ADVANCED NUMERICAL METHODS FOR SIMULATING NONLINEAR MULTIRATE LUMPED PARAMETER MODELS

by

NORBERT HENRY DOERRY

Naval Engineer
S.M. Electrical Engineering and Computer Science
Massachusetts Institute of Technology
(1989)

B.S. Electrical Engineering
United States Naval Academy
(1983)

SUBMITTED TO THE DEPARTMENT OF
OCEAN ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
in the field of

NAVAL ELECTRICAL POWER SYSTEMS

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 1991

Signature of Author _____
Department of Ocean Engineering
12 May 1991

Certified by _____
James L. Kirtley, Thesis Supervisor
Associate Professor of Electrical Engineering

Certified by _____
Marija Ilic, Thesis Supervisor
Senior Research Engineer,
Department of Electrical Engineering and Computer Science

Accepted by _____
A. Douglas Carmichael, Chairman
Ocean Engineering Departmental Graduate Committee

ADVANCED NUMERICAL METHODS FOR SIMULATING NONLINEAR MULTIRATE LUMPED PARAMETER MODELS

by

Norbert H. Doerry

Submitted on May 12, 1991 to the Department of Ocean Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the field of Naval Electrical Power Systems.

ABSTRACT

Naval shipboard electric power systems are transitioning from the relatively simple distribution of ship service electric power to extremely complex integrated electric drive (IED) systems. The optimal design of warships employing IED is presently hampered by the lack of existing simulation computer tools for analyzing the highly coupled and controlled electro-mechanical systems characteristic of IED. As a first step in the development of a viable computer simulation tool, the numerical algorithm testing program WAVESIM was created.

The key contributions of WAVESIM are the systematic treatment of waveforms as an abstract data type, the development of the *terminal description* of devices, and the use of structural jacobians in system reduction.

WAVESIM represents variables by waveforms consisting of a vector of coefficients and a waveform type code indicating how the coefficients should be interpreted. The principal advantage of using waveforms over conventional discrete point methods is the avoidance of unstable integration techniques since for most waveform types, integration and differentiation are linear matrix operations.

Devices are described in WAVESIM by relationships between terminal *interface variables*. WAVESIM recognizes two types of terminals: normal terminals having both potential and flow variables, and information terminals having only a potential variable. In this manner, WAVESIM can simulate processes involving both energy transfer and control signals.

WAVESIM extends the structural jacobian matrix concept to reflect the properties of the dependence of system equations on system variables. The system structural jacobian matrix is constructed from the constitutive device structural jacobian matrices and is used to identfiy a sequence of smaller blocks when can be solved consecutively for all the system variables.

To demonstrate and verify the capabilities of WAVESIM, several simulations were conducted. In all simulations, WAVESIM provide results matching data generated by other simulation methods.

Thesis Supervisor: James L. Kirtley, Associate Professor of Electrical Engineering

Thesis Supervisor: Marija Ilic, Senior Research Engineer, Department of Electrical Engineering and Computer Science

The author hereby grants to the United States Government and the Massachusetts Institute of Technology permission to reproduce and distribute copies of this thesis document in whole or in part.

_____

Norbert Henry Doerry

ACKNOWLEDGEMENTS

I am grateful to the United States Navy for allowing me to continue my education at MIT and complete this thesis. In particular, the assistance of LCDR Rich Martin (DARPA) and CDR Mike Reed (NAVSEA 05Z) in providing financial support for my research is greatly appreciated. I also express my thanks to the numerous individuals at the David Taylor Research Center and Naval Sea Systems Command who shared their tremendous knowledge of shipboard systems with me.

The guidance, challenges, and encouragement of my thesis advisors, Professor James Kirtley and Marija Ilic were invaluable in the completion of this work. John Amy, Mary Tolikas, and Kathy Millis also provided much appreciated feedback.

I would also like to thank my family and many friends who have made the last five years in Boston memorable and treasured.

Table of Contents

Chapter 1 Introduction

A revolution is occurring in modern warship design. The conventional mechanical transmission train for transferring power from the prime movers to the ships screws will be replaced in future warship designs by an integrated electric drive (IED) system. While electric drive is not a new concept, the IED approach differs significantly from previous electric drive implementations in that both propulsion power and ship service power (60 HZ 440 Volts AC) are derived from the same prime movers. The resulting flexibility in the arrangability of the ship, in the design of the electric plant, and in the power available to combat systems provides the naval architect with many opportunities for significantly improving the combat effectiveness of modern warships.

Designing a ship taking full advantage of the opportunities afforded by IED is not an easy task or even obvious. The ship designer has no precedent for an IED ship let alone the design of an IED electric plant. Instead, the designer must rely heavily on simulations of proposed systems to evaluate the soundness of the design. For the electrical power system, a suitable simulation environment must be capable of addressing these questions:

Will the Electric Power System Work?

This is the ultimate question which needs to be answered. Unfortunately defining the term *work* is not an easy task, nor is assuring a system will work under all operating conditions. A strict time domain simulation only provides a solution for a given set of operating conditions. Generalizing the results of relatively few simulations to all operating conditions is both necessary and prone to catastrophic failure. Hence more than just a time response is usually needed.

How Will the System React to Major Disturbances and Faults?

The primary design goal for shipboard electric power systems is continuity of power. To this end, the response of the system to abnormal events such as grounds, stalled motors, and inadvertent opening of breakers is crucial to evaluating the success of the electric power system design.

How Will the System React to Severe Dynamic Conditions?

A number of normal events can cause severe dynamic responses within the system. Rapidly changing the propulsion motor speed or direction, discharging pulse weapons, or starting large motors are all examples of normal dynamic events.

Is the System Stable During a Given Dynamic Scenario?

One import aspect of a system that *works* is its stability. The system should remain stable during all normal dynamic conditions and for as many disturbances and faults as possible.

What is the Stability Margin?

Some measure of how stable the system is should be provided to assist in generalizing the findings of stability about one scenario to other related scenarios.

What is the Sensitivity of the Results to Parameters?

The generation of models for a dynamic system simulation requires some estimation of device parameters. Knowledge of the sensitivity of the simulation results to parameter estimation errors is crucial for correlating simulation results with what can be expected from the physical system.

How Correct are the Answers Provided to the Above Questions?

