Implementation of a Full Frequency-Dependent Transmission Line Model within the Framework of the (OVNI) Real-Time Power System Simulator

by

Jiun-Tsyr Jack Chang

B.A.Sc., The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE in

THE FACULTY OF GRADUATE STUDIES DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THE UNIVERSITY OF BRITISH COLUMBIA

November 2004

© Jiun-Tsyr Jack Chang, 2004
ABSTRACT

This thesis presents the methodology to implement a full frequency-dependent transmission line model in UBC’s OVNI (Object Virtual Network Integrator) real-time power system simulator.

OVNI utilizes an object-oriented approach to represent and implement its element models. Object-oriented programming permits a flexible, reliable, and expandable solution to the simulation program. In particular, OVNI represents each element model as a class inheriting common characteristics and properties from a parent class of basic elements. Each element object interacts with the network solver, the core, only by exchanging such parameters as external node voltages, external node names, the equivalent conductance matrix and the equivalent history source vector through some well-defined member functions. Model developers only needs to correctly and efficiently implement these member functions to successfully incorporate the model with the core. This clean-cut abstraction between element models and the core provides model developers with complete freedom and independence while designing, implementing, and upgrading models of different electric properties and complexities.

The transmission line model implemented in this thesis is a phase-domain full frequency-dependent model (zLine). This model was developed in Ph.D projects by Castellanos [15] and Yu [16]. The model is suitable for time-domain EMTP (Electromagnetic Transient Program) simulations within OVNI’s real-time framework. Z-line is accurate, efficient, numerically stable and strongly suited for multi-circuit asymmetrical line configurations. The model divides the line length into a number of small segments and separates the wave propagating in each segment into a constant ideal-line section and a frequency-dependent loss section. A numerically stable curve fitting routine was modified to synthesize elements of the loss matrix so that the equivalent time-domain model for the loss section can be formulated with an integration rule.
The implementation of zLine in this thesis work is divided into three major tasks: pre­
processing and initialization of model parameters, computation of equivalent conductance
matrix, and update of equivalent history sources. The pre-processing task involves three
subtasks. First of all, it requires the execution of mtLine, a program that generates line
parameter matrices from geometrical conductor configurations. Moreover, it requires the
execution of a modified fitting routine that synthesizes the frequency-dependent loss
matrix with a series of rational functions. Finally a series of I/O routines are required to
organize and generate the input files necessary to run the programs mentioned above. An
ANSI C compatible function is chosen to integrate the execution of all those routines
under the control of the simulator.

The update of history sources and computation of conductance matrices take
advantage of the abstraction offered by object-oriented programming between the line
and the modelling segments. Each history source update and matrix computation is
performed first at the segment level for each zLine segment and then accumulated at the
line level to form the overall history vector and conductance matrix of the line. Internal
nodes are hidden away by applying the node-hiding technique to reduce network
complexity and further streamline solution efficiency.

A comprehensive set of test cases are run and compared with proven results and exact
models in Microtran to ensure the correctness and accuracy of the implementation
methodology.

zLine’s accuracy and absolute numerical stability for any asymmetrical line
configurations render it an ideal candidate as the first frequency-dependent line model to
be included with OVNI.
# TABLE OF CONTENTS

ABSTRACT............................................................................................................. ii
TABLE OF CONTENTS......................................................................................... iv
LIST OF FIGURES................................................................................................. vii
ACKNOWLEDGEMENTS......................................................................................... xi

CHAPTER 1 Introduction......................................................................................... 1
  1.1 Background and Motivation of OVNI......................................................... 1
  1.2 Background and Motivation of the Full Frequency-Dependent Transmission
     Line Model (zLine)...................................................................................... 4
  1.3 Implementation Methodology.................................................................... 9
  1.4 Thesis Organization................................................................................... 12

CHAPTER 2 Full Frequency-Dependent Transmission Line Model Review............ 13
  2.1 Transmission Line Theory Review............................................................ 14
  2.2 Modelling of Ideal Line Section............................................................... 17
  2.3 Modelling of Lumped Loss Section........................................................... 19
     2.3.1 Matrix Synthesis and Fitting............................................................... 19
     2.3.2 Discrete Time Model of the Lumped Loss Section......................... 20
  2.4 Phase-Domain Transmission Line Model.................................................. 23
  2.5 Segment Length and Maximum Frequency of Interest............................. 24

CHAPTER 3 Current OVNI Architecture and Classes........................................... 27
  3.1 Network Partition and Solution................................................................. 28
     3.1.1 Network Segmentation – MATE......................................................... 30
     3.1.2 Node Hiding Technique................................................................. 33
  3.2 Classes in OVNI....................................................................................... 37
     3.2.1 The elm_t Class and Interaction with OVNI.................................. 37
     3.2.2 The blk_t Class............................................................................. 40
     3.2.3 The lnk_t Class............................................................................ 41
     3.2.4 The src_t and isrc_t Class............................................................ 43
LIST OF FIGURES

Figure 2.1 Decoupling of Basic Propagation Effect ........................................... 13
Figure 2.2 Representation of Distributed Line in Frequency Domain .................... 14
Figure 2.3 Multiphase Line Model For Ideal Section ............................................. 17
Figure 2.4 The Model for Lumped Loss Section ..................................................... 21
Figure 2.5 zLine Segment Model ........................................................................... 23
Figure 2.6 The Complete zLine Model .................................................................. 24
Figure 2.7 Maximum Segment Length VS Maximum Frequency of Interest ............ 26
Figure 3.1 Simulation Algorithms for Standard EMTP Type Simulators .................. 28
Figure 3.2 a) Decoupling Effect In Transmission Line model. b) Corresponding [G] Matrix ........................................................................................................... 29
Figure 3.3 Two-Tier Network Segmentation Scheme .............................................. 30
Figure 3.4 zLine Segment Model Highlighting External and Internal Nodes ............. 34
Figure 3.5 Header File for elm_t ............................................................................. 38
Figure 3.6 Header File for ind_t .............................................................................. 39
Figure 3.7 Header File for MATE Blocks ............................................................... 41
Figure 3.8 Model of MATE Link in OVNI .............................................................. 41
Figure 3.9 Header File for MATE Links ................................................................. 42
Figure 3.10 Header File for Sources in OVNI ........................................................ 43
Figure 3.11 Header File for Current Sources in OVNI .......................................... 44
Figure 3.12 Solution Algorithm of OVNI ............................................................... 47
Figure 4.1 Input File Format for the zLine Model ................................................... 50
Figure 4.2 Input File for mtLine ............................................................................. 53
Figure 4.3 Sample mtLine output for the 3 Phase Line of Figure 4.1 and 4.2 .............. 55
Figure 4.4 Input File For the Fitting Routine ........................................................ 56
Figure 4.5 Number of Fitting Blocks for Accurate Synthesis of [Z^{loss}(\omega)] .......... 57
Figure 4.6 Output File for the Fitting Routine ....................................................... 59
Figure 4.7 zLine Model: a) Line Level b)Segment Level ........................................ 63
Figure 4.8 Constructing Conductance Matrix for zLine Model ............................... 65
Figure 4.9 Segment Conductance Matrix ............................................................. 66
Figure 5.13  Steady State and Initial Transients at Receiving End of Phase 2............. 95
Figure 5.14  Circuit Diagram for Double-Conductor 1-km Open Circuit Test.............. 95
Figure 5.15  Steady State and Initial Transients at Receiving End of Phase 1............. 96
Figure 5.16  Steady State and Initial Transients at Receiving End of Phase 2............. 96
Figure 5.17  Circuit Diagram for Double-Conductor 100-km Short Circuit Test............ 97
Figure 5.18  Steady State and Initial Transients at Receiving End of Phase 1............. 97
Figure 5.19  Steady State and Initial Transients at Receiving End of Phase 2............. 98
Figure 5.20  Circuit Diagram for Double-Conductor 100-km Open Circuit Test............ 98
Figure 5.21  Steady State and Initial Transients at Receiving End of Phase 1............. 98
Figure 5.22  Steady State and Initial Transients at Receiving End of Phase 2............. 99
Figure 5.23  Line Configuration and Conductor Parameters for Triple-Conductor

Line.................................................. 100
Figure 5.24  Circuit Diagram for three-phase 100-km Short Circuit Test with π-

Circuit.................................................. 101
Figure 5.25  Steady-state Response for Receiving End of Phase 1 (left) and 2 (right)...... 102
Figure 5.26  Steady-state Response for Receiving End of Phase 3.......................... 102
Figure 5.27  Circuit Diagram for three-phase 100-km Open Circuit Test with π-

Circuit.................................................. 103
Figure 5.28  Steady-state Response for Receiving End of Phase 1 (left) and 2 (right)...... 103
Figure 5.29  Steady-state Response for Receiving End of Phase 3.......................... 103
Figure 5.30  Transposition Scheme for a Three-Phase Line................................. 104
Figure 5.31  Circuit Diagram for 3-phase 100-km Short Circuit Test with

Transposition............................................. 105
Figure 5.32  Initial Transients at Receiving End of Phase 1 (left) and Phase 2 (right)... 105
Figure 5.33  Initial Transients at Receiving End of Phase 3................................. 105
Figure 5.34  Circuit Diagram for 3-phase 100-km Open Circuit Test with

Transposition............................................. 106
Figure 5.35  Initial Transients at Receiving End of Phase 1 (left) and Phase 2 (right)... 106
Figure 5.36  Initial Transients at Receiving End of Phase 3................................. 106
ACKNOWLEDGEMENT

Many people have contributed to the completion of this thesis. I wish to offer my sincere gratitude to all people who have helped me throughout this two-and-half year span.

First and foremost, I would like to express my most gratitude to my thesis supervisor: Dr. Jose R. Marti and Dr. Luis Linares. I would like to thank Dr. Marti for his expertise in the area of transmission line modelling and network simulation, his timely professional advice, unlimited technical guidance and financial support. Following his guidance, I have enriched my technical knowledge base and had the opportunity to work alongside with so many other outstanding colleagues.

I would like to present my sincere gratitude to Dr. Linares for his friendship and expertise in software design and implementation. His assistance and tutoring in program implementation make me confident in writing codes compared to two years ago. His detailed introductions to OVNI and the node hiding technique during meetings last summer prepared me for the development of this thesis work.

Special thanks to all former and current members of UBC power group members for their friendship, support, cooperation and assistance. Among all, I would like to thank Alan Xu for his selfless assistance and valuable advice in power system and mathematics related issues. I particularly respect his broad knowledge base and generosity in offering help. I would also like to thank Peng Zhang for his time and valuable advice on code implementation related issues. Without his help, I would spend far more time debugging.

I would like to offer a tremendous amount of appreciations and respect to Tom De Rybel, a fellow master student in the lab, who takes charge of so many minor but crucial daily business from hardware setup and maintenance to furniture and equipment purchasing. Special thanks to him who has made my life easier and smoother in the lab.
To all the professors and staff in the Department of Electrical Engineering who had taught and assisted me constantly.

Special thanks to my girl friend and best friend Joanne Chang for her patience and constant encouragement. Thanks her for accompanying me when I encountered technical difficulties and emotional downturns during the preparation and writing stage of this thesis. Her company will be forever remembered.

Finally, a lot of thanks to my family members: my parents and my brother. Thanks to my father for his constant support, both mentally and financially. Thanks to my mom for her love and caring and my brother for his knowledge in network and computer maintenance. Thank you all.
CHAPTER 1
INTRODUCTION

This thesis presents the methodology to implement a full frequency-dependent transmission line model (zLine) in the current OVNI framework. This chapter unveils by giving history and motivation of OVNI. Then the motivation and background of the transmission line model to be implemented are provided. Finally the implementation strategy is briefly mentioned to foreshadow the work in this thesis.

1.1 Background and Motivation of OVNI

Network Simulation, as its name suggests, is the process of solving a network for unknowns along the time axis. For a power system the solution entails solving the voltages for each node and the currents flowing between nodes at each point in time. For small systems this equals to solving several coupled differential equations altogether. For large power systems with thousands of nodes, this task is formidable. The EMTP (electromagnetic transient program), with its elegant element models and digital solution methodology, was developed by Dr. Hermman Dommel to streamline the solution process of large and complex power systems [1]. Since its introduction in the late 60’s EMTP has found worldwide recognition and replaced TNAs (Transient Network Analyzers) in power system transient studies. EMTP has surpassed its analog counterpart in the following areas:

- EMTP related programs such as Microtran can be operated using low cost and commercially available off-the-shelf personal computers. This in contrast to a classic TNA, which requires a physical representation of the network, resulting in an expensive and bulky installation.

- Very elegant and sophisticated element models can be developed for EMTP to represent physical characteristics of the elements. Especially, distributed properties of
transmission lines can be modelled accurately in EMTP. Better and improved models can be designed by individuals as old ones become obsolete.

In spite of the advantages so mentioned, the EMTP is unable to simulate a system in "real-time" – the kind of simulation environment that requires the simulator to solve the system "fast enough" so that the equipment being simulated would have the impression that it is indeed connected to a real system. Real-time simulation provides observers and system analysts a substitute system in which all real-world characteristics and performance of the system are accurately represented and mimicked. This avoids running simulation with real systems that are high cost, infeasible and sometimes catastrophic [2].

The need and benefits of real-time simulation have motivated UBC’s Power Group to develop a full-power, versatile and portable real-time digital power system simulator combining the advantages of both TNA and EMTP. In addition to a hardware design approach that most researchers chose to follow, as in the arrangements of DSP boards or transputers [2,5,6] to mimic the topology of power systems, UBC Power Group has been pursuing a software-centred approach. The first EMTP based real-time network simulator was developed by J. R. Martí and L. R. Linares back in 1993 [5,6].

The proposed simulator was written in ANSI C and was completely general to simulate any power system topology. The simulator takes advantage of the decoupling effect of the transmission line model in the EMTP and introduces "one-tier-segmentation" to divide the system into several independent "blocks". The network $[G]$ matrix then becomes a block diagonal matrix. Jesus Calvino [23] implemented a real-time hardware solution based on a single PC workstation.

Jorge A. Hollman followed suit using the RTNS as the core solver to develop a distributed cluster of personal computers as the hardware solution engine (RTDNS) [7,8,9]. This development is based on the belief that simulation with single computer can only deal with a limited number of network nodes. Solving beyond the limit would slow
down the solution process beyond the real-time deadline. Employing the idea of parallel computing, the PC cluster scheme divides the solution tasks into several computer nodes, therefore reducing solution time within the real-time deadline. Most importantly, RTDNS exhibits constant communication overheads between solver nodes independent of the number of PC solvers added to the array. UBC’s research group is the first to incorporate the inexpensive PC solution with the real-time simulation framework.

Using the software solution algorithm of RTNS and the hardware PC cluster scheme of RTDNS as the basic building block, members of UBC Power Group have been continuously researching and advancing the concept of real-time full-size power system simulator to what we have known today as OVNI. The current architecture of OVNI, used for the implementation work in this thesis, will be introduced in a later chapter.
1.2 Background and Motivation of the Full Frequency-Dependent Transmission Line Model (zLine)

Transmission lines are one of the key components in the operation of power networks. They usually extend over hundreds of kilometres linking several substations and are therefore very vulnerable to a wide variety of phenomena such as short circuits and lighting surges. To monitor and study the behaviour of those malicious causes, accurate transmission line models are required to work together with a power system simulator. Several transmission line models have been developed and implemented in simulators such as EMTP and Microtran [1,10], but none has yet been incorporated with OVNI. As the result, it is of utmost importance to implement a reliable and accurate line model under any conductor configurations in the real-time simulation framework.

Since majority of the lines are multiphase for efficient power delivery, many researchers have dedicated their time and efforts to formulating efficient multiphase line models and solutions. One of the most frequently used modelling technique, currently employed by Microtran, is modal domain decomposition. Modal domain modelling requires the use of a transformation matrix to decouple the phase domain model into several independent modal domain models so that each modal domain line model can be treated and solved as a single-phase line. This method of modal decomposition is extremely valuable in transmission line modelling as each decoupled mode contains only single transmission delay as opposed to the mixing delays found in the phase domain due to the coupling between conductors. Multiphase line models of Microtran assume a constant and real transformation matrix, which produces reasonably accurate results most of the time. However, the transformation matrix is actually complex and frequency-dependent in reality, and the constant and real assumption would produce less accurate simulation results particularly in situations of strong conductor asymmetry in multi-conductor transmission line systems. The elegance and accuracy of model transformation mentioned previously are lost.
Several alternatives have been proposed to solve the problems of frequency-dependent transformation matrix. Most prominent ones involved either solving the line directly in phase-domain or synthesizing the frequency-dependent transformation matrix into rational functions. Examples of above line models will be briefly introduced, gradually leading to the zLine model to be implemented in this thesis.

• Constant-Parameter Line Model
The cpLine model [1, 18] of Microtran acquired its name by assuming that the R, L and C line parameters are exact for a single frequency. It is a distributed parameter line model accounting for the travelling delay from the sending to receiving end. In the case of multiphase line modelling, a real and constant transformation matrix is used to decouple a n-phase line into n single-phase lines in the modal domain. Line parameters for a given frequency can be obtained by running the mtLine program [17].

• Π-Circuit
The multiphase, coupled Π-circuit [1, 18] is the simplest representation for transmission lines. A transmission line is represented by a lumped series impedance and a shunt admittance split in half and inserted at both ends of the impedance. Depending on the accuracy desired, there are two versions of Π-circuit available. An approximated nominal Π-circuit assumes the total impedance and admittance to be equal to per-unit-length quantities multiplying by the length. It is most accurate for short line and low frequency conditions. The equivalent Π-circuit applies hyperbolic corrections to the impedance and admittance parameters making it more appropriate for long line simulations. Both models, however, are only valid for one-frequency-at-a-time steady-state simulations.

• fdLine Model
The fdLine model [10, 18, 20] of Microtran successfully models the frequency dependency of line parameters by applying the Bode fitting technique to synthesize the modal characteristic impedance, $Z_c(\omega)$, and propagation function, $e^{i\gamma_\omega}$, into
rational functions connected in cascade. The frequency-domain circuits are converted to time-domain through inverse Laplace transform and discretization with an integration rule. Although fdLine can represent the frequency dependent nature of line parameters very accurately and efficiently, it does not account for the fact that the transformation matrix relating phase and modal domain is also frequency varying. fdLine, like cpLine, still assumes constant and real transformation matrix, which makes simulation under extreme asymmetrical conductor configurations less ideal and accurate.

• Full Frequency-dependent Qcable Model
   The Qcable model developed in [11] is described as a “full” frequency-dependent model in the sense that it solves the problem of the frequency-dependent modal transformation matrix by synthesizing the element of this matrix with rational functions. Because the modal transformation matrix is virtually column arrays of eigenvectors for \([Y][Z]\), any scaling would make the resulting vector a valid eigenvector. Inappropriate scaling could make the synthesis process difficult, and sometimes it may lead to unstable synthesis functions. Moreover, not only the transformation matrix needs to be synthesized, but the characteristic impedance matrix and the propagation matrix all require accurate frequency domain synthesis. This results in a large amount of operations for time domain recursive convolutions.

• Direct Phase-Domain Model
   The most straightforward solution to overcome the problem of modal transformation matrix is to not use it at all. Nguyen, Dommel and J. Martí developed a phase domain transmission line model [12] where elements of the propagation and characteristic impedance matrices are directly synthesized with rational functions in phase domain. Since the line is solved and parameters synthesized in phase domain, the use of transformation matrix is completely avoided. Moreover, because the synthesis of the transformation matrix is prevented, there are fewer time-domain convolution operations compared with the Qcable model. Although this model solves the problems associated with the transformation matrix, it increases numerical
stability concerns due to a variety of matrix fitting. Direct synthesis of the propagation function directly in phase domain is also a hard task. For a N-phase line each of the N modes travels at different speeds and therefore the N^2 elements of the matrix are under the effect of these mixed travelling times and modes. It is difficult to accurately model the phase angles in the fitting procedure.

