the idea of the One-Way Wave Equation approach [23].

In 2D, the wave equation can be written as,

$$LU \equiv (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} - \frac{\partial^2}{\partial t^2})U = 0 \tag{3.14}$$

If L can be factored in the following form,

$$LU = L^+ L^- U = 0, \tag{3.15}$$

either L^+ or L^- can be the One-Way Wave Equation operator, which governs waves traveling in $+x$ or $-x$ direction, respectively. We define,

$$L^\pm \equiv \frac{\partial}{\partial x} \pm \frac{\partial}{\partial t}\sqrt{1 - S^2} \tag{3.16}$$

where,

$$S \equiv \frac{\partial}{\partial y} / \frac{\partial}{\partial t} \tag{3.17}$$

Using a two-term Taylor series approximation to the radical in the above equation yields,

$$L^{\pm} \approx \frac{\partial}{\partial x} \pm \frac{\partial}{\partial t}(1 - S^2/2) \tag{3.18}$$

Thus the One-Way Wave Equation, or the absorbing boundary condition based on it can be given as

$$L^{\pm}U = (\frac{\partial^2}{\partial x \partial t} \pm \frac{\partial^2}{\partial t^2} \mp \frac{1}{2}\frac{\partial^2}{\partial y^2})U = 0 \tag{3.19}$$

Besides the Taylor series, other kinds of approximation have also been discussed in [23].

3.5.3 Liao's Absorbing Boundary Algorithm

This approach has been implemented in 2D and 3D TLM in this thesis. We will fully discuss it in Chapter 5.

Chapter 4

New Developments of Absorbing Boundary Algorithms in 2D-TLM 1: Direct Optimized Dissipation Absorbing Boundary Condition approach

4.1 Introduction

Although it consumes more computer time and memory than many other ABCs such as the Extrapolation ABC, the Dissipation ABC is still being used due to its absolute stability. A Dissipation ABC is a properly arranged lossy area around the computation region (Fig. 4.1) such that the outgoing waves can be gradually dissipated in it. Different ways of finding the arrangement of the lossy region give rise to different Dissipation approaches such as the progressively lossy region approach, the indirect optimization approach[7], etc. In those approaches, indirect simulation or even assumptions were made, e.g., a waveguide was simulated by a transmission line and the optimization was performed on that transmission line [7]. This led to the result that excessive lossy region was involved. Furthermore, the applications of those approaches are limited. They can only be applied on regular structures such as regular waveguide, etc.. In this thesis, a direct optimization approach was introduced.

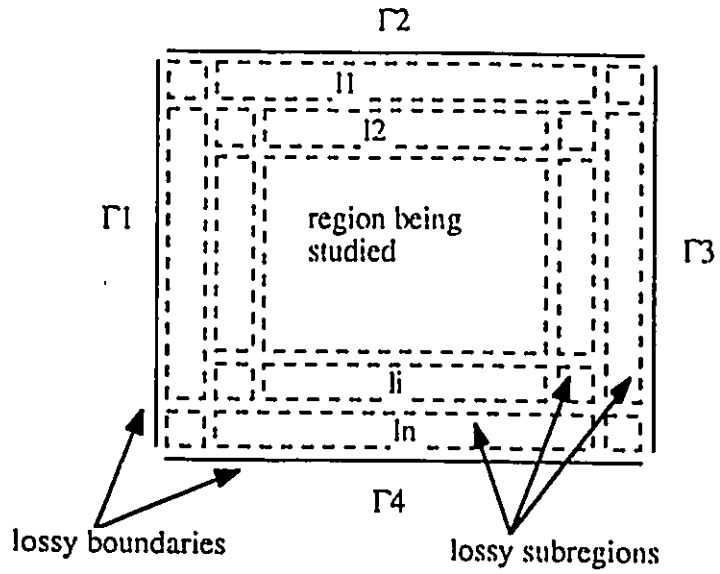


Figure 4.1: An example of the Dissipation Absorbing Boundary Condition

4.2 Formalism of the Direct Optimized Dissipation Absorbing Boundary Condition approach

The block diagram in Fig. 4.2 can help us to understand the idea of this approach. First, the Benchmark results are calculated. Then the lossy area is divided into n subregions(Fig. 4.1) and an initial loss tangent l_i is assigned to subregion $i(i = 1,...,n)$. These initial loss tangents can be any randomly chosen values. The outside of the lossy region is bounded by a set of lossy boundaries with initial reflection coefficients of $\Gamma_k(k = 1,...,m)$. Now we have constructed a Dissipation ABC. Since Γ_k and l_i are randomly chosen, this Dissipation ABC may not be a good one with a return loss smaller than required. Thus, optimization must be done to obtain the optimized Γ_k and l_i. We let Γ_k and l_i to be the optimization variables and calculate the response of the Dissipation ABC with the initial Γ_k and l_i. This response will be compared with the Benchmark results. If the return loss of the Dissipation ABC can not satisfy the required specification, a new set of Γ_k and l_i will be generated by the optimization subroutine for the next round of optimization. The above process iterates until the return loss of the Dissipation ABC either reaches its minimum or the required return loss. In the former case, the lossy region needs to be enlarged, and then

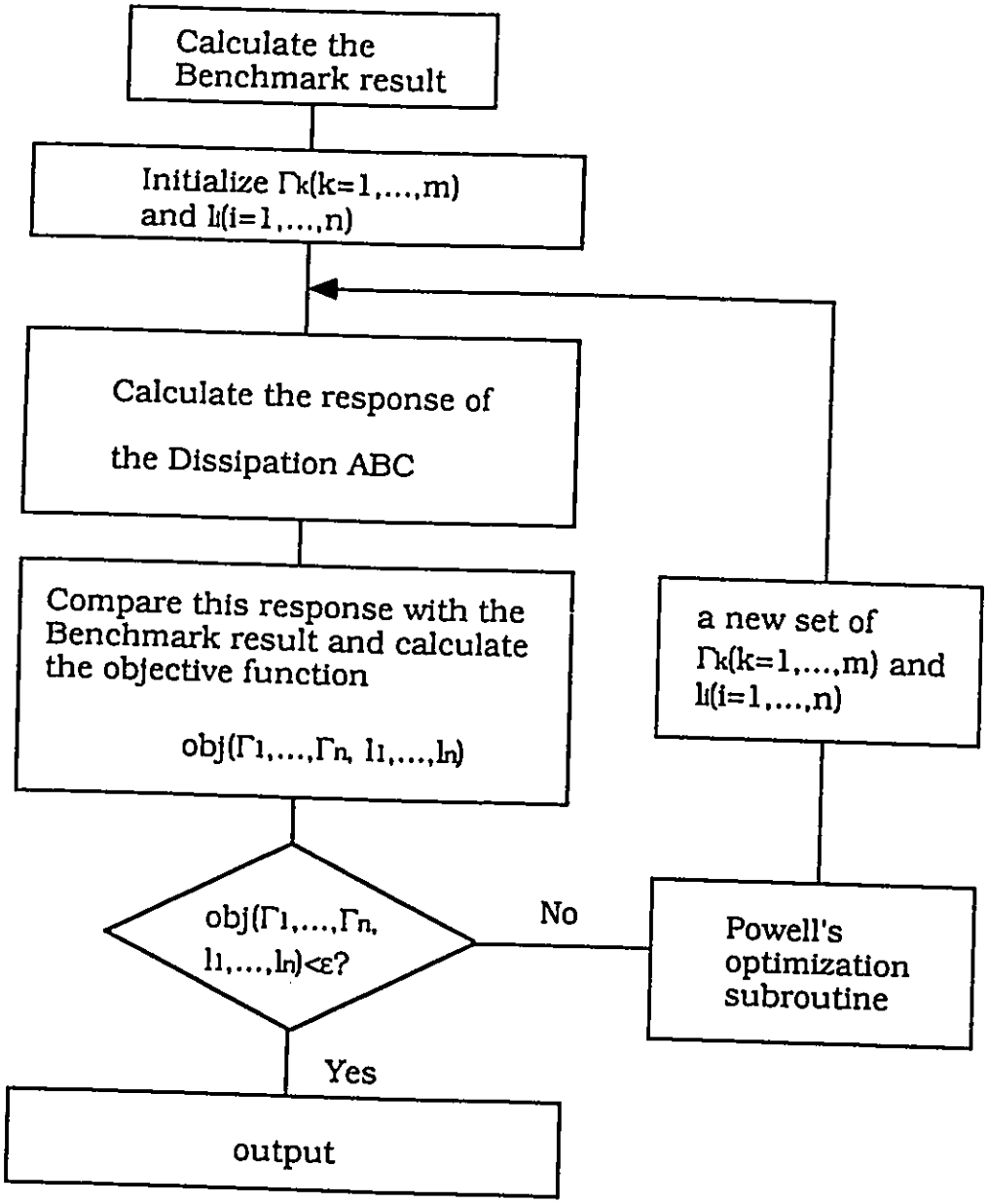


Figure 4.2: Block diagram of Direct Optimization approach

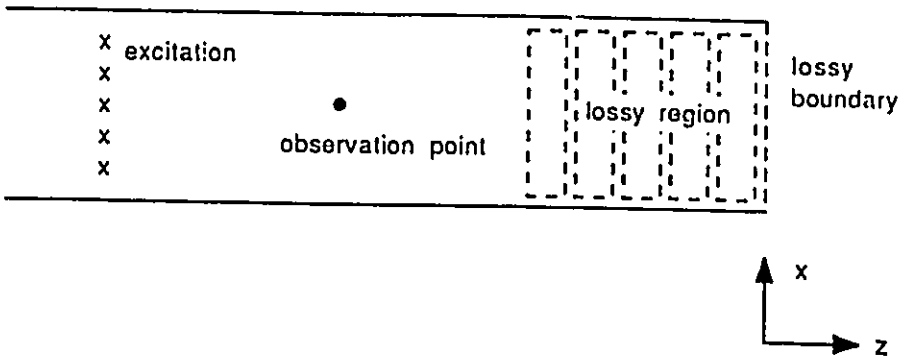


Figure 4.3: Direct Optimization approach applied to a WR28 waveguide structure.

the optimization be performed again.

The advantage of this approach is that the optimization is performed directly on the structure being studied. It gives more accurate result while keeping the lossy region as small as possible.

The optimization method used in this thesis is Powell's method. We applied this approach on a WR28 waveguide structure shown in Fig. 4.3. The discretization in x direction is 20 grid points. The additional lossy area consists of a set of stripes with a width of Δl, which is the distance between two neighboring nodes.

The loss tangents l_i of these stripes and the reflection coefficients Γ_k of the lossy boundaries are the variables of optimization. The objective function can be given as follows,

$$obj(\Gamma_1, \Gamma_2, ..., \Gamma_m, l_1, l_2, ..., l_n) = \sum_{i=1}^{K} W_i(r(f_i) - S) \tag{4.1}$$

where $r(f_i)$ is the return loss in dB at frequency f_i, K is the number of the sampling frequency points in the operating band of WR28 waveguide, W_i is the weighting factor, S is the required specification of the return loss. When the optimization program is run,

Γ_1	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
-1.19e-1	1.67e-1	6.96e-1	1.06e-1	5.48e-1	2.15e-2	2.20e-2	1.37e-3	1.21e-2	4.17e-2	1e-3

l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}
3.4e-2	1.39e-3	1.15e-2	1.52e-1	1.46e-2	1.21e-2	1.02e-3	3.39e-2	4.49e-2	9.42e-3

l_{21}	l_{22}	l_{23}	l_{24}	l_{25}	l_{26}	l_{27}	l_{28}	l_{29}	l_{30}
5.6e-2	8.95e-2	3.29e-2	1.35e-1	2.51e-2	3.02e-6	5.19e-7	2.15e-2	9.89e-3	3.05e-2

l_{31}	l_{32}	l_{33}	l_{34}	l_{35}	l_{36}	l_{37}	l_{38}	l_{39}	l_{40}
9.62e-7	4.46e-6	9.72e-3	3.19e-2	5.65e-3	3.60e-2	6.55e-2	2.99e-6	2.58e-2	8.01e-3

Table 4.1: Optimized values of Γ and l_i.

it will keep searching for the set of Γ_k and l_i which make the objective function above as small as possible.

4.3 Numerical results

For the above WR28 waveguide structure, we choose a lossy stub consisting of 40 lossy stripes. Only the dominant mode(TE_{10}) has been excited. The optimized Γ_k and l_i are given in Tab. 4.1 and the return loss is given in Fig. 4.4. The time domain response of this ABC to a Gaussian pulse has also been obtained and is shown in Fig. 4.5. We can see that the absorption is good. The lossy region here is much shorter than the one discribed in [7] in which Touchstone software has been used to do the optimization indirectly.

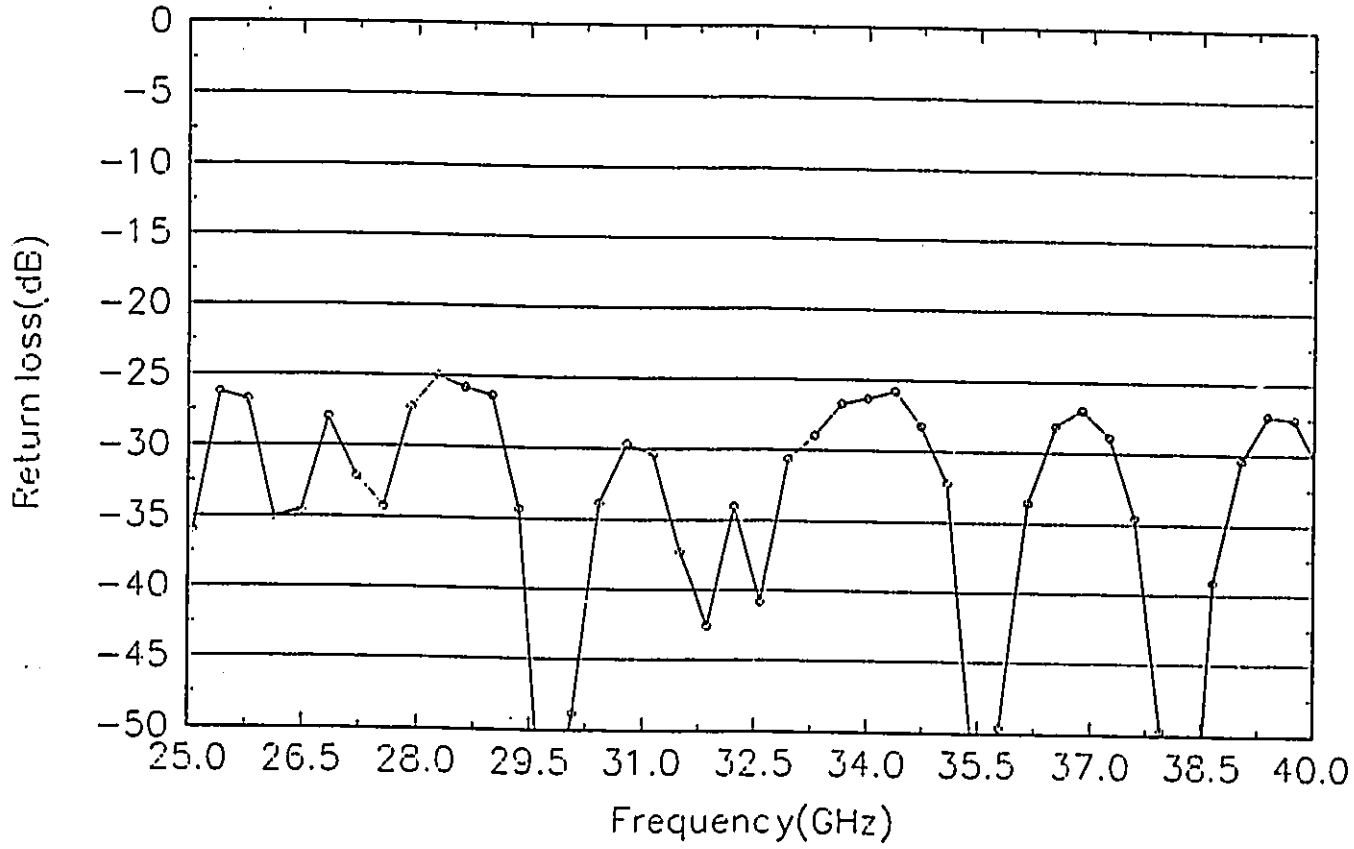


Figure 4.4: Return loss of the Optimized Dissipation Absorbing Boundary Condition on WR28 waveguide structure.

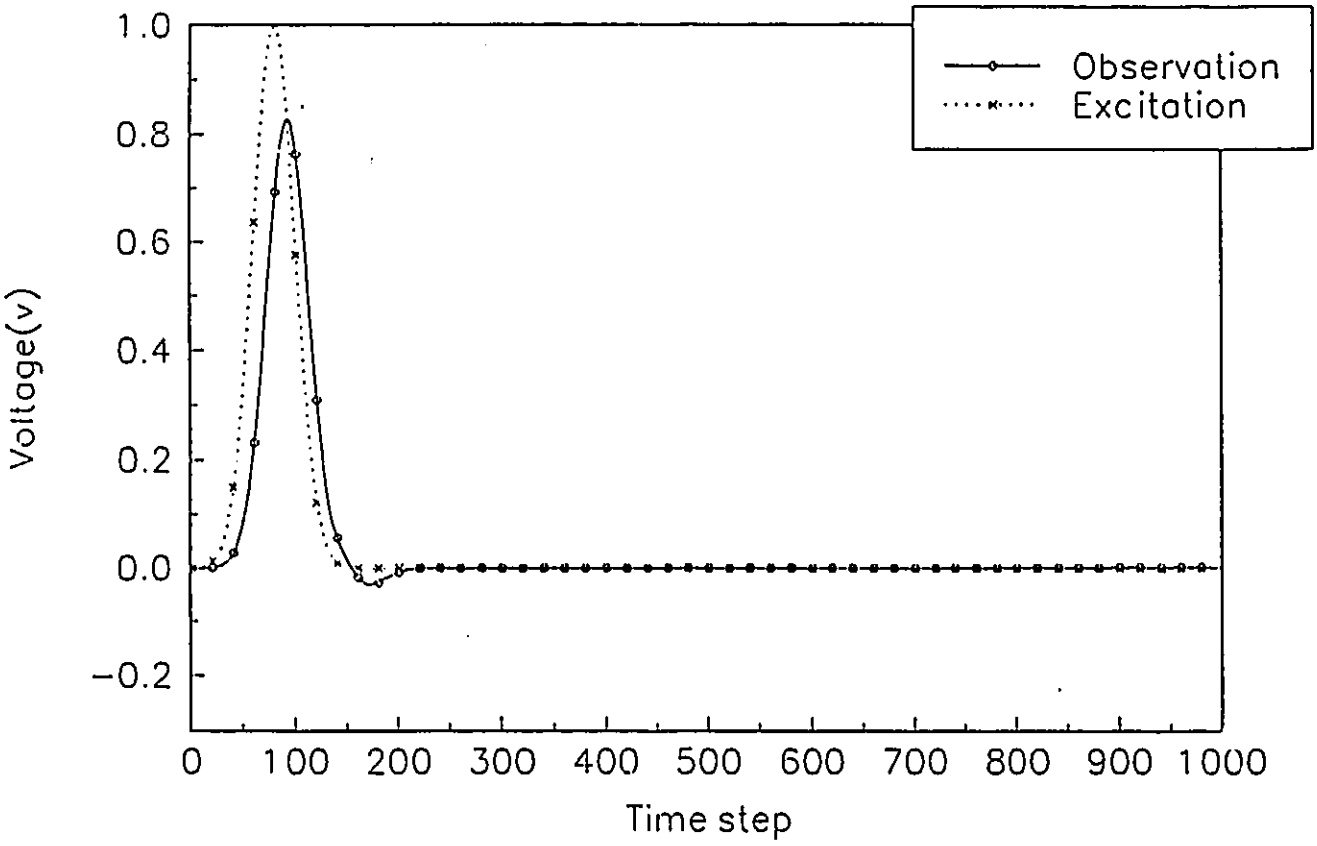


Figure 4.5: Time domain response of the Optimized Dissipation Absorbing Boundary Condition to a Gaussian pulse.

Chapter 5

New Developments of Absorbing Boundary Algorithms in 2D-TLM 2: Liao's Absorbing Boundary Condition

5.1 Introduction

Liao's absorbing boundary is an extrapolation method, which was first proposed by Z. P. Liao et al [17] and implemented in the Finite Element Method (FEM) applied to seismology research. Later, M. Moghaddam et al [21] applied it to the Finite Difference-Time Domain (FD-TD) method. In this thesis, its implementation in TLM will be described. It will be applied to a WR28 empty waveguide and a WR28 waveguide loaded with a dielectric slab in the middle of its wide side. Reflection of less than $3\%(-30dB)$ over the whole operating band has been achieved.

This ABC is unstable. Ways of stabilizing it will be discussed.

5.2 Development of the transmitting formula

Normally, waves incident upon absorbing boundaries are not plane waves. But they can be considered as the summation of a number of plane waves. Hence, we can first study the travelling process of a plane wave and then apply it to general waves.

In Fig. 5.1, a plane wave of velocity c and incident angle θ propagates through the

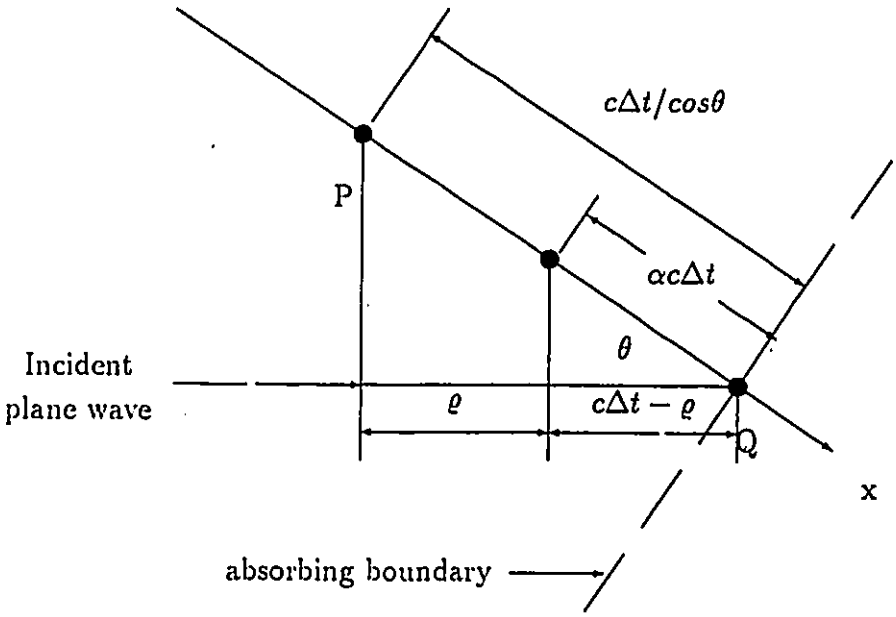


Figure 5.1: The schematic diagram of a transmitting boundary.

absorbing boundary at point Q. PQ is a local coordinate x-axis normal to the boundary at Q.

Following Liao's notation[17], the apparent propagation of the incident wave along the x-axis can be expressed as

$$u(t,x) = u^*(ct - x\cos\theta), \tag{5.1}$$

where u and u^* are the waves along the x-axis and the actual incident wave respectively. It is easy to see that the wave along the x-axis is a plane wave with a phase velocity of $c/\cos\theta$, its propagating process may be expressed in discretized form by,

$$u(t+\Delta t, x) = u(t, x - c\Delta t/\cos\theta), \tag{5.2}$$

which means that the displacement of point Q at time $t+\Delta t$ can be obtained by replacing it with that of point P at time t. This formula can be used as an ABC provided that θ is known. Otherwise, the Multi-Transmitting Theory, which will be explained below, must be used.

From Eq. (5.1), we have,

$$u(t+\Delta t, x) = u^*(ct + c\Delta t - x\cos\theta) \equiv u^*(\xi + \varrho), \tag{5.3}$$

where,

$$\xi = ct - (x - \alpha c\Delta t)\cos\theta, \tag{5.4}$$

$$\varrho = c\Delta t(1 - \alpha\cos\theta), \tag{5.5}$$

α being called the transmitting coefficient. If we choose Δt to be very small and α to be a constant within a certain range which will be discussed later, ϱ can be very small compared to ξ, and thus we can define the mth-order error wave as [17]:

$$\Delta^m u^*(\xi + \varrho) = \Delta^{m-1} u^*(\xi + \varrho) - \Delta^{m-1} u^*(\xi). \tag{5.6}$$

From the above recursive formula, we can easily get,

$$u^*(\xi + \varrho) = \sum_{m=1}^{N} \Delta^{m-1} u^*(\xi) + \Delta^N u^*(\xi + \varrho). \tag{5.7}$$

Using the notation in Eq. (5.1), we have,

$$\Delta^m u(t, x - \alpha c\Delta t) = \Delta^m u^*(\xi). \tag{5.8}$$

From the recursive formula Eq. (5.6), we know that the higher the order, the smaller the error wave. Thus we neglect the last term in the right hand side of Eq. (5.7), and have,

$$u(t+\Delta t, x) \approx \sum_{m=1}^{N} \Delta^{m-1} u(t, x - \alpha c\Delta t). \tag{5.9}$$

The terms of the right hand side of the above expression have been given in Liao's other papers[18, 19],

$$\Delta^m u(t, x - \alpha c\Delta t) = \sum_{j=1}^{m+1} (-1)^{j+1} C_{j-1}^m u(t - (j-1)\Delta t, x - j\alpha c\Delta t), \tag{5.10}$$

where C_j^m is the binomial coefficient. Thus, the transmitting formula for a plane wave can be given as:

$$u(t+\Delta t, x) \approx \sum_{m=1}^{N} \sum_{j=1}^{m+1} (-1)^{j+1} C_{j-1}^m u(t - (j-1)\Delta t, x - j\alpha c\Delta t), \tag{5.11}$$

Making use of the formula,

$$C_{n+1}^{n+k+1} = \sum_{j=0}^{k} C_n^{n+j} \tag{5.12}$$

we have,

$$u(t+\Delta t, x) \approx \sum_{j=1}^{N} (-1)^{j+1} C_j^N u(t-(j-1)\Delta t, x - j\alpha c\Delta t). \tag{5.13}$$

We can see that this formula is independent of the incident angle. Below, we can also see that, if α is varied within a certain range, Eq. (5.13) can also govern the propagation of general non-TEM waves out of the absorbing boundary, with any desired accuracy[17].