Simulations generally use numerical methods to generate solutions. Careful control of error propagation is very important in ensuring accurate conclusions can be drawn from the simulation results. Some form of feedback should be provided to the operator as to the confidence level of the results.

These requirements for performing time domain simulations of proposed and existing electric power systems found on United States naval warships can be quite challenging. The size, complexity, and strong coupling of components all conspire to make the simulator's task difficult. At first glance, one would think the simulation programs designed for the commercial power utilities would be sufficient for handling the smaller shipboard systems. Unfortunately, this is not the case due to the following differences of the shipboard system from commercial power systems:

Variable Frequency

Frequency cannot be assumed constant. Many IED designs have the generators, motors, and ship service power all operating at different frequencies to optimize the performance of individual components. Frequency changers are employed to convert the power from one frequency to another. Even the ship service system onboard mechanical drive ships can experience frequency fluctuations lasting up to 2 seconds. The limited rotational intertia of the prime movers and generators allows for rapid accelerations and decelerations of the shaft and corresponding frequency fluctuations.

Lack of Time Scale Separation

The principal time constants of controls, machine dynamics, and electric dynamics all fall within the same general range of milliseconds to seconds. The practice of decomposing the problem by time scale separation often used in analyzing commercial power systems becomes much more difficult.

Load Sharing instead of Power Scheduling

The commercial power utilities operate by scheduling the power delivered by each of the generating units. The mismatch between scheduled power generation and the actual load is met by a swing generator. Onboard ship however, both real and reactive power are shared equally among all paralleled generators through the very fast exchange of load sharing information. This fast exchange of information strongly couples the dynamics of all the paralleled generators.

Short Electrical Distances

The distances onboard ship are so short (under 1000 ft) as to make the modelling of transmission lines unnecessary for most simulations and to trivialize the load flow problem which is so important to the commercial power sector. The short electrical distances also strengthen the coupling of the various subsystems making up the electrical power system.

Load Dynamics

Commercial utilities usually assume loads are either consuming constant real and reactive power, or are constant impedances. Shipboard systems however, must account for dynamics of loads such as propulsion motors, large pumps, pulsed loads, propeller dynamics, and ship dynamics. Furthermore, the supervisory level control envisioned for future designs may have the ability to control aspects of the loads in addition to generation.

Tighter Control

Because a ship is relatively small, a higher level of control can be exercised over the shipboard power system than can be exercised in the commercial power industry. This higher level of control strengthens the dynamic coupling of components of the system and complicates simulation efforts.

Clearly, shipboard systems are considerably different from commercial power systems; and the inapplicability to shipboard power systems of simulation techniques developed for commercial systems should not be surprising. Other simulation tools exist but for one or

more reasons, all are ill-suited for simulating shipboard systems. A review of currently available commercial software for solving lumped parameter systems reveals no program currenty exists suitable for fulfilling the needs of simulating shipboard electric power systems.

Circuit Simulators

As will be discussed in following sections, circuit simulators almost universally describe devices in terms of branch voltages and currents. The constitutive relationships are based only on the relative difference of the terminal variables and can not depend on the actual nodal potentials. Furthermore, the flow variables must be conserved on the device level. While these restrictions are not of concern for electrical networks, they are a bit constraining on electro-mechanical systems where one would like to deal with energy transformations in a device by employing equations which do not conserve the flow variable on the device level. The torque on the input shaft of a gearbox for example, does not equal the torque on the output shaft. Even electric power models where the flow variable is power and the potential variable is voltage can best be described by constitutive equation which do not enforce conserving power by ignoring the power converted to heat through resistive losses.

Many circuit simulator also have problems modelling the transfer of information which is common in systems employing control systems. Information has only potentials and no flows associated with it.

Signal Analysis Software

Signal Analysis Software is often used to simulate control systems but often have difficulty simulating energy transfer. In particular, these programs often are incapable of solving implicit equations which are typically created by writing Kirchhoff's Current Law when developing systems. Instead much effort must be expended to ensure the models have the appropriate input and output variables for a given system to be built.

Commercial Power Utility Analysis Programs

Software for analyzing commercial power universally do not apply to shipboard systems due to several key differences. The lack of transmission lines, rotational inertia, time scale separation of dynamics and the presence of tightly coupled control loops are all features of the shipboard system which prevent the use of the commercial power system analysis techniques [5] [9] [10] [11].

General Differential Equation solvers

Most general differential equation solving programs cannot handle implicit equations very well. While the development and interconnecting of models into systems is possible, the process can be quite cumbersome [12] [13]. Dynamically stiff systems also pose serious challenges to the general differential equation solvers.

Hybrid Computers

Hybrid computers for studying electrical power systems can answer many of the desired questions for small shipboard systems. Unfortunately, hybrid computers are very expensive and limited in the size of problems which can be addressed. Presently the only hybrid computer in the United States suitable for shipboard power system studies is located at Purdue University. While this machine is capable, the needs of the IED program will require digital computer techniques for performing the desired studies. [92] [93] [94] [95] [96]

As part of an effort to fill the need for simulating shipboard power systems, the WAVESIM program was specially created to develop applicable simulation techniques. Before discussing the numerical methods employed in WAVESIM, a review of existing methods is appropriate.

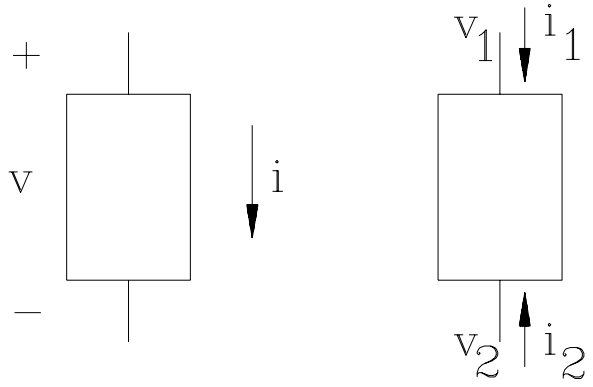
1.1 Simulation Process

The process of simulating a physical system can be broken into three steps. First, a system of equations describing the component device constitutive relationships as well as the network constraints must be developed. While the network constraints are purely linear algebraic equations, the constitutive equations can be nonlinear and dynamic. Together, a system of nonlinear differential algebraic equations is generated. The next step is the conversion of the system of differential algebraic equations into a sequence of purely algebraic equations. Common integration techniques include the forward and backward Euler methods, Trapezoidal rule integration, and the Runge-Kutta methods. Finally, the nonlinear algebraic system is solved either through the Newton-Raphson method or through one of several relaxation techniques.