- **Idempotent Model**

  The idempotent model of [13] is a phase-domain frequency-dependent model avoiding the use of modal transformation and related difficult eigenvector synthesis problems. This model expresses the line propagation function directly in phase domain in terms of the sum of N modal propagation function multiplying by N constant idempotent weighting matrices. Once the idempotent matrices, the propagation function for each mode and the characteristic admittance matrix are synthesized with rational functions, the transmission line model in frequency domain can be converted to time domain through inverse Laplace transform and discretization with an integration rule. This model provides exact and stable simulation results under any conductor configurations.

- **zLine and zCable Model**

  Researching for a full frequency-dependent and numerically stable transmission line model, Castellanos and Martí developed the zLine model for overhead transmission lines [14, 15]. The zLine model divides the length of the line into several small segments. Within each segment, the wave propagation is separated into two parts: a constant ideal line section depending only on the geometry of conductor configuration and a frequency-dependent loss section subject to skin effect. This model represents the frequency dependency of line parameters directly in phase domain, avoiding the use of modal transformation matrix. Furthermore, all modes travel at the same speed of light in ideal line sections, elegantly solving the problems of mixing modes and different transmission delays encountered in other phase domain models. To further improve the stability concern that most phase domain line models have incurred, a new constant-pole fitting procedure for the loss matrix
was introduced to ensure numerically stable solutions. The zLine model is accurate, stable and physically suited for asymmetric line configurations with the only limitations being the requirement of line segmentation to accommodate distributed lumped losses. These qualities make it an ideal candidate to be implemented within OVNI’s real-time framework. Ting-Chung Yu of [16] extended the idea of zLine to develop the zCable model for underground cable simulations.
1.3 Implementation Methodology

OVNI was originally conceived and developed by J. Marti [27] and its general philosophy was defined at that conception. L. Linares [2] realized the OVNI concept into an object-oriented architecture and one important objective he kept in mind was to permit and encourage researchers to develop their own models for elements of particular interests. New models are to be plugged in and updated at ease and independence without hassles of internal details of the simulator engine. In Linares’ implementation the object-oriented approach in C++ allows for a clean-cut segmentation of responsibilities and represents each OVNI network component as an object of a particular class. In particular the elm_t class serves as the parent class of all OVNI element models, and different element models derive the common element characteristics and methods from that class. All element models interact with the core through these well-defined methods to acquire or submit information relevant for network simulation at each simulation step. Information exchanged between an element model and the core includes the external node voltages, the equivalent conductance matrix and the equivalent history sources of the element. Through these common methods, models of different electric properties and complexities, from simple ones of resistor to complicated models of HVDC, all interact with the core in the same unified fashion. Model developers only need to implement those methods and nothing else to successfully integrate his/her model with OVNI.

OVNI’s solution algorithm builds on the MATE partitioning concept [4], (short for Multi-area Thevenin Equivalent), in addition to the original topological segmentation of RTNS. The fundamental concept behind network partitioning is through divide & conquers to solve the network more efficiently. Speed and efficiency could be gained if the inverse conductance matrix of each sub-block were pre-stored, given that it is small enough in dimension for economical storage. If a topological block falls beyond a critical size, defined by the number of nodes in the block, the second-tier MATE segmentation will be used. MATE operates by tearing the network through a number of resistive links. At each solution step, the link currents are first solved independently and later injected to each MATE sub-block to compute its node voltages as if sub-blocks are
fully decoupled from one another. Further simplicity can be achieved by hiding away internal nodes for complex element models. This process in effect reduces the dimensions of MATE sub-blocks to that of external nodes only. Detailed node-hiding technique and OVNI solution algorithm with MATE are introduced in a later chapter.

The pilot OVNI code implementation used for this thesis work was first written by Mazana Armstrong of UBC Power Group and is constantly improved by fellow graduate students as needs arise. This simplified core reduces the complexity of the proposed OVNI simulator by implementing only MATE segmentations and the sinusoidal current source, but maintaining the same element-core interface mentioned above. The purpose of this core is to make understanding of OVNI architecture easy and implementation and testing of various element models by students a more straightforward process. The complexity and versatility of this version of OVNI will gradually increase as new models are added and old ones updated. As a proof, this OVNI simulator contained only simple R, L and C models before this thesis work. Currently, an ideal transformer, induction motor and synchronous machine models are being developed and implemented. The addition of zLine, the first frequency-dependent transmission model to be included, is believed to enhance the functionality of OVNI by a great margin.

The implementation tasks of zLine emphasize on the following tasks: pre-processing and initialization of model parameters, computation of element conductance matrix and update of history sources. The pre-processing and initialization process involves generating from conductor geometry and configurations the line impedance and capacitance matrices, synthesizing the loss impedance matrix and running some file reading routines associated with the above two processing tasks. Model parameters for the line are initialized from the pre-processed data.

The implementation strategy for computing and updating the equivalent conductance matrix and history sources of the line takes advantage of the two-layer encapsulation between line and segments offered by object-oriented programming. The computation and update processes occur first at the segment level for each zLine segment. The entire
history source vector and conductance matrix for each segment are first reduced in
dimensions to external nodes with node-hiding. Reduced conductance matrix and history
vector for each zLine segment are accumulated at the line level to form the entire
conductance matrix and history vector of the line. The same node-hiding procedure is
applied once more to compute the equivalent conductance and history source
contributions of the line.
1.4 Thesis Organization

The work of this thesis is divided into the following chapters for a clear and organized presentation of the implementation strategy for zLine.

- Chapter 1: Presents the background and motivation for the development of OVNI and zLine model. A brief overview on implementation strategy of the zLine model is provided.
- Chapter 2: Presents the literature review on zLine modelling based on the work of [15] and [16].
- Chapter 3: Gives the architecture and solution algorithms of the OVNI simulator used in this thesis. Most importantly, the MATE network partitioning technique, the node-hiding technique and OVNI’s object-oriented classes are introduced here.
- Chapter 4: Provides the detailed implementation methodology.
- Chapter 5: Gives comparisons with a number of test cases of different line configurations to validate the correctness of the above implementation methodology.
- Chapter 6: Recommends future work and research activities.
- Chapter 7: Summarizes the main features and contributions of this thesis.
CHAPTER 2
FULL FREQUENCY-DEPENDENT TRANSMISSION LINE MODEL REVIEW

The transmission line model implemented in this thesis is based on the work of [15,16]. The authors take the space segmentation approach subdividing the total length of the line into several short line segments. Within each short line segment, the wave propagation is separated into two parts: an ideal line section and a lumped loss section. The ideal section represents the effect of the electric and magnetic field outside the conductors and physically corresponds to the external inductance \( L^{\text{ext}} \) and capacitance \( C \). Since \( L^{\text{ext}} \) and \( C \) depend only on the geometry of the line, the ideal section is constant and independent of frequency. The loss section, on the other hand, represents the losses and internal inductance inside the conductors and ground. In terms of line parameters, the loss section can be represented by the internal inductance of conductors \( L^{\text{int}} \) and resistance \( R \). They are frequency dependent due to skin effect and ground return. The separation of basic wave propagation effect is illustrated in Figure 2.1 below:

![Diagram](loss_section_ideal_section.png)

Figure 2.1 Decoupling of Basic Propagation Effect

In this chapter, the literature review on the modelling of both the ideal line section and the loss section that accumulates to form the zLine model will be given.
2.1 Transmission Line Theory Review

Taking into account the distributed nature of transmission lines, Figure 2.1 represents a per-unit length structure of a multi-phase line in frequency domain.

From Figure 2.2, a pair of transmission line equations for a given frequency can be derived as:

\[-\frac{d}{dx}[V] = [Z][I] \quad (2.1a)\]

\[-\frac{d}{dx}[I] = [Y][V] \quad (2.1b)\]

Taking one more derivative against variable x we obtain the wave propagation equation in equation (2.2):

\[\frac{d^2}{dx^2}[V] = [Z][Y][V] \quad (2.2a)\]

\[\frac{d^2}{dx^2}[I] = [Y][Z][I] \quad (2.2b)\]

where \([Z][Y]\) and \([Y][Z]\) are matrices that couple the voltage and current in each phase.

In the above expressions, letters inside brackets denote matrix and vector quantities depending on the context. \([Y]\) is the admittance matrix per unit length of the line where \([Y] = [G] + j\omega[C]\). \([G]\), the conductance matrix, in the case of overhead transmission line
is very small and can be ignored. \([C]\) is the capacitance matrix of the line, with its inverse, \([C]^\dagger\), denoted as \([P]\), the Maxwell coefficient matrix. Since there is no correction needed for ground return, the element of \([C]\) is generally constant and frequency independent. \([Z]\), on the other hand, is the impedance matrix per unit length of the multi-conductor line where \([Z] = [R] + j\omega[L]\). Here \([R]\) represents the resistance matrix and contains the effect of ground return in self and mutual elements. \([R]\) is, therefore, frequency dependent. \([L]\) is the inductance matrix of the line and can be broken down further into two parts: inductance related to flux external to the conductors \([L^\text{ext}]\) and inductance related to flux internal to the conductors and ground return, \([L^\text{int}]\). The expression for \([Z]\) can then be rewritten as:

\[
[Z] = [R] + j\omega([L^\text{ext}] + [L^\text{int}])
\] (2.3)

Note that \(L^\text{ext}\) is constant and frequency independent while \(L^\text{int}\) is subject to skin effect and is a function of frequency. In terms of matrix elements and taking into account the frequency dependence of above parameters, (2.3) becomes:

\[
Z_{ij}(\omega) = R_{ij}(\omega) + j\omega(L^\text{int}_{ij}(\omega) + L^\text{ext}_{ij})
\] (2.4)

Now if we expand expression (2.4) and group frequency dependent terms together we obtain the following:

\[
Z_{ij}(\omega) = (R_{ij}(\omega) + j\omega L^\text{int}_{ij}(\omega)) + j\omega L^\text{ext}_{ij}
\] (2.5)

With \(Z^\text{loss}_{ij}(\omega) = R_{ij}(\omega) + j\omega L^\text{int}_{ij}(\omega)\) and \(Z^\text{ext}_{ij} = j\omega L^\text{ext}_{ij}\) (2.6)

We can rewrite \([Z(\omega)]\) as the following:

\[
[Z(\omega)] = [Z^\text{loss}(\omega)] + [Z^\text{ext}] = ([R(\omega)] + j\omega[L^\text{int}(\omega)]) + j\omega[L^\text{ext}]
\] (2.7)

Using (2.7), the propagation matrix \([YZ]\) can be expressed as
\[ [YZ] = j\omega[C](Z^{loss} + j\omega[L^{ext}]) = j\omega[C][Z^{loss}] - \omega^2[C][L^{ext}] \quad (2.8) \]

Equation (2.8) shows that the propagation matrix can be expressed as the combination of two main components: the first term relates to the internal losses and ground effect and the second term relates to ideal propagation in the external field. In particular, for ideal propagation (\(\rho=0\)):

\[ [C][L^{ext}] = \mu_0\varepsilon_0[I] = \frac{1}{c_0^2}[I] \quad (2.9) \]

where \(\mu_0\) is the permeability in free space, \(\varepsilon_0\) the permittivity in free space, \(c_0\) the speed of light in free space and \([I]\) the identity matrix.

In zLine's sectionalized line model, \([Z^{loss}(\omega)]\) forms the lumped loss section within each small line segment. The elements of \([Z^{loss}(\omega)]\) will be synthesized with rational functions and from the poles and constants of resulting fitting blocks, a corresponding set of time-domain equivalent resistance and history source expression can be formulated with discretization in the EMTP manner [1]. \([Z^{ext}]\) along with \([C]\) contribute to the ideal line section within each small line segment. Since the frequency dependent effect has been represented completely inside \([Z^{loss}(\omega)]\), and (2.9) shows an diagonal propagation matrix, a multiphase cpLine model is sufficient to represent the distributed nature of the line.
2.2 Modelling of Ideal Line Section

After \([Z^{\text{loss}}(\omega)]\) has been extracted from the wave propagation, all modes travel at the same speed of light in the resulting ideal line section, as shown in (2.9). The solution of the ideal line section can therefore be formulated directly in the phase coordinate, avoiding the use of modal decomposition. Dommel's cpLine model [1] can be used to represent the wave propagation in the multi-phase ideal line section, simply replacing voltage and current scalars with vectors and impedance with matrices. The line model in phase domain for the ideal section is given in Figure 2.3.

![Figure 2.3 Multiphase Line Model For Ideal Section](image)

Similar to single-phase cpLine model, the relationship between terminal voltages \([V_m(t)]\) and \([V_k(t)]\), line currents \([I_m(t)]\) and \([I_k(t)]\), and history sources \([h_k(t)]\) and \([h_m(t)]\) can be expressed as

\[
\begin{align*}
[I_k(t)] &= [Z_c]^{-1} [V_k(t)] - [h_k(t)] \\
[I_m(t)] &= [Z_c]^{-1} [V_m(t)] - [h_m(t)]
\end{align*}
\]

(2.10a) \hspace{1cm} (2.10b)

Especially, history sources can be updated at every time step directly in phase domain by the following expression:

\[
\begin{align*}
[h_k(t)] &= [Z_c]^{-1} [V_m(t - \tau)] + [I_m(t - \tau)] \\
[h_m(t)] &= [Z_c]^{-1} [V_k(t - \tau)] + [I_k(t - \tau)]
\end{align*}
\]

(2.11a) \hspace{1cm} (2.11b)
In particular, \([Z_c]\), the characteristic impedance matrix of the line, can be calculated as

\[
[Z_c] = \sqrt{\frac{[Z^\text{ext}]}{[Y]} = [Y]^{-1}(Y[Z^\text{ext}])^{1/2} = \frac{1}{j\omega} [\mathbf{P}] \frac{\omega}{c_0} [\mathbf{I}] = \frac{1}{c_0} [\mathbf{P}]
\]

where \([\mathbf{P}]\) is the Maxwell coefficient matrix and \(c_0\) the speed of light.

Since \([\mathbf{P}]\) is the inverse matrix of \([\mathbf{C}]\), \([Z_c]\) would possess the same property- constant and dependent on line geometry. One significant advantage of this full phase-domain line model is that the time step \(\Delta t\) can be chosen as large as \(\tau\) to have an accurate solution, which is not feasible in the traditional multiphase cpLine model. In the traditional model, each mode travels at a different velocity, hence requiring history sources to be updated in modal domain. \(\Delta t\) of the system in that case can at most be chosen as large as the smallest \(\tau\) among all modes, making the simulation longer and slower.
2.3 Modelling of Lumped Loss Section

The modelling of lumped loss sections for each zLine segment is divided into two parts: synthesis of \([Z^{\text{loss}}(\omega)]\) and discretization of resulting fitting functions.

2.3.1 Matrix Synthesis and Fitting

To generate appropriate model of the lumped loss section for the discrete-time EMTP solution, elements of \([Z^{\text{loss}}(\omega)]\) must first be synthesized with rational functions. In order to ensure a numerically stable fitting functions, according to Castellanos in [14, 15], all elements of the loss matrix must be fitted in a simultaneous manner with the same set of poles. A complete procedure of element synthesis was first developed by Castellanos with ADA and later modified and implemented by Ting-Chung Yu in C++. For the purpose of this thesis, the fitting routine has been tailored to synthesize the loss matrix of overhead lines and later on modified to suit the simultaneous execution with other programs such as mtLine. In the synthesized functions of the zLine model, each element of the loss matrix \([Z^{\text{loss}}(\omega)]\) consists of the following Laplace domain expression, replacing \(s\) with \(j\omega\) to obtain the frequency response:

\[
Z_{f(ii)}^{\text{loss}}(\omega) = R_{\text{ide}} + \sum_{l=1}^{m} \frac{sK_{ii(l)}}{s + P_{ii(l)}} \quad \text{for diagonal elements} \\
Z_{f(ij)}^{\text{loss}}(\omega) = \sum_{l=1}^{m} \frac{sK_{ij(l)}}{s + P_{ij(l)}} \quad \text{for off-diagonal elements}
\]  \hspace{1cm} (2.13)

where subscript \(f\) indicates fitted functions, \(K\) the multiplying constants in numerators and \(P\) for poles. \(R_{\text{ide}}\) denotes DC resistance of conductors or bundles to satisfy the DC condition in diagonal elements. Each fitting block in (2.13) can be represented in circuit schematic by a parallel R-L block. Each element of the loss matrix consists of \(m\) fitting blocks connected in series. The detailed matrix synthesis procedures are explained in details in Appendix I.
2.3.2 Discrete Time Model of the Lumped Loss Section

Once the elements of the loss matrix are synthesized with rational functions in terms of constants and poles, the voltage drop equation across the loss section can be formulated in an EMTP manner by applying an appropriate integration rule.

\[ \left[ V(t) \right] = \left[ R_{eq} \right] \left[ I(t) \right] + \left[ H(t) \right] \] (2.14)

For a single R-L block representing the synthesis of the elements of the loss matrix, its discrete time model, represented by an equivalent resistance and voltage history source, is given below by discretizing with the trapezoidal rule [16]:

\[ R_{eq} = \frac{2K.1}{\Delta t} \] (2.15a)

\[ h_{eq}(t) = \frac{2}{\Delta t + P} h_{eq}(t - \Delta t) - \frac{4KP.1}{\Delta t} I(t - \Delta t) \] (2.15b)

where P and K are the pole and constant of the R-L block. \( l \) stands for the length of the R-L circuit; in this context it is equal to the segment length.

Equation (2.15) is the time-domain discrete time model for a single parallel R-L circuit. It can be extended to generalize the equation for multiphase \([R_{eq}]\) and \([H(t)]\) of the loss section, when each element contains more than one R-L fitting block. The extension is straightforward, simply summing the \( R_{eq} \)'s of all R-L fitting blocks for a particular element, as shown in (2.16) below:

\[ R_{eqit} = R_{iit} + \sum_{n=1}^{m} \frac{2K_{ii(n)}}{\Delta t + P_{ii(n)}} I \text{ for diagonal elements} \] (2.16a)
\[
R_{eqij} = \left[ \sum_{n=1}^{m} \frac{2K_{ij(n)}}{\Delta t + P_{ij(n)}} \right] \quad \text{for off-diagonal elements}
\quad (2.16b)
\]

where, \(P_{ij(n)}\) and \(K_{ij(n)}\) stands for pole and constants of the \(n^{th}\) fitting block for element \(ij\) of the loss matrix. \(m\) denotes the total number of fitting blocks for each element. The addition of \(R_{idec}\) is used to satisfy the DC condition.

The history vector \(H_i(t)\), where \(i\) indicates the row or phase number, is derived by summing up \(h_{eq}(t)\) of (2.15b) for all fitting blocks and across all columns of the loss matrix at row \(i\). This in effect accumulates all voltage history sources for phase \(i\).

\[
H_i(t) = \sum_{j=1}^{\text{NumCond}} \sum_{n=1}^{m} h_{eqij(n)}(t) \quad (2.17a)
\]

\[
h_{eqij(n)} = \frac{2}{\Delta t - P_{ij(n)}} h_{eqij(n)}(t - \Delta t) + \frac{4K_{ij(n)}P_{ij(n)} - 1}{4} I_j(t - \Delta t) \quad (2.17b)
\]

The discrete time model for the lumped loss section can be graphically represented by:

![Diagram](image)

Figure 2.4 The Model for Lumped Loss Section
The voltage history source \([H(t)]\) has been converted to an equivalent current history source by the conversion \([R_{eq}]^{-1}[H(t)]\). The circuit diagram has been converted to the parallel connection subsequently.
2.4 Phase-Domain Transmission Line Model

As mentioned earlier the entire zLine model is represented by several small length segments connecting in cascade to distribute the losses along the line. Each small length segment consists of an ideal line section and a lumped loss section. To combine the loss section with the ideal line section, the loss section is further split in half inside each segment and inserted at both ends of the ideal line section for distributed parameter purposes, similar to the way losses are split and inserted in cpLine models [1]. The model for each small length segment is shown in Fig 2.5.