Let

$$0 \leq \alpha \leq 2. \tag{5.14}$$

For waves incident onto the absorbing boundary,

$$|\theta| \leq 90^{\circ}. \tag{5.15}$$

Thus, from Eq. (5.5), we have,

$$|\varrho|_{max} = c\Delta t. \tag{5.16}$$

Therefore, the Nth-order error wave is

$$\Delta^N u^*(\xi+\varrho) = \frac{d^N}{d\xi^N}(u^*(\xi+(N-1)\varrho))\varrho^N + O(\varrho^{N+1}) = O(\varrho^N). \tag{5.17}$$

This expression reveals that the error is not greater than $O((c\Delta t)^N)$ for waves travelling out of the absorbing boundary at any incident angle. As we mentioned above, all waves can be considered as the summations of many plane waves,

$$u(t,x) = \sum_i u_i(t,x). \tag{5.18}$$

For every $u_i(t,x)$, we choose α within the range $0 \leq \alpha \leq 2$. From Eq. (5.13), we have,

$$u_i(t+\Delta t, x) \approx \sum_{j=1}^{N} (-1)^{j+1} C_j^N u_i(t-(j-1)\Delta t, x - j\alpha c\Delta t). \tag{5.19}$$

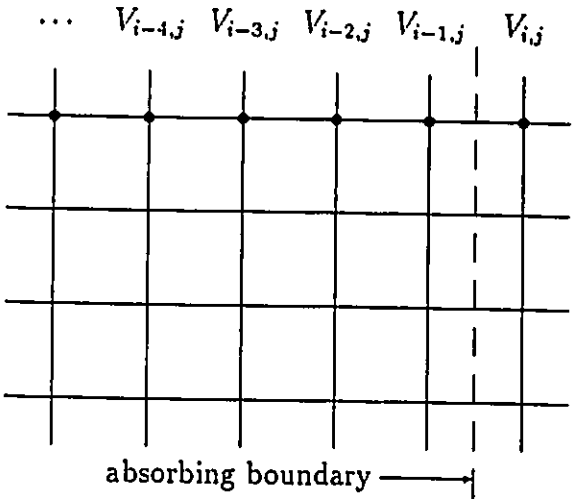


Figure 5.2: The position of an absorbing boundary in a TLM mesh

Substituting it in Eq. (5.18) gives,

$$\begin{aligned} u(t+\Delta t,x) &\approx \sum_i \sum_{j=1}^{N} (-1)^{j+1} C_j^N u_i(t-(j-1)\Delta t, x-jac\Delta t) \\ &= \sum_{j=1}^{N} (-1)^{j+1} C_j^N u(t-(j-1)\Delta t, x-jac\Delta t). \end{aligned} \tag{5.20}$$

This is the Multi-Transmitting formula valid for general waves with an error of the order of $O((c\Delta t)^N)$.

5.3 Implementation in 2D-TLM

In TLM, the absorbing boundary is placed halfway between two nodes (Fig. 5.2) so that the synchronism of impulses reaching each node can be maintained. $V_{i,j}^k$ is assigned to denote the voltage at node (i,j) and at time step $k\Delta t$.

5.3.1 Two ways of introducing the Multi-Transmitting formula into 2D-TLM

The mechanism of the TLM method gives the freedom of choosing how to predict values outside the AB. In Fig. 5.3, the left hand side of the AB is the computation region. Suppose we use values at three nodes to predict values at the unknown point outside the

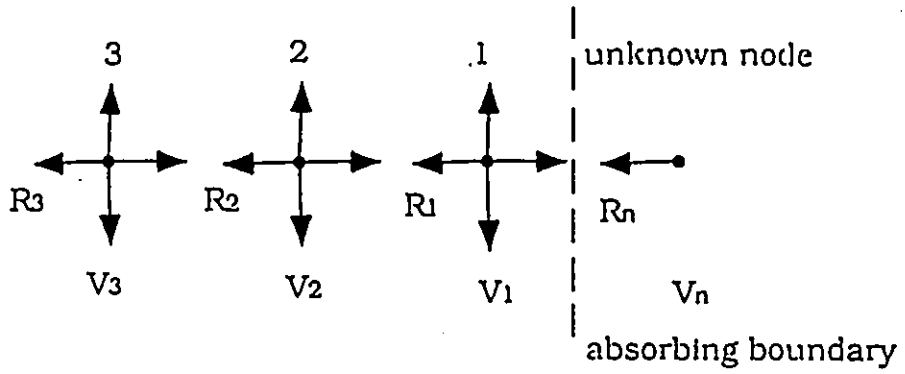


Figure 5.3: Two ways of implementing Liao's Absorbing Boundary Condition in 2D-TLM.

AB. At time step k, we have the node voltages at node 1,2 and 3 which are V_1^k, V_2^k and V_3^k. We also have the reflection voltages at these nodes, which are R_1^k, R_2^k and R_3^k. Using the Multi-Transmitting formula directly, we predict V_n^k which is the node voltage at the unknown node at time k. However, it is easy to see that, to implement ABC in TLM, only R_n^k, which is the reflection voltage of the unknown node at time k, is needed. Thus, the alternative of applying Multi-Transmitting formula to node voltage is applying it to the node reflection voltage.

We have tested these two approaches and no great difference has been found except that the latter approach(predicting node reflection voltages) is slightly more stable than the former when the order of Liao's ABC is high (over 7's order).

5.3.2 Numerical results

We applied this ABC to a WR28 waveguide(Fig. 5.5) and a WR28 waveguide loaded with a centered dielectric slab(Fig. 5.6). In both cases, the layout of the TLM mesh between the absorbing boundary and the excitation points was 30×60 grid points. Only the dominant mode (TE_{10}) was excited . 1000 iterations were chosen. The time domain response of Liao's ABC in the structure shown in Fig. 5.5 to a Gaussian pulse is given in Fig. 5.7.

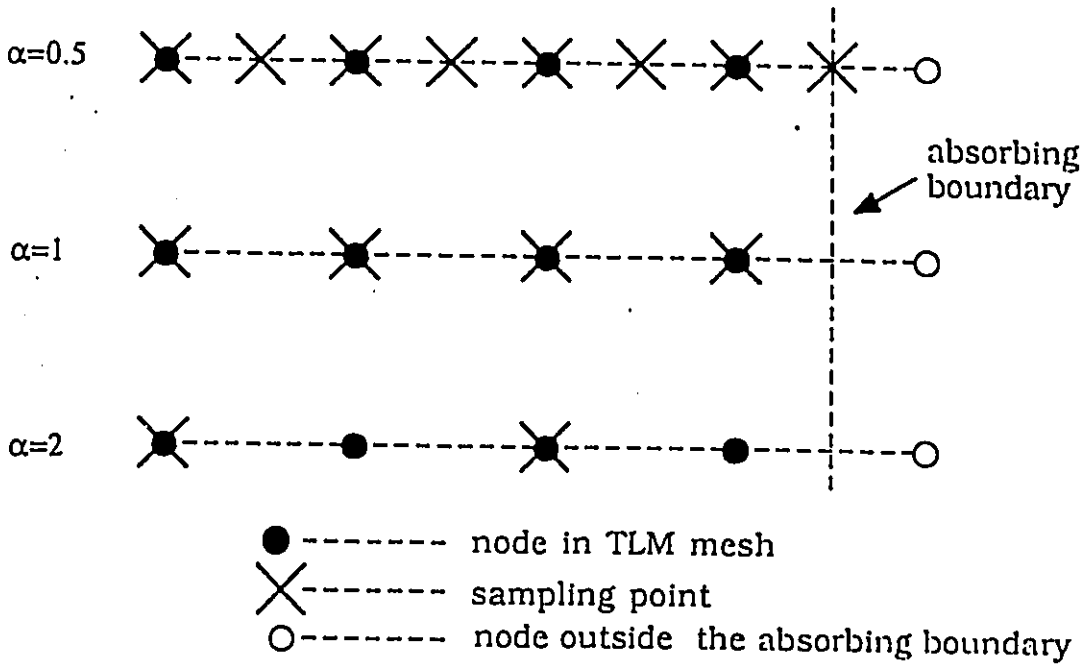


Figure 5.4: Positioning of sampling points of Liao's Absorbing Boundary Condition, (a) $\alpha = .5$, (b) $\alpha = 1$, (c) $\alpha = 2$.

Frequency domain results are shown in Fig. 5.8, Fig. 5.9 and Fig. 5.10. We can see that the return loss of Liao's ABC over the whole operating band of a WR28 waveguide is smaller than $-30dB$. If the order of Liao's ABC is properly chosen, the return loss over the operating band can be as low as $-40dB$. This will be shown later.

5.3.3 The effect of the transmitting coefficient α

As we mentioned above, α can be chosen between 0 and 2. This affects the position of sampling points. Fig. 5.4 shows the sampling point arrangement when $\alpha = 0.5, 1$ and 2. We can see that, when $\alpha \neq 1$, sampling points may be situated somewhere between two neighboring nodes. In such a case, the voltage magnitude $f(x)$ at the sampling point x can be obtained by the following quadratic interpolation formula:

$$f(x) \approx \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}V_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}V_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}V_3 \qquad (5.21)$$

where V_1, V_2, V_3 are the voltage magnitudes on node $1, 2, 3$, respectively.

If we choose $\alpha > 1$, a longer sampling stub is needed for the same order of transmitting formula, and thus more computer storage and time will be involved. If we choose $\alpha < 1$, computer memory and time will be saved, but at the cost of reducing the accuracy due to the interpolation. Thus, normally $\alpha = 1$ is chosen in order to obtain the accuracy required while excessive computer memory and time can be avoided.

5.3.4 The effect of the order of the transmitting formula

We mentioned above that, the higher the order N in the transmitting formula (Eq. 5.20), the better the absorption. However, since the formula is applied numerically in the TLM method, the numerical error can not be ignored when N is large, and thus, instability problems arise. It had been proved by Dalquest [4] in 1963 that, no numerical differential equation solver of order higher than two is absolutely stable.

Numerical experiments have been performed on the waveguide structure in Fig. 5.5 with $N = 4, 8, 9, 12$. Results have been given in Fig. 5.8 and Fig. 5.9. We can see that when $N < 9$, the return loss keeps decreasing as N increases. When $N > 9$, the return loss starts to go up again.

We also applied Liao's ABC to the waveguide structure in Fig. 5.6. The result is given in Fig. 5.10. We can see that a return loss less than $-30dB$ over the operating band has been achieved.

5.3.5 The ability of Liao's Absorbing Boundary Condition to handle waves with different incident angles

When the incident angle is zero, Liao's ABC performs best. Fig. 5.13 shows the return loss at zero incident angle when a 4th order transmitting formula is used. Liao's ABC can also handle waves with large incident angles. This can be demonstrated by the result we have previously got in the waveguide structure. It is commonly known that a single mode (for example a TE_{10} mode) travels in a zigzag fashion along a waveguide structure (Fig.

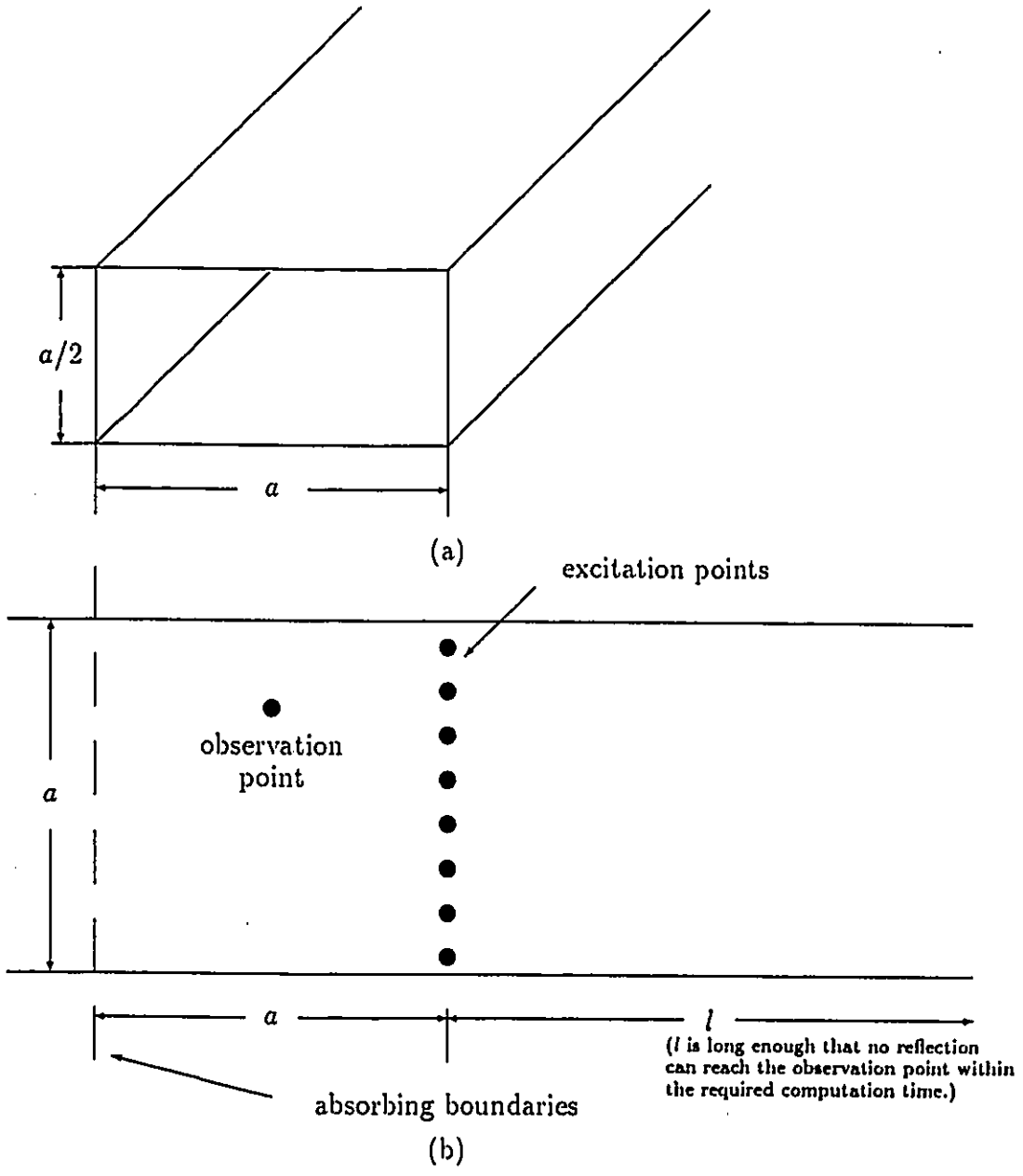


Figure 5.5: (a)WR28 waveguide, (b)Top view.

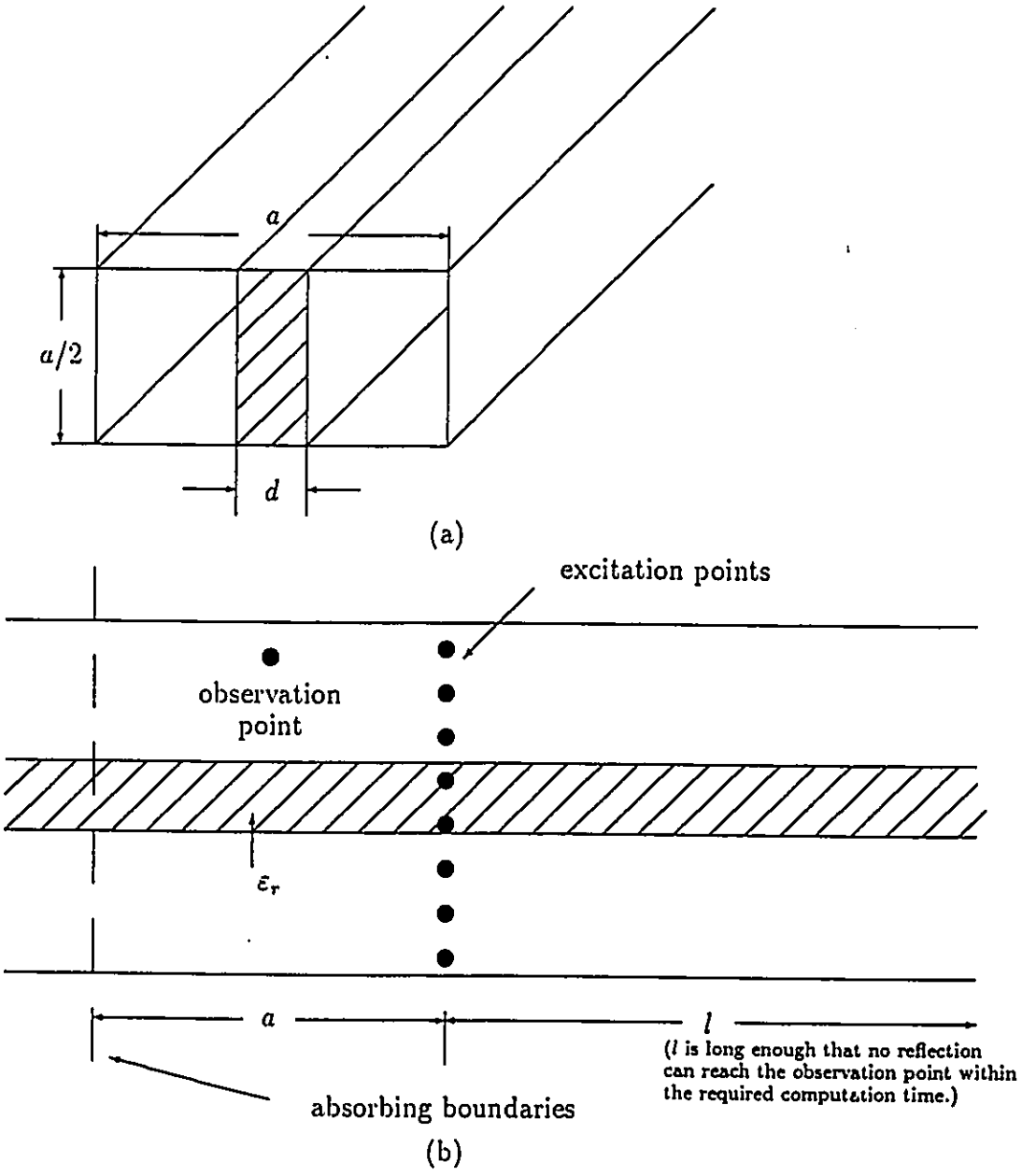


Figure 5.6: (a)WR28 waveguide loaded with a centered dielectric slab, $\varepsilon_r = 3$, $d = a/15$, (b)Top view.

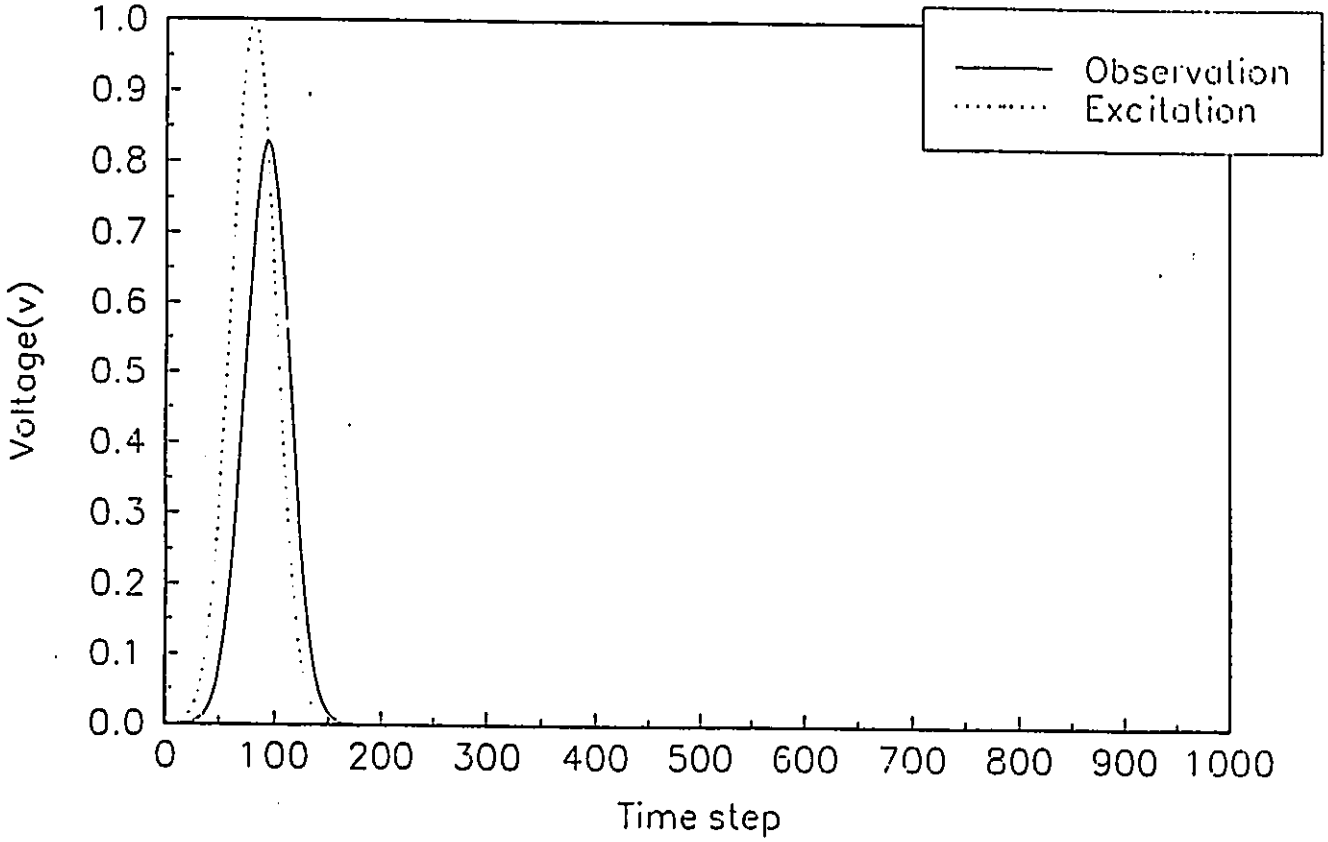


Figure 5.7: Time domain response of the Liao's Absorbing Boundary Condition to a Gaussian pulse. The time shift is due to the observation point being $10\Delta l$ away from the excitation points.

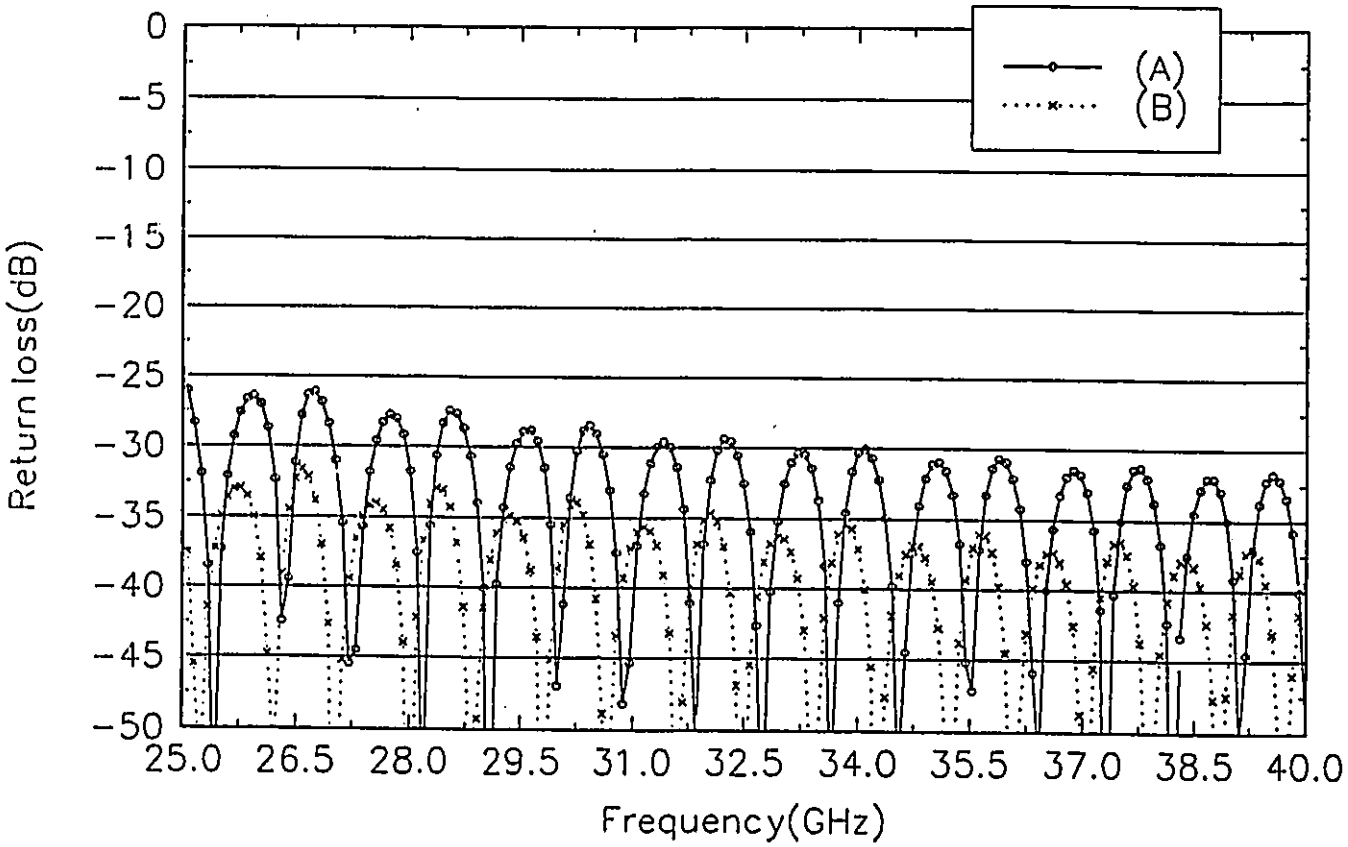


Figure 5.8: The effect of the order of the transmitting formula on the return loss of Liao's Absorbing Boundary Condition (1): (a) $N = 4$, (b) $N = 8$.