Before describing several methods for generating and solving the system of equations corresponding to a physical system, the difference between the branch description and terminal description of devices should be detailed. The branch description of devices requires all the constitutive relationships be based on the relative difference between terminal potentials and all flows entering a device also leave the device. Hence for a two

terminal device, there is a single branch potential variable and a single branch flow variable associated with it. In the terminal description, the potential and flow associated with each terminal are variables. A two terminal device would then have four variables associated with it: two flow variables and two potentials. The terminal description allows the constitutive equations be a function of the actual values of the terminal potentials and not only of the potential difference. In other words, the potential reference can be set at the system level and not necessarily on the device level. Furthermore, the flows are not required to be conserved. A gear box for example, has differing torques entering and exiting it. The branch description method requires a four terminal model of the gear box while the terminal description requires only two terminals. In either case the result would be four variables describing the gearbox, but the branch description requires an explicit declaration of the device potential reference while the terminal description uses an implicit system wide potential reference.

Branch Description vs. Terminal Description



1.2 Developing System Equations

The normal method of describing a dynamic system is to generate a consistent set of possibly nonlinear differential algebraic equations and arrange them into the following canonical form:

$$C\dot{\bar{x}} = f(\bar{x}, \bar{y}, \bar{u})$$

$$0 = g(\bar{x}, \bar{y}, \bar{u})$$

where $\bar{x}$ is the vector of dynamic state variables, $\bar{y}$ is the vector of state variables with no associated dynamics staes, and $\bar{u}$ is the vector of system inputs. This system of differential algebraic equations (DAE) can be generated several different ways with the most common being the Sparse Tableu, Modified Nodal Analysis, and the standard load flow method.

The method employed in WAVESIM does not extract the differential equations from the device constitutive equations but instead forms a system of algebraic equations of the form:

$$0 = g(\bar{x}, g_i(\bar{x}_i, u_i), \bar{u})$$

where $\bar{x}$ is the vector of the system variables and $\boldsymbol{g_i()}$ is a device function having subsets $\bar{x}_i$ and

$$\bar{u}_i$$

of $\bar{x}$ and $\bar{u}$ as arguments. The functions $\boldsymbol{g_i()}$ have the dynamics of the device contained within them, but these dynamics are not expressed on the system level.

1.2.1 Sparse Tableau Method

The Sparse Tableau method is a very general method for describing systems employing the branch description of devices. Proposed in [4] and used in the ASTAP and SPICE simulators [1][15][16], the Sparse Tableau method breaks the system equations and variables each into three groups. The three sets of variables are the branch currents, branch voltages, and the nodal voltages. The three sets of equations are the node Kirchhoff Current Law (KCL) equations in terms of the branch currents, Branch Voltage equations relating branch voltages to nodal voltages, and the Constitutive equations relating branch voltages to branch currents.

Figure 1.2.1-1 RC Example: Sparse Tableau

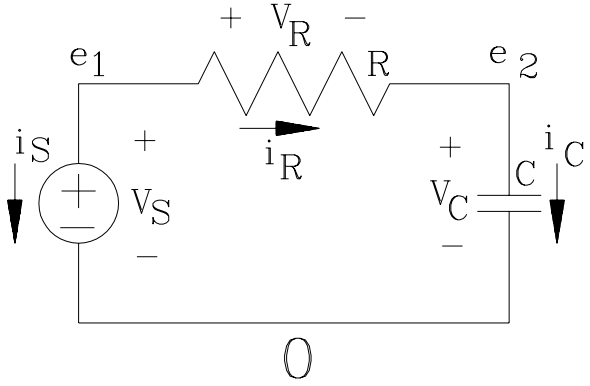


Figure 1.2.1-1 shows an example of a simple RC charging circuit. Using the Sparse Tableau approach, the system variables are:

i_S	Voltage Source branch current
i_R	Resistor branch current
i_C	Capacitor branch current
v_S	Voltage Source branch voltage
v_R	Resistor branch voltage
v_C	Capacitor branch voltage
e_1	Node 1 potential (voltage)
e_2	Node 2 potential (voltage)

Note the reference node **0** is assigned a potential of **0.**

The KCL equations are given by:

$$i_S + i_R = 0 \qquad -i_R + i_C = 0$$

The Branch Voltage equations are:

$$v_S - e_1 = 0 \qquad v_C - e_2 = 0$$

$$v_R - (e_1 - e_2) = 0$$

The Constitutive equations are:

$$v_S - V_S = 0 \qquad i_C - C\frac{dv_C}{dt} = 0$$

$$v_R - i_R R = 0$$

While the Sparse Tableau approach generates a consistent set of network equations, the size of the system is relatively large (eight equations in eight unknowns for this example). Furthermore, the method employs the branch description of devices which complicates the development of electro-mechanical models.

1.2.2 Modified Nodal Analysis

The Modified Nodal Analysis method generates a compact set of system equations for systems of device models using branch descriptions. Modified Nodal Analysis (MNA) was formalized in [6], described in [1][16], and employed in the circuit simulator MSINC. The procedure consists of writing the KCL equations for all but the reference node in terms of the branch currents, replacing the branch currents wherever possible with constitutive equations in terms of the branch voltages, appending any remaining constitutive equations, and substituting the branch voltages with the corresponding nodal voltages.