![ZLine Segment Model](image)

**Figure 2.5 ZLine Segment Model**

The entire zLine model is obtained by connecting several short line segments in series. The complete solution of the line is obtained by combining the solution of ideal line sections and lumped loss sections for all segments of the line. The detailed algorithm for computing and accumulating equivalent segment history sources and conductance matrices will be introduced in detail in Chapter 4. The complete transmission line modelled as zLine in phase coordinate is shown in Figure 2.6.
Figure 2.6 The Complete zLine Model
2.5 Segment Length and Maximum Frequency of Interest

For usual EMTP type simulation including pure R, L, and C components, $\Delta t$ of simulation must be chosen with care to allow accurate simulation of transient behaviour of the system. Normally, the maximum frequency of interest is chosen to be $\frac{1}{5}\frac{1}{f_{\text{Nyquist}}}$ or $rac{1}{10\Delta t}$ to limit the distortion of trapezoidal rule to 3 percent. For cpLine and fdLine of Microtran, $\Delta t$ is usually chosen to be 5 to 10 times less than the smallest $\tau$ of the modes to compensate for interpolation errors and to sufficiently capture the fast transients located in the network. Note that for a long line, the cpLine model must first be sectionalized just like zLine to generate more accurate simulation results. The ballpark rule for an acceptable segmentation is

$$\text{length} \leq \frac{10000}{f}$$

(2.18)

where length is in km and $f$ is frequency of interest in Hz. For networks containing a zLine model, the segment length following approximately the above segmentation rule will determine the largest $\Delta t$ that can be used for simulation. The $\Delta t$ is chosen to match the $\tau$ of each ideal line section in each zLine segment. In other words, the control variable for an accurate zLine simulation is not $\Delta t$ but the length of each zLine segment. The more segments the line model is divided to, the smaller the segment length, hence generating smaller $\tau$, which in turn makes $\Delta t$ of the simulation smaller.

In order to assess the relationship between zLine segment length and maximum frequency of interest, Castellanos of [15] ran simulations with a balanced cpLine model and tailored the model parameters at different source frequencies. A balanced cpLine model gives an exact steady-state response for single-frequency simulations. By comparing the steady-state responses of cpLine with those of zLine at those same
frequencies, a graphical relationship can be established and is presented here for reference.

![Graph](image)

Figure 2.7 Maximum Segment Length VS Maximum Frequency of Interest [15]

Note from the figure that for 60 Hz power frequency, 100 km per segment or even longer are sufficient to simulate the steady-state responses. But usually for faster transient responses, up to thousands of Hz for switching studies, a smaller segment length such as 2.5 km or less is required. In addition, the figure shows that the Δt restriction imposed by line segmentation does not differ too much from $\frac{1}{5} f_{\text{Nyquist}}$ of trapezoidal rule for 3% distortion. To corroborate that statement, a 2.5 km per segment line can simulate up to about 8 KHz, with Δt of 8.333 μs, according to Figure 2.7. The same Δt gives maximum bandwidth of 12 KHz for 3% distortion of trapezoidal rule, which indeed is not too different from zLine’s simulating bandwidth.
CHAPTER 3
CURRENT OVNI ARCHITECTURE AND CLASSES

As mentioned in Chapter 1, OVNI has been implemented in an object-oriented paradigm to maintain a full encapsulation of element models from the core and vice versa. In this manner, model developers can independently create their models, and incorporate with the core without having to worry about the details inside the simulator. In the view of the core developer, this would avoid unnecessary mistakes or changes being made accidentally to damage the core. The only rule that model developers must pay attention to is the constant interface between the two. Adhering to the same interface, all models would look and behave the same in the “eyes” of the simulator engine.

The simplified OVNI simulator, for the purpose of this thesis work and coincidental model developments by peer students, was originally implemented by Mazana Armstrong of UBC Power Group. This OVNI simulator maintains the same interface so mentioned and is less complicated for beginners to grasp the interactions between each OVNI modules as Linares visualized in his thesis [2]. The current core employs the MATE solution algorithm by dividing the network into MATE sub-blocks via some chosen links. MATE segmentation enhances speed and efficiency for real-time solution and opens the door for other network solution techniques such as latency exploitation. In addition to MATE’s improvement in performance, the node-hiding technique is applied in element modules to further simplify the element-core interactions.

In this chapter, the network solution with MATE, the node-hiding technique and the functionalities of each OVNI class, especially the elm_t class, will be discussed. At the conclusion of this chapter, a detailed solution algorithm adopted by current version of OVNI simulator will be listed.
3.1 Network Partition and Solution

As mentioned previously the real-time simulator currently under development imposes very stringent time constraints on the solution algorithm and the hardware components used. The solution steps followed by standard EMTP-type simulators including OVNI is illustrated in Figure 3.1 below [2,3,5,6]:

At each simulation step, the node voltages are computed according to (3.1) below:

\[
[G][V(t)] = [h(t)]
\]  

(3.1)

where \([G]\) is the network conductance matrix, \([V(t)]\) the voltage vector for each node in the network, \([h(t)]\) the accumulated current source vector for each node in the network. Linares [2] indicated in his thesis that the percentage of time consumed by the simulator to perform each of the tasks is as follows: updating history sources 14.7%, accumulating nodal currents 19.6%, nodal voltage computation 65.8%. As can be seen, the step of
nodal voltage computation, for the most part inverting and re-computing the system matrix \([G]\) for topology changes, takes up the most amount of solution time. One solution approach developed in previous RTNS is to pre-calculate and pre-store all possible \([G]^{-1}\) matrices before simulation starts to speed up the solution process. This method may work efficiently in small systems in which the system size is small and complexity simple. For larger systems with thousands of nodes and hundreds of switches, pre-storing all possible system \([G]^{-1}\) matrices, according to Linares, is extremely uneconomical and impractical. Even if all possible \([G]^{-1}\)’s can be pre-stored, the mere solution of (2.1) would require a gigantic amount of floating point operations and still consume most of the solution time.

As the result, network segmentation was proposed to counter the size of the network. In RTNS architecture [5,6], a one-tier segmentation was utilized. The complete power network is decoupled into several independent blocks due to the transmission delays of transmission lines in the system (Figure 3.2a). The resulting network \([G]\) and its inverse becomes block diagonal (Figure 3.2b), with each block sub-matrix containing fewer nodes and fewer topology changes so that \([G]^{-1}\) for each block can be practically pre-stored within memory. Solution of (2.1) can be less formidable as well because each block sub-matrix smaller in size can be solved separately.

Figure 3.2  a) Decoupling Effect in Transmission Line model. b) Corresponding \([G]\)

However, as the size of the network grows, even the size of the topological- segmented blocks grows beyond a limit that renders economical pre-storage not practical. J. Marti suggested a second-tier segmentation scheme, named MATE (multi-area thevenin equivalent) [4].
3.1.1 Network Segmentation - MATE

In ONVI, MATE operates by further dividing the topologically-decoupled blocks into several smaller sub-blocks along the “links” connecting the sub-blocks. The resulting $[G]$ matrix for the network with this two-layer segmentation scheme becomes:

\[
\begin{bmatrix}
A & 0 \\
0 & B
\end{bmatrix}
\begin{bmatrix}
V_A \\
V_B
\end{bmatrix} =
\begin{bmatrix}
h_A \\
h_B
\end{bmatrix}
\]  

(3.2)

For the purpose of illustrating the solution algorithm of MATE, assume that the network only employs MATE segmentation. MATE links divide the network into two sub-blocks, represented by their conductance matrix $[A]$ and $[B]$ respectively. The MATE matrix expressions in (3.2) is created by writing nodal equations assuming sub-network A and B are fully decoupled.
where \( p \) and \( q \) are link matrices for \( A \) and \( B \), having the same columns as the number of links. They contain value of 1 at rows corresponding to nodes where currents flowing out, and \(-1\) corresponding to nodes when currents flowing in. \( z \) is a diagonal matrix containing the resistance of each link in its diagonals. Multiplying \( A^{-1} \) to row \( A \) in (3.4) we have:

\[
\begin{bmatrix}
1 & 0 & a
\end{bmatrix} \begin{bmatrix}
V_A \\
V_B \\
i_A
\end{bmatrix} = e_A
\]

(3.4a)

\[a = A^{-1}p\]

(3.4b)

\[e_A = A^{-1}h_A\]

(3.4c)

Similarly, applying \( B^{-1} \) for row \( B \) in (3.4)

\[
\begin{bmatrix}
0 & 1 & b
\end{bmatrix} \begin{bmatrix}
V_A \\
V_B \\
i_A
\end{bmatrix} = e_B
\]

(3.5a)

\[b = B^{-1}q\]

(3.5b)

\[e_B = B^{-1}h_B\]

(3.5c)

Substituting \( V_A \) and \( V_B \) from (3.5a) and (3.6a) in terms of \( e_A, e_B \) and \( i_A \) into the link equation, row \( \alpha \) of (3.3), the resulting equation becomes:

\[
\begin{bmatrix}
p' & q' & -z
\end{bmatrix} \begin{bmatrix}
e_A - ai_A \\
e_B - bi_A \\
i_A
\end{bmatrix} = 0
\]

(3.6a)
Arranging terms to recover variables $V_A$ and $V_B$

\[
\begin{pmatrix}
V_A \\
0 \\
Z_a \\
i_a
\end{pmatrix}
= \begin{pmatrix}
e_a \\
0 \\
0
\end{pmatrix}
\tag{3.6b}
\]

\[Z_a = p^t a + q^t b + z \tag{3.6c}\]

\[e_a = p^t e_A + q^t e_B \tag{3.6d}\]

Combining (3.4) to (3.6) the equivalent matrix becomes:

\[
\begin{pmatrix}
A & B & \alpha \\
A & 1 & 0 & a & V_A \\
B & 0 & 1 & b & V_B \\
\alpha & 0 & 0 & Z_a & i_a
\end{pmatrix}
= \begin{pmatrix}
e_A \\
e_B \\
e_a
\end{pmatrix}
\tag{3.7}
\]

At this point, nodal voltages at each sub-network A and B can be solved first by solving the link current $i_a$.

\[i_a = \frac{e_a}{Z_a} \tag{3.8}\]

Nodal voltages $V_A$ and $V_B$ can be obtained by substituting $i_a$ into row A and B of (3.7).

\[V_A = e_A - a_i_a \tag{3.9a}\]

\[V_B = e_B - b_i_a \tag{3.9b}\]

The solution algorithm above can be easily extended to cases where the links divide the network into more than two sub-networks. In those cases, there will be as many conductance matrices and link matrices as the number of sub-networks. Equation (3.8) and (3.9) can be applied to solve for node voltages in each sub-network in the same manner.
The advantage of MATE segmentation lies in the fact that the entire network matrix can be effectively represented by MATE sub-block matrices \([A], [B]\) and link matrices \(\alpha (p, q \text{ and } z)\). On top of that, MATE is especially beneficial in representing switch operations, especially when all switches are purposely located in MATE links. The reason for that is when switch state changes, the net influence will only be on the link matrix \([\alpha]\), leaving matrix \([A], [B]\) topologically constant. In the context of real-time simulation, small and constant matrices \([A], [B]\) entail economical and practical pre-calculation of those matrices to alleviate the workload of the core during simulation. The node-hiding technique, to be introduced in the next sub-section, can further reduce the dimension of sub-matrices \([A]\) and \([B]\) to that of connecting nodes in the sub-network only.

Another important area of application where MATE could be extremely advantageous is in latency exploitation. The MATE segmentation separating the network into fast and slow sub-networks by links enables solution of different sub-networks with different simulation steps [17].

The working version of OVNI simulator adopted for this thesis implemented only MATE segmentation; that is, the entire network is divided into sub-blocks by certain amounts of links chosen. Each MATE sub-block instantiation and its corresponding subroutines are defined in class blk_t; each MATE link is constructed with class lnk_t. These classes and the roles they play in OVNI will be introduced in later sections.

3.1.2 The Noding-Hiding Technique

Elements in a power network are connected to other elements through certain amounts of physical nodes, namely the “external nodes”. On the other hand, many complex elements, such as a HVDC controller and the zLine model, possess numerous internal nodes introduced primarily by the modelling process. Figure 3.4 revisits the zLine segment model and highlights the external nodes and internal nodes.
In OVNI, the simulator only needs to solve external nodes with which elements interact with one another. If internal details of elements are hidden away from the core and processed independently within element models, considerable workload could be shed from the core in the process of solving equation (3.1). In this section, the node hiding technique will be introduced to further improve the efficiency of the OVNI solution methodology. Especially, the formula that computes the element conductance matrix and history source contribution in dimensions of only external nodes, encapsulating the contribution made by the internal ones, will be given.

According to Linares's work [2], if external nodes of an element are denoted subscript $a$, arranged next to one another in the conductance matrix and internal nodes denoted $b$, a nodal equation similar to (3.1) can be written for the element:

$$
\begin{bmatrix}
G_{aa} & G_{ab} \\
G_{ba} & G_{bb}
\end{bmatrix}
\begin{bmatrix}
V_a \\
V_b
\end{bmatrix} =
\begin{bmatrix}
h_a \\
h_b
\end{bmatrix} 
$$

(3.10)

where $V_a$ denotes voltages of external nodes and $V_b$ denotes voltages of internal nodes. $h_a$ denotes the accumulated current sources for external nodes and $h_b$ for internal nodes. They are all vector quantities.

From second row of (3.10),
\[ [G_{ba}][V_a] + [G_{bb}][V_b] = [h_b] \]  
\[ (3.11a) \]
\[ [V_b] = [G_{bb}]^{-1}( [h_b] - [G_{ba}][V_a]) \]  
\[ (3.11b) \]

Note that internal voltages can be updated with (3.11b) if the external voltages \( V_a \) and accumulated currents to internal nodes \( [h_b] \) are already known. If the nodal equation of first row of (3.10) is expanded,

\[ [G_{aa}][V_a] + [G_{ab}][V_b] = [h_a] \]  
\[ (3.12) \]

Substituting (3.11) into (3.12) for \( [V_b] \) and arrange terms,

\[ [G_{aa}][V_a] + [G_{ab}][G_{bb}]^{-1}[h_b] - [G_{ab}][G_{bb}]^{-1}[G_{ba}][V_a] = [h_a] \]  
\[ (3.13a) \]
\[ ([G_{aa}] - [G_{ab}][G_{bb}]^{-1}[G_{ba}])[V_a] = [h_a] - [G_{ab}][G_{bb}]^{-1}[h_b] \]  
\[ (3.13b) \]

\[ ([G_{aa}] - [G_{ab}][G_{bb}]^{-1}[G_{ba}])[V_a] = [h_a] - [G_{ab}][G_{bb}]^{-1}[h_b] \]  
\[ (3.13c) \]

The term inside the bracket on the left of the equal sign of (3.13c) shows the element's contribution to the conductance matrix with dimension equal to the number of external nodes. Note that this expression already encapsulates the contribution of internal nodes with the matrix algebra.

\[ [G_{aa}^{element}] = [G_{aa}] - [G_{ab}][G_{bb}]^{-1}[G_{ba}] \]  
\[ (3.14) \]

Similarly, the term on the right side of (3.13c) indicates the element's history source contributions in terms of external nodes. Again, contributions from internal nodes have already been included.

\[ [h_a^{element}] = [h_a] - [G_{ab}][G_{bb}]^{-1}[h_b] \]  
\[ (3.15) \]
In short, the contribution of an element to the entire network is

\[
[G^\text{element}_{aa}] [V_a] = [h^\text{element}_a]
\] (3.16)

Node-hiding plays a big role in the implementation of the zLine model and its application will be explained in more details in the next chapter.
3.2 Classes in OVNI

OVNI is designed with object-oriented programming. Classes are the basic units that define OVNI modules. The most critical module in OVNI is the elm_t class which contains all the element models connected one way or another to form the network. Apart from the element class are the blk_t and lnk_t classes which fulfill the MATE segmentation of the network. There are also a source class, src_t, within which a current source is defined. Although the main topic of this thesis, implementation of the zLine model, does not require any knowledge about the core, some basic understanding on its operation would be helpful especially during the testing stage of model implementation. The current OVNI simulator has actually been modified by several model developers including myself to either improve solution efficiency or to fulfill other model testing purposes. After all, the current core is still an experimental version; changes still need to be made to make it more versatile, optimized and completely general. It is, therefore, considered a big part of this thesis work. In this section, the elm_t class and its interface with the core will be introduced. What follows are the blk_t and lnk_t classes and their interaction with the core. Finally the src_t class and its member functions will be reviewed.

3.2.1 The elm_t Class and Interaction with OVNI

As model developers design their element models, it is essential that they are exempt of the knowledge of the core so that they can work independently and on their own. The core, on the other hand, must protect itself from any accidental changes or introduction of errors by model developers as they incorporate their models with the simulator. The object-oriented approach in C++ allows for this kind of encapsulation and future expansion as all current and to be developed element models are the "child" of the elm_t class, independent of details of the core. The elm_t class contains all the necessary methods that an element model in OVNI needs to interact with the simulator. A glimpse of the elm_t class in current OVNI is shown in Fig 3.5.
In OVNI's object-oriented structure, each element is given a letter id to identify one from another; for example, 'Z' represents the zLine model, 'R' for the resistor model, etc. Each element model also specify the number of external nodes it is connected to, cNumNod, and fill up a vector of numerical node id's corresponding to each external node, aNodId. An element would also fill up the conductance matrix aG before or at each simulating step depending on if there are any topological changes in the network. Within every time step, the history source vector aExHsrc is updated and submitted to the core for voltage solutions. To be more specific, all element models interact with the simulator engine through exchange of the following information:

- The request by the core for the element to submit its conductance matrix in dimensions equal to its “external nodes” only. This entails to applying the node hiding technique (3.14) summarized in the previous sub-section. In terms of programming language, the core would invoke the submitG() function of every element. The element model returns the pointer aG to the corresponding conductance matrix.
• The request by the core for the element to update its history sources to the external nodes. Node hiding technique (3.15) would be used to compute the equivalent $h_a$ contribution in dimensions of external nodes. In Figure 4.3, the pointer $a_{ExHsrc}$ would be returned to the core when the function UpdateHistory() is invoked.

• The request by the core for the element to return the number of external nodes it is connected to, GetNumNod(), and the numerical external node ids that the element is connected to, submitNodId(). They are used by the core to correctly insert the element conductance matrix and history source vector to the correct location in the network $[G]$ matrix and $[h_h]$ vector of (3.1).

• The request by the element for the core to return the external nodal voltages that are solved in the previous time step. These external voltages are utilized to compute all the internal voltages (3.11b), which in turn are used to update all history sources for the network solution.

All element classes in OVNI are descendent of elm_t. In other words, all elements inherit the public subroutines and protected member variables in elm_t. The “virtual” key word indicates that the very function will be redefined in the child class as different models may have distinct ways of computing and updating their conductance matrices and history sources. An already existed inductor model would indicate how a child class inherits from elm_t in better details.

class ind_t:public elm_t{
    // Derived class from elm_t class
    public:
        ind_t(fstream&);  // constructor
        double** submitG();  // virtual functions redefined here
        double* UpdateHistory();
    private:
        double** aGind;  // Conductance matrix of the inductor
    };

Figure 3.6 Header File for ind_t

39
Note that the scope of inheritance for ind_t is indicated by key word ":public elm_t". The functionality of the default constructor, ind(fstream&), is to compute and initialize model parameters relevant to an inductor. For a simple element like inductor, parameters to be initialized include mainly inherited member variables and methods from elm_t mentioned in the last page. For more complicated element models, unique member variables beside those inherited from elm_t need to be customized and initialized.

3.2.2 The blk_t Class

An object of the blk_t class represents a MATE block in the network. In MATE’s formulation, MATE blocks are separated from one another by MATE links and contain the associated block conductance matrices. The header file of blk_t is shown in Fig 4.5 below.

The blk_t class contains a default constructor that automatically reads in parameters related to the MATE block every time a new block object is created. A numerical id of 5 has been assigned to MATE blocks to recognize that blocks of data in the input file belong to and will be read by a MATE block. Each blk_t object possesses a variable containing the total number of nodes in that block. As well, each blk_t object maintains a pointer to a vector listing the node ids that are located in the block. The simulator interacts with blk_t by invoking each block to return the number of nodes, GetNumNod(), and node ids of those nodes, submitNodId(), in order to create the block matrix $A^{-1}$ and $B^{-1}$ in equation (3.3).
3.2.3 The lnk_t Class

An object of the lnk_t class represents a MATE link in the network to be solved. Physically, MATE links connect two MATE blocks in an OVNI network. In the MATE formulation, MATE links contribute to the bottom few rows and rightmost few columns of the network conductance matrix given in equation (3.3), depending on the number of links in the network. A MATE link in current OVNI implementation is modeled by a resistor in series with a voltage source as shown in Figure 4.6.