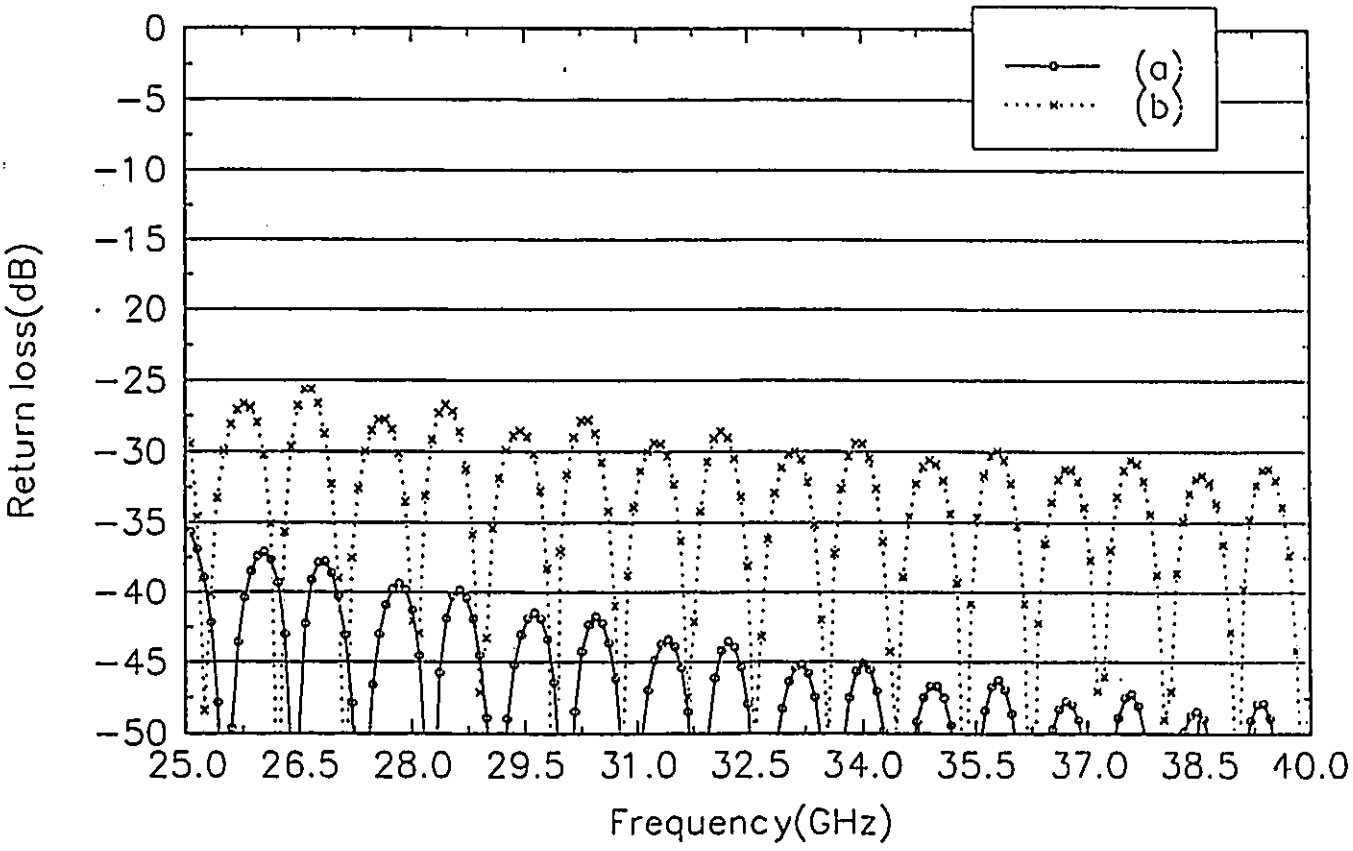


Figure 5.9: The effect of the order of the transmitting formula on the return loss of Liao's Absorbing Boundary Condition (2): (a) $N = 9$, (b) $N = 12$.

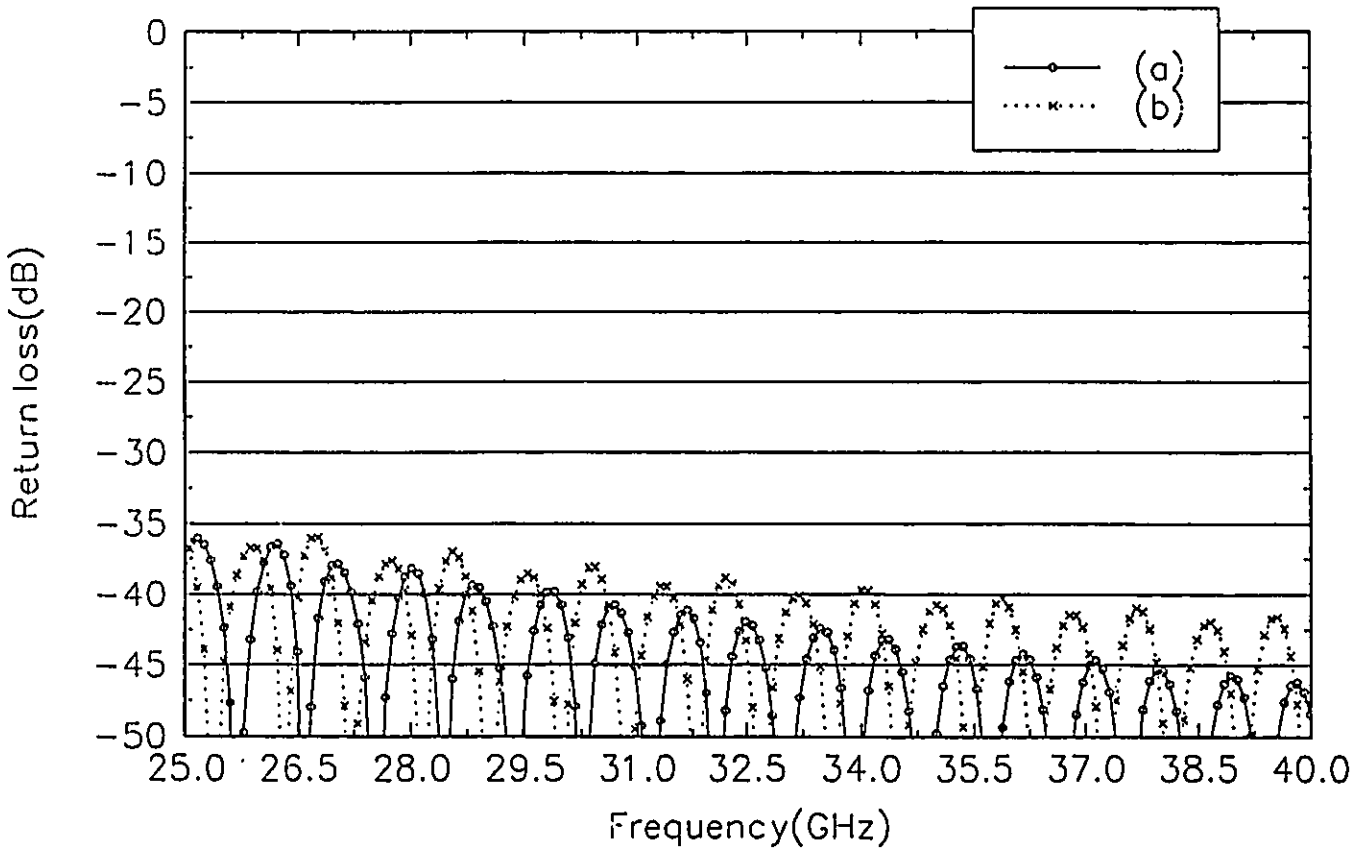


Figure 5.10: Return loss of Liao's Absorbing Boundary Condition applied to a WR28 waveguide loaded with dielectric slab, (a) $N = 9$, (b) $N = 11$.

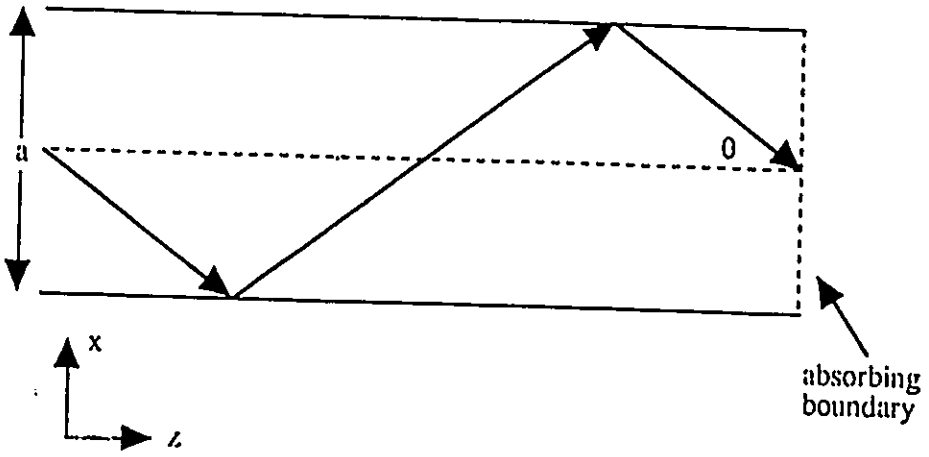


Figure 5.11: A single mode plane wave travels along a waveguide structure.

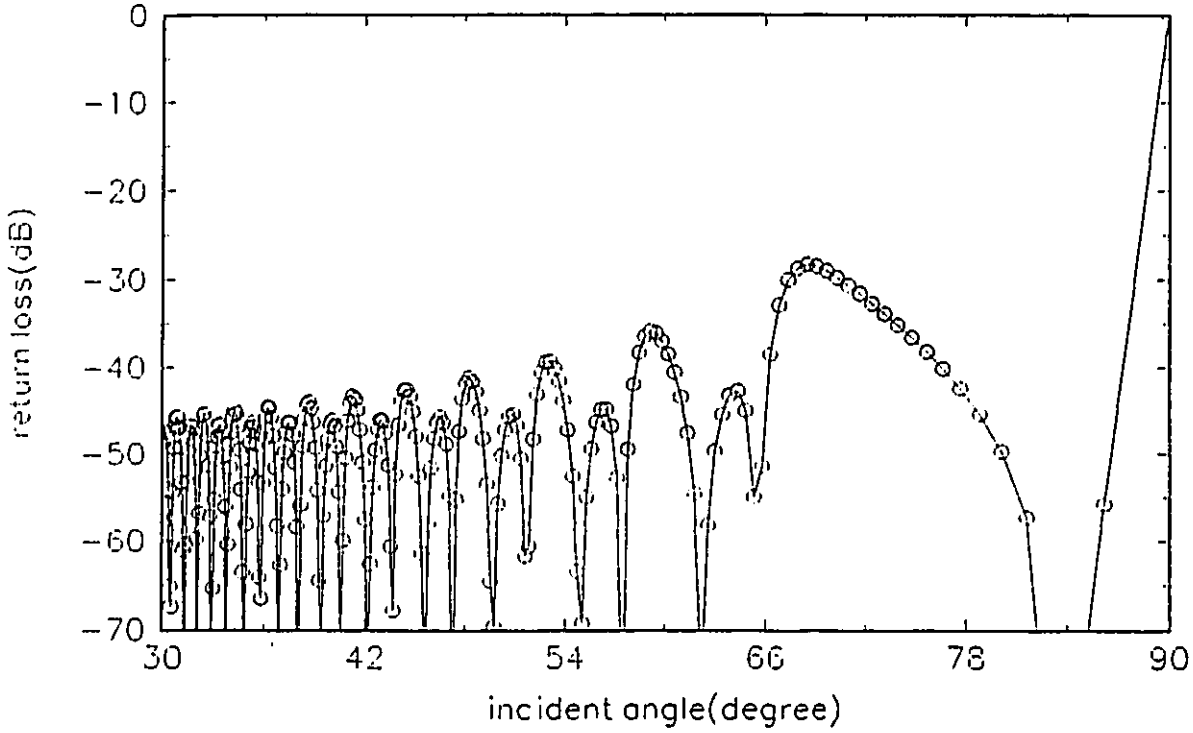
5.11). The value of angle θ is depended on the frequency f of this plane wave,

$$\cos\theta = \frac{\lambda}{\lambda_g} = \sqrt{1 - (\frac{c}{f\lambda_c})^2} \tag{5.22}$$

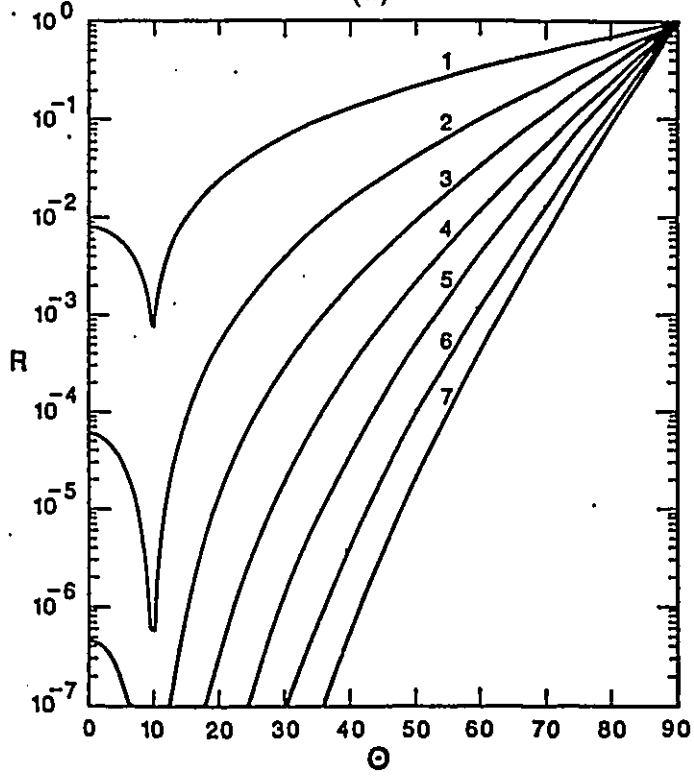
where c, λ_c are the light velocity and the cutoff wavelength of the mode, respectively.

Using this formula we can convert the relation between the return loss of Liao's ABC and frequency, into the relation between return loss and incident angles. Results have been given in Fig. 5.12. We can see that, the return loss is less than -20dB even when θ is as large as 85^0.

An analytical evaluation of Liao's ABC has been given in [3] and is redrawn in Fig. 5.12. We can see that the results we obtained agree well with the analytical one except for the ripples. These ripples result from the numerical errors and higher order evanescent modes in our simulation.



(a)



(b)

Figure 5.12: Return loss of Liao's Absorbing Boundary Condition with respect to incident angles, (a) obtained by TLM, (b) obtained analytically.

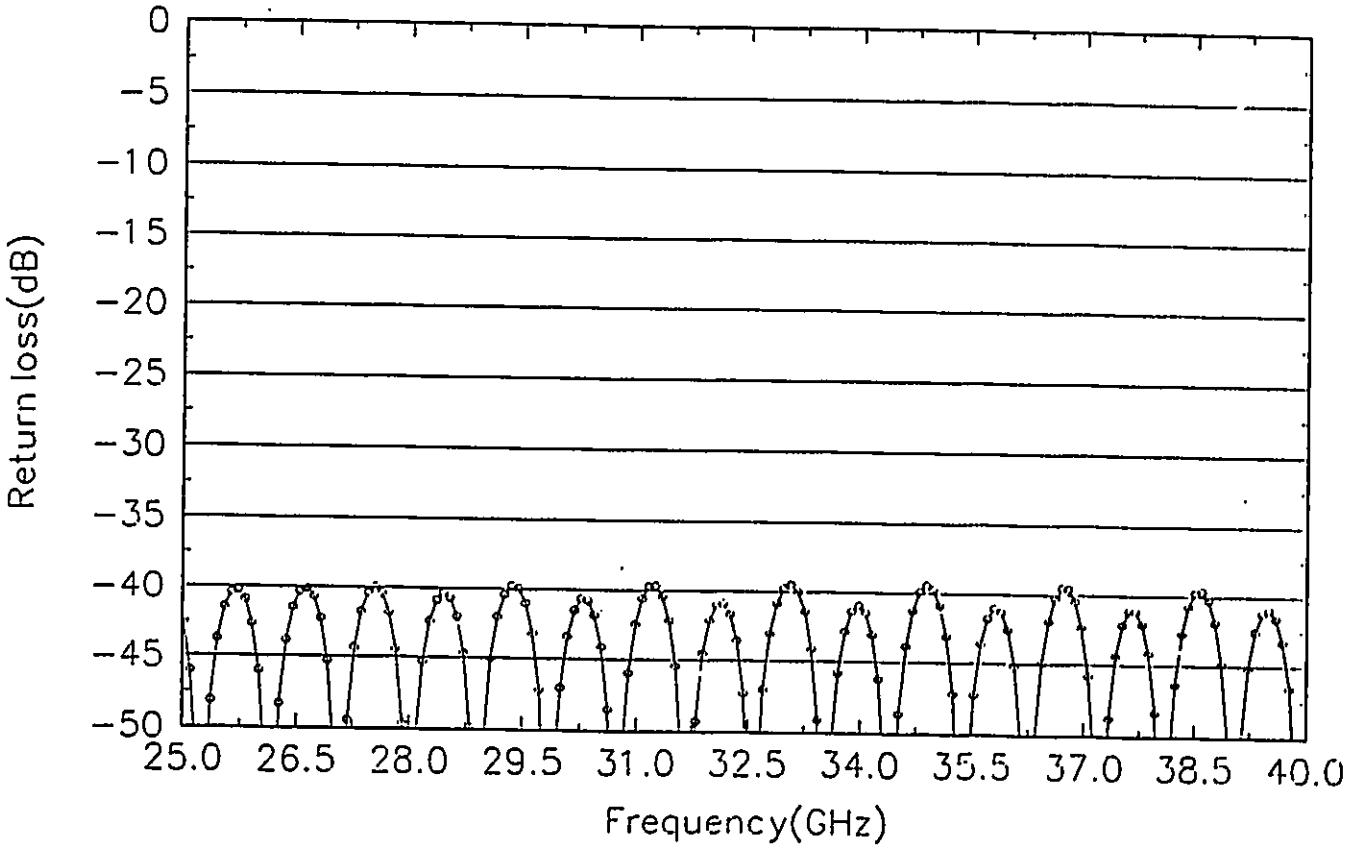


Figure 5.13: Return loss of Liao's Absorbing Boundary Condition when the incident wave is a TEM plane wave with zero incident angle.

Chapter 6

New Developments of Absorbing Boundary Algorithms in 2D-TLM 3: Combination approach

6.1 Introduction

We mentioned above that every numerical differential equation solver of order higher than two is not absolutely stable. However, in the vast majority of practical problems, a high order differential equation solver is a must to obtain the accuracy required. Therefore, ways to suppress instability are of great importance. In this thesis, a new approach by combining Dissipation ABC and Liao's ABC has been proposed. This concept can also be implemented in other extrapolation methods.

We know that a Dissipation ABC is stable under any circumstance. However, it is less accurate than some of other ABCs. The extrapolation approach has features that are just opposite to those of the Dissipation ABC. Thus , combining them together may yield an ABC which has good stability while maintaining the accuracy required.

Fig. 6.1 shows the idea of this combination approach. We made the sampling zone (where the sampling points of the extrapolation approach exist) lossy. It is obvious that, the higher the loss tangent of such a region, the higher the stability and the lower the accuracy. Thus a properly chosen lossy stub is required.

We suggest a stub with tapered loss which is shown in Fig. 6.1. The intermediate region

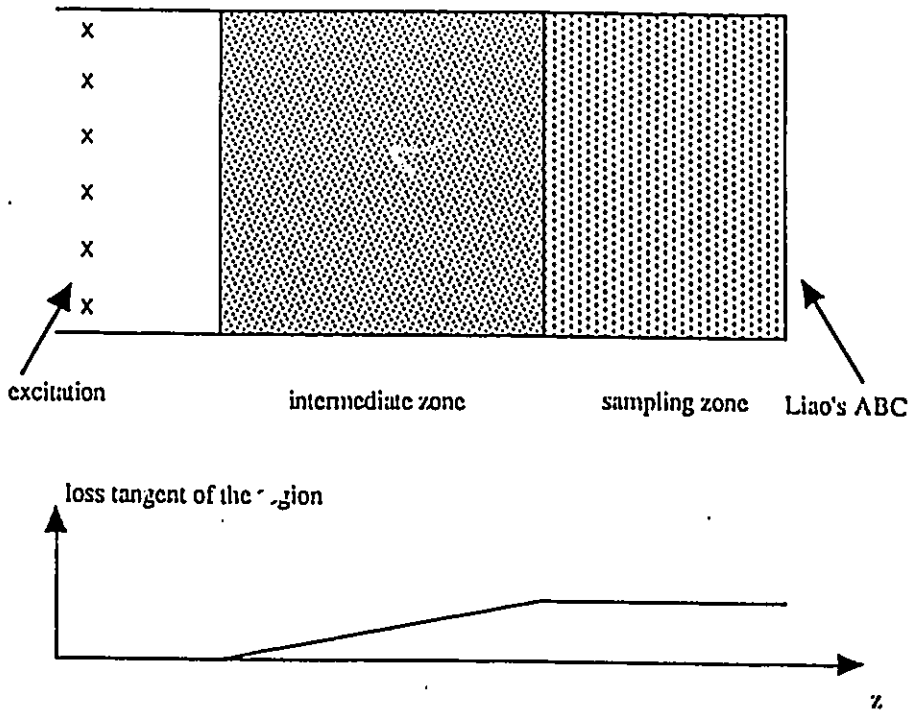


Figure 6.1: Schematic diagram of the Combination appproach

between the computation zone and the sampling zone has a slowly increasing loss tangent. It absorbs part of the energy from the incoming wave before it reaches the sampling zone. In the sampling zone, the loss tangent is the same everywhere in order to fit the sampling nature of the extrapolation approach.

6.2 Numerical results

A simple example was chosen to show the effect of such a combination approach. A TEM plane wave excited by a Gaussian pulse travels perpendicularly towards an absorbing boundary.

The intermediate zone is chosen 30 grid points long. 8th order Liao's ABC is used. Several values of the loss tangent in the sampling zone were chosen. Results are shown in Fig. 6.2. We can see that the 8th order Liao's ABC is unstable after 600 iterations and this instability is successfully suppressed by the lossy area. However, as expected, the absorption of the combination approach is worse.

In Fig. 6.3, frequency domain results have been compared among 8th order Liao's ABC,

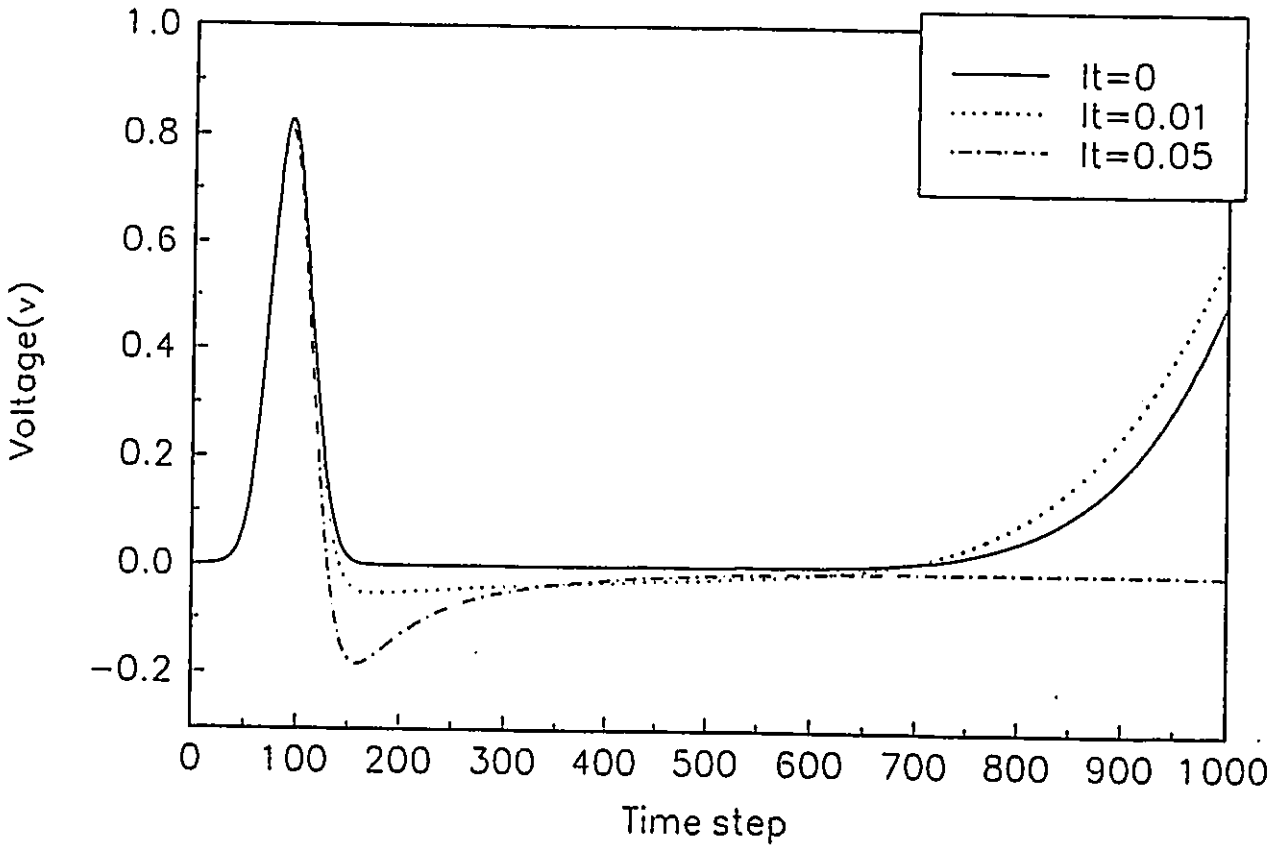


Figure 6.2: Time domain response of a 8th order Liao's Absorbing Boundary Condition and Combination Absorbing Boundary Conditions to a Gaussian excitation("*lt*" is the loss tangent in the sampling zone).