Figure 1.2.2-1 RC Example: Modified Nodal Analysis

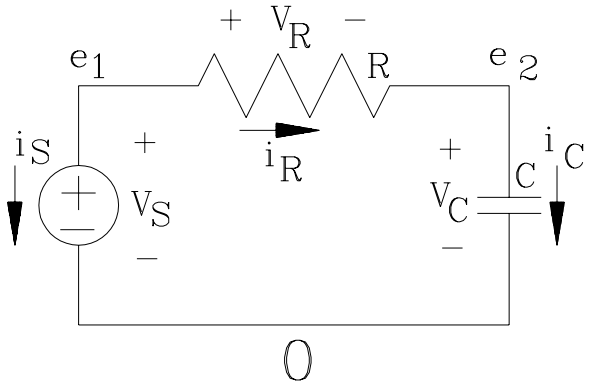


Figure 1.2.2-1 shows a simple example of a simple RC charging circuit, the MDA approach would first write the KCL equations:

$$i_S + i_R = 0 \qquad\qquad -i_R + i_C = 0$$

Substituting the constitutive relations results in:

$$i_S + \frac{1}{R} v_R = 0 \qquad\qquad -\frac{1}{R} v_R + C\frac{dv_C}{dt} = 0$$

The extra constitutive equation is given by:

$$v_s - V_S = 0$$

Substituting the nodal voltage results in the system of three equations

$$i_S + \frac{1}{R}(e_1 - e_2) = 0 \qquad -\frac{1}{R}(e_1 - e_2) + C\frac{de_2}{dt} = 0$$

$$e_1 - V_S = 0$$

While the Modified Nodal Analysis Method generates a compact set of equations, it does require the use of the branch description as well as the explicit definition of flow variables. Both restrictions can complicate teh modelling of electro-mechancial devices.

1.2.3 Standard Load Flow

The Load Flow approach is traditionally used in the analysis of commercial power systems. For this application, the flow variables are usually real and reactive power while the potential variables are the voltage magnitude and phase angle. The Load Flow approach is a variation of nodal analysis described in many papers and texts on power systems including [14] [29] [31] [35] [49] [50] [76] [101]. The terminal description of devices is used since power flow is not conserved on the device level (The power entering a transmission line is not the same as the power leaving the same line). The basic procedure is to write the *KCL* equations in terms of the node potentials. Nodes with ideal potential sources are treated specially since their corresponding flow variable is not a function of the device voltages.

Figure 1.2.3-1 RC Example : Load Flow

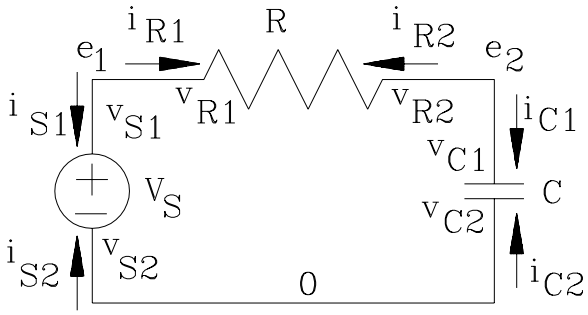


Figure 1.2.3-1 shows a simple RC charging circuit using the terminal description of the devices. A load flow approach using currents as the flow variable would result in the following procedure:

Write the KCL Equation at nodes without potential sources

$$i_{R2} + i_{C1} = 0$$

Substitute Constitutive relationships for the flow variables

$$\frac{1}{R}(v_{R2} - v_{R1}) + C\frac{d(v_{C1} - v_{C2})}{dt} = 0$$

Substiute the nodal potentials

$$\frac{1}{R}(e_2 - V_S) + C\frac{de_2}{dt} = 0$$

All the remaining variables can be calculated from the solution of this differential equations. The load flow method definitely creates a very compact set of equations (only one in this case) but requires the flow variables be defined explicitly in terms of the potential variables, and must treat ideal potential sources as special exceptions. Neither of these restrictions is attractive for a general electro-mechanical system simulator.

1.2.4 WAVESIM Terminal Description

The method employed in this thesis is similar to Modified Nodal Analysis with the exception that terminal potentials are used instead of branch voltages and that the constitutive equations are only expressed on the device level and never expressed on the system level. Potential difference equations are appendended to the system of KCL equations to equate explicitly defined potentials with their node potentials. For the RC example, the system variables are given by:

i_{S1} Voltage Source terminal 1 current

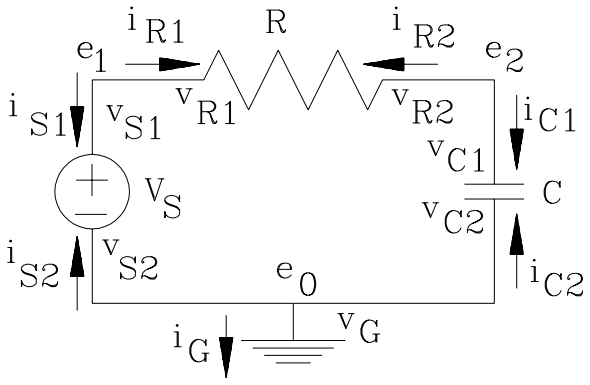
i_{C1} Capacitor terminal 1 current

e_0 Node 0 potential (voltage

e_1 Node 1 potential (voltage)

e_2 Node 2 potential (voltage)

Figure 1.2.4-1: RC Example: Terminal Description



The KCL Equations for the RC example are given by:

$$i_{S1} + g_{R\_iR1}(e_1, e_2) = 0$$

$$i_{C1} + g_{R\_iR2}(e_1, e_2) = 0$$

$$i_G + g_{S\_iS2}(i_{S1}, e_0) + g_{C\_iC2}(i_{C1}, e_0) = 0$$

The Potential Difference Equations are given by:

$$e_2 - g_{C\_vC1}(i_{C1}, e_0) = 0$$

$$e_1 - g_{S\_vS1}(i_{S1}, e_0) = 0$$

$$e_0 - g_{G\_vG}(i_G) = 0$$

Note that a reference device allowing for a more general method of setting the system reference points is employed rather than a reference node. While the number of equations is twice that of the Modified Nodal Analysis method, flows need not be conserved on the device level. Furthermore, the system of equations is easily partitioned into a sequence of five blocks for a more rapid solution (two 1×1 blocks, followed by a 2×2 block, followed by two more 1×1 blocks).

1.3 Solving System Equations

As stated earlier, the standard approach to simulating a physical system is to generate a system of differential algebraic equation of the form:

$$C\dot{\bar{x}} = f(\bar{x}, \bar{y}, \bar{u})$$

$$0 = g(\bar{x}, \bar{y}, \bar{u})$$

To solve this system, it must first be converted to a system of purely algebraic equations by substituting the differential equations with discrete approximations. The time history of a variable is expressed as a series of discrete points in time where dynamics are expressed as algebraic relationships between the values of a variable at different discrete times. Standard methods for performing this approximation include the forward and backward Euler, Trapezoidal rule integration and Runge-Kutta methods.