![Figure 3.8: Model of MATE Link in OVNI](image)

If the voltage source in the links of Fig 4.2 were not zero, the last row of (3.3) would become

$$
\begin{bmatrix}
    p^t & q^t & -z & V_A \\
    V_B \\
    i_a
\end{bmatrix}
= -e_{\text{link}},
$$

(3.17)
where \(-e_{\text{link}}\) denotes a vector containing the value of voltage sources in the links plus a negative sign for the orientation shown in Figure 3.8. Following the same manipulation as in equation (3.6) a modified expression for \(e_a\) become:

\[
e_a = p^t e_A + q^t e_B + e_{\text{link}}
\]  

(3.18)

Expressions for \(e_A\), \(e_B\) and \(Z_a\) remains the same. As in blk_t, the lnk_t class possesses a default constructor to initialize the parameters of every link object. A link object is assigned an id of 4 to recognize the data to be read in the input file. The header file of lnk_t is shown in Fig 3.9 below.

```cpp
// Each link in the system is an instantiation of this object.
class lnk_t {
public:
    lnk_t(fstream&);
    double submitVLnk();
    double submitRLnk();
    int* submitNodId();
    int Lnkld(){return id;};
    int GetNumNod(){return cNumNod;};
protected:
    int id; // Link id
    int cNumNod; // Number of nodes the link is connected to
    int* aNodld; // Vector of nodes id (from, to)
    double rLnk; // Link resistance
    double vLnk; // Link voltage source
};
```

Figure 3.9 Header File for MATE Links

As can be seen, a link object possesses two double precision variables rLnk, which stores the link resistance and vLnk, which stores the link voltage source. Similar to blk_t, a link object maintains a variable cNumNod to keep track of the number of nodes a link is connected to (usually two) and a pointer *aNodId to maintain a vector containing the numerical ids of those nodes. The simulator would invoke the member function submitNodId() and GetNumNod() to obtain from each link its relative node ids and node counts to establish the link matrix \(p^t\) and \(q^t\). It would invoke submitRLnk() to construct
the matrix $-z$ in equation (3.3). Likewise the vector $-e_{\text{ink}}$ is formed by calling submitVLnk() of each link object to obtain its link voltage source.

### 3.2.4 The src_t and isrc_t Class

The current version of OVNI implements a src_t class which, just like the elm_t class, acts as the ancestor or base class for all source types. All degenerated source classes inherit the public and protected member functions and variables from src_t. In the current core, only the current source has been implemented so the isrc_t class represents the structure for all current sources and is the descendent of src_t. The voltage source can be represented by arranging a current source in Norton form in parallel with a very small resistor. A glimpse of the src_t structure is provided in Figure 3.10 below. Figure 3.11 shows the isrc_t structure.

```cpp
// Each source in the system is an instantiation of this class.
class src_t {
public:
    src_t(fstream&);
    virtual double* submitSrc();
    int* submitNodId();
    int SrcId() {return id;}; // return source id to the core
    int GetNumNod() {return cNumNod;};

protected:
    int id; // Source id that identifies data for sources
    int cNumNod; // Number of nodes the source is connected to
    int* aNodId; // Vector of node ids
    double* aSrc; // Vector of source contributions to each node
};
```

Figure 3.10 Header File for Sources in OVNI
class isrc_t: public src_t // Derived class from src_t class
public:
    isrc_t(fstream &);
    double* submitSrc(); // Virtual member redefined
private:
    double* alsrc;
    double freq; // frequency of the sinusoidal source in Hz
    double amp; // amplitude of the sinusoidal source
    double angle; // phase shift of the sinusoidal source

Figure 3.11 Header File for Current Sources in OVNI

The current source in OVNI is assigned an id of 3 to differentiate their parameters from other OVNI components. Fig 3.10 shows that a source possesses a variable cNumNod to store the number of nodes it is connected to (usually two), a pointer aNodId to the vector storing the numerical node names at both ends of the source, and yet another pointer aSrc to the vector storing the contribution of source values at its two nodes. The polarity of source contributions is defined in the way that the node with current flowing in is granted a positive source value. As an example, a 2 amp current source flowing from node 1 to node 2 would fill the vector *aSrc with [-2 2]. The isrc_t object inherits all member variables in src_t, with the addition of three private members: freq, amp and angle to represent the frequency, amplitude and phase (radian) of a sinusoidal current source. The core interact with the src_t and isrc_t by invoking at every time step, for sinusoidal sources, or only once, for constant DC sources, the submitSrc() function. The core would accumulate the contributions from each source to the correct positions in the nodal current vector $h_A, h_B$ to $h_N$ (N is the number of block in the network).
3.3 The Simulator Engine and the Solution Algorithm

Although OVNI follows basically the same solution steps as in Figure 3.1 when solving a network in discrete time steps, inclusion of MATE segmentation for better solution speed and efficiency introduces further pre-processing and communications between the core and MATE modules. In this section, the solution algorithm for OVNI that highlights those communication and initialization tasks will be explained in numbered steps and illustrated in a flowchart. In particular, each numbered task is elaborated below.

- Preprocessing
  1. Simulation parameters such as step size ($\Delta t$), simulation time ($t_{final}$), number of elements, number of nodes, number of MATE blocks, and number of MATE links in the network are read in directly from the input file. An example of the input file is given in Appendix II.

  2. The core constructs the objects of each element, current source, MATE block, and MATE link object and initializes corresponding parameters for each type of OVNI components from the input file.

  3. The core invokes submitG() and submitNodId() of each element model and accumulate the resulting $[G]$ contributions from elements to the correct positions of the $G$ matrix in (3.3). Link matrices (rightmost few columns in matrix of (3.3)) are filled by calling submitRLnk() and submitNodId of each MATE link.

  4. Block matrices $A$, $B$ and etc of (3.3) are constructed by extracting directly from the network $[G]$ of step 3 at correct positions. Link matrices $p$, $q$, and $-z$ are formed by extracting from link matrices of step 3.
5. The core invokes submitSrc() and submitNodId of each current source object and accumulate respective contributions at correct positions in the accumulated current source vector (vector right of "=" in equation (3.3). $e_{\text{link}}$ vector is filled by invoking submitVLnk() of each link object.

6. The core invokes UpdateHistory() and submitNodId() of each element model and accumulate respective history source contributions at correct positions in the accumulated current source vector of step 5.

7. Accumulated current source vectors for each MATE block ($h_A$, $h_B$) are filled by extracting directly from the accumulated current source vector of step 6.

8. The core computes $Z_a$, $e_a$, and solves for $i_a$ with (3.6c), (3.18) and (3.8) respectively.

9. The core solves for node voltages in each block with (3.9) and output results.
1. Read in simulation parameters
2. Initializing elements, sources, blocks and links
3. Filing up matrix of (3.3)
4. Creating block matrix A, B, link matrix, p, q, and c of (3.3)
5. Accumulating contribution of current sources
6. Updating element history sources and accumulating their contribution
7. Creating current source vector corresponding to each MATE block (h_A, h_B, etc)
8. Compute Z_s and c, and solve for link current i_a
9. Solve for node voltages at each MATE block and output voltages onto file

$t = t + \Delta t$
$t < t_{\text{final}}$
$t \geq t_{\text{final}}$
End of Simulation

Figure 3.12 Solution Algorithm of OVNI
CHAPTER 4
IMPLEMENTATION METHODOLOGY

This chapter studies the implementation methodology of the full frequency-dependent transmission line model, zLine, in complete details. The zLine model, similar to many other element models, belongs to the descendent or child class of elm_t introduced in section 3.2.1. The class zLine_t defines objects of the zLine model. Since a complete transmission line is sectionalized into several equal-length segments in zLine, another class section_t is naturally defined to represent each zLine segment. For instance, if the line length is divided into 10 segments, then 10 objects of type section_t will be generated inside zline_t to represent each segment. The interaction between segments and line dominates the model implementation process in areas of conductance matrix computation and history sources update.

This chapter unveils by giving the input file format and data required for the implementation and execution of the transmission line model. In particular, some pre-processing work, including the execution of mtLine program and synthesis routine, need to be performed to generate the required transmission line model parameters for the time domain simulation. Next, the methods used to compute and update the equivalent line conductance matrix and history sources are discussed in details. Complete data structures for zLine_t and section_t are included in APPENDIX III for reference.
4.1 The Default Constructor and Pre-Processing

In order to simulate the network containing transmission lines modeled as zLine, it is necessary to obtain the required line model parameters. Some examples of those would include \([R(\omega)]\), \([L^\text{int}(\omega)]\) and \([L^\text{ext}]\) for element synthesis, \([Z_c]\) for the modelling of ideal line sections and poles and constants of each R-L fitting block for the modelling of loss sections. It is inside the default constructor of the zLine object where the initialization of these model parameters occurs. To obtain above model parameters, some sorts of step-by-step pre-processing work must first be conducted. In this section the pre-processing tasks, namely reading input data from the input file, running the mtLine, and executing the element synthesis routine, along with constructing zLine segments, will be introduced in the same order as they occur. The preparation work for \([G]\) matrix computation, though occurs inside the constructor as well, is delayed to another section dedicated solely for \([G]\) matrix computation for coherence and clarity.

4.1.1 Input Data and the Input File

At the outset of the simulation as the core invokes each element to initialize its model parameters (Figure 3.12, step 2), the default constructor of zLine_t performs the first pre-processing task – reading from the input file the relevant conductor parameters and configuration. The complete set of input data necessary for the pre-processing of the zLine model is outlined in Figure 4.1 below.
Z1 1 2 3 4 5 6 TLINE
  .MODEL Z1 TLINE (Unit = METRIC,
  Len = 100,
  SecLength = 1,
  NumCondInBundle=1,
  NumGndWire=0,
  Earth-Resistivity=100,
  NumFrequency=9,
  Frequency = [0.1, 1, 10, 100, 1000, 1e4, 1e5, 1e6, 1e-5],
  Rdc = [0.105, 0.105, 0.105],
  Diameter=[24.2062, 24.2062, 24.2062],
  Thickness/Diameter = [.2854, .2854, .2854],
  Height=[12, 12, 12],
  Midspan=[12, 12, 12],
  HzDistance=[0, 2, 4])

Figure 4.1 Input File Format for the zLine Model

The input file format shown in Fig 4.1 mimics the input format used in SPICE as suggested by supervisor Martí. Each line of data in Fig 4.1 will be briefly introduced in the following bullets.

- The first line of the input file is organized in the following manner: model name, node names and model type. Z1 indicates the name of the model- the first zLine model. There can be more line models in the network by naming them Z2 and Z3 etc. The core first reads Z1 to recognize that it is a zLine model and hands the control over to zLine_t so it can read the remaining data. The “node names” field represents the external node ids that the model is connected to at its two ends in the network to be simulated. The node ids are used by the core to insert the conductance matrix and history source vector of the line to appropriate positions in the network conductance matrix and current source vector. A three-phase transmission line, as shown in Fig 4.1, is connected to 6 external nodes, three at each end. The last parameter, model type, indicates that model Z1 is a transmission line, denoted by type TLINE.

- The .MODEL keyword in the second line of Fig 5.1 indicates that data following the open bracket is the data to be read in and processed for a particular element model. In the case of the zLine model, the data consists of tower geometry and
conductor parameters for the transmission line. They later are fed to other routines to generate appropriate model parameters for simulation.

- The **Unit** key word specifies the unit used throughout the input file. There are two choices: **British** stands for British unit (mile, foot) and **Metric** stands for metric unit (kilometer, meter).

- **Len** indicates the length of the line in km or mile.

- **SecLength** represents the length of each zLine segment. Note that the number of segments is equal to **Len/SecLength**.

- **NumCondInBundle** represents the number of conductors that are grouped to form a phase. Line Constant (mtLine) would automatically reduce impedance and admittance matrices of physical conductors to equivalent phase form.

- **NumGndWire** represents the number of ground wires on the tower. In the matrices of physical conductors, rows and columns of ground wires are first eliminated before reducing the matrices to the equivalent phase form.

- **Earth-Resistivity** represents the earth resistivity in Ω-m; the typical value is 100.

- **NumFrequency** along with the **Frequency** vector indicates the number of frequency points that mtLine would generate for its frequency-dependent line parameters. These frequency points are ranging one decade from one another ranging from 0.1 hz to $10^6$ hz. The corresponding impedance matrix for each frequency point is inputed to the matrix fitting routine. Note that the last frequency point (1e-5 Hz) is used to obtain the DC resistance matrix when the frequency is close to DC. The DC resistance matrix contains resistance of each conductor in diagonal elements and zeros in off-diagonal elements.
• **Rdc** and the following 5 vectors contain \((\text{NumConductor} + \text{NumGndWire})\) elements for each conductor and ground wire. **NumConductor** is the parameter representing the total number of conductors for the line, and is equal to \(\text{NumCondInBundle} \times \text{NumPhase}\). **NumPhase** is determined from the first line of the input file, equivalent to half of the number of external nodes. **Rdc** represents the DC conductor resistance per unit length \((\Omega/\text{km} \text{ or } \Omega/\text{mile})\). In a bundle of conductors, the conductors are placed in parallel and the equivalent resistance per phase can be found accordingly.

• **Diameter** contains the diameter in mm or inches for each conductor and ground wire of the line.

• **Thickness/Diameter** contains the ratio between conductor thickness and diameter for each conductor and ground wire. This parameter is used by mtLine for the correction of skin effect in resistance and internal inductance.

• **Height, Midspan, and HzDistance** are geometries for each conductor and ground wire. Height and Midspan specify the height of the conductor at the tower and at midspan in meters or feet. The average height can be calculated from the two parameters by \(h_{\text{average}} = h_{\text{midspan}} + \frac{1}{3}(h_{\text{tower}} - h_{\text{midspan}})\). Note that, when Height and Midspan are equal, \(h_{\text{average}} = h_{\text{midspan}} = h_{\text{tower}}\). Usually the heights specified on a tower diagram are average conductor heights. **HzDistance** specifies the horizontal distance of each conductor with respect to a reference point \(x = 0\) (m or ft). The reference point can be chosen arbitrarily along the x-axis with a negative value to the left of the reference point, and a positive value to the right.

A subroutine **ReadSPICEInput()** is created to facilitate the input file reading process. Furthermore, since the second stage of pre-processing involves execution of mtLine, the data read from Figure 4.1 must be organized into a compatible format readable by the program. **ReadSPICEInput()** would also perform this task by arranging conductor...
characteristics of Figure 4.1 into separate conductor and frequency cards as defined in the mtLine reference manual [19]. The input file generated by ReadSPICEInput for the execution of mtLine is shown below:

```
*Input File For mtLine
Case Example For The Z-line Model

*Unit Card
METRIC

*Conductor Card
Rdc
diameter height midspan
1 0.285 0.10504 24.2062 0.000 12.00 12.00
2 0.285 0.10504 24.2062 2.000 12.00 12.00
3 0.285 0.10504 24.2062 4.000 12.00 12.00
&END

*Frequency Card
rhofreq[C][P][Z]lengthPlModal
100 1.00e-001 1 1 1 1-100.000 1 1
100 1.00e+000 1 1 1 1-100.000 1 1
100 1.00e+001 1 1 1 1-100.000 1 1
100 1.00e+002 1 1 1 1-100.000 1 1
100 1.00e+003 1 1 1 1-100.000 1 1
100 1.00e+004 1 1 1 1-100.000 1 1
100 1.00e+005 1 1 1 1-100.000 1 1
100 1.00e+006 1 1 1 1-100.000 1 1
100 1.00e-005 1 1 1 1-100.000 1 1
&END
&END
```

Figure 4.2 Input File for mtLine

### 4.1.2 Running mtLine Program

The program mtLine, developed by Microtran Power System Analysis Corporation, calculates parameters of an overhead transmission line based on conductor characteristics and tower geometry given of Figure 4.1. mtLine requires one conductor card for each conductor and ground wire. More than one frequency cards are allowed to obtain line parameters at different frequencies. The input file for mtLine is shown in Figure 4.2. Note that for each conductor card, the first column indicates the phase number that a conductor belongs to. For bundled configurations, conductor with the same phase number will be grouped to the same phase. The "1" for each frequency card indicates the type of line parameters that will be generated by mtLine at each given frequency. For the purpose of this thesis work, mtLine will generate the capacitance matrix, [C], Maxwell coefficient Matrix, [P], and the impedance matrix, [Z], all in phase coordinate for each frequency point.
In order to execute mtLine inside the default constructor of zLine_t automatically at run-time, an executive routine is required to remotely execute the .exe file of the program. Several exec routines [19, 20] are available and were tested for their performance and versatility until finally the routine \texttt{system(const char *command)} was selected as the tool for the task at hand. The major advantage for \texttt{system} lies in its simplicity to use and most importantly, the fact that it is ANSI C supported. In other words, unlike most exec routines tested so far that only operate under the MS Windows environment, \texttt{system} will function in different platforms because its header file inclusion originates from the stdlib.h of ANSI C. It is believed that it can be compiled with the general g++ compiler while other similar routines cannot. The function call to \texttt{system} looks like:

\begin{equation}
\texttt{system("C:\MT310\MTLINE.exe mtline.dat")}
\end{equation}

where the first formal parameter of the routine specifies the path to the executable file, mtline.exe, which runs the Line Constant program. The second formal parameter is the file name for the input file of Figure 4.2. mtLine would automatically execute the input file mtline.dat with the above function call, relieving users the trouble of typing. More arguments are allowed to follow if necessary.

The output file generated by mtLine is shown in Figure 4.3 below. Note that Figure 4.3 only shows the transmission line parameters at frequency of 0.1 Hz. Parameters for eight other frequencies follow the order of respective frequency cards. Moreover, only the elements below the diagonal line of the matrices are displayed because transmission line matrices are symmetrical. This characteristic is very advantageous to later synthesis of the loss matrix since only elements above or below the diagonal line need to be synthesized. Other symmetrical elements would share the same fitted poles and constants. Exploitation of this property would save considerable amount of floating point operations in matrix synthesis and computation of loss section parameters.
FREQUENCY CARD REQUEST: Earth resistivity = 0.10E+03 ohms-m, Frequency = 0.10 Hz.

Carson formula used for earth return impedances

INVERTED CAPACITANCE MATRIX (km/F) FOR THE SYSTEM OF EQUIVALENT PHASE CONDUCTORS

Rows and columns proceed in same order as sorted input.

1 0.13647E+09 //P
2 0.44729E+08 0.13647E+09 //P
3 0.32453E+08 0.44729E+08 0.13647E+09 //P

CAPACITANCE MATRIX (F/km) FOR THE SYSTEM OF EQUIVALENT PHASE CONDUCTORS

Rows and columns proceed in same order as sorted input.

1 0.83883E-08 //C
2 -0.23476E-08 0.88663E-08 //C
3 -0.12253E-08 -0.23476E-08 0.83883E-08 //C

IMPEDEANCE MATRIX (ohm/km) FOR THE SYSTEM OF EQUIVALENT PHASE CONDUCTORS

Rows and columns proceed in same order as sorted input.

1 0.10510E+00 //R, 0.18271E-02 //(oL,
2 0.98570E-04 0.10510E+00 //R, 0.11627E-02 0.18271E-02 //aL12 \aL22
3 0.98570E-04 0.98570E-04 0.10510E+00 //R, 0.10756E-02 0.11627E-02 0.18271E-02 //aL13 \aL23 \aL33

Figure 4.3 Sample mtLine output for the 3 Phase Line of Figure 4.1 and 4.2

At this stage of pre-processing, another data reading subroutine ReadFileMTLine( ) is implemented to decipher the output file of Figure 4.3 and read in the required transmission line matrices such as [R(ω)], [L(ω)], [C] and [P] for [Z_{loss}(ω)] synthesis. Matrices elements are arranged by ReadFileMTLine to create another input file in the format required by the matrix fitting routine.