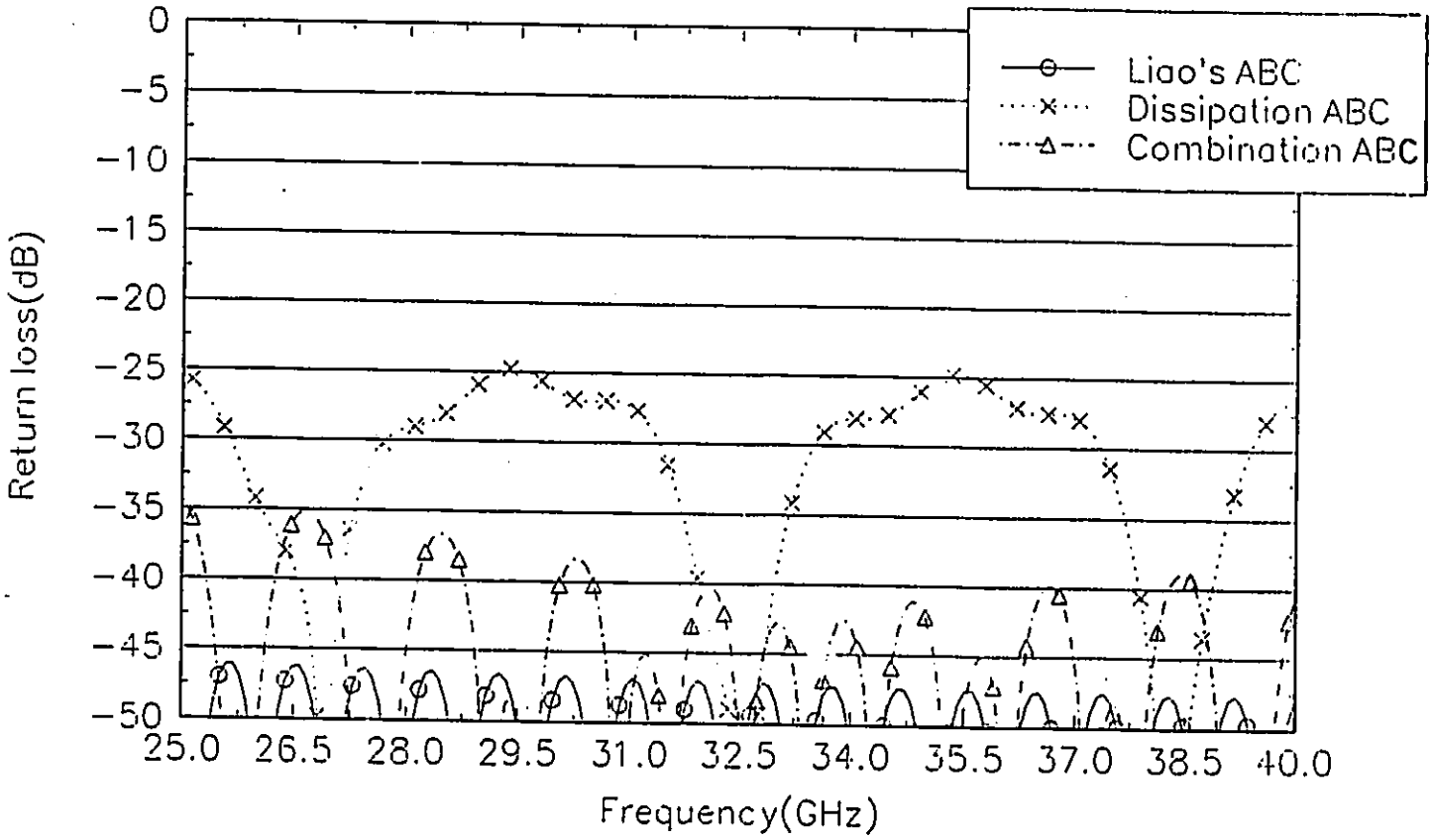


Figure 6.3: Comparison of return loss of Liao's Absorbing Boundary Condition, Combination Absorbing Boundary Condition and the Dissipation Absorbing Boundary Condition.

the Dissipation ABC and the Combination ABC, when applied to the above structure with an impulse excitation (8th order Liao's ABC is stable for impulse excitation). This figure proves our previous assumption, which is that the Combination ABC has an absorption worse than Liao's ABC and better than the Dissipation ABC.

6.3 Conclusion and future work

The combination approach can ease the instability problems affecting the extrapolation approach for a certain degree. However, as we can see above, the absorption becomes worse.

We can easily understand from the mechanism of an extrapolation approach that the instability is due to the errors of the prediction. Such errors are injected back into the system and accumulate there. If the accumulated error energy is greater than the real signal energy within the computation time, instability problems arise. Such instability can be controlled by reducing the error energy going back to the system. The combination approach is a method based on this idea.

The author has also tried to reduced the magnitude of the reflection voltages from points outside the AB. However, the absorption by this method is not as good as that obtained with the combination approach.

Finding a way to cure the instability of the extrapolation approach without losing much accuracy still remains an open field for researchers.

Chapter 7

New Developments of Absorbing Boundary Algorithms in 3D-TLM with Distributed Nodes

7.1 Introduction

In the above chapters, we have presented three kinds of new AB algorithms in 2D TLM. However, more practical microwave and electromagnetic structures are three-dimensional. In this thesis, we implemented all of the above ABCs in 3D TLM with distributed nodes. Ways to suppress the instability in Liao's ABC have been discussed.

7.2 Direct Optimized Dissipation approach in 3D TLM

The basic idea and formalism of this approach in 3D TLM is the same as in 2D TLM. Loss is introduced into a 3D distributed node by adding an infinitely long shunt stub of normalized characteristic admittance g_0 at shunt nodes.

The approach has been applied to a WR28 empty waveguide structure show in Fig. 7.1. 40 "lossy boards" were chosen. The optimized values of the loss tangents of these "lossy boards" and the reflection coefficient of the lossy boundary is given in Tab. 7.1. The time domain response to a Gaussian pulse is shown in Fig. 7.2. Frequency domain return loss can be seen in Fig. 7.3.

This ABC is unconditionally stable in 3D TLM.

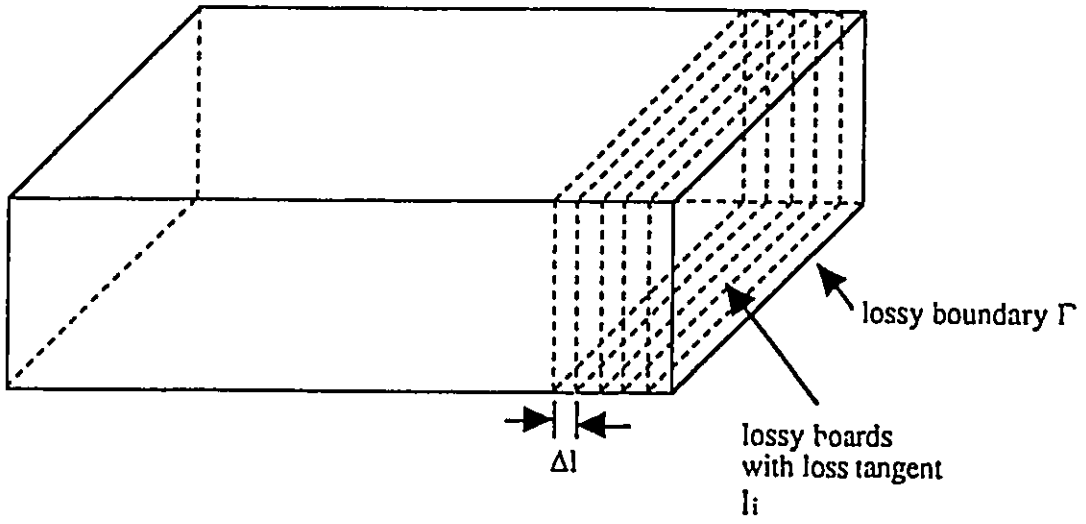


Figure 7.1: Applying the Direct Optimized Dissipation Absorbing Boundary Condition approach to a WR28 waveguide.

Γ_1	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
1.02e-2	2.54e-1	3.2e-2	1.79e-1	2.54e-2	1.12e-2	2.38e-2	3.33e-3	2.98e-1	2.7e-1	1.6e-3

l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}
2.4e-2	3.3e-3	1.14e-2	3.21e-1	2.34e-2	2.98e-2	5.2e-3	7.1e-2	9.06e-2	7.5e-3

l_{21}	l_{22}	l_{23}	l_{24}	l_{25}	l_{26}	l_{27}	l_{28}	l_{29}	l_{30}
5.43e-3	7.6e-2	2.54e-2	2.35e-1	2.99e-2	3.87e-6	5.43e-5	3.09e-2	1.90e-3	1.87e-2

l_{31}	l_{32}	l_{33}	l_{34}	l_{35}	l_{36}	l_{37}	l_{38}	l_{39}	l_{40}
5.44e-5	3.32e-6	2.91e-2	1.20e-2	5.43e-3	3.22e-2	3.15e-2	2.32e-6	2.43e-2	4.41e-3

Table 7.1: Optimized values of Γ and l_i.

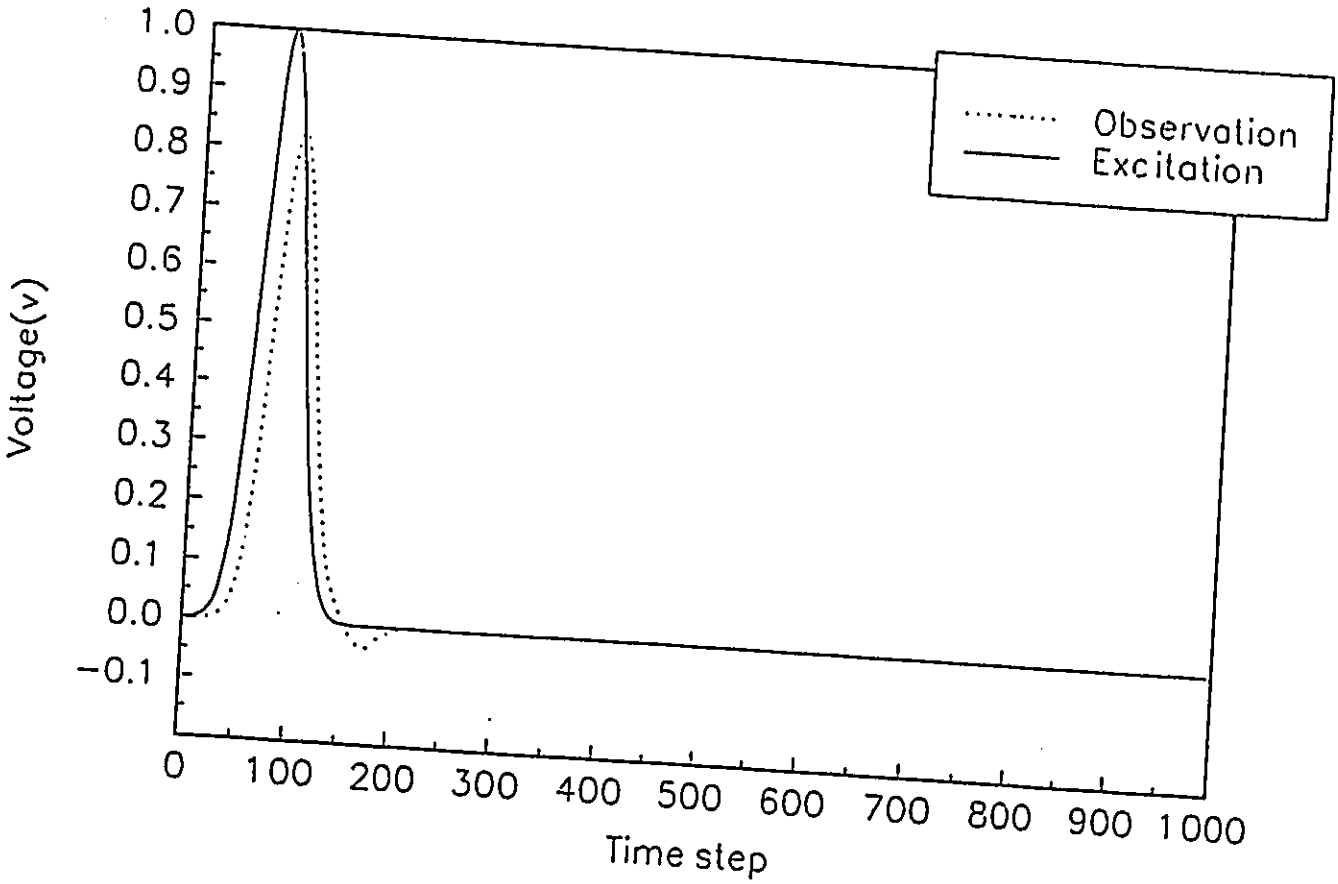


Figure 7.2: Time domain response of a 3D Direct Optimized Dissipation Absorbing Boundary Condition to a Gaussian pulse.

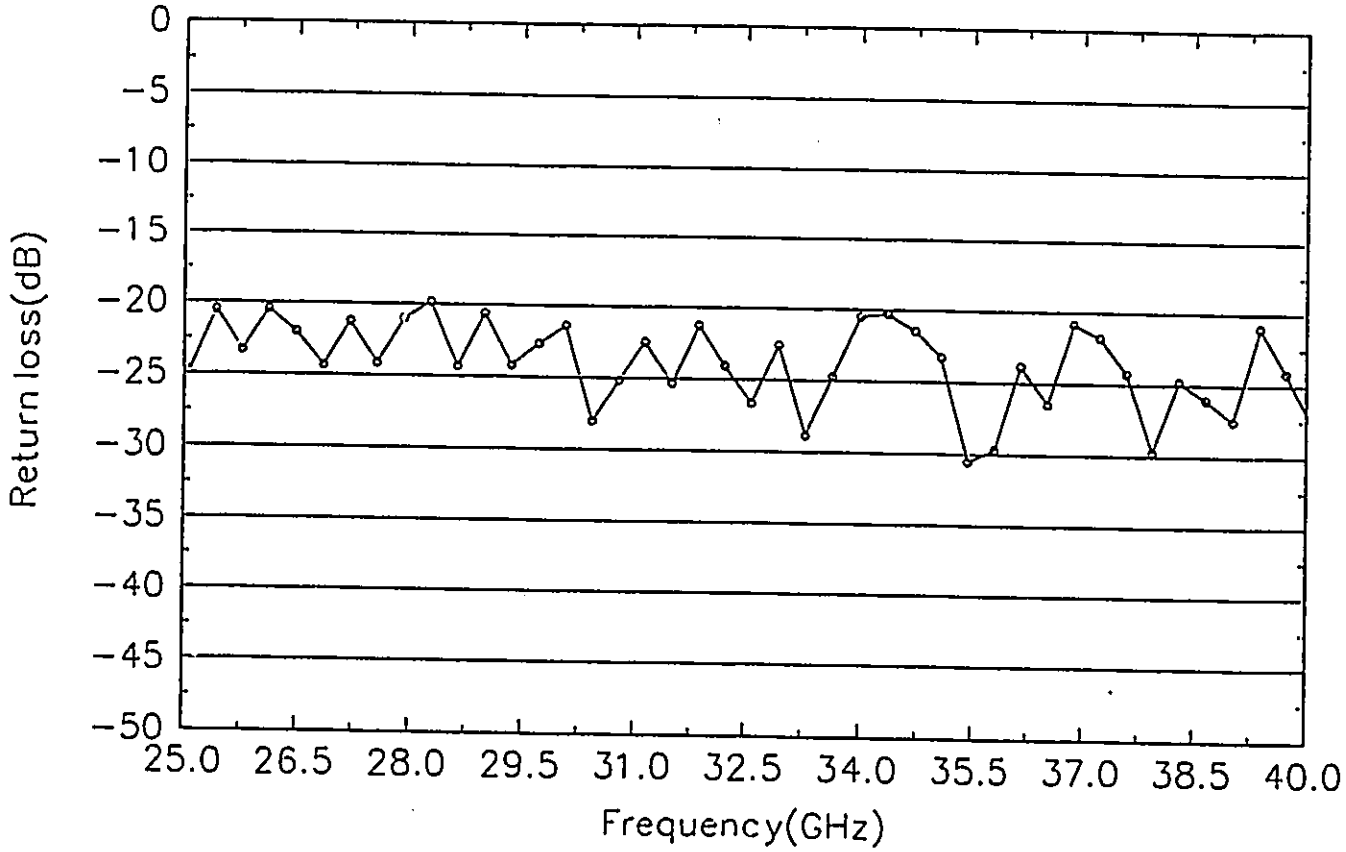


Figure 7.3: Return loss of a 3D Direct Optimized Dissipation Absorbing Boundary Condition.

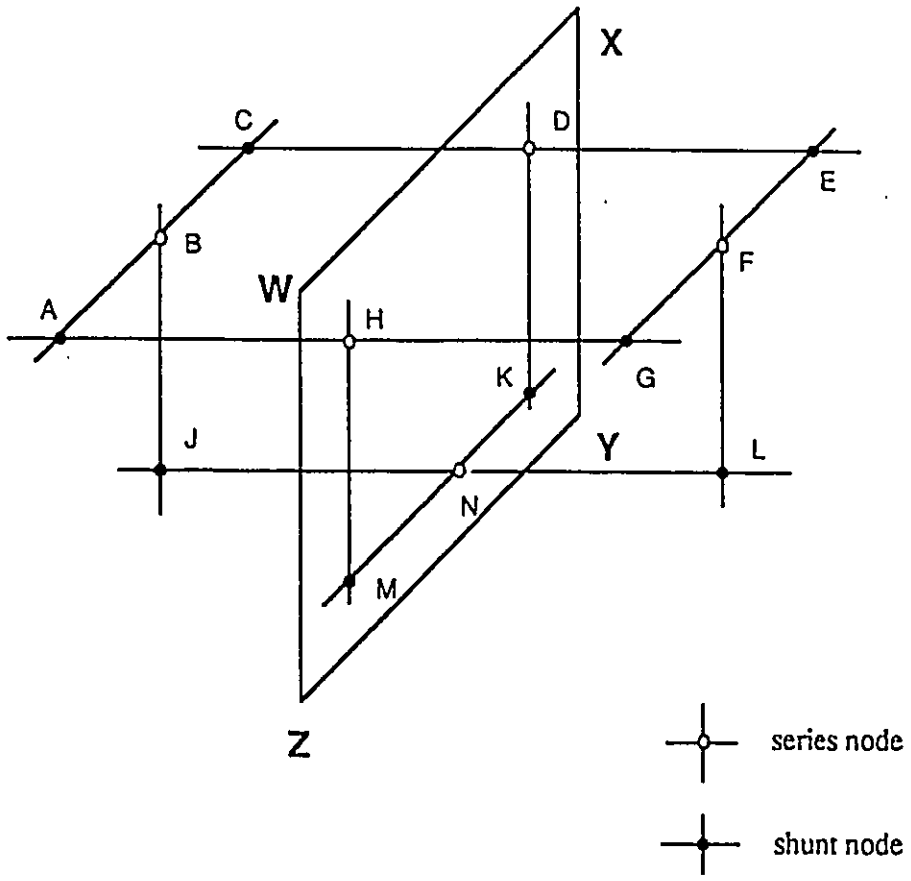


Figure 7.4: Schematic diagram showing the implementation of Liao's Absorbing Boundary Condition in 3D-TLM

7.3 Liao's Absorbing Boundary Condition in 3D TLM

7.3.1 Formalism

Applying Liao's ABC in 3D TLM is similiar to 2D except the way the absorbing boundary is placed. Fig. 7.4 shows a 3D absorbing boundary($WXYZ$ plane). The left hand side of this absorbing boundary is the computation region. If we define $ABJNMH$ as a 3D TLM "node", we can see that this node and the outer space separated by the absorbing boundary are connected by two branches, which are JL and AG. Thus, if we can properly simulate the reflection of an infinitively large outer space into the computation region through these two branches, we implement ABC in 3D-TLM. In other words, prediction is

performed along one branch for a node in 2D and along two branches for a node in 3D.

7.3.2 Numerical Results

3D Liao's ABC has been applied to a WR28 waveguide. Only the dominant mode(TE_{10}) has been excited. The time domain response to a Gaussian pulse has been given in Fig. 7.5. Frequency domain results have been shown in Fig. 7.6. The absorption is found to be good within the operating band.

7.3.3 Discussion of the stability problem

The instability problem of Liao's ABC is serious in 3D. We found that the instability arose when the order of Liao's ABC is greater than 3. In order to suppress this instability, like in the 2D case, a combination approach can be used. Results similar to 2D have been obtained and are shown in Fig. 7.7. The combination approach can suppress the instability to a certain degree. If the order of Liao's ABC is high, this approach is no longer useful. Finding a good way to suppress the instability is still a task for future researchers.

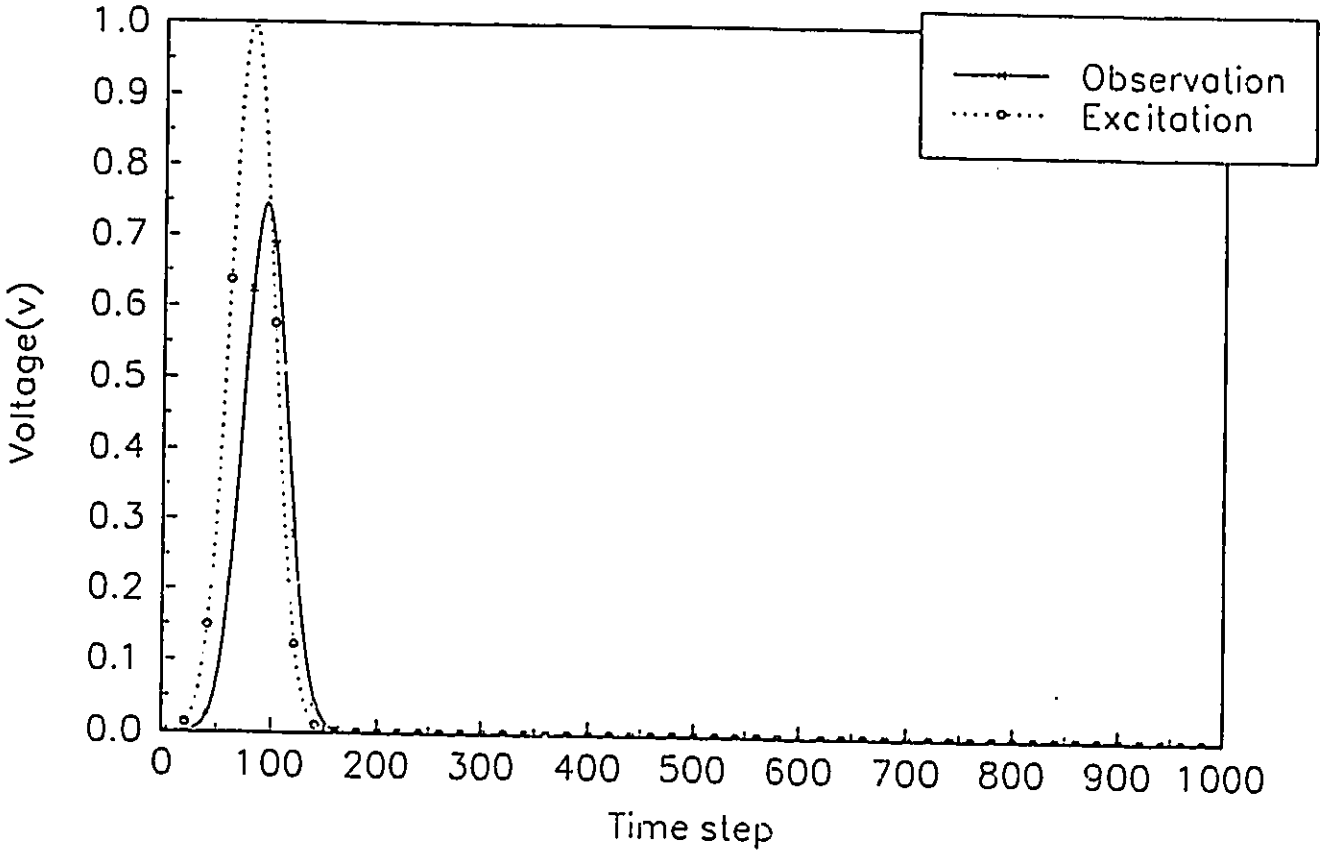


Figure 7.5: Time domain response of a 3D Liao's Absorbing Boundary Condition to a Gaussian pulse.