The major problem with this approach is the dependence of the time step on the fastest mode (smallest eigenvalue) of the dynamic system. This forces the entire system be solved with a very fine discretization of time, even though large portions of the system are not affected by the fast mode.

In any case, the system of nonlinear algebraic equations must be solved. The two classes of solvers most commonly used are variations of the Newton-Raphson Method and several relaxation methods.

1.3.1 Newton-Raphson Method

The Newton-Raphson method works well for most systems as long as the initial guess for all of the variables are within the convergence region of the final solution. This method is used in SPICE and ASTAP and is based on a Taylor series expansion of the system of equations:

$$\bar{F}(\bar{x}) = 0 = \bar{F}(\bar{x}_k) + \bar{J}(\bar{x}_k)\bar{\Delta}_k + \ldots$$

$$\bar{x}_{k+1} = \bar{x}_k + \bar{\Delta}_k$$

$$\bar{\Delta}_k = -\bar{J}^{-1}(\bar{x}_k)\bar{F}(\bar{x}_k)$$

The matrix $\bar{J}$ is called the **Jacobian Matrix** and its inverse must exist for the method to work.

1.3.2 Relaxation

Relaxation methods assign one of the system variables to each of the system equations. After initial guesses are made for each of the variables, the variables are updated by solving their corresponding equation assuming none of the other variables have changed. The two most popular relaxation methods are the Gauss-Jacobi (popular with parallel processing computers) and the Gauss-Seidel method (usually used with serial processing computers). The Gauss-Jacobia calculates updates for all the system variables before actually performing the update:

$$\overline{F}(\overline{x}) = 0$$

$$\overline{F}_i([x_{1,k}, x_{2,k}, \ldots, x_{i,k+1}, \ldots, x_{n-1,k}, x_{n-1,k}]^T) = 0$$

The Gauss-Seidel method updates the system variables as the updates are calculated:

$$\overline{F}(\overline{x}) = 0$$

$$\overline{F}_i([x_{1,k+1}, x_{2,k+1}, \ldots, x_{i,k+1}, \ldots, x_{n-1,k}, x_{n-1,k}]^T) = 0$$

1.3.3 Waveform Relaxation

An alternate method to solving the dynamic equations system wide is to solve them equation by equation over a given time interval. The Waveform Relaxation method represents variables by a sequence of points representing the time history of the waveform over a given time interval. Each variable can be discretized differently and is assigned one of the system equations. The system equations are solved over the waveform interval for their assigned variable with the other variables held at their current waveform values.

Waveform Relaxation works well with loosely or directionally coupled systems, but does not work well for tightly coupled systems. The method does however, have good multirate performance since each differential equation can be solved using a time increment appropriate to it.

1.3.4 WAVESIM Approach

To summarize the traditional solving methods, the standard methods employing Netwon-Raphson can handle tightly coupled systems but perform poorly with multirate systems while waveform relaxation performs poorly with tightly coupled systems but

efficiently solves multirate problems. Unfortunately, the shipboard systems have both multirate and tightly coupled properties. For this reason, WAVESIM combines the Newton-Raphson method with a waveform representation of variables.

In WAVESIM variables are represented over a time interval by a vector of coefficients along with a type indicator for specifying how the coefficients should be interpreted. Common interpretations include Legendre Series coefficients, Chebyshev Series coefficients, and polynomial series coefficients. For these representations, integration and differentiation are linear matrix operations and the issue of numerical stability of an integration technique disappears. Waveforms can usually be converted from one type to another with a linear matrix operation as well.

With variables represented as vectors of coefficients, the Newton-Raphson method can be employed for solving tightly coupled systems. Good multirate performance is achieved through the linear matrix operator for integration along with waveform smoothing to average out phenomena faster than the time scale of interest.

1.4 Thesis Outline

This thesis focuses on developing a digital computer simulation environment suitable for studying shipboard electric power systems. WAVESIM, a simulation program written in the C programming language demonstrates algorithms for simulating systems of nonlinear lumped parameter models representing the electro-mechanical components composing an IED system. The key features of WAVESIM are:

Devices defined independent of the encompassing systems

Devices can be developed and tested without an exact knowledge of the topology of the systems incorporating the devices.

Devices described using the Terminal Representation of devices

Device constitutive relationships are written in terms of the actual values of the terminal potentials and not in terms of relative potentials. In this manner, device equations can be written in terms of a system reference when such a reference level is unambiguous. Furthermore, the flow variables are not required to be conserved on a device level. This greatly eases the task of modelling flows which also depend on a reference potential (power for example).

Devices defined independent of the manner in which terminal interface variables are expressed.

Devices can be developed without specifying how the interface variables are specified. In WAVESIM, variables can be represented many different ways, all of which are irrelevant to the specification of the constitutive equations making up the device.

System equations instead of the devices resolve *input-output* conflicts.

WAVESIM does not constrain normal terminals where energy is transferred from having more than one output hooked together at a node.

Interface Variables represented by waveforms

Waveforms are a vector of coefficients which specify a given variable over a given time interval instead of a single value describing the variable at a given point in time. The waveform type determines how the coefficients should be interpreted for generating values of the variable within the time interval. Representing variables as waveforms has the primary benefit of removing the issue of numerical stability of integration techniques from the simulation. Integration and differentiation are merely operators on waveforms, no different from addition, subtraction, or any of the trigonometric operators.

Differentiation and Integration performed on the device level instead of the system level.

Most circuit simulators as described in the previous sections solve the differential equations associated with device constitutive equations on a system level. This method eases the task of evaluating the stability of linear systems but introduces new problems. If the eigenvalues of a dynamic system are widely separated in value, the simulation time step must be made very small for the entire system if conventional integration techniques are employed. WAVESIM solves the differential equations on the device level and employs waveform smoothing to remove dynamics which occur faster than the time scale of interest.