4.1.3 Running Matrix Synthesis Routine

The original matrix fitting routine, written by Ting-Chung Yu [15] for underground cable synthesis, has been modified in this thesis to synthesize the loss impedance matrix of overhead transmission lines. The fitting algorithm is provided in Appendix I. The matrix fitting routine takes an input file containing the following parameters:
1. The elements $R_{ij}(\omega)$ and $L_{ij}(\omega)$ above the diagonal line of respective per-unit inductance matrix, $[L]$, and resistance matrix, $[R]$. For each upper diagonal element there are 8 R-L pairs corresponding to 8 different frequency points in the frequency cards of Figure 4.2.

2. $R_{ij}^{dc}$ above the diagonal line of the DC resistance matrix, $[R^{dc}]$. $[R^{dc}]$ contains DC resistance of each conductor or bundle of conductors in its diagonal elements and zeros in off-diagonal elements. $[R^{dc}]$ is the resistance matrix generated by mtLine with the last frequency card of Figure 4.2.

3. $L_{ij}^{ext}$ above the diagonal line of the external inductance matrix, $[L^{ext}]$. $[L^{ext}]$ are calculated by applying equation (2.6). Since $[L^{ext}]$ is constant and independent on frequency, there is only one value for each upper diagonal element.

4. The tolerance of how accurate the fitted functions are to match the loss matrix data at each frequency point.

A sample input file for the matrix fitting routine is shown in Figure 4.4 below:

```
6.66000e-006 //Delta
0.10510000000 0.00290792001 //R_{i1}(\omega) L_{i1}(\omega)
0.10598000000 0.00267810002 //R_{i1}(\omega) L_{i1}(\omega)
0.11476000000 0.00244923546
0.20132000000 0.00223370628
1.06410000000 0.0019596218
7.78420000000 0.00178474964
48.14900000000 0.00156386092 //R_{i1}(\omega) L_{i1}(\omega)
219.52000000000 0.00139173042
0.00099870000
0.00099830100
0.00097470000
0.00094910000
0.00094848390
7.24020000000 0.00075507881
46.40000000000 0.00061282612
213.38000000000 0.00053816567 //R_{i1}(\omega) L_{i1}(\omega)

//R_{ij}(\omega) L_{ij}(\omega) of line parameters for all upper diagonal elements
0.105000 0.000000 0.000000 0.105000 0.000000 0.105000 0.000000 0.105000 //R_{ij}^{00} for upper diagonal elements
0.001516353 0.00049409455 0.001516358 0.00049409455 0.001516353 0.00049409455 //L_{ij}^{00}
0.0010 //tolerance
```

Figure 4.4 Input File for the Fitting Routine
The matrix fitting process is divided into the following tasks:

- **Pre-processing**

  The fitting routine initializes storage vectors for $R_{ij}(\omega)$, $L_{ij}(\omega)$ at 8 fitting frequencies and variables for $L_{ij}^{\text{ext}}$ and $R_{ij}^{\text{ext}}$ by reading from the input file of Figure 4.4. It also creates a frequency array to store the 8 fitting frequencies, $\omega_1$ to $\omega_8$, corresponding to the 8 frequencies in the frequency card. Next the fitting routine determines the number of fitting blocks necessary for an accurate fitting of original $Z_{ij}^{\text{loss}}$ for a particular $\Delta t$ chosen to simulate the network. From sampling theorem, the maximum frequency that can theoretically be represented for a given $\Delta t$ is $0.5/\Delta t$ Hz. Since the approximation of $[Z^{\text{loss}}(\omega)]$ with R-L fitting blocks is modelled in time domain with trapezoidal rule in (2.15), even more stringent restriction is imposed on the maximum frequency attainable. In fact, it is $0.1/\Delta t$ Hz for discretization with trapezoidal rule. Consequently, it is wasteful in synthesis time to have more fitting blocks than necessary to achieve a given bandwidth. Since there are a total of 8 fitting frequencies (0.1Hz to 1MHz) for each decade, corresponding to 8 fitting blocks capable of simulating up to the MHz range, the number of fitting blocks needed to represent up to a maximum frequency of interest can be selected by:

\[
\begin{align*}
\text{if } &\text{ } f_{\text{max}} < 10 \text{ MHz } \text{ then numblk } = 8; \quad \text{and} \\
\text{if } &\text{ } f_{\text{max}} < 1 \text{ MHz } \text{ then numblk } = 7; \quad \text{and} \\
\text{if } &\text{ } f_{\text{max}} < 100 \text{ KHz } \text{ then numblk } = 6; \quad \text{and} \\
\text{if } &\text{ } f_{\text{max}} < 10 \text{ KHz } \text{ then numblk } = 5; \quad \text{and} \\
\text{if } &\text{ } f_{\text{max}} < 1 \text{ KHz } \text{ then numblk } = 4; \quad \text{and} \\
\text{if } &\text{ } f_{\text{max}} < 100 \text{ Hz } \text{ then numblk } = 3; \quad \text{and} \\
\ldots
\end{align*}
\]

**Figure 4.5 Number of Fitting Blocks for Accurate Synthesis of $[Z^{\text{loss}}(\omega)]$**

As a numerical example, $\Delta t$ of 6.666 $\mu$sec can effectively simulate up to 15 KHz in time domain with trapezoidal integration. Figure 4.5 shows that 6 fitting blocks are sufficient to synthesize $[Z^{\text{loss}}(\omega)]$ for time-domain simulation up to that frequency.
Fitting and Optimization

After determining the number of fitting blocks required, the fitting routine would start the fitting process. It first calculates $Z_{ij \omega_{numblk}}^{loss} = R_{ij \omega_{numblk}} + j\omega_{numblk}L_{ij \omega_{numblk}}$ with the following equation:

$$Z_{ij \omega_{numblk}}^{loss} = R_{ij \omega_{numblk}} + j\omega_{numblk}L_{ij \omega_{numblk}} \left[ L_{ij \omega_{numblk}} - L_{ij}^{ext} \right] \quad (4.2)$$

The fitting routine applies the synthesis procedures (I.1 to I.11) summarized in Appendix I to compute the fitting blocks that synthesize $Z_{ij}^{loss}$. Optimization iterations are next applied until the magnitude response of the fitting functions at each fitting frequency falls within the acceptable tolerance set in the input file. In mathematical forms using expressions of equation (2.13),

$$|Z_{ij \omega_{numblk}}^{loss} - Z_{rij(i)}^{loss}| \leq \text{tolerance} \quad (4.3a)$$

and

$$|Z_{i\omega_{numblk}}^{loss} - Z_{rf(ij)}^{loss}| \leq \text{tolerance} \quad (4.3b)$$

where

$$Z_{rij(i)}^{loss} = R_{i\omega_{numblk}} + \sum_{l=1}^{m} \frac{s K_{r(i)l}}{s + P_{r(i)l}} \quad (4.3c)$$

$$Z_{rf(ij)}^{loss} = \sum_{l=1}^{m} \frac{s K_{r(j)l}}{s + P_{r(j)l}} \quad (4.3d)$$

The resulting pole-constant and resistor-inductor pairs of each fitting block that synthesizes $Z_{ij}^{loss}$ are printed to an output file. Also included are the $\Delta t$ used for the simulation, the maximum frequency attainable and the number of fitting blocks selected accordingly. Part of the output file generated by the fitting routine is shown in Figure 4.6 below:
Recall from Chapter 2, in order to ensure numerical stability of fitted functions, all elements must be fitted with the same set of poles. Figure 4.6 shows that the fitting output for $Z_{11}^{\text{loss}}$ and $Z_{22}^{\text{loss}}$ shares a very close set of poles. For fitting results of off-diagonal elements, the same agreement is also demonstrated.

Similar to the way mtLine is run, the fitting routine must also be executed inside the default constructor at run time. The ANSI C supported routine system is used to execute the .exe file of the fitting routine in the same manner as described in Section 4.1.1.

4.1.4 Initializing Line Model Parameters

The tasks to initialize line model parameters in preparation for the time-domain simulation are two folds – initializing parameters for the ideal line sections and computing the parameters for the loss sections. These parameters will be passed on to
each zLine segment later when segment objects are created. The initialization process will be discussed in this subsection.

The parameter for the ideal line section includes the section length for each ideal section, the transmission delay, $\tau$, and the characteristic impedance $[Z_c]$. The section length is virtually the length of each zLine segment and is read from the input file of Figure 4.1. The number of segments the zLine model is divided into can be calculated by:

$$\text{Number of segments} = \frac{\text{total line length}}{\text{segment length}} \quad (4.4)$$

The transmission delay, $\tau$, for the ideal line section can be computed by

$$\text{Transmission delay} = \frac{\text{section length}}{c_0} \quad (4.5)$$

where $c_0$ is the speed of light depending on the unit specified in the input file.

The routine ReadMTLine reads in the Maxwell Coefficient Matrix from the mtLine output file (Figure 4.3) and stores its value in a matrix $P[i][j]$. The characteristic impedance matrix $Z_c$ can be calculated by applying equation (2.12):

$$[Z_c] = \frac{1}{c_0} [P] \quad (2.12)$$

The model parameters for the loss section, on the other hand, include the equivalent resistance, $[R_{eq}]$, and equivalent history source, $[H(t)]$, in phase coordinate introduced in Section 2.3.2. $[R_{eq}]$ and $[H(t)]$ are computed from poles and constants of the fitting blocks that synthesize elements of $[Z^{\text{loss}}(\omega)]$.

A subroutine `ReadFileFitRoutine` is written to read the poles and constants corresponding to each fitting block of $Z^{\text{loss}}_{(ii)}$ from the fitting routine output file (Figure 4.6). In C++ implementations, two 3-dimensional arrays are designed to store the values
of these poles and constants. As an example, a 3-D array, Pole\([i][j][k]\), effectively stores the poles for all fitting blocks. In particular, indices \(i\) and \(j\) make references to element \(Z_{ij}^{loss}\) while index \(k\) make reference to the \(k^{th}\) fitting block of that element. Note that for the matrix symmetry reason, poles for lower diagonal elements are the same as upper diagonal counterparts. This means that Pole\([i][j][k]\) and Pole\([j][i][k]\) contain the same values. Same property applies to constant \(K[i][j][k]\) as well.

Once all poles and constants have been stored into respective arrays, eqn (2.16) and (2.17) can be applied to compute \([R_{eq}]\). Same equations are revisited below in terms of the array structure mentioned above:

\[
R_{eq}[i][i] = R_{ide} + \sum_{k=1}^{m} \frac{2K[i][j][k]}{\Delta t} . 1 \quad \text{for diagonal element} \quad (4.6a)
\]

\[
R_{eq}[i][j] = \sum_{k=1}^{m} \frac{2K[i][j][k]}{\Delta t} . 1 \quad \text{for off - diagonal element} \quad (4.6b)
\]

The equivalent history sources \(h_{eq[i][j]}(t)\) for each fitting block (2.17b), which is used to compute the equivalent phase-domain history vector \([H(t)]\) (2.17a), is stored with the same 3-D array, \(h_{eq[i][j][k]}\). Reiterating eqn. (2.17) in 3D arrays, we obtain:

\[
H_{i}(t) = \sum_{j=1}^{NumPhase} \sum_{K=1}^{NumBlock} h_{eq[i][j][k]}(t)
\]  

\[
h_{eq[i][j][k]}(t) = \frac{2}{\Delta t} P[i][j][k] h_{eq[i][j][k]}(t - \Delta t) - \frac{4K[i][j][k]P[i][j][k]}{\Delta t} . 1 .
\]

Note in eqn (4.6) and (4.7) that “1” is equal to one-half segment length because the loss section model has been split in half and inserted to both ends of the ideal line section.
Since the equivalent resistance \([R_{eq}]\) is a constant parameter, it can be pre-calculated before simulation starts. \([H(t)]\), however, needs to be updated at every time step because it depends on the history source and currents at the previous time step. A considerable amount of floating point operations are required to compute all elements of \(h_{eq}[i][j][k](t)\) at every time step. However, if we look into equation (4.7b) more carefully, it shows that the majority amount of computations occur in manipulations on poles and constants. In fact, if those operations can be pre-calculated for every array element \(i, j, k\) before the simulation loop actually starts, considerable amount of saving in numerical computations can be gained. In other words, we can simplify \(h_{eq}[i][j][k]\) into the following expression:

\[
h_{eq}[i][j][k](t) = CoeffA[i][j][k].h_{eq}[i][j][k](t - \Delta t) - CoeffB[i][j][k].I_j(t - \Delta t) \quad (4.8)
\]

where \(CoeffA\) and \(CoeffB\) are now two other 3-D storage matrices containing the results of operations on poles and residues in (4.7b) and can be pre-calculated in the constructor. Later on when history sources are updated with (4.8), the two coefficient matrices can be plugged in directly. Further saving in computation can be gained if \([R_{eq}]\), \(CoeffA\) and \(CoeffB\) are computed together for some common multiplying terms.

### 4.1.5 Constructing zLine Segments

In zLine's sectionalized model, the segment forms the basic building block for the modelling of the line. Each zLine segment is represented in objected-oriented paradigm as an object of the section_t class whose data structure is provided in Appendix III. Initialization of zLine segments at this stage is necessary as all operations for the transmission line, such as conductance matrix computation and history source update, occur first in the segment level. Each segment returns its equivalent conductance matrix and history sources to the line level where they are accumulated to form the line conductance and history source contribution to the entire network. The structure of section_t is very similar to its upper layer companion zLine_t in the manner that it also contains a constructor in charge of initializing parameters relevant to each zLine segment, a submitG and an UpdateSecHistory function responsible for returning the conductance.
matrix and history source contributions to the transmission line level. This subsection focuses mainly on creation of zLine segments. The method to construct the line conductance matrix and update history sources will be relayed to later sections of this chapter when appropriate.

A recap of the zLine model and the segment model is provided in Figure 4.7 below. As can be seen, the full-length transmission line model is created by connecting \( N \) physically identical segment models of Figure 4.7(b) in series. \( N \), the number of segments in the zLine model, can be found with equation (4.5).

![Diagram showing transmission line level and segment level models](image)

(a) Transmission Line Level

Figure 4.7 ZLine Model: a) Line Level  b)Segment Level

The zLine constructor initializes each segment by first creating an array of \( N \) pointers to type section_t, where \( N \) is the number of segment. These \( N \) pointers are used to dynamically allocate \( N \) objects of type section_t. Dynamic allocation in C++ can be done with the \texttt{new} operator. A short line of code would best illustrate the process:

\[
apSection[i] = \text{new section}_t(\text{model parameter already computed})
\]

\( i = 1 \) to \( N \);
The N objects created corresponds to N zLine segments and each of them is accessible with the “handle” apSection[i].

Upon creation of segment object with (4.9), the constructor for each segment object is automatically invoked. Since all segments possess the same ideal line section and two half-segment-length loss section models, the constructor of zLine_t would simply pass corresponding model parameters to the segment constructors as arguments so that ideal line section and loss section parameters for every segment can be initialized immediately upon creation.
4.2 Constructing Line Conductance Matrix

The tasks of constructing the line conductance matrix begin at the segment level. The segment conductance matrix contribution for each zLine segment is first computed inside the constructor of each segment model. These segment matrices are returned to the line level by the call to SubmitG() of each segment model. Segment matrices are accumulated and reduced to form the equivalent conductance matrix of the line. A hierarchical structure would illustrate this process clearly:

![Diagram](image)

Figure 4.8 Constructing Conductance Matrix for ZLine Models

The method used to create segment conductance matrix would be given in Section 4.2.1. In Section 4.2.2, the method used to accumulate matrix contributions from the segments are provided.

4.2.1 Creating Segment Conductance Matrices

For a segment model of Figure 4.7b, the conductance matrix in terms of composite external nodes a1, a2 and internal nodes b1, b2 can be created. For a multi-phase model as in Figure 4.7b, nodes a1 to a2 shown in the figure each consists of multi-nodes and the
impedances are in matrix quantities. The node hiding technique introduced in Section 3.1.2 is used to reduce the equivalent segment conductance contribution to dimensions of external nodes only. The segment conductance matrix would be filled in the same manner as single-phase networks would; that is, diagonal elements consist of sum of conductance connected to each node and off-diagonal elements contain the negative conductance between the two neighbouring nodes in the network. The complete segment conductance matrix for the segment model in Figure (4.7b) would look like as follows:

![Segment Conductance Matrix](image)

To compute the equivalent conductance matrix, \([G^\text{segment}_{aa}]\), in terms of external nodes a1 and a2, the following sub-matrices are required. In the following expressions, a represents external nodes a1, a2; b represents internal nodes b1, b2.

\[
[G_{aa}] = \begin{pmatrix}
a1 & a2 \\
\[R_{eq}\]^{-1} & 0 \\
0 & \[R_{eq}\]^{-1}
\end{pmatrix} \quad (4.10)
\]

\[
[G_{ab}] = \begin{pmatrix}
b1 & b2 \\
-\[R_{eq}\]^{-1} & 0 \\
0 & -\[R_{eq}\]^{-1}
\end{pmatrix} \quad (4.11)
\]

\[
[G_{ba}] = \begin{pmatrix}
b1 & b2 \\
-\[R_{eq}\]^{-1} & 0 \\
0 & -\[R_{eq}\]^{-1}
\end{pmatrix} \quad (4.12)
\]
Once the four sub-matrices have been created and initialized, \( [G_{bb}] \) can be formulated with eqn (3.14):

\[
[G_{aa}] = [G_{aa}] - [G_{ab}][G_{bb}]^{-1}[G_{ba}]
\]  

(4.14)

Since each sub-matrix is zero in off-diagonal elements and contains the same diagonal elements, \( G_{aa}^{segment} \) can be expressed in the similar form:

\[
[G_{aa}^{segment}] = \begin{bmatrix}
    a1 & a2 \\
    a1 X & 0 \\
    a2 & 0 X
\end{bmatrix}
\]  

(4.15)

where \( X \) denotes a non-zero diagonal element matrix.

### 4.2.2 Accumulating and Reducing Segment Matrices

When the zLine constructor invokes SubmitG of each zLine segment object, the equivalent segment conductance matrices of (4.15) are returned to the line level and accumulated to form the conductance matrix of the line. The simplified zLine model marked with node names for clarity is shown in Fig 4.10 below:
The number of nodes represented by each node name depends on the phase number of the transmission line. In general for a \( n \) phase transmission line dividing into \( N \) segments, there will be \((N+1)n\) nodes in total; among them \(2n\) are external nodes, \((N-1)n\) are internal ones. These quantities are kept track of by the \( z \text{Line} \) constructor to allocate matrices of right size for \([G_{aa}], [G_{ab}], [G_{ba}]\) and \([G_{bb}]\) so that the node-hiding technique can be utilized to formulate the equivalent conductance matrix of the line.

In order to assemble the conductance matrix for the transmission line, the constructor would first allocate a storage matrix, \([G_{line}]\), with size \((N+1)n \times (N+1)n\). Next, the constructor would invoke the \( \text{submitG()} \) function of each \( z \text{Line} \) segment. Each segment returns its \([G_{segment}]\)'s back to the constructor. The constructor accumulates the \([G_{aa}]\)'s from each segment to the proper positions in \([G_{line}]\) according to the nodes the segment is connected to in Figure 4.10. Visual diagrams would best illustrate this matrix filling procedure.

1. Fill \([G_{line}]\) from first segment connected between composite nodes \( a1 \) and \( b1 \)

![Figure 4.11 Filling \([G_{line}]\) From Segment 1](image-url)
The conductance matrix, $[G_{\text{line}}]$, is initialized with zeros when first allocated. The conductance matrix for segment 1 can be directly copied to the appropriate location in $G_{\text{line}}$; that is, in rows $a1,b1$ and columns $a1,b1$.

2. Fill $[G_{\text{line}}]$ from second segment connected between composite nodes $b1$ and $b2$

![Diagram of segment and conductance matrix](image)

Figure 4.12 Filling $[G_{\text{line}}]$ From Segment 2

The conductance matrix contribution from segment two is inserted to rows $b1$, $b2$ and columns $b1$, $b2$ the same way as segment 1 except that there will be two elements overlapping each other at position $(b1,b1)$. The two overlapped elements are accumulated to form a new element as shown in the figure above.