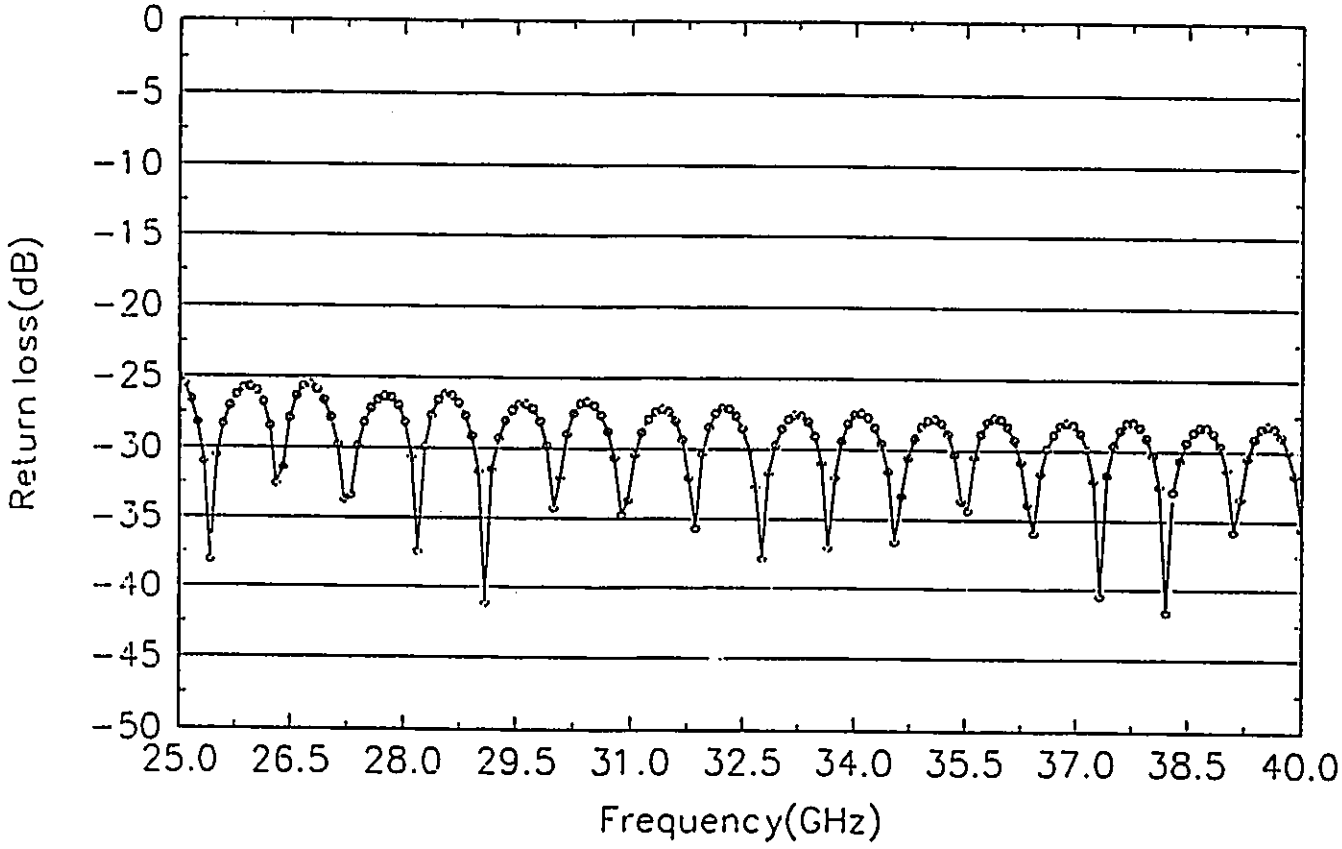


Figure 7.6: Return loss of 3D Liao's Absorbing Boundary Condition applied on a WR28 waveguide.

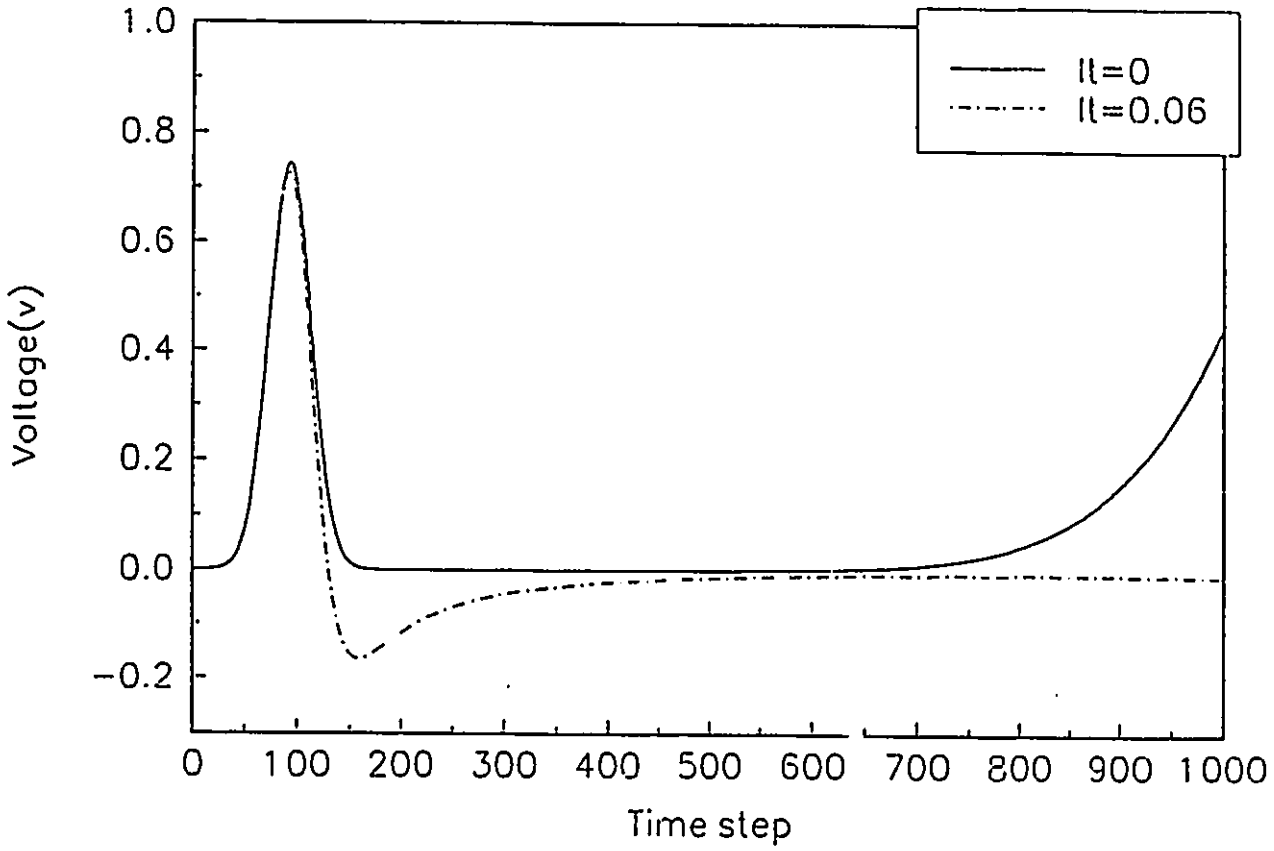


Figure 7.7: Instability of 3D Liao's Absorbing Boundary Condition can be suppressed by the combination approach ("*lt*" is the loss tangent in the sampling zone).

Chapter 8

Application of New Absorbing Boundary Algorithms

8.1 Simulation of an E-plane filter

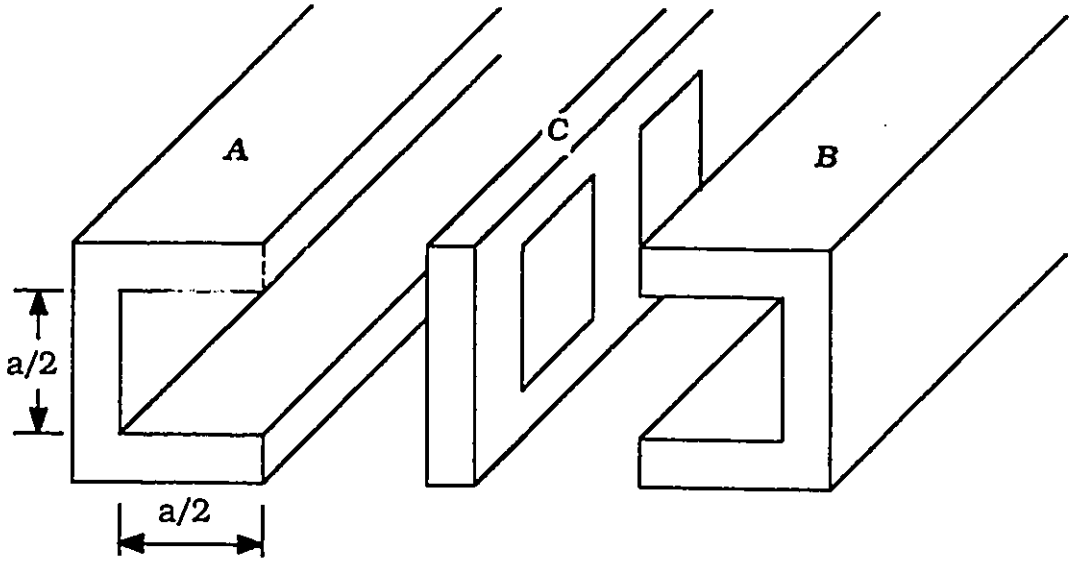
E-plane filters are popular in microwave devices. Its configuration can be seen in Fig. 8.1. Part A and B are obtained by simply cutting a waveguide in the middle of the wide side. C is a dielectric slab covered by metal on both sides in some region. Fig. 8.1 (b) and (c) give the side and top view of such a slab. By putting A, B and C together and carefully choosing the size of the metal-covered region on the slab, one can obtain an E-plane bandpass filter.

To simulate such a filter by a time domain method, the waveguide must be terminated by matched load at its two ends. Numerically, such matched loads can be simulated by an AB.

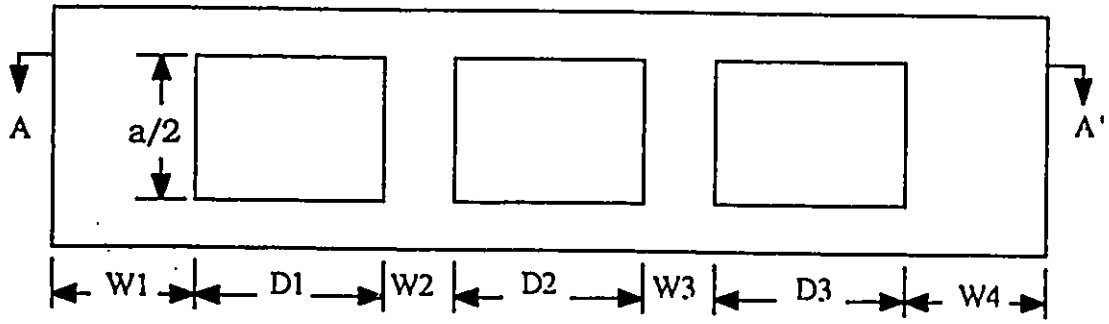
8.2 Applying new Absorbing Boundary algorithms to the simulation of an E-plane filter

The E-plane filter in this thesis has the inner cross-section dimensions of a WR28 waveguide. The parameters of the metal covered dielectric slab are given in Tab. 8.1.

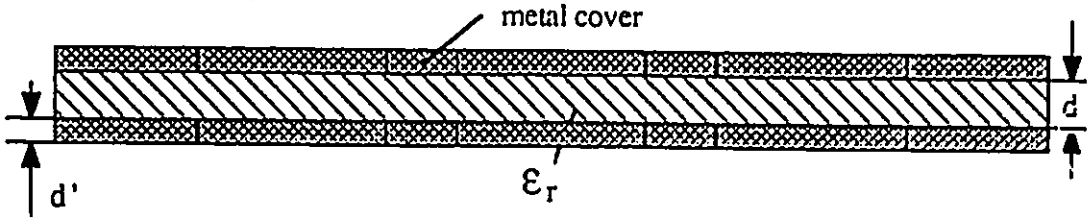
The TLM method is used to simulate this filter and Direct Optimized Dissipation AB as well as Liao's AB are used to simulate the matched load at the two ends of the waveguide. We excite the dominant mode (TE_{10}) by a Dirac impulse on one side of the filter and



(a)



(b)



A--A' view

(c)

Figure 8.1: The configuration of an E-plane filter, (a) schematic diagram, (b) side view of the metal covered dielectric slab, (c) top view of the slab.

W_1	D_1	W_2	D_2	W_3	D_3	W_4
2.22 mm	3.45 mm	5.57 mm	3.45 mm	5.57 mm	3.46 mm	2.23 mm

ε_r	d	d'
2.22	0.127 mm	0.0175 mm

Table 8.1: Parameters of an E-plane filter.

observe the time response at a point on the other side. The insertion loss of the filter can be obtained by performing Fourier Transform on this time domain results.

8.3 Comparison between experimental and theoretical results

Fig. 8.2 gives the insertion loss obtained by using Liao's ABC and the Direct Optimized Dissipation ABC. The experimental results which were given in the author's undergraduate thesis [24] have also been included in this figure. We can see that except a slight shift of the central frequency,the theoretical results agree reasonably well with the experimental ones. In the pass band, the experimental insertion loss is greater than the theoretical results. This is due mainly to losses in the metal-covers (skin effect) which were not taken into account in the theoretical model. The frequency shift between the measurement and the computation can be explained as follows:

1. Errors in frequency measurement.
2. Misalignment between the filter section and the external measuring circuit.
3. Metallization thickness: zero-thickness was used by the theoretical model. Due to the limit of our computer resources, only a coarse mesh (with respect to the

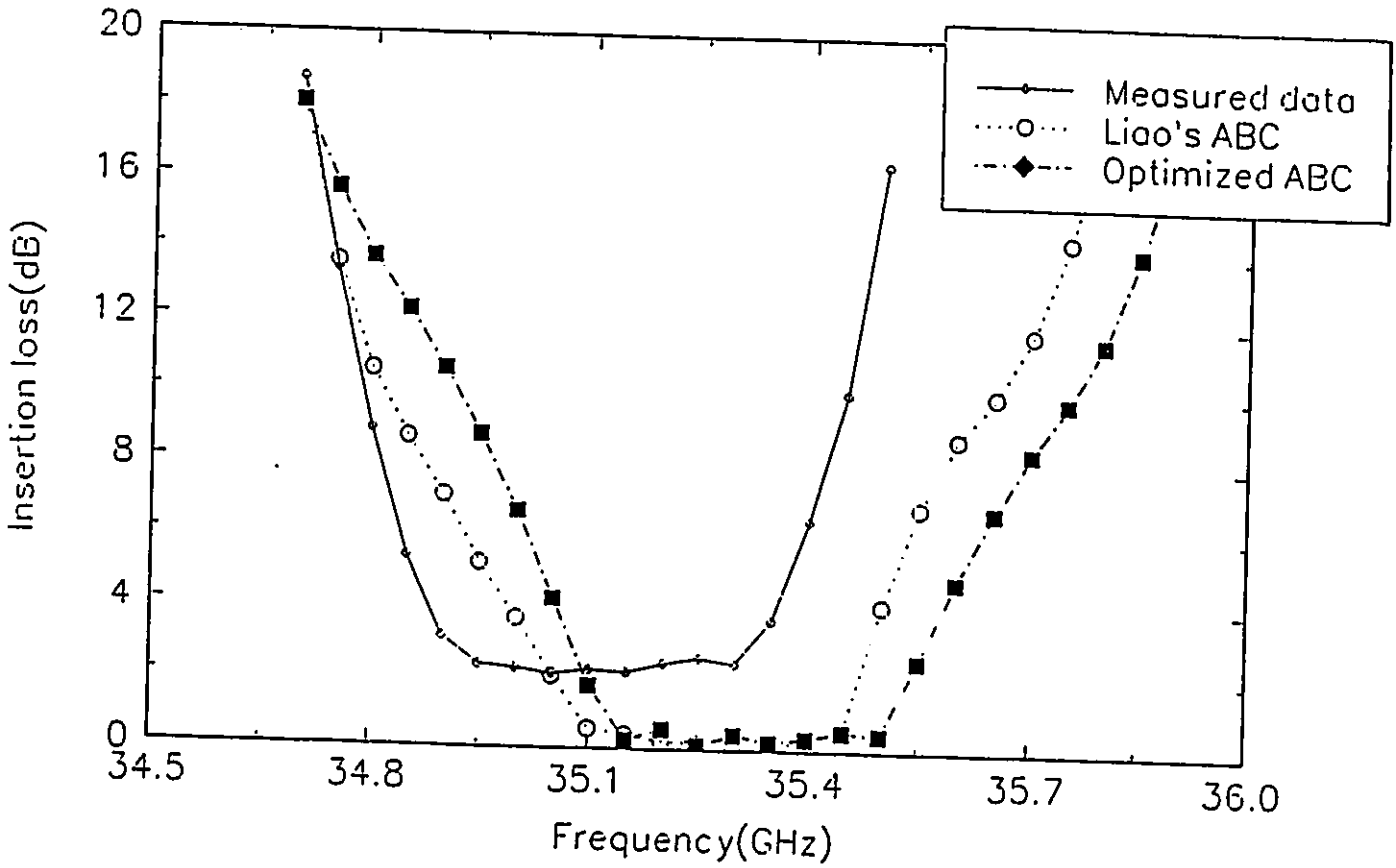


Figure 8.2: Comparison of experimental and theoretical results using Liao's ABC and Direct Optimized Dissipation Absorbing Boundary Condition.

thickness of the metal-covers) was used.

4. Errors in dimension within the manufacturing tolerances. Also, the size of the TLM mesh was not exactly an integer multiple of the dimensions of the filter. This also introduced errors in the theoretical model.

Chapter 9

Discussions and Conclusions

As the rapid development of microwave and electromagnetic techniques, frequency domain analysis becomes quite limited. More and more designers have turned to time domain approaches, such as TLM and FD-TD, etc.. To solve open space problems, ABC must be used to save computer time and memory. In this thesis, the author introduced the basic formalism of the TLM method and the recent developments of time domain AB algorithms. Furthermore, three kinds of new AB algorithms in TLM, which are the original contribution of the author, have been fully described. These new AB algorithms are the Direct Optimized Dissipation ABC approach, Liao's ABC approach and the combination approach. They are implemented both in 2D TLM and 3D TLM with distributed nodes.

The Direct Optimized Dissipation ABC is more accurate than any other Dissipation ABC because the optimization is performed directly on the structure being studied. It requires much less extra computation space than other Dissipation approaches. It is absolutely stable. This advantage makes it possible for this ABC to be used in many cases where the extrapolation approach has instability problems. The only shortcoming of this approach is that it takes long time to do the optimization if the TLM mesh layout of the structure being studied is large.

Liao's AB algorithm is an extrapolation approach proposed by Liao et al [17] in the FEM method. In this thesis, it has been implemented in the TLM method. Numerical results show that it has good absorption even if the AB is very close to the excitation

sources. It has been applied to two WR28 waveguide structures, where a return loss of less than $-30dB$ over the whole operating band has been achieved. It can also absorb waves with incident angles as large as 85°.

Due to the errors of the prediction of Liao's AB algorithm accumulating during the computation time, it is potentially unstable, especially for 3D cases. For 2D, the system can remain stable when the order of it is smaller than 9. Thus, Liao's ABC is a good ABC and can be a first-to-choose approach for solving 2D problems.

Ways to suppress instability of Liao's ABC have been proposed and discussed in this thesis. Combining the Dissipation ABC and the Liao's ABC is one of the approaches. From the numerical results we can see that, the instability of the Liao's ABC was suppressed by this approach. However, the absorption became worse. The author has also tried to reduce the reflection voltages from points outside the AB. However, absorption is found to be not as good as the combination approach. How to reduce the accumulation of the error energy of the extrapolation approach while maintaining its good absorption remains an open field for future researchers.

The Direct Optimized Dissipation ABC and Liao's ABC have been applied to a practical E-plane filter. Results have been compared to the published data. Good agreement has been achieved.

Since the extrapolation approach has good absorption and can be placed close to the excitation sources, the author suggests that, in future research, a greater effort should be made to find a way to stabilize the extrapolation approach while maintaining its good absorption characteristics.

Bibliography

[1] S. Akhtarzad, and P. B. Johns, "Numerical Solution of Lossy Waveguide: T.L.M. Computer Program", Electron. Lett., Vol. 10, No. 15, pp.309-311, 25th July 1974.

[2] S. Akhtarzad, "Analysis of Lossy Microwave Structures and Microstrip Resonators by the TLM Method", Ph.D dissertation, University of Nottingham, England, July 1975.

[3] Z. Z. Chen, Ph.D thesis, University of Ottawa, Ottawa, Canada,1992.

[4] Dalquest, Ph.D. thesis, 1963.

[5] B. Engquist and A. Majda, "Radiation Boundary Conditions for Acoustic and Elastic Wave Calculations", Comm. Pure Appl. Math., v.32, 1979, pp.313-357.

[6] B. Engquist and A. Majda, "Absorbing Boundary Conditions for the Numerical Simulation of Waves", Math. Comp., v.31, 1977, pp.629-651.

[7] Eswarappa, George I. Costache, Wolfgang J. R. Hoefer, "Transmission Line Matrix Modeling of Dispersive Wide-Band Absorbing Boundaries with Time-Domain Diakoptics for S-Parameter Extraction", IEEE-MTT, vol. 38, No.4, April 1990, pp379-385.

[8] R. L. Higdon, "Absorbing boundary conditions for difference approximations to the multi-dimensional wave equation", Math. Comp., v. 47, 1986, pp.437-459.

[9] R. L. Higdon, "Numerical Absorbing Boundary Conditions for the Wave Equation", Math. Comp., v. 49, 1987, pp.65-90.

[10] W. J. R. Hoefer and Poman So, "The Electromagnetic Wave Simulator,"John Wiley & Sons Ltd, 1991.

[11] W. J. R. Hoefer, " The Discrete Time Domain Green's Function or Johns Matrix–A New Powerful Concept in TLM", Intl. Journal of Numerical Modelling, Vol 2, No. 4, pp.215-225, 1989.

[12] Tatsuo Itoh, "Numerical Techniques for Microwave and Millimeter-Wave Passive Structures",John Wiley & Sons,1989.

[13] P. B. Johns and R. L. Beurle, "Numerical solution of 2-dimensional scattering problems using a transmission-line matrix," Proc. Inst. Electr. Eng., vol. 118, pp. 1203-1208, Sept. 1971.

[14] P. B. Johns, "Application of the Transmission-Line Matrix Method to Homogeneous Waveguides of Arbitrary Cross-Section", Proc. IEE, Vol. 119, No. 8, pp.1086-1091, August 1972.

[15] P. B. Johns, "The Solution of Inhomogeneous Waveguide Problems Using a Transmission-Line Matrix", IEEE-MTT, Vol. MTT-22, No. 3, pp.209-215, March 1974.

[16] P. B. Johns, " A New Mathematical Model to Describe the Physics of Propagation", The Radio and Electronic Engineer, Vol. 44, No. 12, pp.657-666, Dec. 1974.

[17] Z. P. Liao, H. L. Wong, B. P. Yang, and Y. F. Yuan, "A transmitting boundary for transient wave analysis," Scientia Sinica, vol. XXVII, no. 10, p. 1065, October 1984

[18] Liao Z. P., & Wong, H. L., "Proceedings of the ASCE Eng. Mech. Div. Specialty Conference," 1983, May, 23-25.

[19] Liao, Z. P. & Wong, H. L., "Proceedings of International Workshop on Earthquake Engineering," Shanghai, March 1984.

[20] E. L. Lindman, " "Free-space" Boundary Conditions for the Time Dependent Wave Equation", J. Comput. Phys., v. 18, 1975, pp.66-78.

[21] M. Moghaddam, E. J. Yannakakis, and W. C. Chew "Modeling of the subsurface interface radar," J. Electromagn. Waves Appl., Vol. 5, No. 1, 17-39, 1991

[22] M. Moghaddam and W. C. Chew, "Stabilizing Liao's Absorbing Boundary Conditions Using Single-Precision Arithmetic,"1991

[23] T. G. Moore, J. G. Blaschak, A. Taflove and G. A. Kriegsmann, "Theory and Application of Radiation Boundary Operators," IEEE-AP, vol. 36, No. 12, December 1988, pp1797-1812.

[24] Qiyang Li, "Studies of MM-Wave Filters", Bachelor thesis, Tsinghua University, Beijing, China, June 1987.

[25] Zhenghe Feng, Qiyang Li and Jie Lai, "The Accurate Design of MM-Wave Fin-Line Filter",Proceedings of International Conference on Millimeter Wave and Far-Infrared Technology, June 1989, Beijing, China. pp338-341

[26] P. Saguet, "Analyse des milieux guidés -la méthode MTLM," Doctoral Thesis, Inst. Natl. Polytech., Grenoble, 1985.

[27] A. Sommerfeld, "Partial Differential Equations in Physics", New York: Academic, 1949.