While many of the pieces of WAVESIM are not new, several key concepts are presented in this thesis for the first time:

The Terminal Description of devices

Instead of specifying the interface of devices by ports consisting of a potential difference (branch voltage) and the flow through the potential difference (branch

current), the terminal description of a device assigns a potential and a flow entering the device for each normal terminal. Simulators based on branch voltages and currents require all of the flow entering a device to also leave the device. In this sense, the flow is conserved. The terminal description however, does not require conservation of flow within a device (Conservation of flow as expressed by Kirchhoff's Current Law - KCL is required at connection points called nodes). The ability to construct models which do not conserve flows can simplify models where energy transformations occur, the reference potential is clearly known for the system and not just for the device, and certain forms of energy are not of interest. In many mechanical simulations for example, the amount of energy lost in friction is not of interest to the modeler. A simulation model based on branch potentials and flows of a device experiencing friction would be required to reject the frictional heat through one of its branches.

The terminal description also allows for the transfer of information between devices through information nodes and information terminals. This feature is essential for successfully modelling many control algorithms. The ability to mix control signals and energy transfer through flow variables within the same simulation environment is a major advantage of the terminal description.

The Structural Jacobian method for building and reducing systems

The concept of the *connection* matrix for specifying the participation of system variables in system equations is expanded to include the structural form (i.e. diagonal, linear, nonlinear, etc.) of the dependence of the system equations on the system variables. The codes for the structural Jacobian adhere to a simple set of algebraic rules which can be used to construct a system structural Jacobian matrix from the individual device structural Jacobians. The system structural Jacobian facilitates the reduction of the numerical effort required to solve the system by identifying and characterizing a set of smaller blocks which when sequentially solved, determine all of the system variables. The system structural Jacobian can also be used to detect unconnected systems and indicate possible potential reference problems.

The Systematic Treatment of Waveforms as an abstract data type

WAVESIM departs from the conventional paradigm of representing variables in a dynamic simulation by a series of discrete points in time with a new paradigm based on representing variables as a sequence of waveform intervals. Within each waveform interval, the value of the waveform can be directly determined for any time

based on a vector of coefficients, a waveform type indicator for specifying how the vector of coefficients should be interpreted, and the time boundaries of the waveform interval. Devices are defined independent of the waveform type of the terminal variables. The principle advantage of using waveforms is that integration and differentiation are simple operators. The integral of a waveform is just another waveform. Simulation time steps are no longer controlled by the requirement for numerical stability of the integration technique. Instead, series truncation error control becomes the primary concern of the simulation environment. The ability to use arbitrary waveform types and convert between types allows the modeler to use the most appropriate waveform representation for the modeling problem.

This thesis is composed of six chapters including this introduction. Chapter Two describes in some detail the specific properties of current shipboard electric power systems and proposed integrated electric drive systems. Chapter Three provides a framework of theory for developing the simulation environment WAVESIM and is broken into five subsections. The first subsection details the Terminal Description method for modelling devices. The second subsection demonstrates how to interconnect device models into systems, construct the system structural Jacobian, and generate a sequence of blocks for solving the system equations. The third subsection covers the treatment of waveforms as an abstract data type. Solving the system of equations employing waveforms is detailed in the fourth subsection. The fifth and final subsection of the third chapter covers modelling techniques and considerations not covered in previous sections. The actual WAVESIM implementation of the concepts developed in the third chapter are described in the fourth chapter. The fifth chapter presents results of several simulations conducted with WAVESIM. The final chapter provides an assessment of the work presented here as well as possible future developments.

The appendices support the main chapters. Appendix A is a glossary of terms used through out this thesis. Appendix B details some possible problems with using continuation parameters. Appendices C and D are Load Flow examples of the terminal description method. Appendix E provides examples of waveform types and a number of operators for them. Appendix F presents a number of models useful for conducting shipboard power system simulations. Finally, Appendix G details the program files making up WAVESIM.

This thesis introduces a number of new terms. To assist the reader, the first occurance of a new term is indicated by the distinctive **Helvetica** typeface. MATLAB variable names and sample sections of C programs are printed in **`Courier`**.

Chapter 2 Shipboard Electric Systems

2.1 Typical Shipboard Electric Distribution System

The electric power systems onboard naval warships differ considerably from the integrated power utilities found in developed countries. The differences arise from the small size of the shipboard systems and contrasting standards for optimization. Shipboard systems are optimized for survivability and minimization of weight and volume. Power utilities on the other hand optimize for reliability and minimization of cost. The unique characteristics of the shipboard systems result in markedly different design requirements and standards as compared to power utilities.

Frigates, destroyers and cruisers are relatively small warships with corresponding small electric power systems. Frigates normally displace from 2000 to 4000 long tons (1 long ton = 2240 lbs) and have a primary mission of escorting merchant convoys. In the U.S. Navy, frigates have only one propulsion shaft and about half the armament of a destroyer. Destroyers displace from 4000 to 7500 long tons and are designed as escorts for aircraft carrier battle groups. Cruisers are larger than destroyers, displacing from 6000 to 16000 long tons, carry more weapons, and are used to provide aircraft carrier battle groups with integrated anti-aircraft and anti-cruise missile defenses. U.S. Navy cruisers and destroyers all have two propulsion shafts.

The installed electric plant capacity for U.S. warships has varied from 3000 KW to 4500 KW per propulsion shaft over the past twenty years. Generally, the newer ships have more installed capacity. Figure 2.1-1 shows the electric plant characteristics for the major classes of conventionally fueled frigates, destroyers and cruisers constructed in the past twenty years. All the listed ships with the exception of the Knox class frigates use mechanically coupled gas turbine propulsion. The Knox class frigate is the last class of conventionally fueled warships to use 1200 psi steam for main propulsion. (All nuclear powered ships use 600 psi steam). Most of the Knox class frigates are presently being transferred to the reserve forces or being decommissioned.