3. Follow the same procedure until the conductance matrices for all segments are inserted. Resulting $[G_{\text{line}}]$ will look like as follows:

![Table of conductance matrix](image)

69
As soon as \([G_{\text{line}}]\) of Figure 4.12 has been constructed, the last initialization task of the zLine constructor, similar to its segment counterpart, is to compute the equivalent conductance matrix contribution of the line to the entire network. As before, submatrices \([G_{aa}],[G_{ab}],[G_{ba}]\) and \([G_{bb}]\) are to be extracted from \([G_{\text{line}}]\) of Figure 4.12.

\[
G_{aa} = \begin{bmatrix} a1 & a2 \\ a1 & X & 0 & 0 & 0 & 0 \\ a2 & 0 & X \end{bmatrix}
\]

\[(4.16)\]

\[
G_{ab} = \begin{bmatrix} b1 & b2 & b3 & \ldots & bN \\ a1 & 0 & 0 & 0 & 0 & 0 \\ a2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\]

\[(4.17)\]

\[
G_{ba} = \begin{bmatrix} a1 & a2 \\ b1 & 0 & 0 \\ b2 & 0 & 0 \\ b3 & 0 & 0 \\ \vdots & 0 & 0 \\ bN & 0 & 0 \end{bmatrix}
\]

\[(4.18)\]
Once more, equation (3.14) are applied to compute for \([G_{ba}^{\text{line}}]\), the equivalent conductance matrix of the line.

\[
\begin{bmatrix}
 b1 & b2 & b3 & \ldots & bN \\
 b1 & 2X & 0 & 0 & 0 \\
 b2 & 0 & 2X & 0 & 0 \\
 b3 & 0 & 0 & 2X & 0 \\
 \vdots & 0 & 0 & 0 & 2X \\
 bN & 0 & 0 & 0 & 2X \\
\end{bmatrix}
\] (4.19)

Once more, equation (3.14) are applied to compute for \([G_{aa}^{\text{line}}]\), the equivalent conductance matrix of the line.

\[
[G_{aa}^{\text{line}}] = [G_{aa}] - [G_{ab}] [G_{bb}]^{-1} [G_{ba}]
\] (4.20)

\([G_{aa}^{\text{line}}]\) for the line is returned to the core when the member function submitG( ) of the zLine module is invoked. The core inserts \([G_{aa}^{\text{line}}]\) into the proper positions of network \([G]\) according to the nodes the line is connected to.
4.3 Updating History Sources

Recall from Chapter 3 on simulation algorithm of the current OVNI engine, the simulator at each time step would prompt each element model to update and return its history source vector at external nodes to the core. It is the responsibility of each element model to retrieve and process necessary information to compute its history source contributions. The steps to update history sources for the zLine model, like the tasks of computing equivalent conductance matrix, begin at the segment level. Equivalent history sources for each segment include the combination of history sources from both the ideal line section and loss sections. History sources for the ideal line and loss sections have to be updated separately, and assembled to form the equivalent history source vector for each zLine segment. History sources from the segment levels are accumulated and reduced with node-hiding to generate the equivalent history vector of the line. A graph can best illustrate this updating process.

The tasks to update history sources for zLine models involve: a) retrieving and updating external node voltages for the line and segments; b) updating and assembling history sources for each zLine segment; c) accumulating and reducing segment history sources. Each of the above tasks will be explained in details.
4.3.1 Retrieving and Updating Node Voltages

The first step in the attempt to update element history sources at every time step is to retrieve external node voltages from the core. External node voltages solved in previous time steps are used to compute branch currents and update history sources. The node voltage for every node in the network is stored in an global array whose elements are managed in increasing order of numerical node names. The zLine model retrieves external node voltages corresponding to the node names it is connected to. Taking the model of Figure 4.10 for example, external node voltages $V_{a1}$ and $V_{a2}$ are retrieved from the global storage array.

However, as mentioned in the outset of this section, segment history sources, including history sources for both the ideal line section and loss section, need to be updated first for all zLine segments. Simply having node voltages $V_{a1}$ and $V_{a2}$ is not sufficient for updating history sources inside every zLine segment. All internal node voltages for the line model are also required as they represent external node voltages for zLine segments. Equation (3.11b) of the node hiding technique, repeated here for reference, is applied to update the internal voltages of the line model.

$$[V_b] = [G_{bb}]^{-1}([h_b] - [G_{ba}][V_a])$$

(3.11b)

Note that for every time step, $[V_a]$ and $[h_b]$ are known vectors representing external node voltages and internal history source vector for the line solved at the previous time step. $[G_{bb}]$ and $[G_{ba}]$ are also constant sub-matrices formulated already in the constructor in (4.17) and (4.19). $[V_b]$, therefore, can be uniquely determined with (3.11b) at every time step.

Once all external and internal node voltages have been obtained at the line level, they are organized in a storage vector in the order they appear in the line model of Figure 4.9. A glimpse of the storage vector would look like the following:
Figure 4.15 Storage Vector for Node Voltages of the Line

The vector of Figure 4.15 is passed as an argument in the function call UpdateSecHistory of each zLine segment. The first segment would extract from Figure 4.15 the external voltages at its two ends, \( V_{a1} \) and \( V_{b1} \). Next it applies equation (3.11b) to compute its internal voltages.

Internal voltages for a zLine segment represent node voltages, \( V_k \) and \( V_m \), for the ideal line section in Figure 4.7b. In order to update history sources for the ideal line section, the currents flowing through the left and right branches are needed. Eqn (2.10), outlined here for reference, is applied to calculate the branch currents, \( I_k \) and \( I_m \).

\[
[I_k(t)] = [Z_c]^{-1} [V_k(t)] - [h_k(t)] \tag{2.10a}
\]
\[
[I_m(t)] = [Z_c]^{-1} [V_m(t)] - [h_m(t)] \tag{2.10b}
\]

The same steps are followed by the remaining zLine segments, one after another, each extracting its corresponding external node voltages and computing its internal node voltages and branch currents. Each segment would in turn use these voltages and currents to update its history sources for both the ideal line section and loss sections and later on return these history sources back to the line level. The methods to update history sources will be explained in the next two sub-sections.

4.3.2 Updating History Sources for Ideal Line Sections

Eqn (2.11) shows the equations to update history sources of ideal line sections.

\[
[h_k(t)] = [Z_c]^{-1} [V_m(t - \tau)] + [I_m(t - \tau)] \tag{2.11a}
\]
\[
[h_m(t)] = [Z_c]^{-1} [V_k(t - \tau)] + [I_k(t - \tau)] \tag{2.11b}
\]
Eqn (2.11) indicates that voltage and current vectors at $\tau$ seconds before are required to update history sources $[h_k(t)]$ and $[h_m(t)]$ of the ideal line section. If the time step, $\Delta t$, is purposely chosen to be a sub-multiple of the transmission delay, $\tau$, voltage and current solutions at $\tau$ seconds before can be found by storing the solutions of previous time steps in tables. However, in order to avoid wasting valuable memory space to store voltage and current solutions for every time step, a "circular array" storage approach is taken. In this scheme, only solutions up to $\tau/\Delta t$ time steps need be stored. A cursor moves around these storage arrays to access the storage value so that at every time step the elements being accessed are the solutions $\tau$ seconds ago.

Assuming $\tau$ is five times $\Delta t$, an storage array for $[V_k(t)]$ of size $n \times 5$ can be created, where $n$ denotes the number of phase of the line. The array is initialized with zeros and an index pointing at its first column. Figure 4.16 shows this array:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{k1}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$V_{k2}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$V_{k3}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$V_{kn}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.16 Example of the Circular Storage Array

Recall from the previous subsection, a zLine segment would extract from a vector of line level node voltages for its corresponding external node voltages. It then computes the internal voltages, $[V_k]$ and $[V_m]$, and branch currents, $[I_k]$ and $[I_m]$. Every time these vector values are obtained, they are stored in the correct positions of their respective storage arrays. The general procedures for updating history sources of the ideal line section using the circular array scheme is explained as follows:

1. For the first time step, as there are no solutions for the previous time step, it is not necessary to retrieve external and compute internal voltages for a zLine segment.
Instead, as can be seen in Figure 4.17, \([V_k(t-\tau)], [V_m(t-\tau)], [I_k(t-\tau)]\) and \([I_m(t-\tau)]\) values are taken directly from the first column of their respective storage arrays. At the first simulation step, \([V_k(t-\tau)], [V_m(t-\tau)], [I_k(t-\tau)]\) and \([I_m(t-\tau)]\) are expected to be zero, and this explains the reason the storage arrays are initialized with zeros when created. Equation (2.11) is applied to update history sources, \([h_k]\) and \([h_m]\).

![Figure 4.17 History Source Update Scheme at t = Δt](image)

2. From the second time step on, each zLine segment would first retrieve and compute node voltages and branch currents solved at the previous time step. They are stored in columns of their respective storage arrays pointed by the current index pointer. For example, \([V_k(\Delta t)], [V_m(\Delta t)], [I_k(\Delta t)]\) and \([I_m(\Delta t)]\) values obtained at the second time step are stored in column 1 of their respective storage arrays because column 1 is the position where vector values were extracted in the previous time step to update history sources. Next, the index pointer is incremented by 1 position. \([V_k(t-\tau)], [V_m(t-\tau)], [I_k(t-\tau)]\) and \([I_m(t-\tau)]\) values are taken from the columns pointed by the new index positions. Equation (2.11) is again applied to update the history sources. The update process at \(t = 2\Delta t\) is illustrated in Figure 4.18. Note that \(V_{k,1\Delta t}\) denotes \(V_k\) solutions solved at the first time step, but retrieved at the second.
3. The same procedures outlined in step 2 are repeated for every time steps. In this manner, every time a vector value is taken from a column of the storage array, it is the value solved at $\tau$ seconds or $5\Delta t$ before. As the index pointer reaches the end of the storage array, it wraps around and points again to the first element. Figure 4.19 shows the history source update process when $t = 6\Delta t$. Note that as voltage solutions for the previous time step ($t = 5\Delta t$) have been stored in the appropriate position of the storage array, the index wraps around to the first column. The voltage vector to be extracted at this position, as we recall, was the solutions for the very first time step ($t = \Delta t$). Therefore, this update scheme is proven to be mathematically and logically correct as $V_k(t-\tau)$ is equal to $V_k(6\Delta t-5\Delta t)$ or $V_k(\Delta t)$, which is exactly what we are extracting from column 1.

One important assumption for the above history update scheme to become valid is that $\tau$ is an exact multiple of $\Delta t$. In this manner, solutions at $\tau$ seconds before can always
be located in appropriate columns of the storage array. For a transmission line modelled
as zLine, the maximum $\Delta t$ possible for time-domain simulation corresponds to the
travelling delay of the ideal line section. $\Delta t$, therefore, are selected as large as $\tau$ for the
zLine segment to enhance simulation efficiency and reduce simulation time. If faster
transient phenomena are to be simulated, shorter segments, which indicate smaller $\tau$ and
$\Delta t$, can be utilized.

Making $\Delta t$ as large as $\tau$ is ideal only when there is only one transmission line
modelled as zLine or many zLine models equivalent in segment length. Imagine that
there are two zLine models different in segment length, $\tau$ would apparently be different,
too. In this scenario, $\Delta t$ can no longer be chosen to match $\tau$ of both transmission line
segments. One of the $\tau$’s may not be an exact multiple of $\Delta t$, which makes solutions at $(t-
\tau)$ locate in between two neighbouring integer time steps. To counter this problem, a
linear interpolation technique is devised to approximate the solutions between those two
neighbouring time steps. The history source update scheme using linear interpolation
will be introduced next.

Assume now that $\tau$ is no longer an integer multiple of $\Delta t$, say $\tau = 5.2\Delta t$, the same
circular array update scheme are used with the addition of linear interpolation. With
linear interpolation, voltages and currents up to $\text{int}(\tau/\Delta t) + 1$ time steps before need to be
stored. In the example of $\tau = 5.2\Delta t$, six previous solutions are stored in the storage array.
The principle of linear interpolation is outlined in figures below:

![Figure 4.20 Principle of Linear Interpolation](image-url)
Based on Figure 4.20, the purpose of linear interpolation is to locate the approximated voltage solutions, \([V_k(t-\tau)]\), between two integrating steps. To linearly approximate the values of \([V_k(t-\tau)]\), the following mathematical deduction is taken based on illustration of Figure 4.21.

First the slope of the linear line can be found by:

\[
\text{slope} = \frac{V_k(t-5\Delta t) - V_k(t-6\Delta t)}{\Delta t}
\]  
(4.21)

The same slope can also be expressed as

\[
\text{slope} = \frac{V(t-\tau) - V(t-6\Delta t)}{6\Delta t - \tau}.
\]  
(4.22)

Substituting (4.21) to (4.22) for slope and solving for \(V(t-\tau)\), we have

\[
V(t-\tau) = \text{slope} \times (6\Delta t - \tau) + V(t-6\Delta t)
\]  
(4.23)

A subroutine \textbf{Interpolation} is created to perform the computation of eqn (4.21) to (4.23). For every time step in each segment object, the storage arrays for \([V_k(t)]\), \([V_m(t)]\), \([I_k(t)]\) and \([I_m(t)]\) and the current index position are passed as arguments to the subroutine.
for processing. Note that since above voltages and currents are vector quantities, one element of the vector is being interpolated at a time. The detailed updating scheme with interpolation is outlined below.

1. For the first time step, instead of taking \([V_k(t-x)], [V_m(t-x)], [I_k(t-x)]\) and \([I_m(t-x)]\) directly from first columns of respective storage arrays, elements of first and second columns are linearly interpolated to find the approximated value. These approximated vector solutions are used to update history sources \([h_k]\) and \([h_m]\) with eqn (2.11).

2. For the second time step, similar to the case without interpolation, each segment would first retrieve node voltages and compute branch currents for the last time step. These previous vector solutions are stored in first columns of respective arrays pointed by the current index pointer. The index pointer is incremented by 1 position. Voltage and currents of second and third columns are extracted for linear approximation. Resulting vectors \([V_k(t-x)], [V_m(t-x)], [I_k(t-x)]\) and \([I_m(t-x)]\) are used to update history sources with eqn (2.11). Figure 4.23 shows the update process for \(t = 2\Delta t\).
Figure 4.23 History Update Scheme with Interpolation $t = 2\Delta t$

3. Same procedures are repeated for every time step. As the index pointer reaches the end of the array, it wraps around to the first column. Note that for every storage array, there will be two columns being extracted – the column pointed by the index and the column next to it. If the index pointer points to the last column of the storage array, the second column to be extracted would be the first column of the array. Vectors from the two columns are linearly interpolated to determine the in-between approximated vector value at $\tau$ seconds before. Figure 4.24 shows the update process for $t = 6\Delta t$.

Figure 4.24 History Update Scheme with Interpolation $t = 6\Delta t$

As can be seen from Figure 4.20, the approximated $V_k(t-\tau)$ can deviate from the exact $V_k(t-\tau)$ considerably depending on the shape of the curve. In fact, if the size between two neighbouring time steps ($\Delta t$) were reduced, that error could be significantly lowered.
Generally speaking, reducing $\Delta t$ to 5 to 10 times lower than the smallest $\tau$ in the network is sufficient. Unfortunately, this method would lower simulation efficiency.

### 4.3.3 Update History Sources for Loss Sections

The method to update history sources for loss sections of the zLine model is more straightforward, compared to tasks outlined for ideal line sections. Recall that the model for the multiphase loss section can be formulated in the time-domain by discretizing and combing each R-L fitting block with an integration rule. The equivalent resistance matrix, $[R_{eq}]$, has already been pre-calculated in the constructor. The equivalent history source, $[H(t)]$, however, depends on past branch currents so that it needs to be updated for every time step.

The equations for updating history sources of loss sections were introduced first in eqn (2.17). To implement (2.17) in C++, the equivalent history source of each fitting block is stored in a 3D array, $h_{eq}[i][j][k](t)$. To reduce unnecessary floating-point operations, computations involving mathematical operations of poles and constants can be pre-calculated and stored for every fitting block (4.8).

The procedures used in this thesis work to update history sources of loss sections are given below:

1. For the first time step voltage and current solutions for the previous time step are assumed to be zero. $h_{eq}[i][j][k](t)$ and $H_i(t)$ are therefore zero for all $i,j,k$.

2. For the second time step on, history sources need to be computed with (4.7a) and (4.8). In fact, two history source vectors must be updated since the loss section model has been split in half in every zLine segment to distribute the lumped losses. From (4.8), the branch current through the loss section is required to update its history sources. A close look at the segment model in Figure 4.7b shows that the branch currents flowing through the loss section is equal to those flowing through the ideal
line section. Since $[I_k(t-\Delta t)]$ and $[I_m(t-\Delta t)]$ for the ideal line section have already been computed, they can be plugged into (4.7a) directly. However, special attention must be paid to the polarity of the branch current when updating history sources of the loss section at the right end. The reason for that is, $[I_m(t-\Delta t)]$, flowing from right to left, is opposite in polarity to that defined in (4.8). Equation (4.8) was formulated by assuming current direction of left to right. As the result, the polarity for $[I_m(t-\Delta t)]$ must be reversed when applying (4.8). (4.7a) and (4.8) can be implemented easily with a triple-layer for loop in C++.

3. History sources of the loss sections are converted to the current source form for easy combination with the ideal line section and also for later network solutions (Figure 2.4).

Once history sources for the ideal line section and loss sections have been updated for each zLine segment, they are combined, reduced and accumulated at the line level. This last update procedure will be explained in details next.

4.3.4 Accumulating and Reducing Segment History Sources

Once the history sources for the ideal line section and loss sections of each zLine segment have been updated with procedures summarized previously, they are organized into external and internal history vectors $[h_{asec}]$ and $[h_{bsec}]$. For the segment model of Figure 4.7b, history sources for a zLine segment can be divided in the following manner:

$$[h_{a1}] = [H_{Ik}(t)]$$
$$[h_{a2}] = [-H_{Im}(t)]$$
$$[h_{b1}] = [h_k(t)] - [H_{Ik}(t)]$$
$$[h_{b2}] = [h_m(t)] + [H_{Im}(t)]$$

Eqn (4.25) can be further organized into external and internal history source vectors.
\[
[h_{\text{sec}}] = \begin{bmatrix}
 h_{a1} \\
 h_{a2}
\end{bmatrix} \quad \text{(4.26a)}
\]
\[
[h_{\text{bsec}}] = \begin{bmatrix}
 h_{b1} \\
 h_{b2}
\end{bmatrix} \quad \text{(4.26b)}
\]

The contribution of history sources from each segment to the line at the connecting nodes can be computed by applying the node-hiding equation (3.15)

\[
[h^\text{segment}_a] = [h^\text{sec}_a] - [G_{ab}][G_{bb}]^{-1}[h^\text{sec}_b]
\quad \text{(4.27)}
\]

where matrices \([G_{aa}], [G_{ab}], [G_{ba}]\) and \([G_{bb}]\) have already been defined in the constructor of each segment in Figure 4.10-4.13. A figure showing the contribution of segment history sources at external nodes is given below:

![Figure 4.25 History Source Contribution - Segment Level](image)

When segment models are connected in cascade to form the transmission line model, history sources at right end of a segment are combined with history sources at left end of the next segment. The global picture of the line model including history source contribution from each segment is given in the Figure 4.26.
Similar to the way the conductance matrix of the line model is accumulated from those of segments, the entire history source vector for the line model are filled by accumulating segment history source contributions at connecting nodes. Before accumulating, a vector $[h_{\text{line}}]$ is created and initialized with zero as the storage for history sources. History sources from each segment are inserted and accumulated into correct positions of $[h_{\text{line}}]$ corresponding to connecting nodes labeled in Figure 4.26. For instance, history contribution from segment 1 would be inserted into first few positions corresponding to nodes $a_1$ and $b_1$. Similarly, contributions from segment 2 would enter the positions corresponding to nodes $b_1$ and $b_2$. However, history sources to node $b_1$ from segment 2 would overlap with that from segment 1. As the result, they are accumulated and combined to generate a new history source for node $b_1$. Same procedures are followed until the history source contributions from all $z$Line segments have been entered into $[h_{\text{line}}]$. The resulting $[h_{\text{line}}]$ would look like as follows:

\[
[h_{\text{line}}] = \begin{array}{ccc}
\text{nodes} & a_1 & b_1 \\
\text{history sources} & h_{a_1} & h_{b_1} + h_{b_1} + h_{b_2} & h_{b_2} + h_{b_2} \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{b3} & \ldots & bN & a_2 \\
h_{b_3} + h_{b_3} & \ldots & h_{b_N} + h_{b_N} & h_{a_2} \\
\end{array}
\]  

(4.28)

Up to this point, the history source vector for the entire line model at all connecting nodes have been established, but our ultimate goal is to compute the equivalent history
source contributions of the line model. Following the same preparation for the node-
hiding technique as in (4.26), the resulting external and internal history source vectors are:

\[
[h_a] = \begin{bmatrix} h_{a1}^{\text{segment}} \\ h_{a2}^{\text{segmentN}} \end{bmatrix} \quad (4.29a)
\]

\[
[h_b] = \begin{bmatrix} h_{b1}^{\text{segment1}} + h_{b1}^{\text{segment2}} \\ h_{b2}^{\text{segment2}} + h_{b2}^{\text{segment3}} \\ (h_{b3}^{\text{segment3}} + h_{b3}^{\text{segment4}}) \\ \vdots \\ (h_{bN}^{\text{segment(N-1)}} + h_{bN}^{\text{segmentN}}) \end{bmatrix} \quad (4.29b)
\]

The node-hiding equation (3.15) is again used to compute the equivalent history
source contribution of the zLine model, \([h_a^{\text{line}}]\), at external nodes a1, a2 connecting to
other elements.

\[
[h_a^{\text{line}}] = [h_a] - [G_{ab}] [G_{bb}]^{-1} [h_b] \quad (4.30)
\]

where \([G_{aa}], [G_{ab}], [G_{ba}], [G_{bb}]\) matrices have already been obtained in the constructor
of the line model in Figure 4.16-4.19.