Appendix A

Source code of Liao's Absorbing Boundary Condition in 2D-TLM

```
/* Liao's ABC in 2D-TLM. */
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define PI=3.1415926

FILE        *inf,*tmf,*time;
int         nx,ny;
int         io,it,jo,ni,deltat;
int         kb,kc,kd,ke;
double      v[7][602][33];
double      r[40];
double      rc[40],rd[40];
double      a,va[40];
double      c,kka,kkb;
double      ehre,ehim,d,tow;
double      eh[9000];
int         ib[6][9];
int         ibd[6][9];
int         ie[6][9];
int         ia[6][9];
char        header[80];
int         l,jd,j,m,i,k,ic,pt,ptp,ptm,nn;
double      pcf,cf,d1,d2,ds;
double      peak,cs,max,mmax,mmin,md,yo;
int         npt;
double      data[301][3];
char        buf[80];

main()
{
  inf=fopen("tlm_inh1.inp","r");
  tmf=fopen("liaod.out","wt");
  time=fopen("time.out","wt");
  readnxny();
  readbound();
  readdielbound();
  readcompbox();
  readexcitation();
  readfreq();
  fclose(inf);
  iterate();
  fourier();
  curvefit();
```

```
    correct();
    results();
    close(time);
    return 0;
}

readnxny()
{
    fgets(header,80,inf);
    printf("\n%s",header);
    fgets(header,80,inf);
    printf("%s",header);
    fscanf(inf,"%d %d\n", &nx,&ny);
    printf("%4d%4d\n", nx, ny);
    return 0;
}

readbound()
{
    fgets(header,80,inf);
    printf("%s",header);
    fgets(header,80,inf);
    printf("%s",header);
    kb=0;
    do {
      kb=kb+1;
      for(m=1;m<=8;m++) {
        fscanf(inf,"%d",&ib[kb][m]);
        printf("%4d",ib[kb][m]);
        if(m==4) printf("     ");
      }
      fscanf(inf,"%lf %d\n",&r[kb],&it);
      printf("\t%8.4lf\t%4d\n",r[kb],it);
    }while(it!=0);
    return 0;
}

readdielbound()
{
    kd=0;
    fgets(header,80,inf);
    printf("%s",header);
    fgets(header,80,inf);
    printf("%s",header);
```

```
  do {
    kd=kd+1;
    for(m=1;m<=8;m++) {
      fscanf(inf,"%d",&ibd[kd][m]);
      printf("%4d",ibd[kd][m]);
      if(m==4) printf("     ");
    }
    fscanf(inf,"%lf %d\n",&rc[kd],&it);
    printf("\t%8.4lf\t%4d\n",rc[kd],it);
  }while(it!=0);
  return 0;
}

readcompbox()
{
  kc=0;
  fgets(header,80,inf);
  printf("%s",header);
  fgets(header,80,inf);
  printf("%s",header);
  do {
    kc=kc+1;
    for(m=1;m<=4;m++) {
      fscanf(inf,"%d",&ia[kc][m]);
      printf("%4d",ia[kc][m]);
    }
    fscanf(inf,"%lf %d\n",&rd[kc],&it);
    printf("\t%8.4lf\t%4d\n",rd[kc],it);
  }while(it!=0);
  return 0;
}

readexcitation()
{
  ke=0;
  fgets(header,80,inf);
  printf("%s",header);
  fgets(header,80,inf);
  printf("%s",header);
  do {
    ke=ke+1;
    for(m=1;m<=7;m++) {
      fscanf(inf,"%d",&ie[ke][m]);
      printf("%4d",ie[ke][m]);
```

```
        if(m==4) printf("      ");
      }
      fscanf(inf,"%lf %d\n",&va[ke],&it);
      printf("\t%8.4lf\t%4d\n",va[ke],it);
   }while(it!=0);
   return 0;
}

readfreq()
{
   fgets(header,80,inf);
   fgets(header,80,inf);
   fscanf(inf,"%lf %lf %lf\n",&d1,&d2,&ds);
   fgets(header,80,inf);
   fscanf(inf,"\n%d %d %d %d %lf %d %lf %lf %lf %lf",
     &io,&jo,&l,&ni,&yo,&deltat,&tow,&c,&kka,&kkb);
   printf("Output point is(%4d,%4d) and l=%4d\n",io,jo,l);
   printf("Number of iterations is %4d\n",ni);
   printf("Permittivity stub admittance is %6.4lf\n",yo);
   printf("   D1        D2          Step Size  deltat   tow\n");
   printf("%8.6lf %8.6lf %8.6lf %4d %8.6lf\n",d1,d2,ds,deltat,tow);
   printf("Finished reading input. Working hard on TLM now.\n");
   printf("Please be patient!\n");
   return 0;
}

iterate()
{
   double c,a,a1,a2,vx,vy,vxy,vv[20][33][20],vvv[20][33][20];
   void excite();

   for(j=1;j<=ny;j++) {
      for(i=1;i<=nx;i++) {
         for(m=1;m<=5;m++) {
            v[m][i][j]=0.0;
         }
      }
   }
   for(j=1;j<=ny;j++) {
      for(i=1;i<=14;i++) {
        for(m=1;m<=14;m++) {
           vvv[m][j][i]=0.0;
           vv[m][j][i]=0.0;
         }
```

```
    }
  }

  for(ic=1;ic<=ni;ic++) {
    if(ic==1) excite();
    for(nn=1;nn<=kb;nn++) {
      for(j=ib[nn][3];j<=ib[nn][4];j++) {
        for(i=ib[nn][1];i<=ib[nn][2];i++) {
          vxy=v[ib[nn][6]][i][j];
          v[ib[nn][6]][i][j]=r[nn]*
                        v[ib[nn][5]][i+ib[nn][8]][j+ib[nn][7]];
          v[ib[nn][5]][i+ib[nn][8]][j+ib[nn][7]]=r[nn]*vxy;
        }
      }
    }

    if(ibd[1][1]!=0) {
      for(nn=1;nn<=kd;nn++) {
        for(j=ibd[nn][3];j<=ibd[nn][4];j++) {
          for(i=ibd[nn][1];i<=ibd[nn][2];i++) {
            vx=v[ibd[nn][6]][i][j];
            vy=v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]];
            v[ibd[nn][6]][i][j]=
                              -rc[nn]*vy+(1.0+rc[nn])*vx;
            v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]]=
                    rc[nn]*vx+(1.0-rc[nn])*vy;
          }
        }
      }
    }

    for(nn=1;nn<=1;nn++) {
      for(j=ia[nn][3];j<=ia[nn][4];j++) {
        for(i=ia[1][1];i<=ia[2][2];i++) {
          if(i==ia[nn][1]+1) v[2][i][j]=vvv[0][j][1];
          if(i==(ia[2][2])) v[2][i+1][j]=vv[0][j][1];
          if(i>=ia[1][2]) c=rd[2];
          else c=rd[1];
          a=(v[1][i][j+1]+v[1][i][j]+v[2][i][j]+v[2][i+1][j]
              )*2./(c+4.0);
          v[5][i][j]=a-v[5][i][j];
          v[1][i][j]=a-v[1][i][j];
          v[2][i][j]=a-v[2][i][j];
          vy=a-v[1][i][j+1];
```

```
vx=a-v[2][i+1][j];
v[2][i+1][j]=v[4][i][j];
v[1][i][j+1]=v[3][i][j];
v[3][i][j]=vy;
v[4][i][j]=vx;
 if(i==(ia[2][2]-14)) {
   vv[15][j][1]=vv[15][j][2];
   vv[15][j][2]=vv[15][j][3];
   vv[15][j][3]=vv[15][j][4];
   vv[15][j][4]=vv[15][j][5];
   vv[15][j][5]=vv[15][j][6];
   vv[15][j][6]=vv[15][j][7];
   vv[15][j][7]=vv[15][j][8];
   vv[15][j][8]=vv[15][j][9];
   vv[15][j][9]=vv[15][j][10];
   vv[15][j][10]=vv[15][j][11];
   vv[15][j][11]=vv[15][j][12];
   vv[15][j][12]=vv[15][j][13];
   vv[15][j][13]=vv[15][j][14];
   vv[15][j][14]=vv[15][j][15];
   vv[15][j][15]=v[2][i][j];
}
 if(i==(ia[2][2]-13)) {
   vv[14][j][1]=vv[14][j][2];
   vv[14][j][2]=vv[14][j][3];
   vv[14][j][3]=vv[14][j][4];
   vv[14][j][4]=vv[14][j][5];
   vv[14][j][5]=vv[14][j][6];
   vv[14][j][6]=vv[14][j][7];
   vv[14][j][7]=vv[14][j][8];
   vv[14][j][8]=vv[14][j][9];
   vv[14][j][9]=vv[14][j][10];
   vv[14][j][10]=vv[14][j][11];
   vv[14][j][11]=vv[14][j][12];
   vv[14][j][12]=vv[14][j][13];
   vv[14][j][13]=vv[14][j][14];
   vv[14][j][14]=v[2][i][j];
}
 if(i==(ia[2][2]-12)) {
   vv[13][j][1]=vv[13][j][2];
   vv[13][j][2]=vv[13][j][3];
   vv[13][j][3]=vv[13][j][4];
   vv[13][j][4]=vv[13][j][5];
   vv[13][j][5]=vv[13][j][6];
```

```
      vv[13][j][6]=vv[13][j][7];
      vv[13][j][7]=vv[13][j][8];
      vv[13][j][8]=vv[13][j][9];
      vv[13][j][9]=vv[13][j][10];
      vv[13][j][10]=vv[13][j][11];
      vv[13][j][11]=vv[13][j][12];
      vv[13][j][12]=vv[13][j][13];
      vv[13][j][13]=v[2][i][j];
    }
    if(i==(ia[2][2]-11)) {
      vv[12][j][1]=vv[12][j][2];
      vv[12][j][2]=vv[12][j][3];
      vv[12][j][3]=vv[12][j][4];
      vv[12][j][4]=vv[12][j][5];
      vv[12][j][5]=vv[12][j][6];
      vv[12][j][6]=vv[12][j][7];
      vv[12][j][7]=vv[12][j][8];
      vv[12][j][8]=vv[12][j][9];
      vv[12][j][9]=vv[12][j][10];
      vv[12][j][10]=vv[12][j][11];
      vv[12][j][11]=vv[12][j][12];
      vv[12][j][12]=v[2][i][j];
    }
    if(i==(ia[2][2]-10)) {
      vv[11][j][1]=vv[11][j][2];
      vv[11][j][2]=vv[11][j][3];
      vv[11][j][3]=vv[11][j][4];
      vv[11][j][4]=vv[11][j][5];
      vv[11][j][5]=vv[11][j][6];
      vv[11][j][6]=vv[11][j][7];
      vv[11][j][7]=vv[11][j][8];
      vv[11][j][8]=vv[11][j][9];
      vv[11][j][9]=vv[11][j][10];
      vv[11][j][10]=vv[11][j][11];
      vv[11][j][11]=v[2][i][j];
    }
  if(i==(ia[2][2]-9)) {
      vv[10][j][1]=vv[10][j][2];
      vv[10][j][2]=vv[10][j][3];
      vv[10][j][3]=vv[10][j][4];
      vv[10][j][4]=vv[10][j][5];
      vv[10][j][5]=vv[10][j][6];
      vv[10][j][6]=vv[10][j][7];
      vv[10][j][7]=vv[10][j][8];
```

```
    vv[10][j][8]=vv[10][j][9];
    vv[10][j][9]=vv[10][j][10];
    vv[10][j][10]=v[2][i][j];
  }
  if(i==(ia[2][2]-8)) {
    vv[9][j][1]=vv[9][j][2];
    vv[9][j][2]=vv[9][j][3];
    vv[9][j][3]=vv[9][j][4];
    vv[9][j][4]=vv[9][j][5];
    vv[9][j][5]=vv[9][j][6];
    vv[9][j][6]=vv[9][j][7];
    vv[9][j][7]=vv[9][j][8];
    vv[9][j][8]=vv[9][j][9];
    vv[9][j][9]=v[2][i][j];
  }
  if(i==(ia[2][2]-7)) {
    vv[8][j][1]=vv[8][j][2];
    vv[8][j][2]=vv[8][j][3];
    vv[8][j][3]=vv[8][j][4];
    vv[8][j][4]=vv[8][j][5];
    vv[8][j][5]=vv[8][j][6];
    vv[8][j][6]=vv[8][j][7];
    vv[8][j][7]=vv[8][j][8];
    vv[8][j][8]=v[2][i][j];
  }
  if(i==(ia[2][2]-6)) {
    vv[7][j][1]=vv[7][j][2];
    vv[7][j][2]=vv[7][j][3];
    vv[7][j][3]=vv[7][j][4];
    vv[7][j][4]=vv[7][j][5];
    vv[7][j][5]=vv[7][j][6];
    vv[7][j][6]=vv[7][j][7];
    vv[7][j][7]=v[2][i][j];
  }
  if(i==(ia[2][2]-5)) {
    vv[6][j][1]=vv[6][j][2];
    vv[6][j][2]=vv[6][j][3];
    vv[6][j][3]=vv[6][j][4];
    vv[6][j][4]=vv[6][j][5];
    vv[6][j][5]=vv[6][j][6];
   vv[6][j][6]=v[2][i][j];
  }
  if(i==(ia[2][2]-4)) {
    vv[5][j][1]=vv[5][j][2];
```

```
    vv[5][j][2]=vv[5][j][3];
    vv[5][j][3]=vv[5][j][4];
    vv[5][j][4]=vv[5][j][5];
    vv[5][j][5]=v[2][i][j];
  }
  if(i==(ia[2][2]-3)) {
    vv[4][j][1]=vv[4][j][2];
    vv[4][j][2]=vv[4][j][3];
    vv[4][j][3]=vv[4][j][4];
    vv[4][j][4]=v[2][i][j];
  }
  if(i==(ia[2][2]-2)) {
    vv[3][j][1]=vv[3][j][2];
    vv[3][j][2]=vv[3][j][3];
    vv[3][j][3]=v[2][i][j];
  }
  if(i==(ia[2][2]-1)) {
    vv[2][j][1]=vv[2][j][2];
    vv[2][j][2]=v[2][i][j];
  }
  if(i==(ia[2][2])) {
    vv[1][j][1]=v[2][i][j];
    vv[0][j][1]=vv[0][j][2];
      vv[0][j][2]=8.*vv[1][j][1]-28.*vv[2][j][1]
        +56.*vv[3][j][1]-70.*vv[4][j][1]+
                    56.*vv[5][j][1]-28.*vv[6][j][1]
        +8.*vv[7][j][1]-1.*vv[8][j][1];
  }
  if(i==(ia[nn][1]+15)) {
    vvv[15][j][1]=vvv[15][j][2];
    vvv[15][j][2]=vvv[15][j][3];
    vvv[15][j][3]=vvv[15][j][4];
    vvv[15][j][4]=vvv[15][j][5];
    vvv[15][j][5]=vvv[15][j][6];
    vvv[15][j][6]=vvv[15][j][7];
    vvv[15][j][7]=vvv[15][j][8];
    vvv[15][j][8]=vvv[15][j][9];
    vvv[15][j][9]=vvv[15][j][10];
    vvv[15][j][10]=vvv[15][j][11];
    vvv[15][j][11]=vvv[15][j][12];
    vvv[15][j][12]=vvv[15][j][13];
    vvv[15][j][13]=vvv[15][j][14];
    vvv[15][j][14]=v[4][i][j];
  }
```

```
if(i==(ia[nn][1]+14)) {
  vvv[14][j][1]=vvv[14][j][2];
  vvv[14][j][2]=vvv[14][j][3];
  vvv[14][j][3]=vvv[14][j][4];
  vvv[14][j][4]=vvv[14][j][5];
  vvv[14][j][5]=vvv[14][j][6];
  vvv[14][j][6]=vvv[14][j][7];
  vvv[14][j][7]=vvv[14][j][8];
  vvv[14][j][8]=vvv[14][j][9];
  vvv[14][j][9]=vvv[14][j][10];
  vvv[14][j][10]=vvv[14][j][11];
  vvv[14][j][11]=vvv[14][j][12];
  vvv[14][j][12]=vvv[14][j][13];
  vvv[14][j][13]=v[4][i][j];
}
if(i==(ia[nn][1]+13)) {
  vvv[13][j][1]=vvv[13][j][2];
  vvv[13][j][2]=vvv[13][j][3];
  vvv[13][j][3]=vvv[13][j][4];
  vvv[13][j][4]=vvv[13][j][5];
  vvv[13][j][5]=vvv[13][j][6];
  vvv[13][j][6]=vvv[13][j][7];
  vvv[13][j][7]=vvv[13][j][8];
  vvv[13][j][8]=vvv[13][j][9];
  vvv[13][j][9]=vvv[13][j][10];
  vvv[13][j][10]=vvv[13][j][11];
  vvv[13][j][11]=vvv[13][j][12];
  vvv[13][j][12]=v[4][i][j];
}
if(i==(ia[nn][1]+12)) {
  vvv[12][j][1]=vvv[12][j][2];
  vvv[12][j][2]=vvv[12][j][3];
  vvv[12][j][3]=vvv[12][j][4];
  vvv[12][j][4]=vvv[12][j][5];
  vvv[12][j][5]=vvv[12][j][6];
  vvv[12][j][6]=vvv[12][j][7];
  vvv[12][j][7]=vvv[12][j][8];
  vvv[12][j][8]=vvv[12][j][9];
  vvv[12][j][9]=vvv[12][j][10];
  vvv[12][j][10]=vvv[12][j][11];
  vvv[12][j][11]=v[4][i][j];
}
if(i==(ia[nn][1]+11)) {
  vvv[11][j][1]=vvv[11][j][2];
```

```
    vvv[11][j][2]=vvv[11][j][3];
    vvv[11][j][3]=vvv[11][j][4];
    vvv[11][j][4]=vvv[11][j][5];
    vvv[11][j][5]=vvv[11][j][6];
    vvv[11][j][6]=vvv[11][j][7];
    vvv[11][j][7]=vvv[11][j][8];
    vvv[11][j][8]=vvv[11][j][9];
    vvv[11][j][9]=vvv[11][j][10];
    vvv[11][j][10]=v[4][i][j];
  }
  if(i==(ia[nn][1]+10)) {
    vvv[10][j][1]=vvv[10][j][2];
    vvv[10][j][2]=vvv[10][j][3];
    vvv[10][j][3]=vvv[10][j][4];
    vvv[10][j][4]=vvv[10][j][5];
    vvv[10][j][5]=vvv[10][j][6];
    vvv[10][j][6]=vvv[10][j][7];
    vvv[10][j][7]=vvv[10][j][8];
    vvv[10][j][8]=vvv[10][j][9];
    vvv[10][j][9]=v[4][i][j];
  }
  if(i==(ia[nn][1]+9)) {
    vvv[9][j][1]=vvv[9][j][2];
    vvv[9][j][2]=vvv[9][j][3];
    vvv[9][j][3]=vvv[9][j][4];
    vvv[9][j][4]=vvv[9][j][5];
    vvv[9][j][5]=vvv[9][j][6];
    vvv[9][j][6]=vvv[9][j][7];
    vvv[9][j][7]=vvv[9][j][8];
    vvv[9][j][8]=v[4][i][j];
  }
  if(i==(ia[nn][1]+8)) {
    vvv[8][j][1]=vvv[8][j][2];
    vvv[8][j][2]=vvv[8][j][3];
    vvv[8][j][3]=vvv[8][j][4];
    vvv[8][j][4]=vvv[8][j][5];
    vvv[8][j][5]=vvv[8][j][6];
    vvv[8][j][6]=vvv[8][j][7];
    vvv[8][j][7]=v[4][i][j];
  }
  if(i==(ia[nn][1]+7)) {
    vvv[7][j][1]=vvv[7][j][2];
    vvv[7][j][2]=vvv[7][j][3];
    vvv[7][j][3]=vvv[7][j][4];
```

```
            vvv[7][j][4]=vvv[7][j][5];
            vvv[7][j][5]=vvv[7][j][6];
            vvv[7][j][6]=v[4][i][j];
          }
          if(i==(ia[nn][1]+6)) {
            vvv[6][j][1]=vvv[6][j][2];
            vvv[6][j][2]=vvv[6][j][3];
            vvv[6][j][3]=vvv[6][j][4];
            vvv[6][j][4]=vvv[6][j][5];
           vvv[6][j][5]=v[4][i][j];
          }
          if(i==(ia[nn][1]+5)) {
            vvv[5][j][1]=vvv[5][j][2];
            vvv[5][j][2]=vvv[5][j][3];
            vvv[5][j][3]=vvv[5][j][4];
            vvv[5][j][4]=v[4][i][j];
          }
          if(i==(ia[nn][1]+4)) {
            vvv[4][j][1]=vvv[4][j][2];
            vvv[4][j][2]=vvv[4][j][3];
            vvv[4][j][3]=v[4][i][j];
          }
           if(i==(ia[nn][1]+3)) {
            vvv[3][j][1]=vvv[3][j][2];
            vvv[3][j][2]=v[4][i][j];
          }
          if(i==(ia[nn][1]+2)) {
            vvv[2][j][1]=v[4][i][j];
          }
          if(i==(ia[nn][1]+1)) {
           vvv[1][j][1]=v[4][i][j];
            vvv[0][j][1]=vvv[0][j][2];
            vvv[0][j][2]=4.*vvv[1][j][1]-6.*vvv[2][j][1]
                +4.*vvv[3][j][1]-1.*vvv[4][j][1];
           }
        }
      }
    }
  switch(l) {
    case 3 : eh[ic]=(v[1][io][jo]+v[2][io][jo]
              +v[3][io][jo]+v[4][io][jo]
                      )*2./(4.+yo); break;
    case 2 : eh[ic]=v[3][io][jo]-v[1][io][jo];break;
    case 1 : eh[ic]=v[4][io][jo]-v[2][io][jo];break;
```

```
    }
    printf("%8d %10.6lf\n",ic,eh[ic]);
    fprintf(time,"%8d %10.6lf\n",ic,eh[ic]);
   }
  return 0;
}

fourier()
{
  double r,ra,rb,t;
  double cs,u,uk;
  double ehre,ehim,ehmod;
  double d;
  npt=0;
  t=0.0;
  max=0;
  r=0.5*sqrt(1.0+t*t);
  if(t!=0.0) ra=6.283184*sqrt(-0.5*r);
  else ra=0.0;
  rb=6.283184*sqrt(0.5+r);
  d=d1;
  while(d<=d2) {
    ehre=0.0;
    ehim=0.0;
    uk=exp(-d*ra);
    u=uk;
    for(ic=1;ic<=ni;ic++) {
      cs=ic*rb*d;
      ehre=ehre+(eh[ic]*cos(cs)*uk);
      ehim=ehim-(eh[ic]*sin(cs)*uk);
      uk=uk*u;
    }
    ehmod=sqrt(ehre*ehre+ehim*ehim);
    npt=npt+1;
    data[npt][1]=d;
    data[npt][2]=ehmod;
    d=d+ds;
  }
  for(j=1;j<=npt;j++) {
    if(data[j][2]>max) {
      max=data[j][2];
      peak=data[j][1];
      pt=j;
```

```
    }
  }
  return 0;
}

correct()
{
  double a,p,ga,gb;
  char  ve;
  printf("Velocity error correction?(y/n)  ");
  gets(buf);
  sscanf(buf,"%c",&ve);
  if((ve=='y')||(ve=='Y')) {
    p=3.141592653*peak;
    ga=sqrt(2)*sin(p);
    gb=sqrt(1-a*ga);
    cf=p/atan(ga/gb);
    if(cf!=0) pcf=peak/cf;
  }
  if((ve=='n')||(ve=='N')) {
    cf=1/sqrt(2);
    pcf=peak/cf;
  }
  return 0;
}

curvefit()
{
  double ai,bmax;
  if((pt!=1)&&(pt!=jd)) {
    ptp=pt+1;
    ptm=pt-1;
    ai=((data[ptm][2]-max)*(data[ptm][1]-
        data[ptp][1])-(data[ptm][2]-data[ptp][2])*
          (data[ptm][1]-data[pt][1]));
    ai=ai/((data[ptm][1]-data[pt][1])*
         (data[pt][1]-data[ptp][1])*(data[ptm][1]-
              data[ptp][1]));
    bmax=(data[ptm][2]-max)/(data[ptm][1]-data[pt][1])
              -ai*(data[ptm][1]+data[pt][1]);
    peak= -bmax/(2*ai);
  }
  return 0;
}
```

```
results()
{
  if(cf==0) printf("cf=0\n");
  else {
    cf=1./cf;
    printf("  The velocity correction factor (Do/D) is %8.6lf\n\n",cf);
  }
  for(j=1;j<=npt;j++) {
    fprintf(tmf,"       %8.6lf                  %8.6lf\n"
          ,1265.466817*cf*data[j][1],
          data[j][2]);
  }
  return 0;
}

/*void excite()
{
  for(nn=1;nn<=ke;nn++){
    for(j=ie[nn][3];j<=ie[nn][4];j++){
      for(i=ie[nn][1];i<=ie[nn][2];i++){
        m=ie[nn][5];
        a=(ic-deltat)*(ic-deltat);
        va[nn]=exp(-a/2./tow/tow);
        while(m<=ie[nn][7]){
          v[m][i][j]=va[nn];
          m=m+ie[nn][6];
        }
        v[5][i][j]=va[nn];
      }
    }
  }
}*/
/*half sin*/
/*void excite()
{
  for(nn=1;nn<=ke;nn++) {
    for(j=ie[nn][3];j<=ie[nn][4];j++) {
      for(i=ie[nn][1];i<=ie[nn][2];i++) {
        m=ie[nn][5];
        va[nn] = sin(PI*(15.-(double)(j)+1.5)/30.);
        printf("v[nn]=%lf\t",va[nn]);
        while(m<=ie[nn][7]) {
          v[m][i][j]=va[nn];
```

```
            m=m+ie[nn][6];
          }
          v[5][i][j]=va[nn];
        }
      }
    }
}*/

/*SIN excitation*/
/*
void excite()
{
  for(nn=1;nn<=ke;nn++) {
    for(j=ie[nn][3];j<=ie[nn][4];j++) {
      for(i=ie[nn][1];i<=ie[nn][2];i++) {
        m=ie[nn][5];
        va[nn] = sin(PI*((double)(j)-1.5)/30.);
        while(m<=ie[nn][7]) {
          v[m][i][j]=va[nn];
          m=m+ie[nn][6];
        }
        v[5][i][j]=va[nn];
      }
    }
  }
}
*/
void excite()
{
  for(nn=1;nn<=ke;nn++){
    for(j=ie[nn][3];j<=ie[nn][4];j++){
      for(i=ie[nn][1];i<=ie[nn][2];i++){
        m=ie[nn][5];
        while(m<=ie[nn][7]){
          v[m][i][j]=va[nn];
          m=m+ie[nn][6];
        }
        v[5][i][j]=va[nn];
      }
    }
  }
}
```