Figure 2.1-1 U.S. Navy Ship Characteristics				
Ship Class (Nbr)	**Name**	**KW**	**Generator Type**	**Year**
FF-1052 Frigate (46)	Knox	4 × 750 KW	3 Steam Turbine 1 Diesel	1969
FFG-7 Frigate (51)	Oliver Hazard Perry	4 × 1000 KW	4 Diesel	1977
DD-963 Destroyer (31)	Spruance	3 × 2000 KW	3 Gas Turbine	1975
DDG-993 Destroyer (4)	Kidd	3 × 2000 KW	3 Gas Turbine	1981
DDG-51 Destroyer (1 + 28)	Arleigh Burke	3 × 3000 KW	3 Gas Turbine	1991
CG-47 Cruiser (19 + 8)	Ticonderoga "Aegis"	3 × 2500 KW	3 Gas Turbine	1983

Figure 2.1-2 Shipboard Electric Distribution System

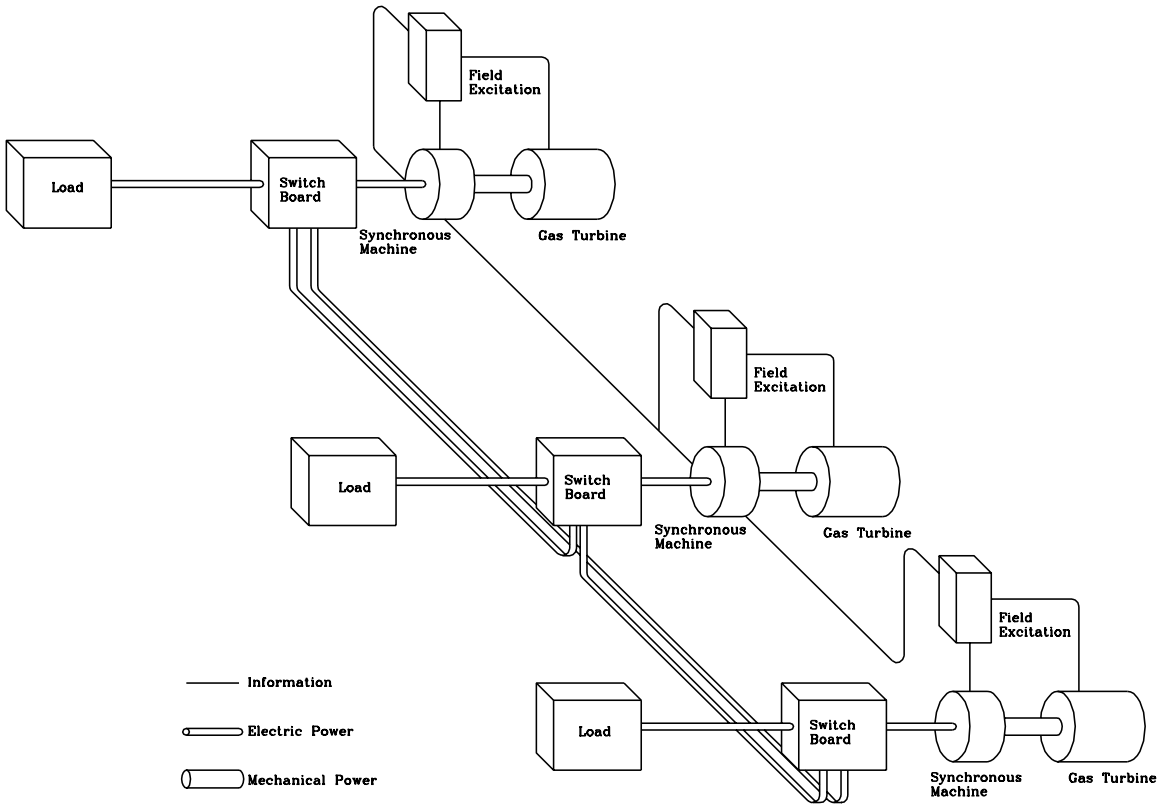


Figure 2.1-2 shows a typical ring bus architecture found on modern warships. The small size of the shipboard system results in many differences with respect to commercial systems. As a consequence the analysis of the shipboard plant requires recognition of these differences:

1. Power Quality requirements relaxed relative to commercial standards. Constant frequency and voltage assumptions can not be made. See section 2.2 for more details.
2. Very little Rotational Inertia require fast controls to maintain frequency. Infinite bus assumption does not hold.
3. Transmission lines are very short and for the most studies, can be ignored.
4. No scheduling of real or reactive power. All generators are loaded in equal proportion to their rating. Load Flow solution has little meaning.
5. Load sharing information communicated to all online generators.
6. Large loads (relative to the size of generation plant) present. Start up transients (load dynamics) are important.
7. Power Electronic Switching loads are significant.
8. Load shedding strategies are minimal.

Figure 2.1-2 also indicates the requirement for a simulation environment to include the ability to model more than just electric power phenomena. Modelling shipboard systems also requires extensive representation of mechanical dynamics as well as energyless information transfer between components. This requirement is significant in that simulation packages for commercial power systems do not include this capability as an integral part of the simulation environment design.

2.2 Shipboard Electric Plant Standards

The primary standards for designing a shipboard electric plant are contained in the following references:

Department of Defense, **Interface Standard for Shipboard Systems, Section 300A, Electric Power, Alternating Current (Metric)**, MIL-STD-1399(NAVY), 13 October 1987.

Department of the Navy, **General Specifications for Ships of the United States Navy, Section 300, General Requirements for Electric Plant**, Naval Sea Systems Command, 1987.

Department of the Navy, **General Specifications for Ships of the United States Navy, Section 320, General Requirements for Electric Power Distribution Systems**, Naval Sea Systems Command, 1987

The goal of electric power utilities is to provide a reliable source of high quality electric power at minimum cost. Shipboard systems on the other hand are designed to provide a survivable and continuous source of electricity. Quality and cost are secondary issues. Figure 2.2-1 summarizes the minimum quality of power a shipboard system must provide

Figure 2.2-1 clearly demonstrates the quality of power guaranteed onboard a warship is considerably lower than the quality of service provided by power utilities. Figure 2.2-1 does not show however, how often the transient conditions occur. This information is provided by MIL-STD-1399 and summarized in figure 2.2-2. A major ramification of the low quality of power provided by the ship service electric system is that loads must be designed to operate and survive wide ranges of voltage and frequency fluctuations. This is one of the reasons why commercial equipment often can not be directly installed onboard ships (Shock requirements are also a major factor). Sensitive loads must provide their own filtering and protection circuitry. This *militarization* of equipment can add considerable cost and complexity to warship design, outfitting and maintenance.