For every time step, the tasks for updating history sources of zLine models are
complete when the equivalent history source vector \([h_a^{\text{line}}]\) is properly computed. \([h_a^{\text{line}}]\)
is the history vector submitted to the core at every time step for the network solution.
CHAPTER 5
SIMULATION AND VALIDATION

A complete set of test cases and simulation results to validate the implementation algorithm and line modeling introduced in previous chapters are provided in this chapter. Simulation results with zLine OVNI are compared with the fdLine, cpLine, and the π-circuit model of Microtran to verify transient and steady-state responses. Test cases presented in this chapter emphasize open circuit and short circuit studies, and follow the order of increasing conductor counts and line length. Performing simulations on single-circuit lines (up to three phases) would prove the modeling and implementation correctness. To demonstrate the erroneous simulation results of fdLine due to the assumption of a constant transformation matrix and to reflect the accuracy of zLine under extreme asymmetrical tower configuration, an asymmetric double-circuit line is used. Since F. Marcano and F. Castellanos had conducted the same double-circuit simulation in their work [13,15], the simulation results with zLine using OVNI as the core will compare directly with simulation results in their work. In both of their work, the simulating results were compared with not only the fdline model, but also the FDTP [28] model, a frequency domain model which gives exact results and is not affected by issues of frequency-dependent transformation matrix. Consequently, further validation in accuracy can be obtained if simulation results in this thesis work match that of FDTP in [13, 15].
5.1 Single Conductor Simulations

The single conductor test provides the simplest line combination possible with no coupling between conductors and a unity modal transformation matrix. In this configuration cpLine generates an exact and accurate steady-state response while fdLine gives an exact transient response. Two tests with different line length will be conducted to provide an example of line segmentation. Each test would study both open circuit and close circuit conditions. The results of zLine with the OVNI simulator are compared with fdLine and cpLine of Microtran. The line geometry and conductor parameters for the single conductor line are shown in Figure 7.1 below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio Thickness/Diameter</td>
<td>0.2854</td>
</tr>
<tr>
<td>DC Resistance (Ω/km)</td>
<td>0.1050</td>
</tr>
<tr>
<td>Diameter (mm)</td>
<td>24.2062</td>
</tr>
</tbody>
</table>

Figure 5.1 Line Configuration and Conductor Parameters for Single Conductor Line

5.1.1 Short Line Simulation

For the short line study, a 1-km line will be used. Because the line is extremely short, no line segmentation is necessary especially for frequency as low as 60 Hz. For modelling of zLine, Figure 2.6 shows that the segment length of 1 km per segment is sufficient to simulate up to 8 KHz. At for the simulation will be chosen to match τ of the zLine segment, which equals to 3.333 μsec. In this scenario, the one-segment zLine model would resemble a cpLine model in structure. As the result, all three line models are expected to generate similar steady-state results but with zLine matching fdLine more closely in the transient region as both are more capable of simulating frequency-varying fast transient phenomena. Simulation results with short circuit are given below.
Short Circuit Simulation

The network configuration for the short circuit test is shown in Figure 5.2. A 60 Hz sinusoidal voltage source with magnitude of 1 volt is connected at time zero, energizing the circuit. Because there is no voltage source available in current OVNI simulator yet, a current source utilizing the Norton equivalence is used instead. As long as the parallel resistor is chosen small enough, its effect on the general circuit can be ignored. The $10^{-5} \, \Omega$ resistor serves as the link for MATE partitioning. Again, it is chosen small so as not to affect overall simulation results.

![Figure 5.2 Circuit Diagram for the Single-Conductor 1-km Short Circuit Test](image)

![Figure 5.3 Steady-State Response and Initial Transients at Receiving End](image)
• Open Circuit Simulation

Following the same configuration of Figure 5.2, except changing the load resistor to a very large value \(10^8 \Omega\) to mimic the open-circuit condition, the network configuration is shown Figure 5.4.

Figure 5.4  Circuit Diagram for the Single-Conductor 1-km Open Circuit Test

Figure 5.5  Steady-State Response and Initial Transients at Receiving End

As expected in both tests, the three models match closely in the steady-state responses. However for the transient responses, cpline deviates considerably from that of fdline and zLine. The transient response for fdLine and zLine is very similar because both models are frequency-dependent and more capable of capturing large-scale multi-frequency transient effects.
5.1.2 Long Line Simulation

For a 100-km long line, line segmentation is necessary to simulate transient frequency up to a desired accuracy. Based on Figure 2.6, 50 zLine segments of 2 km each can simulate up to 7 KHz which should be sufficient for normal switching transient studies. Δt for the simulation will be chosen to match τ of 6.666 μsec. Since fdLine possesses internal modeling of line segmentation already, results of zLine are expected to match those of fdLine with high similarity. The cpLine model without segmentation may be inaccurate, but at such low frequency as 60 Hz, the steady-state response should be still very reliable. If the same line configuration as in Figure 5.1 is used, the short circuit and open circuit test results are provided as follows:

- Short Circuit Simulation

![Circuit Diagram for the Single-Conductor 100-km Short Circuit Test](image)

Figure 5.6 Circuit Diagram for the Single-Conductor 100-km Short Circuit Test

![Steady-State Response and Initial Transients at Receiving End](image)

Figure 5.7 Steady-State Response and Initial Transients at Receiving End
Open Circuit Simulation

Following the same configuration of Figure 5.6, except changing the load resistor to a very large value \((10^8 \, \Omega)\) to mimic the open-circuit condition, the network configuration is shown Figure 5.8.

![Circuit Diagram for the Single-Conductor 100-km Open Circuit Test](image)

Figure 5.8 Circuit Diagram for the Single-Conductor 100-km Open Circuit Test

![Steady-State Response and Initial Transients at Receiving End](image)

Figure 5.9 Steady-State Response and Initial Transients at Receiving End

As can be seen, the transient components for the open circuit test take more time to fade, especially for the cpLine simulation. Once the transient response completely settles, the steady-state responses of all three models are very similar. Again, transient responses of fd and zLine are very close, but not for the case of cpLine.
5.2 Double-Conductor Simulations

The double-conductor transmission line provides a more complex test case. There is mutual coupling between each conductor and the effect of a 2 by 2 modal transformation matrix. Again, implementation correctness of zLine with OVNI will be validated against fdLine and cpLine of Microtran. Short line and long line scenarios will be studied with each scenario examining short circuit and open circuit conditions. Figure 5.10 shows the configuration and distance between conductors of this transmission line. Since it is still a single-circuit transmission line system and therefore no conductor asymmetry presents between conductors of different circuits, the steady-state responses of fdLine and cpLine would still expect to be very similar to those of zLine. zLine and fdLine tends to simulate fast transients more accurately given their frequency-dependent properties.

![Line Configuration and Conductor Parameters for Double-Conductor Line](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio Thickness/Diameter</td>
<td>0.2854</td>
</tr>
<tr>
<td>DC Resistance (Ω/km)</td>
<td>0.1050</td>
</tr>
<tr>
<td>Diameter (mm)</td>
<td>24.2062</td>
</tr>
</tbody>
</table>

Figure 5.10 Line Configuration and Conductor Parameters for Double-Conductor Line

5.2.1 Short Line Simulation

For the short line simulation of 1km, no segmentation is necessary as 1 km per segment is sufficient to simulate up to 8 KHz. Δt of the simulation is chosen to match τ of the segment, that is, 3.333 µsec. Simulation results for the short circuit condition are first provided then followed by the open circuit test.
Short Circuit Simulation

The network configuration of the two-conductor line for short circuit test is shown in Figure 5.11. As can be seen, the network contains two coupled conductors running parallel to each other. The first conductor is fed with a 60 Hz sinusoidal source and grounded through a 10 Ω resistor. The second conductor is not energized but grounded through the 10 Ω resistor at both sending and receiving ends. Simulation results at receiving end of both phases (conductors) will be provided in Figure 5.12 and 5.13.

![Circuit Diagram for Double-Conductor 1-km Short Circuit Test](image)

Figure 5.11 Circuit Diagram for Double-Conductor 1-km Short Circuit Test

![Steady State and Initial Transients at Receiving End of Phase 1](image)

Figure 5.12 Steady State and Initial Transients at Receiving End of Phase 1
Open Circuit Simulation

The open circuit condition is similar to the short circuit case except that the receiving end of phase 1 is left open by grounding through a large $10^8 \, \Omega$ resistor now. The configuration of the second conductor remains intact. Network configuration and simulation results at the receiving end are given in Figure 5.14 to 5.16.

Figure 5.14 Circuit Diagram for Double-Conductor 1-km Open Circuit Tests
All three line models agree closely in steady state responses but not in capturing the transient response. Due to the large amount of fast transients components generated from the open circuit configurations, the single-frequency cpLine is unable to fully simulate the transient response. zLine and fdLine, frequency dependent in nature, are more capable of accurately capturing the transient response.
5.2.2 Long Line Simulation

The long line simulations follow the same network configuration as short line simulations except that the line length is changed to 100 km. Line segmentation and step size chosen remain the same as in the single-conductor case.

- Short Circuit Simulation

![Circuit Diagram for Double-Conductor 100-km Short Circuit Test](image)

Figure 5.17 Circuit Diagram for Double-Conductor 100-km Short Circuit Test

![Steady State and Initial Transients at Receiving End of Phase 1](image)

Figure 5.18 Steady State and Initial Transients at Receiving End of Phase 1
- Open Circuit Simulation

Figure 5.19 Steady State and Initial Transients at Receiving End of Phase 2

![Figure 5.19](image1.png)

Figure 5.20 Circuit Diagram for Double-Conductor 100-km Open Circuit Test

![Figure 5.20](image2.png)

Figure 5.21 Steady State and Initial Transients at Receiving End of Phase 1

![Figure 5.21](image3.png)
Figure 5.22 Steady State and Initial Transients at Receiving End of Phase 2

As can be seen, although the transient response generated by cpLine takes longer to fade, the steady-state response after initial transients have settled still agree very closely with fdLine and zLine. Again, initial transients for fdLine and zLine match closely with each other while it is not quite the case for cpLine.
5.3 Steady-state and Initial Transients Validation

Simulation results of previous two sections unanimously show that the steady-state responses at receiving ends of networks under test agree very closely among all three models and initial transients between fdLine and zLine are also quite similar. These findings could have concluded the correctness of implementation methodologies introduced in Chapter 4. However, to further confirm this belief, another set of tests will be conducted. In this section, simulation results with a single-circuit triple-conductor zLine model are compared with a three-phase π-circuit of same tower and conductor configurations. Since the π-circuit is considered an exact proof of steady-state response, it elegantly validates the steady-state results drawn from the two previous sections. In order to validate the transient response, supervisor Martí suggested the use of transposition to transform the non-balanced line to a balanced one. Since a balanced line applies a constant and real Clarke transformation matrix, the assumption of real and frequency independent transformation matrix by fdLine is always true and valid. As a result, there will be no errors inherited from matrix approximations and the transient response under the balanced line condition should be exact, too. Comparing simulation results of zLine with fdLine would validate the accuracy of transient response. For the π-circuit test, both open circuit and short circuit tests will be examined on a 100km line. For the transient response validation, a 30 km line is used for its convenience to divide into three equal sections for the transposition. The tower configuration and conductor characteristics are provided in Figure 5.23.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio Thickness/Diameter</td>
<td>0.2854</td>
</tr>
<tr>
<td>DC Resistance (Ω/km)</td>
<td>0.1050</td>
</tr>
<tr>
<td>Diameter (mm)</td>
<td>24.2062</td>
</tr>
</tbody>
</table>

Figure 5.23 Line Configuration and Conductor Parameters for Triple-Conductor Line
5.3.1 π-circuit Steady-State Simulation

The [R], [L] and [C] parameters for the π-circuit are generated by mtLine at 60 Hz. mtLine generates two types of parameters for the multiphase π-circuit: a corrected parameters for long lines and approximated nominal π for medium and short lines. Nominal π-circuit produces very accurate results in our simulations.

- Short Circuit Simulation

For the short circuit configuration, the π-circuit with parameters calculated at 60 Hz is energized by a balanced set of sinusoidal current sources in Norton form. The receiving end of each phase is grounded through a 10 Ω resistor. For the zLine simulation, the 100 km line is divided into 50 equal segments and Δt of the simulation is chosen to be 6.666μsec. Network configuration and simulation results are shown in Figure 5.24 to 5.26 below.

![Circuit Diagram for three-phase 100-km Short Circuit Test with π-Circuit](image)

Figure 5.24 Circuit Diagram for three-phase 100-km Short Circuit Test with π-Circuit
Figure 5.25 Steady-state Response for Receiving End of Phase 1 (left) and 2 (right)

Figure 5.26 Steady-state Response for Receiving End of Phase 3

- Open Circuit Simulation

For the open circuit simulation, instead of grounding receiving end nodes 7, 8 and 9 through 10 Ω resistors, they are left open. Network configuration and steady state response at receiving nodes are provided in Figure 5.27 – 5.29.
Figure 5.27 Circuit Diagram for three-phase 100-km Open Circuit Test with $\pi$-Circuit

As can be seen from the short circuit test, the steady-state response for zLine agrees very closely with the response for $\pi$-circuit. For the open circuit test, although the final steady-state takes more than 0.1 second to arrive, the comparison between zLine and $\pi$
after that time also shows an uniform fit. These results affirm the implementation correctness in the area of steady-state response.

5.3.2 Balanced Line Simulations

For the transient response validation, an unbalanced line is transformed to a balanced line via transposition to validate the transient response for the receiving end of each phase. Simulation results with a balanced zLine model are compared with a balanced fdLine model. The transposition scheme for a single three-phase circuit is given in Figure 5.30 below. Note from the figure that the line length is divided into three sections, in our case, 10 km per section. Phase 1 of the first section is connected to phase 2 of the second section, phase 2 connected to phase 3, and phase 3 connected to phase 1. Same connection joins the second and third sections. A 30 km line that facilitates easy division into three sections is used for transient simulation in this subsection. Again, short circuit and open circuit tests will be revisited.

![Figure 5.30 Transposition Scheme for a Three-Phase Line](image)

- Short Circuit Simulation

For both fdLine and zLine simulations, the 30 km line is represented by 3 equal length line models connected as shown in Figure 5.31. The zLine model further breaks each 10-km line model into 10 equal zLine segments. \( \Delta t \) for the simulation is chosen to be 3.3333 \( \mu \)sec that matches \( \tau \) of each zLine segment. Simulation results for the short circuit simulation are provided in Figure 5.31 to 5.33 below.
Figure 5.31  Circuit Diagram for 3-phase 100-km Short Circuit Test with Transposition

Figure 5.32  Initial Transients at Receiving End of Phase 1 (left) and Phase 2 (right)

Figure 5.33  Initial Transients at Receiving End of Phase 3
- Open Circuit Simulation

Replacing 10 Ω resistors at receiving end of each phase with large resistors ($10^8$ Ω), the network configuration and simulation results for the open circuit test are summarized in Figure 5.34 to 5.36 below:

Figure 5.34  Circuit Diagram for 3-phase 100-km Open Circuit Test with Transposition

Figure 5.35  Initial Transients at Receiving End of Phase 1 (left) and Phase 2 (right)

Figure 5.36  Initial Transients at Receiving End of Phase 3
As can be observed from the short and open circuit tests, the transient response for zLine agrees very closely with that for fdLine at each phase of the receiving end. Since the 30km transmission line is a balanced line, the results obtained with fdLine are therefore exact and accurate. Simulation results in this section validates the correctness of transient responses of zLine.
5.4 Asymmetric Six-Phase Double-Circuit Tests

Having mentioned throughout this thesis, the assumption of real and constant transformation matrix by fdLine and cpLine of Microtran would encounter severe inaccuracy especially in situations of asymmetric conductor configuration between neighbouring circuits. In this section, a 100 km double-circuit 6-phase transmission line with asymmetric conductor configurations is modelled as both fdLine and zLine. Simulation results in short circuit and open circuit conditions are studied. For the zLine simulation, the transmission line is divided into 40 sections of 2.5 km each, capable of simulating up to 6 KHz. $\Delta t$ of the simulation is chosen as 8.333 $\mu$sec to match exactly $\tau$ of each segment. Note that in the work of [13] and [15], Marcano and Castellanos performed the same tests and compared their results with the frequency domain FDTP program. Since FDTP solves the network in frequency domain, its accuracy is exact and not affected by the degree of asymmetry associated with the line configuration. The results with zLine using OVNI’s solution method will be compared directly with Castellanos’ zLine simulations using the same segmentation, Marcano’s Idempotent model and the FDTP [28] model to verify the implementation accuracy. The transmission line configuration and conductor parameters are given in Figure 5.37.

![Figure 5.37 Configuration and Conductor Parameters for Double-Circuit Six-Conductor Line](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio Thickness/Diameter</td>
<td>0.2854</td>
</tr>
<tr>
<td>DC Resistance (Ω/km)</td>
<td>0.1050</td>
</tr>
<tr>
<td>Diameter (mm)</td>
<td>24.2062</td>
</tr>
</tbody>
</table>
Short Circuit Simulation

Network configuration for the short circuit test is given in Figure 5.38. Note in the figure that each solid conductor represents a "circuit" that contains a three-phase line. The current source is therefore a three-phase balanced source and the resistors are three-phase as well. The two circuits are combined to form a six-phase line as shown. For the short circuit test, the three-phase current source energizes phase 1, 2 and 3 at time zero. The receiving ends of these phases are grounded through \( 10 \, \Omega \) resistors. Phases 4, 5 and 6 are grounded at both ends with \( 10 \, \Omega \) resistors as well. The simulation results recorded correspond to the middle phase of the first circuit (phase 2) at the receiving end, and all phases for the second circuit (phase 4, 5, 6). The results for the second circuit validate the accuracy of modelling the coupling effect between different circuits and conductors. In Castellanos' zLine simulation results, different curves are marked with labels: \( f = \text{fdLine} \), \( w = \text{FDTP} \) and \( z = \text{Castellanos' zLine implementation} \).