Appendix B

Source code of Liao's Absorbing Boundary Condition in 3D-TLM

```
/* Liao's ABC in 3D-TLM. */
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define PI=3.141592653589793354626

FILE        *inf.*tmf,*time;
int         nx,ny,nz,it,io,jo,ko,ni,deltat;
int         kb,kc,kd,kee,khe;
double      v[16][24][14][14],Ratio,r[40];
double      rc[40],rd[40],vea[40],vha[40];
double      c,kka,kkb,ehre,ehim,d,tow,eh[10001];
int         ib[10][11],ibd[60][10],npt;
int         iee[40][15],ihe[40][15],ia[60][10];
char        header[80],buf[80];
int         lee=3,l,jd,j,m,i,k,ic,pt,ptp,ptm,nn,nnn,flag;
double      pcf,cf,d1,d2,ds,peak,cs,max,mmax,mmin,md,yo,data[300][3];

main()
{
  inf=fopen("3d-tlm1.inp","r");
  tmf=fopen("3d-tlm.out","wt");
  time=fopen("time.out","wt");
  readnxnynz();readbound();readdielbound();
  readcompbox();readeexcitation();readhexcitation();
  readfreq();fclose(inf);iterate();fourier();
  curvefit();correct(); results();
  fclose(tmf); fclose(time);
  return 0;
}

readnxnynz()
{
  fgets(header,80,inf);printf("\n%s",header);
  fgets(header,80,inf);printf("%s",header);
  fscanf(inf,"%d %d %d\n", &nx,&ny,&nz);
  printf("%4d %4d %4d\n", nx, ny,nz);
  return 0;
}

readbound()
{
  fgets(header,80,inf);printf("%s",header);
  fgets(header,80,inf);printf("%s",header);
```

```
  kb=0;
  do
  {  kb=kb+1;
     for(m=1;m<=6;m++)
     {
        fscanf(inf,"%d",&ib[kb][m]);
        printf("%4d",ib[kb][m]);
        if(m==6) printf("      ");
     }
     fscanf(inf,"%lf %d\n",&r[kb],&it);
     printf("\t%8.4lf\t%4d\n",r[kb],it);
  }while(it!=0);
  return 0;
}

readdielbound()
{
  kd=0;
  fgets(header,80,inf);printf("%s",header);
  fgets(header,80,inf);printf("%s",header);
  do
  {  kd=kd+1;
     for(m=1;m<=8;m++)
     {
        fscanf(inf,"%d",&ibd[kd][m]);
        printf("%4d",ibd[kd][m]);
        if(m==4) printf("      ");
     }
     fscanf(inf,"%lf %d\n",&rc[kd],&it);
     printf("\t%8.4lf\t%4d\n",rc[kd],it);
  }while(it!=0);
  return 0;
}

readcompbox()
{
  kc=0;
  fgets(header,80,inf);printf("%s",header);
  fgets(header,80,inf);printf("%s",header);
  do
  {  kc=kc+1;
     for(m=1;m<=6;m++)
     {
        fscanf(inf,"%d",&ia[kc][m]);
```

```
        printf("%4d",ia[kc][m]);
      }
      fscanf(inf,"%lf %d\n",&rd[kc],&it);
      printf("\t%8.4lf\t%4d\n",rd[kc],it);
    }while(it!=0);
    return 0;
}

readeexcitation()
{
    kee=0;
    fgets(header,80,inf);printf("%s",header);
    fgets(header,80,inf);printf("%s",header);
    do
    {  kee=kee+1;
       for(m=1;m<=9;m++)
       {
          fscanf(inf,"%d",&iee[kee][m]);
          printf("%4d",iee[kee][m]);
          if(m==6) printf("      ");
       }
       fscanf(inf,"%lf %d\n",&vea[kee],&it);
       printf("\t%8.4lf\t%4d\n",vea[kee],it);
    }while(it!=0);
    printf("Please input the type of excitation\n");
    printf(" (1: sinasoidal; 2: half-sinasoidal; 3: gaussian; 4: pulse):\n");
    gets(buf);
    sscanf(buf,"%d\n", &lee);
    printf("%4d\n",lee);
    return 0;
}

readhexcitation()
{
    khe=0;
    fgets(header,80,inf);printf("%s",header);
    fgets(header,80,inf);printf("%s",header);
    do
    {  khe=khe+1;
       for(m=1;m<=7;m++)
       {
          fscanf(inf,"%d",&ihe[khe][m]);
          printf("%4d",ihe[khe][m]);
          if(m==6) printf("      ");
```

```
    }
    fscanf(inf,"%if %d\n",&vha[khe],&it);
    printf("\t%8.4lf\t%4d\n",vha[khe],it);
  }while(it!=0);
  return 0;
}

readfreq()
{
  fgets(header,80,inf);fgets(header,80,inf);
  fscanf(inf,"%lf %lf %lf\n",&d1,&d2,&ds);
  fgets(header,80,inf);
  fscanf(inf,"\n%d %d %d %d %d %lf %d %lf %lf %lf %lf",
    &io,&jo,&ko,&l,&ni,&yo,&deltat,&tow);
  printf("Output point is(%4d,%4d,%4d) and l=%4d\n",io,jo,ko,l);
  printf("Number of iterations is %4d\n",ni);
  printf("Permittivity stub admittance is %8.4lf\n",yo);
  printf("   D1        D2          Step Size  deltat   tow\n");
  printf("%8.4lf %8.4lf %8.4lf yo=%8.4lf   %8.4lf\n",d1,d2,ds,yo,tow);
  printf("Finished reading input. Working hard on TLM now.\n");
  printf("Please be patient!\n");
  return 0;
}

iterate()
{
  double ve,vh,vv1[9][14][14][9],vvv1[9][14][14][9];
  double vv2[9][14][14][9],vvv2[9][14][14][9];
  void excite(),seriesx(),seriesy(),seriesz();
  void shuntx(),shunty(),shuntz();

    for(j=0;j<=ny;j++) {
      for(k=0;k<=nz;k++) {
        for(i=0;i<=nx;i++) {
          for(m=0;m<=15;m++) {
            v[m][i][j][k]=0.0;
          }
        }
        for(nn=0;nn<=8;nn++) {
          vv1[nn][j][k][nn]=0.0;
          vvv1[nn][j][k][nn]=0.0;
          vv2[nn][j][k][nn]=0.0;
          vvv2[nn][j][k][nn]=0.0;
        }
```

```
    }
  }
for(ic=1;ic<=ni;ic++) {
   Ratio=1.;
  switch(lee) {
    case 1 : if(ic==1) sine();break;
    case 2 : if(ic==1) halfsin();break;
    case 3 : gauss();break;
    case 4 : if(ic==1) impulse();break;
  }
  switch(l) {
    case 1 : eh[ic]=v[1][io][jo][ko];break;
    case 2 : eh[ic]=v[2][io][jo][ko];break;
    case 3 : eh[ic]=v[3][io][jo][ko];break;
    case 4 : eh[ic]=v[4][io][jo][ko];break;
    case 5 : eh[ic]=2.*(v[1][io][jo][ko]+v[2][io][jo][ko]
                +v[3][io][jo][ko]+v[4][io][jo][ko])/4.;break;
    case 6 : eh[ic]=2.*(v[5][io][jo][ko]+v[6][io][jo][ko]
                +v[7][io][jo][ko]+v[8][io][jo][ko])/4.;break;
    case 7 : eh[ic]=(v[9][io][jo][ko]+v[10][io][jo][ko]
              +v[11][io][jo][ko]+v[12][io][jo][ko])/2.;
  }
  printf("%8d %14.8lf\n",ic,eh[ic]);
  fprintf(time,"%8d %14.8lf\n",ic,eh[ic]);
  if(ic==1) excite();

  if(ibd[1][1]!=0) {
    for(nn=1;nn<=kd;nn++) {
      for(j=ibd[nn][3];j<=ibd[nn][4];j++) {
        for(i=ibd[nn][1];i<=ibd[nn][2];i++) {
          vx=v[ibd[nn][6]][i][j];
          vy=v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]];
          v[ibd[nn][6]][i][j]=(-rc[nn]*vy)+(1.0+rc[nn])*vx;
          v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]]=
                                  rc[nn]*vx+(1.0-rc[nn])*vy;
        }
      }
    }
  }

  for(nn=1;nn<=kc;nn++) {
    for(k=ia[nn][5]+1;k<=ia[nn][6]-1;k++) {
      for(j=ia[nn][3]+1;j<=ia[nn][4]-1;j++) {
        for(i=ia[nn][1];i<=ia[nn][2];i++) {
```

```
                if(i!=ia[nn][2]) seriesx();
                if(i!=ia[nn][2]) seriesy();
                if(i!=ia[nn][1]) seriesz();
            }
        }
    }
}

for(nn=1;nn<=kb;nn++) {
  for(k=ib[nn][5];k<=ib[nn][6];k++) {
    for(j=ib[nn][3];j<=ib[nn][4];j++) {
      for(i=ib[nn][1];i<=ib[nn][2];i++) {

          if(i==ib[nn][1]&&i==ib[nn][2]) {
            seriesz();
            v[10][i][j][k]= r[nn]*v[10][i][j][k];
            v[12][i+1][j][k]= r[nn]*v[12][i+1][j][k];
            v[6][i][j][k]= r[nn]*v[6][i][j][k];
            v[8][i+1][j][k]= r[nn]*v[8][i+1][j][k];
          }
          if(j==ib[nn][3]&&j==ib[nn][4]){
            seriesx();
           v[3][i][j][k]=r[nn]*v[3][i][j][k];
            v[1][i][j+1][k]=r[nn]*v[1][i][j+1][k];
            v[11][i][j][k]=r[nn]*v[11][i][j][k];
            v[9][i][j+1][k]=r[nn]*v[9][i][j+1][k];
          }
          if(k==ib[nn][5]&&k==ib[nn][6]){
            seriesy();
            v[4][i][j][k]= r[nn]*v[4][i][j][k];
            v[2][i][j][k+1]= r[nn]*v[2][i][j][k+1];
            v[7][i][j][k]= r[nn]*v[7][i][j][k];
            v[5][i][j][k+1]= r[nn]*v[5][i][j][k+1];
          }
        }
      }
    }
  }

for(nn=1;nn<=kc;nn++) {
  for(k=ia[nn][5];k<=ia[nn][6];k++) {
    for(j=ia[nn][3];j<=ia[nn][4];j++) {

        vvv1[1][j][k][1]=v[8][ia[nn][1]+2][j][k];
```

```
vvv1[2][j][k][1]=vvv1[2][j][k][2];
vvv1[2][j][k][2]=v[8][ia[nn][1]+3][j][k];

vvv1[3][j][k][1]=vvv1[3][j][k][2];
vvv1[3][j][k][2]=vvv1[3][j][k][3];
vvv1[3][j][k][3]=v[8][ia[nn][1]+4][j][k];

vvv1[4][j][k][1]=vvv1[4][j][k][2];
vvv1[4][j][k][2]=vvv1[4][j][k][3];
vvv1[4][j][k][3]=vvv1[4][j][k][4];
vvv1[4][j][k][4]=v[8][ia[nn][1]+5][j][k];

vvv1[5][j][k][1]=vvv1[5][j][k][2];
vvv1[5][j][k][2]=vvv1[5][j][k][3];
vvv1[5][j][k][3]=vvv1[5][j][k][4];
vvv1[5][j][k][4]=vvv1[5][j][k][5];
vvv1[5][j][k][5]=v[8][ia[nn][1]+6][j][k];

vvv1[6][j][k][1]=vvv1[6][j][k][2];
vvv1[6][j][k][2]=vvv1[6][j][k][3];
vvv1[6][j][k][3]=vvv1[6][j][k][4];
vvv1[6][j][k][4]=vvv1[6][j][k][5];
vvv1[6][j][k][5]=vvv1[6][j][k][6];
vvv1[6][j][k][6]=v[8][ia[nn][1]+7][j][k];

vvv1[7][j][k][1]=vvv1[7][j][k][2];
vvv1[7][j][k][2]=vvv1[7][j][k][3];
vvv1[7][j][k][3]=vvv1[7][j][k][4];
vvv1[7][j][k][4]=vvv1[7][j][k][5];
vvv1[7][j][k][5]=vvv1[7][j][k][6];
vvv1[7][j][k][6]=vvv1[7][j][k][7];
vvv1[7][j][k][7]=v[8][ia[nn][1]+8][j][k];

vvv1[0][j][k][0]=vvv1[0][j][k][1];
vvv1[0][j][k][1]=vvv1[0][j][k][2];
vvv1[0][j][k][2]=vvv1[0][j][k][3];
vvv1[0][j][k][3]=vvv1[0][j][k][4];
vvv1[0][j][k][4]=                1.  *vvv1[1][j][k][1]
        -0.*vvv1[2][j][k][1]+0.*vvv1[3][j][k][1]
        -0.*vvv1[4][j][k][1]+0.  *vvv1[5][j][k][1]
        -0.  *vvv1[6][j][k][1]+0.  *vvv1[7][j][k][1];
v[8][ia[nn][1]+1][j][k]=vvv1[0][j][k][3]*Ratio;
```

```
vvv2[1][j][k][1]=v[12][ia[nn][1]+2][j][k];

vvv2[2][j][k][1]=vvv2[2][j][k][2];
vvv2[2][j][k][2]=v[12][ia[nn][1]+3][j][k];

vvv2[3][j][k][1]=vvv2[3][j][k][2];
vvv2[3][j][k][2]=vvv2[3][j][k][3];
vvv2[3][j][k][3]=v[12][ia[nn][1]+4][j][k];

vvv2[4][j][k][1]=vvv2[4][j][k][2];
vvv2[4][j][k][2]=vvv2[4][j][k][3];
vvv2[4][j][k][3]=vvv2[4][j][k][4];
vvv2[4][j][k][4]=v[12][ia[nn][1]+5][j][k];

vvv2[5][j][k][1]=vvv2[5][j][k][2];
vvv2[5][j][k][2]=vvv2[5][j][k][3];
vvv2[5][j][k][3]=vvv2[5][j][k][4];
vvv2[5][j][k][4]=vvv2[5][j][k][5];
vvv2[5][j][k][5]=v[12][ia[nn][1]+6][j][k];

vvv2[6][j][k][1]=vvv2[6][j][k][2];
vvv2[6][j][k][2]=vvv2[6][j][k][3];
vvv2[6][j][k][3]=vvv2[6][j][k][4];
vvv2[6][j][k][4]=vvv2[6][j][k][5];
vvv2[6][j][k][5]=vvv2[6][j][k][6];
vvv2[6][j][k][6]=v[12][ia[nn][1]+7][j][k];

vvv2[7][j][k][1]=vvv2[7][j][k][2];
vvv2[7][j][k][2]=vvv2[7][j][k][3];
vvv2[7][j][k][3]=vvv2[7][j][k][4];
vvv2[7][j][k][4]=vvv2[7][j][k][5];
vvv2[7][j][k][5]=vvv2[7][j][k][6];
vvv2[7][j][k][6]=vvv2[7][j][k][7];
vvv2[7][j][k][7]=v[12][ia[nn][1]+8][j][k];

 vvv2[0][j][k][3]=vvv2[0][j][k][4];
 vvv2[0][j][k][4]=                    1. *vvv2[1][j][k][1]
          -0.*vvv2[2][j][k][1]+0.*vvv2[3][j][k][1]
          -0.*vvv2[4][j][k][1]+0.  *vvv2[5][j][k][1]
          -0.  *vvv2[6][j][k][1]+0.  *vvv2[7][j][k][1];
v[12][ia[nn][1]+1][j][k]=vvv2[0][j][k][3]*Ratio;

vv1[1][j][k][1]=v[6][ia[nn][2]-1][j][k];
```

```
vv1[2][j][k][1]=vv1[2][j][k][2];
vv1[2][j][k][2]=v[6][ia[nn][2]-2][j][k];

vv1[3][j][k][1]=vv1[3][j][k][2];
vv1[3][j][k][2]=vv1[3][j][k][3];
vv1[3][j][k][3]=v[6][ia[nn][2]-3][j][k];

vv1[4][j][k][1]=vv1[4][j][k][2];
vv1[4][j][k][2]=vv1[4][j][k][3];
vv1[4][j][k][3]=vv1[4][j][k][4];
vv1[4][j][k][4]=v[6][ia[nn][2]-4][j][k];

vv1[5][j][k][1]=vv1[5][j][k][2];
vv1[5][j][k][2]=vv1[5][j][k][3];
vv1[5][j][k][3]=vv1[5][j][k][4];
vv1[5][j][k][4]=vv1[5][j][k][5];
vv1[5][j][k][5]=v[6][ia[nn][2]-5][j][k];

vv1[6][j][k][1]=vv1[6][j][k][2];
vv1[6][j][k][2]=vv1[6][j][k][3];
vv1[6][j][k][3]=vv1[6][j][k][4];
vv1[6][j][k][4]=vv1[6][j][k][5];
vv1[6][j][k][5]=vv1[6][j][k][6];
vv1[6][j][k][6]=v[6][ia[nn][2]-6][j][k];

vv1[7][j][k][1]=vv1[7][j][k][2];
vv1[7][j][k][2]=vv1[7][j][k][3];
vv1[7][j][k][3]=vv1[7][j][k][4];
vv1[7][j][k][4]=vv1[7][j][k][5];
vv1[7][j][k][5]=vv1[7][j][k][6];
vv1[7][j][k][6]=vv1[7][j][k][7];
vv1[7][j][k][7]=v[6][ia[nn][2]-7][j][k];

vv1[0][i][k][3]=vv1[0][j][k][4];
vv1[0][j][k][4]=                    1. *vv1[1][j][k][1]
        -0.*vv1[2][j][k][1]+0.*vv1[3][j][k][1]
        -0.*vv1[4][j][k][1]+0.  *vv1[5][j][k][1]
        -0.  *vv1[6][j][k][1]+0.  *vv1[7][j][k][1];
v[6][ia[nn][2]][j][k]=vv1[0][j][k][3]*Ratio;

vv2[1][j][k][1]=v[10][ia[nn][2]-1][j][k];

vv2[2][j][k][1]=vv2[2][j][k][2];
vv2[2][j][k][2]=v[10][ia[nn][2]-2][j][k];
```

```
        vv2[3][j][k][1]=vv2[3][j][k][2];
        vv2[3][j][k][2]=vv2[3][j][k][3];
        vv2[3][j][k][3]=v[10][ia[nn][2]-3][j][k];

        vv2[4][j][k][1]=vv2[4][j][k][2];
        vv2[4][j][k][2]=vv2[4][j][k][3];
        vv2[4][j][k][3]=vv2[4][j][k][4];
        vv2[4][j][k][4]=v[10][ia[nn][2]-4][j][k];

        vv2[5][j][k][1]=vv2[5][j][k][2];
        vv2[5][j][k][2]=vv2[5][j][k][3];
        vv2[5][j][k][3]=vv2[5][j][k][4];
        vv2[5][j][k][4]=vv2[5][j][k][5];
        vv2[5][j][k][5]=v[10][ia[nn][2]-5][j][k];

        vv2[6][j][k][1]=vv2[6][j][k][2];
        vv2[6][j][k][2]=vv2[6][j][k][3];
        vv2[6][j][k][3]=vv2[6][j][k][4];
        vv2[6][j][k][4]=vv2[6][j][k][5];
        vv2[6][j][k][5]=vv2[6][j][k][6];
        vv2[6][j][k][6]=v[10][ia[nn][2]-6][j][k];

        vv2[7][j][k][1]=vv2[7][j][k][2];
        vv2[7][j][k][2]=vv2[7][j][k][3];
        vv2[7][j][k][3]=vv2[7][j][k][4];
        vv2[7][j][k][4]=vv2[7][j][k][5];
        vv2[7][j][k][5]=vv2[7][j][k][6];
        vv2[7][j][k][6]=vv2[7][j][k][7];
        vv2[7][j][k][7]=v[10][ia[nn][2]-7][j][k];

         vv2[0][j][k][3]=vv2[0][j][k][4];
         vv2[0][j][k][4]=                    1. *vv2[1][j][k][1]
                        -0.*vv2[2][j][k][1]+0.*vv2[3][j][k][1]
                        -0.*vv2[4][j][k][1]+0.  *vv2[5][j][k][1]
                        -0.  *vv2[6][j][k][1]+0.  *vv2[7][j][k][1];
  v[10][ia[nn][2]][j][k]=vv2[0][j][k][3]*Ratio;
     }
   }
}

for(nn=1;nn<=kc;nn++) {
   for(k=ia[nn][5];k<=ia[nn][6];k++) {
      for(j=ia[nn][3];j<=ia[nn][4];j++) {
```

```
            for(i=ia[nn][1];i<=ia[nn][2];i++) {
                if(i!=ia[nn][1]) shuntx();
                if(i!=ia[nn][1]) shunty();
                shuntz();
            }
          }
        }
      }
    }
}

void seriesz()
{
  double a;
            a=(-v[7][i][j][k]+v[11][i][j][k]+v[5][i][j][k+1]
              -v[9][i][j+1][k])/2.;
            v[7][i][j][k]=v[7][i][j][k]+a;
            v[11][i][j][k]=v[11][i][j][k]-a;
            v[5][i][j][k+1]=v[5][i][j][k+1]-a;
            v[9][i][j+1][k]=v[9][i][j+1][k]+a;
}

void seriesx()
{
  double a;
            a=(-v[4][i][j][k]+v[10][i][j][k]+v[2][i][j][k+1]
              -v[12][i+1][j][k])/2.;
            v[4][i][j][k]=v[4][i][j][k]+a;
            v[10][i][j][k]=v[10][i][j][k]-a;
            v[2][i][j][k+1]=v[2][i][j][k+1]-a;
            v[12][i+1][j][k]=v[12][i+1][j][k]+a;
}

void seriesy()
{
  double a;
            a=(-v[3][i][j][k]+v[6][i][j][k]+v[1][i][j+1][k]
              -v[8][i+1][j][k])/2.;
            v[3][i][j][k]=v[3][i][j][k]+a;
            v[6][i][j][k]=v[6][i][j][k]-a;
            v[1][i][j+1][k]=v[1][i][j+1][k]-a;
            v[8][i+1][j][k]=v[8][i+1][j][k]+a;
}
```

```
void shuntz()
{
  double a;

              a=(v[1][i][j][k]+v[2][i][j][k]+v[3][i][j][k]
                +v[4][i][j][k])/2.;
              v[1][i][j][k]=a-v[1][i][j][k];
              v[2][i][j][k]=a-v[2][i][j][k];
              v[3][i][j][k]=a-v[3][i][j][k];
              v[4][i][j][k]=a-v[4][i][j][k];
}

void shuntx()
{
  double a;
              a=(v[5][i][j][k]+v[6][i][j][k]+v[7][i][j][k]
                +v[8][i][j][k])/2.;
              v[5][i][j][k]=a-v[5][i][j][k];
              v[6][i][j][k]=a-v[6][i][j][k];
              v[7][i][j][k]=a-v[7][i][j][k];
              v[8][i][j][k]=a-v[8][i][j][k];
}

void shunty()
{
  double a;
              a=(v[9][i][j][k]+v[10][i][j][k]+v[11][i][j][k]
                +v[12][i][j][k])/2.;
              v[9][i][j][k]=a-v[9][i][j][k];
              v[10][i][j][k]=a-v[10][i][j][k];
              v[11][i][j][k]=a-v[11][i][j][k];
              v[12][i][j][k]=a-v[12][i][j][k];
}


fourier()
{
  double r,re,rb,t;
  double cs,u,uk;
  double ehre,ehim,ehmod;
  double d;
  npt=0;
  t=0.0;
  max=0;
```

```
r=0.5*sqrt(1.0+t*t);
if(t!=0.0) ra=6.283184*sqrt(-0.5*r);
else ra=0.0;
rb=6.283184*sqrt(0.5+r);
d=d1;
while(d<=d2)
{
   ehre=0.0;
   ehim=0.0;
   uk=exp(-d*ra);
   u=uk;
   for(ic=1;ic<=ni;ic++)
   {
      cs=ic*rb*d;
      ehre=ehre+(eh[ic]*cos(cs)*uk);
      ehim=ehim-(eh[ic]*sin(cs)*uk);
      uk=uk*u;
   }
   ehmod=sqrt(ehre*ehre+ehim*ehim);
   npt=npt+1;
   data[npt][1]=d;
   data[npt][2]=ehmod;
   d=d+ds;
 }
 for(j=1;j<=npt;j++)
 {
   if(data[j][2]>max)
   {
      max=data[j][2];
      peak=data[j][1];
      pt=j;
   }
 }
 return 0;
}

correct()
{
   double a,p,ga,gb;
   char  ve;
   printf("Velocity error correction?(y/n)  ");
   gets(buf);
   sscanf(buf,"%c",&ve);
   if((ve=='y')||(ve=='Y'))
```

```
  {
    p=3.141592653*peak;
    ga=sqrt(2)*sin(p);
    gb=sqrt(1-ga*ga);
    cf=p/atan(ga/gb);
    if(cf!=0) pcf=peak/cf;
  }
  if((ve=='n')||(ve=='N'))
  {
    cf=1/sqrt(2);
    pcf=peak/cf;
  }
  return 0;
}

curvefit()
{
  double ai,bmax;
  if((pt!=1)&&(pt!=jd))
  {
    ptp=pt+1;
    ptm=pt-1;
    ai=((data[ptm][2]-max)*(data[ptm][1]-data[ptp][1])-
        (data[ptm][2]-data[ptp][2])*(data[ptm][1]-data[pt][1]));
    ai=ai/((data[ptm][1]-data[pt][1])*
        (data[pt][1]-data[ptp][1])*(data[ptm][1]-data[ptp][1]));
    bmax=(data[ptm][2]-max)/
        (data[ptm][1]-data[pt][1])-ai*(data[ptm][1]+data[pt][1]);
    peak=(-bmax)/(2*ai);
  }
  return 0;
}


results()
{
  if(cf==0) printf("cf=0\n");
  else
  {
  cf=1./cf;
  printf("  The velocity correction factor (Do/D) is %8.4lf\n\n",cf);
  }
  for(j=1;j<=npt;j++)
  {
```

```
    fprintf(tmf,"        %8.4lf                    %8.4lf\n"
          ,data[j][1],data[j][2]);
  }
  return 0;
}

/*void excite()
{
  for(nn=1;nn<=ke;nn++) {
    for(j=ie[nn][3];j<=ie[nn][4];j++) {
      for(i=ie[nn][1];i<=ie[nn][2];i++) {
        m=ie[nn][5];
        va[nn] = sin(kka*.0002371*(15.-(double)(j)+1.5));
         if(j==2) va[nn]=c*cos(kkb*.0002371*.5);
        printf("v[nn]=%lf",va[nn]);
        while(m<=ie[nn][7]) {
          v[m][i][j]=va[nn];
          m=m+ie[nn][6];
        }
        v[5][i][j]=va[nn];
      }
    }
  }
}*/
halfsin()
{
double a;

    for(nn=1;nn<=kee;nn++) {
      for(k=iee[nn][5];k<=iee[nn][6];k++) {
        for(j=iee[nn][3];j<=iee[nn][4];j++) {
          for(i=iee[nn][1];i<=iee[nn][2];i++) {
            m=iee[nn][7];
            a = vea[nn]*sin(PI*(((double)(iee[nn][4])-
                (double)(j))*2.+1.)/4./((double)(iee[nn][4])
                -(float)(iee[nn][3])+1.));
            printf("v[nn]=%lf\t",a);
            while(m<=iee[nn][9]) {
              v[m][i][j][k]=a;
              m=m+iee[nn][8];
            }
          }
        }
      }
```

```
    }

}
/*sin excitation*/
sine()
{

    for(nn=1;nn<=kee;nn++) {
      for(k=iee[nn][5];k<=iee[nn][6];k++) {
        for(j=iee[nn][3];j<=iee[nn][4];j++) {
          for(i=iee[nn][1];i<=iee[nn][2];i++) {
            m=iee[nn][7];
            vea[nn] = sin(PI*((double)(j)-1.5)/30.);
            while(m<=iee[nn][9]) {
              v[m][i][j][k]=vea[nn];
              m=m+iee[nn][8];
            }
          }
        }
      }
    }

}

gauss()
{
  double ve;

  for(nn=1;nn<=kee;nn++) {
    ve=vea[nn]*exp(-(double)(ic-deltat)*
                   (double)(ic-deltat)/2./tow/tow);
    for(k=iee[nn][5];k<=iee[nn][6];k++) {
      for(j=iee[nn][3];j<=iee[nn][4];j++) {
        for(i=iee[nn][1];i<=iee[nn][2];i++) {
          m=iee[nn][7];
          while(m<=iee[nn][9]) {
            v[m][i][j][k]=ve;
            m=m+iee[nn][8];
          }
        }
      }
    }
  }
}
```

```
impulse()
{

    for(nn=1;nn<=kee;nn++) {
      for(k=iee[nn][5];k<=iee[nn][6];k++) {
        for(j=iee[nn][3];j<=iee[nn][4];j++) {
          for(i=iee[nn][1];i<=iee[nn][2];i++) {
            m=iee[nn][7];
            while(m<=iee[nn][9]) {
              v[m][i][j][k]=vea[nn];
              m=m+iee[nn][8];
            }
          }
        }
      }
    }

}
```

Appendix C

Source code of the Direct Optimized Dissipation Absorbing Boundary Condition

```
/* Direct Optimized Dissipation ABC. */
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<malloc.h>

#define     ITMAX 200
#define     TOL 2.0e-4
#define     GOLD 1.618034
#define     CGOLD 0.3819660
#define     GLIMIT 100.0
#define     TINY 1.0E-20
#define     ZEPS 1.0e-10
#define     MAX(a,b) ((a) > (b) ? (a) : (b))
#define     SIGN(a,b) ((b)>0.0 ? fabs(a) : -fabs(a))
#define     SHFT(a,b,c,d) (a)=(b); (b)=(c); (c)=(d);
static      float sqrarg;
#define     SQR(a) (sqrarg=(a),sqrarg*sqrarg)

FILE        *init,*inio;
float       func(),tlmrun();
void        powell();
void        linmin();
float       brent();
void        mnbrak(),tlmread();
int         ntotal,itnum=0,n_open=0;

int main()
{
  float *p,**xi,**matrix();
  float ifret=0.0,*fret,*vector(),ob;
  char inbuf[130],cf;
  int  i,j,n,ini=0,*iter;

  iter = &ini;
  fret = &ifret;
  init=fopen("init.inp","r");
  inio=fopen("init.out","w");
  *iter= 1;
  printf("How amny section you want?\n");
  gets(inbuf);
  sscanf(inbuf,"%d",&n);
  printf("Do optimization (y/n)?\n");
  scanf("%c",&cf);
```

```
  printf("cf=%c\n",cf);
  ntotal=n+1;
  xi=matrix(1,n+1,1,n+1);
  p=vector(1,n+1);
  for (i=1;i<=n+1;i++) {
     for (j=1;j<=n+1;j++) {
        xi[i][j]=0.;
     }
     fscanf(init,"%f",&p[i]);
     xi[i][i]=1.;
  }
  fclose(init);
  tlmread(p,n+1);
  if(cf=='n'||cf=='N') {
    ob=tlmrun(p); printf("obj=%f\n",ob);exit();
  }
  powell(p,xi,n+1,.1,iter,fret,func,cf);
  fprintf(inio,"stop");
  for(i=1;i<=ntotal;i++) {
     fprintf(inio,"  %10.6f",p[i]);
  }
  fclose(inio);
}

void powell(p,xi,n,ftol,iter,fret,func,cf)
float *p,*fret,**xi,ftol,(*func)();
int n,*iter;
char cf;
{
   int i,ibig,j;
   float t,fptt,fp,del;
   float *pt,*ptt,*xit,*vector();
   void linmin(),nrerror(),free_vector();

   pt=vector(1,n);
   ptt=vector(1,n);
   xit=vector(1,n);
   *fret=(*func)(p);
   for(j=i;j<=n;j++) pt[j]=p[j];
   for(*iter=1;;(*iter)++) {
      printf("*iter=%4d (*iter)=%4d\n",*iter,(*iter));
      fp=(*fret);
      ibig=0;
      del=0.0;
```

```
    for(i=1;i<=n;i++) {
      for(j=1;j<=n;j++) xit[j]=xi[j][i];
      fptt=(*fret);
      linmin(p,xit,n,fret,func);
      if(fabs(fptt-(*fret))>del) {
        del=fabs(fptt-(*fret));
        ibig=i;
      }
    }
    if(2.0*fabs(fp-(*fret))<=ftol*(fabs(fp)+fabs(*fret))) {
      free_vector(xit,1,n);
      free_vector(ptt,1,n);
      free_vector(pt,1,n);
      return;
    }
    if(*iter==ITMAX) nrerror("Too many iterations in routine POWELL");
    for(j=1;j<=n;j++) {
      ptt[j]=2.0*p[j]-pt[j];
      xit[j]=p[j]-pt[j];
      pt[j]=p[j];
    }
    fptt=(*func)(ptt);
    if(fptt<fp) {
      t=2.0*(fp-2.0*(*fret)+fptt)*SQR(fp-(*fret)-del)-del*SQR(fp-fptt);
      if(t<0.0) {
        linmin(p,xit,n,fret,func);
        for(j=1;j<=n;j++) xi[j][ibig]=xit[j];
      }
    }
  }
}

int ncom=0;
float *pcom=0,*xicom=0,(*nrfunc)();

void linmin(p,xi,n,fret,func)
float p[],xi[],*fret,(*func)();
int n;
{
  int j;
  float xx,xmin,fx,fb,fa,bx,ax;
  float brent(),f1dim(),*vector();
  void mnbrak(),free_vector();
```

```
 ncom=n;
 pcom=vector(1,n);
 xicom=vector(1,n);
 nrfunc=func;
 for(j=1;j<=n;j++) {
    pcom[j]=p[j];
    xicom[j]=xi[j];
 }
 ax=0.0;
 xx=1.0;
 mnbrak(&ax,&xx,&bx,&fa,&fx,&fb,f1dim);
 *fret=brent(ax,xx,bx,f1dim,TOL,&xmin);
 for(j=1;j<=n;j++) {
    xi[j] *= xmin;
    p[j] += xi[j];
 }
 free_vector(xicom,1,n);
 free_vector(pcom,1,n);
}

extern int ncom;
extern float *pcom,*xicom,(*nrfunc)();

float f1dim(x)
float x;
{
   int j;
   float f,*xt,*vector();
   void free_vector();

   xt=vector(1,ncom);
   for(j=1;j<=ncom;j++) xt[j]=pcom[j]+x*xicom[j];
   f=(*nrfunc)(xt);
   free_vector(xt,1,ncom);
   return f;
}

void mnbrak(ax,bx,cx,fa,fb,fc,func)
float *ax,*bx,*cx,*fa,*fb,*fc;
float (*func)();
{
   float ulim,u,r,q,fu,dum;

   *fa=(*func)(*ax);
```

```
*fb=(*func)(*bx);
if (*fb>*fa){
  SHFT(dum,*ax,*bx,dum)
  SHFT(dum,*fb,*fa,dum)
}
*cx=(*bx)+GOLD*(*bx-*ax);
*fc=(*func)(*cx);
while (*fb>*fc) {
   r=(*bx-*ax)*(*fb-*fc);
   q=(*bx-*cx)*(*fb-*fa);
   u=(*bx)-((*bx-*cx)*q-(*bx-*ax)*r)/(2.0*SIGN(MAX(fabs(q-r),TINY),q-r));
   ulim=(*bx)+GLIMIT*(*cx-*bx);
   if((*bx-u)*(u-*cx)>0.0) {
      fu=(*func)(u);
      if(fu<*fc){
         *ax=(*bx);
         *bx=u;
         *fa=(*fb);
         *fb=fu;
         return;
      } else if (fu>*fb) {
         *cx=u;
         *fc=fu;
         return;
      }
      u=(*cx)+GOLD*(*cx-*bx);
      fu=(*func)(u);
    }else if ((*cx-u)*(u-ulim) >0.0) {
      fu=(*func)(u);
      if(fu<*fc){
         SHFT(*bx,*cx,u,*cx+GOLD*(*cx-*bx))
         SHFT(*fb,*fc,fu,(*func)(u))
      }
    }else if ((u-ulim)*(ulim-*cx)>=0.0) {
      u=ulim;
       fu=(*func)(u);
    } else {
      u=(*cx)+GOLD*(*cx-*bx);
       fu=(*func)(u);
    }
    SHFT(*ax,*bx,*cx,u)
    SHFT(*fa,*fb,*fc,fu)
  }
}
```

```
float brent(ax,bx,cx,f,tol,xmin)
float ax,bx,cx,tol,*xmin;
float (*f)();
{
  int iter;
  float a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
  float e=0.0;
  void nrerror();

  a=((ax<cx)?ax:cx);
  b=((ax>cx)?ax:cx);
  x=w=v=bx;
  fw=fv=fx=(*f)(x);
  for(iter=1;iter<=ITMAX;iter++){
    xm=0.5*(a+b);
    tol2=2.0*(tol1=tol*fabs(x)+ZEPS);
    if(fabs(x-xm)<=(tol2-0.5*(b-a))){
      *xmin=x;
      return fx;
    }
    if(fabs(e)>tol1){
      r=(x-w)*(fx-fv);
      q=(x-v)*(fx-fw);
      p=(x-v)*q-(x-w)*r;
      q=2.0*(q-r);
      if(q>0.0) p= -p;
      q=fabs(q);
      etemp=e;
      e=d;
    if(fabs(p)>=fabs(0.5*q*etemp)||p<=q*(a-x)||p>=q*(b-x))
      d=CGOLD*(e=(x>=xm?a-x:b-x));
      else{
        d=p/q;
        u=x+d;
        if(u-a<tol2||b-u<toi2) d=SIGN(tol1,xm-x);
      }
    } else{
      d=CGOLD*(e=(x>=xm?a-x:b-x));
    }
    u=(fabs(d)>=tol1?x+d:x+SIGN(tol1,d));
    fu=(*f)( );
    if(fu<=fx){
      if(u>=x) a=x;else b=x;
      SHFT(v,w,x,u)
```

```
        SHFT(fv,fw,fx,fu)
      } else{
        if(u<x) a=u;else b=u;
        if(fu<=fw||w==x){
          v=w;
          w=u;
          fv=fw;
          fw=fu;
        } else if (fu<=fv||v==x || v==w){
          v=u;
          fv=fu;
        }
      }
    }
    nrerror("Too many iteration in BRENT");
    *xmin=x;
    return fx;
}

void free_vector(v,n1,nh)
float *v;
int n1,nh;
{
    free((char*)(v+n1));
}


float func(xx)
float *xx;
{
    float ffx;
    int i,flag=0;

    itnum+=1;
    if(itnum>=500){
    itnum=0;
    n_open+=1;
    fclose(inio);
    inio=fopen("init.out","w");
    }
    for(i=1;i<=ntotal;i++) {
      if(i<ntotal && xx[i]<0) flag=1;
      if(i==ntotal && fabs(xx[i])>1) flag=1;
      printf(" %7.5e",xx[i]);
```

```
    fprintf(inio,"  %10.6e",xx[i]);
  }
  if(flag==0) ffx=tlmrun(xx);
  else ffx=1.e10;
  printf("\n itnum=%d n_open=%d   fx=%10.6e\n",itnum,n_open,ffx);
  fprintf(inio,"\n  itnum=%d n_open=%d  fx=%10.6e\n",itnum,n_open,ffx);
  return   ffx;
}


void nrerror(error_text)
char error_text[];
{
  void exit();

  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}

float *vector(nl,nh)
int nl,nh;
{
  float *v;

  v=(float *)malloc((unsigned)(nh-nl+1)*sizeof(float));
  if(!v) nrerror("allocation failure in vector()");
  return v-nl;
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  float **m;

  m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
  if(!m) nrerror("allocation failure 1 in matrix()");
  m -= nrl;
  for(i=nrl;i<=nrh;i++) {
    m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
    if(!m[i]) nrerror("allocation failure 2 in matrix()");
    m[i] -= ncl;
```

```
  }
  return m;
}
```

```
/* This is a 2D-TLM subroutine. */
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define PI=3.1415926

void        tlmread();
void        readcompbox();
float       tlmrun(),results();
FILE        *inf,*in1,*outf,*readoutf,*dbout;
int         nx,ny;
int         io,it,jo,ni,deltat;
int         kb,kc,kd,ke;
float       v[6][151][13];
float       r[30];
float       rc[30],rd[50];
float       va[15];
float       ehre,ehim,d,tow;
float       eh[1000];
int         ib[6][10];
int         ibd[6][10];
int         ie[10][15];
int         ia[50][10];
char        header[80];
int         jd,ic,pt,ptp,ptm,nt,nf,optype;
float       pcf,cf,d1,d2,ds;
float       peak,cs,max,mmax,mmin,md,yo,assum;
```

```
int             npt;
float           data[300][4],*vector();
char            out[3][70];
char            buf[80];



void tlmread(float *p ,int nnn)
{
  inf=fopen("tlm_inho.inp","r");
  readoutf=fopen("tlmread.out","w");
  in1=fopen("bmark.out","r");
  nt=nnn-1;
  readnxny();
  readbound();
  readdielbound();
  readcompbox(p);
  readexcitation();
  readfreq();
  readdata1();
  fclose(inf);
  fclose(readoutf);
  fclose(in1);
}

float tlmrun(xx)
float *xx;
{
  float obj;
  int i;
  outf=fopen("tlm.out","wt");
  dbout=fopen("db.out","wt");
  for(i=1;i<=nt;i++) {
    rd[kc-i+1]=xx[i];
    rd[i+1]=xx[i];
  }
  r[2]=xx[nt+1];
  r[4]=xx[nt+1];
  iterate();
  fourier();
  curvefit();
  correct();
```

```
  obj=results();
  fclose(outf);
  fclose(dbout);
  return obj;
}

readdata1()
{
  int i;
  for(i=1;i<=nf;i++)
  {
    fscanf(in1,"%f %f",&assum,&data[i][3]);
    printf("freq=%10.6f   mag1=%10.6f\n",assum,data[i][3]);
   }
  return 0;
}

readnxny()
{
  fgets(header,80,inf);
  fprintf(readoutf,"\n%s",header);
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  fscanf(inf,"%d %d\n", &nx,&ny);
  fprintf(readoutf,"%4d%4d\n", nx, ny);
  return 0;
}

readbound()
{
  int m;
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  kb=0;
  do
  { kb=kb+1;
    for(m=1;m<=8;m++)
    {
       fscanf(inf,"%d",&ib[kb][m]);
       fprintf(readoutf,"%4d",ib[kb][m]);
       if(m==4) fprintf(readoutf,"     ");
    }
```

```
    fscanf(inf,"%f %d\n",&r[kb],&it);
    fprintf(readoutf,"\t%8.4f\t%4d\n",r[kb],it);
  }while(it!=0);
  return 0;
}

readdielbound()
{
  int m;
  kd=0;
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  do
  {  kd=kd+1;
    for(m=1;m<=8;m++)
    {
       fscanf(inf,"%d",&ibd[kd][m]);
       fprintf(readoutf,"%4d",ibd[kd][m]);
       if(m==4) fprintf(readoutf,"     ");
    }
    fscanf(inf,"%f %d\n",&rc[kd],&it);
    fprintf(readoutf,"\t%8.4f\t%4d\n",rc[kd],it);
  }while(it!=0);
  return 0;
}

void readcompbox(p)
float *p;
{
  int m,i;
  kc=0;
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  do
  {  kc=kc+1;
    for(m=1;m<=4;m++)
    {
       fscanf(inf,"%d",&ia[kc][m]);
    }
    ia[kc+nt][1] =ia[kc][1]+nt+1;
```

```
    ia[kc+nt][2] =ia[kc][2]-nt;
    ia[kc+nt][3] =ia[kc][3];
    ia[kc+nt][4] =ia[kc][4];
    fscanf(inf,"%f %d\n",&rd[kc+nt],&it);
  }while(it!=0);
  for(i=1;i<=nt;i++) {
    ia[i][1]=i+1;
    ia[i][2]=i+1;
    if(i==1) ia[i][1]=1;
    ia[kc+2*nt-i+1][1]= ia[kc+nt][2]+nt-i+1;
    ia[kc+2*nt-i+1][2]= ia[kc+nt][2]+nt-i+1;
    ia[i][3]=ia[kc+nt][3];
    ia[kc+2*nt-i+1][3]=ia[kc+nt][3];
    ia[i][4]=ia[kc+nt][4];
    ia[kc+2*nt-i+1][4]=ia[kc+nt][4];
    rd[i]=p[i];
    rd[kc+2*nt-i+1]=p[i];
  }
  for(i=1;i<=(2*nt+kc);i++) {
    fprintf(readoutf,"%4d %4d %4d %4d ",ia[i][1],ia[i][2],ia[i][3],
          ia[i][4]);
    fprintf(readoutf,"\t%8.4f\t%4d\n",rd[i],it);
  }
  kc=kc+2*nt;
}

readexcitation()
{
  int m;
  ke=0;
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  fgets(header,80,inf);
  fprintf(readoutf,"%s",header);
  do
  { ke=ke+1;
    for(m=1;m<=7;m++)
    {
      fscanf(inf,"%d",&ie[ke][m]);
      fprintf(readoutf,"%4d",ie[ke][m]);
      printf("%4d",ie[ke][m]);
      if(m==4) fprintf(readoutf,"    ");
    }
    fscanf(inf,"%f %d\n",&va[ke],&it);
```

```
    fprintf(readoutf,"\t%8.4f\t%4d\n",va[ke],it);
    printf("\t%8.4f\t%4d\n",va[ke],it);
  }while(it!=0);
  return 0;
}

readfreq()
{
  fgets(header,80,inf);
  fgets(header,80,inf);
  fscanf(inf,"%f %f %f %d\n",&d1,&d2,&ds,&nf);
  fgets(header,80,inf);
  fscanf(inf,"\n%d %d %d %d %f%d %f",&io,&jo,&optype,&ni,&yo,&deltat,&tow);
  fprintf(readoutf,"Output point is(%4d,%4d) and l=%4d\n",io,jo,optype);
  fprintf(readoutf,"Number of iterations is %4d\n",ni);
  fprintf(readoutf,"Permittivity stub admittance is %6.4f\n",yo);
  fprintf(readoutf,"  D1       D2          Step Size  deltat   tow\n");
  fprintf(readoutf,"%8.6f    %8.6f%8.6 %4d %8.6f\n",d1,d2,ds,deltat,tow);
  return 0;
}

iterate()
{
  float a,vx,vy,vxy,rrr;
  int i,j,nn,m;

  for(j=1;j<=ny;j++){
    for(i=1;i<=nx;i++){
      for(m=1;m<=5;m++){ v[m][i][j]=0.0;}
    }
  }
  for(nn=1;nn<=ke;nn++) {
    for(j=ie[nn][3];j<=ie[nn][4];j++) {
      for(i=ie[nn][1];i<=ie[nn][2];i++) {
        m=ie[nn][5];
        va[nn] = sin(PI*(12.-(float)(j)+1.5)/24.);
        printf("v[nn]=%lf\t",va[nn]);
        while(m<=ie[nn][7]) {
          v[m][i][j]=va[nn];
          m=m+ie[nn][6];
        }
        v[5][i][j]=va[nn];
      }
    }
```

```
}
mmax=0.;
mmin=0.;
for(ic=1;ic<=ni;ic++)
{
  for(nn=1;nn<=kb;nn++)
  {
    for(j=ib[nn][3];j<=ib[nn][4];j++)
    {
      for(i=ib[nn][1];i<=ib[nn][2];i++)
      {
        vxy=v[ib[nn][6]][i][j];
        v[ib[nn][6]][i][j]=r[nn]*
          v[ib[nn][5]][i+ib[nn][8]][j+ib[nn][7]];
        v[ib[nn][5]][i+ib[nn][8]][j+ib[nn][7]]=r[nn]*vxy;
      }
    }
  }

  if(ibd[1][1]!=0)
  {
    for(nn=1;nn<=kd;nn++)
    {
      for(j=ibd[nn][3];j<=ibd[nn][4];j++)
      {
        for(i=ibd[nn][1];i<=ibd[nn][2];i++)
        {
          vx=v[ibd[nn][6]][i][j];
          vy=v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]];
        v[ibd[nn][6]][i][j]=(-rc[nn]*vy)+(1.0+rc[nn])*vx;
        v[ibd[nn][5]][i+ibd[nn][8]][j+ibd[nn][7]]=rc[nn]*vx+
          (1.0-rc[nn])*vy;
        }
      }
    }
  }
    for(j=1;j<=13;j++){
      for(i=1;i<=120;i++){
        for(nn=1;nn<=kc;nn++){
          if(i>=ia[nn][1]&&i<=ia[nn][2]) rrr=rd[nn];
        }
        a=(v[1][i][j+1]+v[1][i][j]+v[2][i][j]+v[2][i+1][j])
        *2.0/(rrr+4.0);
        v[1][i][j]=a-v[1][i][j];
```

```
            v[2][i][j]=a-v[2][i][j];
            vy=a-v[1][i][j+1];
            vx=a-v[2][i+1][j];
            v[2][i+1][j]=v[4][i][j];
            v[1][i][j+1]=v[3][i][j];
            v[3][i][j]=vy;
            v[4][i][j]=vx;
         }
      }
      switch(optype)
      {
         case 1 : eh[ic]=v[1][io][jo];break;
         case 2 : eh[ic]=v[2][io][jo];break;
         case 3 : eh[ic]=v[3][io][jo];break;
         case 4 : eh[ic]=v[4][io][jo];break;
         case 5 : eh[ic]=v[3][io][jo]-v[1][io][jo];break;
         case 6 : eh[ic]=v[4][io][jo]-v[2][io][jo];break;
         case 7 : eh[ic]=0.5*(v[1][io][jo]+v[2][io][jo]
                  +v[3][io][jo]+v[4][io][jo]);break;
      }
   }
}



fourier()
{
   float r,ra,rb,t;
   float cs,u,uk;
   float ehre,ehim,ehmod;
   float d;
   int j;
   npt=0;
   t=0.0;
   max=0;
   r=0.5*sqrt(1.0+t*t);
   if(t!=0.0) ra=6.283184*sqrt(-0.5*r);
   else ra=0.0;
   rb=6.283184*sqrt(0.5+r);
   d=d1;
   while(d<=d2)
   {
      ehre=0.0;
```

```
    ehim=0.0;
    uk=exp(-d*ra);
    u=uk;
    for(ic=1;ic<=ni;ic++)
    {
       cs=ic*rb*d;
       ehre=ehre+(eh[ic]*cos(cs)*uk);
       ehim=ehim-(eh[ic]*sin(cs)*uk);
       uk=uk*u;
    }
    ehmod=sqrt(ehre*ehre+ehim*ehim);
    npt=npt+1;
    data[npt][1]=d;
    data[npt][2]=ehmod;
    d=d+ds;
  }
  for(j=1;j<=npt;j++)
  {
    if(data[j][2]>max)
    {
       max=data[j][2];
       peak=data[j][1];
       pt=j;
    }
  }
  return 0;
}

correct()
{
   float a,p,ga,gb;
   char  ve;
   ve='y';
   if((ve=='y')||(ve=='Y'))
   {
      p=3.141592653*peak;
      ga=sqrt(2)*sin(p);
      gb=sqrt(1-ga*ga);
      cf=p/atan(ga/gb);
      if(cf!=0) pcf=peak/cf;
   }
   if((ve=='n')||(ve=='N'))
   {
      cf=1/sqrt(2);
```

```
    pcf=peak/cf;
  }
  return 0;
}

curvefit()
{
  float ai,bmax;
  if((pt!=1)&&(pt!=jd))
  {
    ptp=pt+1;
    ptm=pt-1;
    ai=((data[ptm][2]-max)*(data[ptm][1]-data[ptp][1])-
        (data[ptm][2]-data[ptp][2])*(data[ptm][1]-data[pt][1]));
    ai=ai/((data[ptm][1]-data[pt][1])*
        (data[pt][1]-data[ptp][1])*(data[ptm][1]-data[ptp][1]));
    bmax=(data[ptm][2]-max)/
        (data[ptm][1]-data[pt][1])-ai*(data[ptm][1]+data[pt][1]);
    peak= -bmax/(2*ai);
  }
  return 0;
}

float results()
{
  float obj= 0;
  int i;
  for(i=1;i<=nf;i++)
  {
    data[i][1]=1431.712268*data[i][1];/*deltl=.7112*/
    eh[i]=fabs((data[i][3]-data[i][2])/data[i][3]);
    if(eh[i]==0.) eh[i]= -100.;
    else eh[i]=20*log10(eh[i]);
    if(eh[i]>=-30.) obj=obj+eh[i]+30;
    fprintf(dbout,"%10.6f   %10.6f\n",data[i][1],eh[i]);
    printf("%10.6f   %10.6f\n",data[i][1],eh[i]);
  }
  printf("obj=%f\n",obj);
  return obj;
}
```