![Circuit Diagram for the Double-Circuit Short Circuit Test](image)

Figure 5.38 Circuit Diagram for the Double-Circuit Short Circuit Test
Figure 5.39 Steady State and Initial Transients at Receiving End of Phase 2

Figure 5.40 Castellanos' Initial Transients at Receiving end of Phase 2

Figure 5.41 Steady State and Initial Transients at Receiving End of Phase 4
Figure 5.39 Steady State and Initial Transients at Receiving End of Phase 2

Figure 5.40 Castellanos' Initial Transients at Receiving end of Phase 2

Figure 5.41 Steady State and Initial Transients at Receiving End of Phase 4
Figure 5.42 Castellanos’ and Marcano’s Initial Transients at Receiving End of Phase 4

Figure 5.43 Steady State and Initial Transients at Receiving End of Phase 5

Figure 5.44 Castellanos’ and Marcano’s Initial Transients at Receiving End of Phase 5
Simulation results of Figure 5.39 to 5.46 show noticeable difference in magnitude and phase between zLine and fdLine of Microtran. As expected, the difference is due to the assumption by fdLine that the modal transformation matrix is real. The imaginary components are important in situations where the conductor configuration is extremely asymmetrical. Single-circuit tests in Section 5.1 and 5.2 do not show such considerable differences. Simulation results of zLine OVNI shows very close agreement with Castellanos' zLine simulation and with Marcanos' Idempotent model. Especially, the simulation in this thesis shows even closer agreement with FDTP, the frequency-domain model that gives exact solutions in asymmetrical tower.
configurations. The conclusion can be drawn that the short circuit simulation results are accurate and reliable.

- **Open Circuit Simulation**

The network configuration for the open-circuit test is shown in Figure 5.47. The receiving end of the first circuit is left open, while both ends of the second circuit are still grounded through 10 Ω resistors. Simulation results at phase 2 of the first circuit and phase 4, 5 and 6 of the second circuit are provided in figures below. Simulation results in this thesis are again compared with Castellanos’ and Marcano’s and the FDTP model to validate the simulation accuracy.

![Figure 5.47 Circuit Diagram for the Double-Circuit Open Circuit Test](image)

**Figure 5.47 Circuit Diagram for the Double-Circuit Open Circuit Test**

![Figure 5.48 Steady State and Initial Transients at Receiving End of Phase 2](image)

**Figure 5.48 Steady State and Initial Transients at Receiving End of Phase 2**
Figure 5.49 Castellanos' Initial Transients at Receiving End of Phase 2

Figure 5.50 Steady State and Initial Transients at Receiving End of Phase 4

Figure 5.51 Castellanos' and Marcano's Initial Transients at Receiving End of Phase 4
Figure 5.52  Steady State and Initial Transients at Receiving End of Phase 5

Figure 5.53  Castellanos' and Marcano's Initial Transients at Receiving End of Phase 5

Figure 5.54  Steady State and Initial Transients at Receiving End of Phase 6
Simulation results of zLine do not deviate as drastically from fdLine in the open circuit test as in the short circuit test. It is found that the fdLine model tends to deviate from zLine as the height of the conductor from the ground increases. Once again, zLine implemented with OVNI in this thesis agrees closely with Castellanos and Marcano’s results, and even more so with the exact FDTP model. These findings further confirm the correctness and accuracy of the zLine model in extreme asymmetrical conductor configurations and the implementation methodologies introduced in this thesis.
CHAPTER 6
RECOMMENDATION FOR FUTURE WORK

The work presented in this thesis is based on work of numerous graduate students and professors as listed in the bibliography. Using network partitioning, OVNI opens a new page in the realm of real-time digital network simulation. The implementation and interfacing of the zLine model with OVNI's real-time framework also sets an example for future inclusion of complex models containing internal nodes. In the previous chapter, the implementation of node-hiding strategies has been successfully verified by proving the correctness of the time domain simulations. We hope that the work in this thesis can be used as a model for future interfacing of more complex models with OVNI. Some suggested future works are listed as follows:

- **Incorporate other element models to OVNI to enhance its functionality and capability**
  As of today OVNI only possesses simple R, L, and C models. A coupled R-L π-circuit and a cpLine model were implemented prior to this thesis work as exercises. OVNI is lacking element models that are key components for a power network. Examples of those may include transformer models, synchronous machines, induction machines and power electronic devices. Good news is that a lot of the models mentioned are being implemented by graduate students of UBC power group. More are believed to come as the simulator becomes more matured and computationally efficient.

- **Implementation of latency technique to streamline the simulation process**
  In order to represent the distributed nature of the losses, the zLine model divides the total line length into a number of short segments. This segmentation unnecessarily forces Δt for the simulation to the travelling time of each short segment. If external networks use the same Δt as the zLine model, it unnecessarily increases the computation burden and slows down the solution process. The latency technique developed by Moreira and Martí [17] suggests the use of different Δt’s for different
part of network so that a larger $\Delta t$ can be used for external and slower sub-networks, while the sub-networks containing zLine models can be solved with a smaller $\Delta t$. Moreira commented that gains in the order of ten to a hundred times can be achieved applying latency in networks containing zLine models.
CHAPTER 7
SUMMARY AND CONCLUSION

The zLine model based on the work of [15] and [16] has been implemented and interfaced with OVNI in this thesis work. zLine is a frequency-dependent transmission line model formulated entirely in phase-domain, elegantly avoiding the problem associated with the frequency-dependent transformation matrix in most modal domain line models. It circumvents the problems of mixed mode and different travelling delays by separating the wave propagation into the constant ideal line sections and the frequency-dependent loss sections. For distributed-parameter reasons, the entire line length is divided into a number of short segments and each segment consists of one ideal line section and two half-length loss sections. The zLine segments form the basic building blocks for model implementation in C++.

The implementation of zLine model within OVNI’s real-time framework follows naturally the way zLine is modelled - from segment level upward toward the line level, but must also adhere to OVNI’s unique element - core interface. In OVNI, each element instantiation possesses a member function to initialize parameters relevant to the operation of the element, a function to submit the element conductance matrix and a function to update and submit the equivalent history sources at its external nodes. It is therefore natural for the implementation tasks to emphasize the above three criteria.

The initialization tasks occur inside the default constructor of the zLine model. The default constructor invokes and executes the mtLine programs, the matrix fitting routine and some extra I/O routines to arrange and process the primitive tower geometry and conductor configurations to produce appropriate model parameters for time-domain simulation. The function system (const *char command) is chosen to integrate these multiple reading and processing tasks. In addition to the pre-processing tasks, the default constructor is also responsible for constructing zLine segments and initializing model
parameters for ideal line and loss sections. In general, computations that can be dealt with once and for all occur inside the constructor.

The elegance of the history source update and conductance matrix computation scheme lies in the two-level abstraction between segments and the line. History source and conductance matrix contributions are first updated and computed at the segment level for each zLine segment, hiding away internal nodes using the node-hiding technique and then accumulated at the line level to form the overall history sources and conductance matrix of the line. The equivalent conductance matrix and history source vector of the line can be obtained by applying node-hiding formula, one more time, to lump the effect of internal nodes onto external ones. A linear interpolation routine is written to account for situations when $\Delta t$ cannot be chosen as an exact sub-multiple of $\tau$.

Simulation results for the single-circuit configurations show very good fit between zLine and fdline. In addition, the $\pi$-circuit and three-phase balanced line tests verify the correctness of steady-state and transient responses. For asymmetrical double-circuit simulations, zLine OVNI displays evident variations from that of fdLine due to the constant transformation matrix assumption by fdLine. However, comparing with the same simulation case in [13] and [15], zLine OVNI shows very good agreement with the zLine simulation in [15] and the Idempotent model in [13], and most importantly, it agrees with the results of FDTP, a frequency-domain model giving accurate and exact solutions. Above agreements indicate that the implementation methods introduced in this thesis for zLine ONVI produce correct and accurate transient and steady-state responses.

Although at this point of initial implementation zLine ONVI is slower than fdLine, its high accuracy and absolute numerical stability for all possible line configurations make it a valuable transmission line model to be included with OVNI and possibly Microtran of EMTP. Its inclusion also serves as a good example for future implementation of complex element models containing internal nodes. It is believed that the computational efficiency of the model will improve with the implementation of latency technique and with more optimized programming.
BIBLIOGRAPHY


121


APPENDIX I
CURVE FITTING PROCEDURE

The fitting routine includes two parts: a fitting process and an optimization process.

For the fitting process, we first assume a frequency $\omega_1$. Computing $K_{ii(1)}$, $K_{ij(1)}$, $P_{ii(1)}$ and $P_{ij(1)}$ for each element of $[Z^{\text{loss}}]$ with equations (1.1-1.4) gives the expressions of first fitting block that matches the line data at $\omega_1$, $[Z^{\text{loss}}(\omega_1)]$.

\[
R_{fij(1)} = \frac{(\omega_1 L_{ij}(\omega_1))^2}{R_{ij}(\omega_1)} + R_{ij}(\omega_1) \quad \text{for all } i,j \tag{1.1}
\]

\[
L_{fij(1)} = \frac{(R_{ij}(\omega_1))^2}{\omega_1^2 L_{ij}(\omega_1)} + L_{ij}(\omega_1) \quad \text{for all } i,j \tag{1.2}
\]

\[
K_{ij(1)} = R_{fij(1)} \tag{1.3}
\]

\[
P_{ij(1)} = \frac{R_{fij(1)}}{L_{fij(1)}} \tag{1.4}
\]

where

$R_{ij}(\omega_1)$, $L_{ij}(\omega_1)$ are the resistance and inductance at frequency $\omega_1$ per unit length of the loss matrix (acquired from mtLine).

$R_{fij(1)}, L_{fij(1)}$ are the resistance and inductance per unit length of the first RL fitting block for elements of the loss matrix.

Same Equation (1.1) is applied to frequency $\omega_2$ to calculate $P_{ij}$ and $K_{ij}$ for fitting block 2 with the following corrections: $K_{ij}$'s and $P_{ij}$'s for the second R-L block are calculated to match not just $Z_{ij}^{\text{loss}}(\omega_2)$, but the difference between $Z_{ij}^{\text{loss}}(\omega_2)$ and the influence of the first fitting block at $\omega_2$. In other words, we have the following:

\[
\bar{Z}_{ij}^{\text{loss}}(\omega_2) = Z_{ij}^{\text{loss}}(\omega_2) - \frac{j\omega_2 R_{fij(1)} L_{fij(1)}}{R_{fij(1)} + j\omega_2 L_{fij(1)}} \tag{1.5}
\]

from which

124
\[
\overline{R}_{ij}(\omega_2) = \text{Re}(\overline{Z}_{ij}^{\text{loss}}(\omega_2)) = R_{ij}(\omega_2) - \frac{j\omega_2 R_{fij(1)} L_{fij(1)}}{R_{fij(1)}^2 + \omega_2^2 L_{fij(1)}^2}
\]

\[
\overline{L}_{ij}(\omega_2) = \frac{\text{Im}(\overline{Z}_{ij}^{\text{loss}}(\omega_2))}{\omega_2} = L_{ij}(\omega_2) - \frac{L_{fij(1)} R_{fij(1)}^2}{R_{fij(1)}^2 + \omega_2^2 L_{fij(1)}^2}
\]

Substituting \( \overline{R}_{ij}(\omega_2) \) and \( \overline{L}_{ij}(\omega_2) \) into equation (A.1) we obtain the R and L for the second fitting block for all elements i,j.

\[
R_{fij(2)} = \frac{(\omega, \overline{L}_{ij}(\omega_2))^2}{\overline{R}_{ij}(\omega_2)} + \overline{R}_{ij}(\omega_2)
\]

\[
L_{fij(2)} = \frac{\overline{R}_{ij}^2(\omega_2)\omega_2^2}{\omega_2^2 L_{ij}(\omega_2)} + \overline{L}_{ij}(\omega_2)
\]

\[
K_{ij(2)} = R_{fij(2)}
\]

\[
P_{ij(2)} = \frac{R_{fij(2)}}{L_{fij(2)}}
\]

\( R_{ij}(\omega_2), L_{ij}(\omega_2) \) are the resistance and inductance at frequency \( \omega_2 \) per unit length of the loss Matrix.
\( R_{fij(2)}, L_{fij(2)} \) are the resistance and inductance per unit length of the second fitting block for each element of the loss matrix.
\( \overline{R}_{ij}(\omega_2), \overline{L}_{ij}(\omega_2) \) are the resistance and inductance at frequency \( \omega_2 \) per unit length of the loss matrix, subtracting the influence of the first block at this frequency.

Same procedures (I.1) to (I.4) can be generalized to synthesize at frequency \( \omega_m \) and compute for R and L elements of the \( m^{th} \) fitting block, except that the pole and constant for the \( m^{th} \) block are computed to match the line data at \( \omega_m \) subtracting the influence of previous blocks at this frequency, as shown in (I.5-I.11).

After evaluating the fitted blocks for each element following the procedure mentioned, an optimization procedure using the Gauss-Seidel iteration is applied to further minimize the difference between the fitted functions and the actual \( Z_{ij}^{\text{loss}} \). First an acceptable tolerance between the actual \([Z^{\text{loss}}]\) element and the fitted functions is decided. Then a
similar procedure to I.1 to I.4 is followed except that the RL elements at \( \omega_m \) for the \( m^{th} \) fitting block is calculated to match the line data of the element at \( \omega_m \) subtracting the influence of all "m-1" other blocks at this frequency \( \omega_m \). Once a complete set of poles and constants are computed for \( m \) fitting blocks with the above procedure, the errors in terms of magnitude between the synthesized functions and the actual \([Z^{loss}]\) elements are compared against the tolerance at each fitting frequency, \( \omega_1 \) to \( \omega_m \). The same optimization process will be repeated until the error at each frequency for each \([Z^{loss}]\) element is within the tolerance. In general, acceptable fitting results within tolerance can be obtained with ten iterations or less.
APPENDIX II
INPUT FILE FOR OVNI

0.2 6.666e-6 9 7 3 3 2 //t nod At NumNod NumElm NumSrc NumLnk NumBlk

R1
1 0
100 -100 -100 100

R2
2 0
100 -100 -100 100

R3
3 0
100 -100 -100 100

R4
7 0
1e-8 -1e-8 -1e-8 1e-8

R5
8 0
1e-8 -1e-8 -1e-8 1e-8

R6
9 0
1e-8 -1e-8 -1e-8 1e-8

Z1 1 2 3 4 5 6 TLINE //parameters for the zline model
.MODEL Z1 TLINE ( Unit = METRIC, Len = 100, SecLength = 2, NumCondInBundle=1, NumGndWire=0, Earth-Resistivity=100, NumFrequency=9, Frequency = [0.1,1.1, 10 ,100,1000,1e4,1e5,1e6,1e-5], Rdc = [0.105,0.105,0.105], Diameter=[24.2062,24.2062,24.2062], Thickness/Diameter = [2854,2854,2854], Height = [12,12,12], Midspan = [12,12,12], HzDistance = [0,2,4] )

3 //id for current source
1 0 //connecting nodes
100 60 0 //amplitude frequency phase shift

3
2 0
100 60 -2.0944

3
3 0
100 60 2.0944

continue to next page
continue from last page

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>//id for MATE link</td>
<td></td>
</tr>
<tr>
<td>4 7</td>
<td>//connecting nodes</td>
<td></td>
</tr>
<tr>
<td>1 e-5 0</td>
<td>//link resistance link voltage source</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 e-5 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 e-5 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>//id for MATE block</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>//number of nodes in the block</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6</td>
<td>//node ids in the block</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure II.1 Sample Input File For OVNI
APPENDIX III

Header Files for Class zline_t & section_t

```cpp
class zline_t: public elm_t { // Derived class from elm_t class
public:
    zline_t(fstream&);
    double** submitG();
    double* UpdateHistory();

private:
    int cNumPhase; // number of phase
    int cNumFitBlk; // number of fitted blocks for the loss matrix
    int cNumSec; // number of small line sections
    double vo; // speed of light
    int cNumExNod; // number of external nodes a
    int cNumIntNod; // number of internal nodes b
    section_t** apSection; // Array of pointers to each section of the line
    double alpha; // damping constant for trapezoidal rule
    double tao; // travelling time for the ideal line section
    double length; // length of the transmission line
    double seclength; // line length for each zline section
    double** aC; // capacitance matrix in phase coordinate
    double** aP; // Maxwell Coefficient Matrix
    double*** aK; // constant for numerator of each fitted block
    double*** aPole; // poles for each fitted block
    double** aReq; // equivalent resistance matrix of the loss section in each zline segment
    double** aZc; // characteristic impedance of the ideal line section
    double** aRdc; // Matrix storing the DC conductor resistance in phase domain
    double** aVa; // vector storing voltage of external nodes of the line
    double** aVb; // vector storing voltage of internal nodes of the line
    double* aNodV; // storage for all the internal and external nodal voltages
    double* aExHsrc; // Vector of external history sources after node hiding
    double* ahb; // vector of internal history sources before node hiding
    double* ah; // vector of external history sources before node hiding
    double* ah; // vector for storing all the history sources passed from

    double** agline; //G matrix of the whole line before node hiding
    double** agaA; //arrays for node hiding calculation to deliver the overall
    double** agbb; //matrix of the line (hide internal nodes) a for external nodes,
    double** agab; //for internal nodes
    double** agba;
    double*** acoeffA; // coeffA and coeffB are for computation of Rea and heq before
    double*** acoeffB; //loop starts

Figure III.1 Header File of zline_t
```

129
class section_t
{
  // Each section of the line is an instantiation of this object.
  // Each section of the line is an instantiation of this object.
  public:
  section_t(double** Req, double** Zc, double travtime, int cNumPhase, double** twoReq,
             double** negtwoReq, double** ZcAndTwoReq, double** zero, int NumBlk,
             double*** K, double*** P, double sectionlen, double*** coeffA, double***
             coeffB);

  // constructor. Pointer to Req and Zc matrices are passed when a section object is created.
  double* UpdateSecHistory(double* NodV, int& count); // Update the history source of
  // each section

  private:
  int cNumPhase; // number of phase of the section
  int cNumBlk; // number of fitting blk for each element of the zloss matrix
  int cNumExNodSec; // number of external nodes of the section
  int cNumIntNodSec; // number of internal nodes of the section
  double tao; // travelling time of the ideal line section
  double alpha; // damping constant for trapezoidal rule
  double seclength;
  int index; // index to reference the voltage & current storage.
  double** aVka; // voltage of the left external node, n phase
  double** aVma; // voltage of right external node
  double** aVmb; // voltage of right internal nodes
  double** aVkb; // voltage of left internal nodes
  double** aVkbStor; // Storage for Voltage of the left internal node, n phase, up to
                     // (tao/deltaT) previous deltaT
  double** aVmbStor; // Storage for Voltage of the right internal node
  double** aVaSec; // external voltages of the section, [left, right]
  double** aVbSec; // internal voltages of the section
  double** alkStor; // current storage of dimension nx(tao/deltaT) to store last (tao/deltaT)
                     // deltaT passing thru left part of the section
  double** almStor; // current storage of dimension nx(tao/deltaT) to store last (tao/deltaT)
                     // deltaT passing thru right part of the section
  double** alk; // current flows in the left half of the section, from left to right
  double** alm; // current flows in the right half of the section, from left to right
  double** ahk; // left history source for the ideal section
  double** ahm; // right history source for the ideal section
  double** aHk; // left history source for the loss section (voltage and current)
  double** aHm; // right history source for the loss section
  double*** aheqk; // 3D history source used to calculate actual history source of the loss
                   // section, aHk
  double*** aheqm; // 3D history source used to calculate actual history source of the loss
                   // section, aHm
  double*** aKs; // constant of numerator of each fitted block 3D array per zloss
                 // element
  double*** aPs; // poles for denominator of each fitted block 3D array per zloss
                 // element
  double** aReq; // inverse of equivalenet resistance matrix of discrete representation of
                 // the loss section in each section
  double** aZc; // inverse of characteristic impedance of the ideal line section in phase
                // domain
double** ahbSec;  // internal history sources of the section
double** ahaSec;  // external history sources of the section
double* aExHsrcSec;  // Vector of external history sources
double** aGsec;  // G matrix of the section after node hiding
double** aGbbSec;  // arrays for node hiding calculation to deliver the overall G matrix of
double** aGbaSec;  // line (hide internal nodes)
double** aGabSec;
double** aGaaSec;
double*** acoeffA;
double*** acoeffB;

Figure III.2  Header File of section_t