Figure 2.2-1 : Shipboard Electric Power Quality Standards (MIL-STD-1399)	
Frequency	
Nominal	60 Hz
Tolerance	± 3 %
Modulation[1]	0.5 %
Transient Tolerance	± 4 %
Transient Recover Time	2 seconds
Worst Case Excursion	± 5.5 %
Voltage	
Nominal	440/115 Volts
Tolerance of 3 Phase Ave	± 5 %
Tolerance of any 1 Phase	± 7 %
Line Voltage Unbalance[2]	3 %
Voltage Modulation	2 %
Transient Tolerance	± 16 %
Maximum Departure Voltage from combination of 3 Phase Ave. and Voltage Modulation	± 6 %
Worst Case Excursion	± 20 %
Recovery Time	2 Seconds
Voltage Spike[3]	2500 / 1000 Volts
Voltage Waveform	
Max Total Harmonic Distortion[4]	3 %
Max Single Harmonic	2 %
Max Deviation Factor[5]	5 %
Emergency	
Frequency Excursion	-100 % to +12 %
Voltage Excursion	-100 % to +35 %
Duration	2 Minutes

1 Modulation (percent) = $\frac{E_{\max} - E_{\min}}{2E_{nominal}} 100$ measured over a period of 1 to 10 seconds.

2 Line Voltage Unbalance is the difference between the largest line to line voltage and the smallest line to line voltage divided by the nominal voltage.

3 A Voltage Spike is a voltage change of less than 1 ms duration.

4 Total Harmonic distortion is the ratio of the rms value of the residue (after elimination of the fundamental) to the rms value of the fundamental.

5 Deviation Factor is the ratio of the maximum difference between corresponding ordinates of the waveform and an equivalent sine wave to the magnitude of the equivalent sine wave. The equivalent sine wave is defined as having the same frequency and rms voltage as the wave being tested.

Figure 2.2-2 Shipboard Electrical Reliability	
Voltage Transients of 10% or less	Several times an hour
Voltage Transients of 10% to 16%	Several times a day
Voltage Spikes above 200 Volts	About once every 3 hours

The basic reason for the low quality of power onboard ship is the lack of rotational inertia in the power system. In the commercial sector, the inertia of all the generators in the network add up to such a large number that no single fault can cause a frequency disturbance system wide. Onboard ship however, generators are often operated independently. Other than the inertia provided by motors, the only source of rotational inertia is the one generator. Since the generators are not very large, sudden load changes and faults can cause significant disturbances. Although speed governors and voltage regulators have improved significantly in the past twenty years, there is presently no way to prevent the transients from happening.

The frequency tolerance limits in the steady state are rarely ever approached in modern warships. The rather loose tolerances allowed the use of droop governors to stably share loads. The electric plant operator on older ships could increase the load on a paralleled generator by increasing the base frequency set point on the mechanical speed governor. Adjusting the system frequency without changing the load sharing ratios required adjusting the base frequency set points on all the generator speed governors. On modern warships, all the generators normally operate isosynchronously and perform load sharing by transmitting load current information to Governor Control Units which provide feedback to the isosynchronous governors.

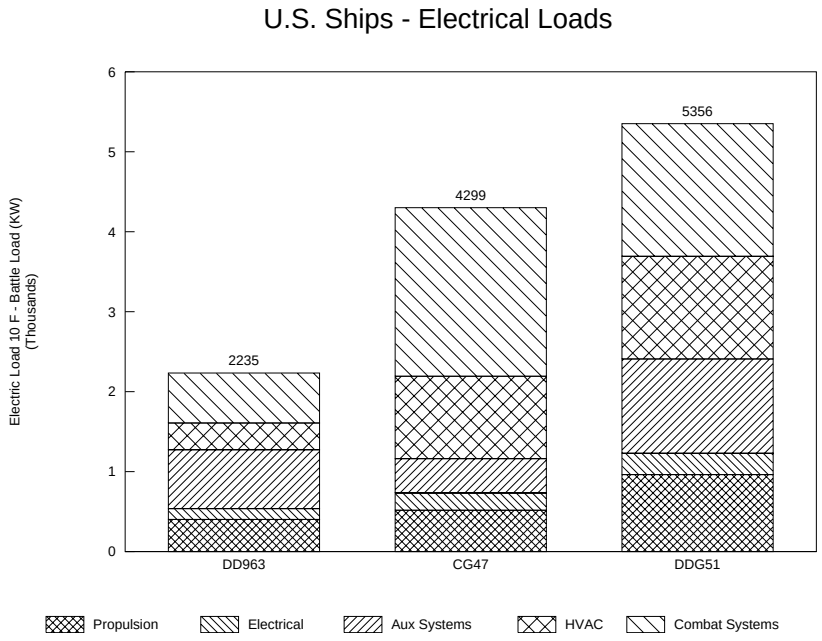
2.3 Shipboard Electric Plant Design

In the commercial sector, the design of electric generation and transmission capacity are done continuously. Ships on the other hand, have a finite life (typically thirty years) and the expense of upgrading the capacity of the electric plant and distribution system once the ship is built is usually prohibitive. In this sense, capacity expansion onboard ships is not done. Instead, excess capacity is initially installed to account for projected growth in load.

The maximum load for a ship design is determined by tabulating every load in an Electrical Load Summary and summing up the power requirements under different operating conditions. The maximum projected load usually occurs when the ship is in battle condition and the ambient temperature is low (Electric heaters are used in many areas of a ship). To account for uncertainty in estimating loads, a 20 % margin is added to the

maximum projected load. Another 20 % margin is added for capacity expansion requirements. Ninety percent of the capacity of all but one of the installed generators must meet or exceed the margined maximum projected load. The ninety percent requirement allows for imprecise load sharing when at maximum load while the *all but one* requirement accounts for taking one generator off line for maintenance.

Figure 2.3-1



Once the size of the electric plant is determined, there are a number of other considerations that must be accounted for. GENSPECS[6] require the system be ungrounded and based on Split Plant Operation (Each generator operating independently) with the capability for parallel operation. Electromagnetic Interference (EMI) and Electromagnetic Pulse (EMP) requirements place further constraints on the electric plant design and are detailed in MIL-STD-461 and MIL-STD-1310. Since warships are designed for combat, they must also be capable of surviving severe mechanical shocks from exploding ordnance. The shock requirements are particularly important for electrical equipment such as circuit breakers and generators. Specific requirements for shock are listed in MIL-STD-901.

6 General Specifications for Ships of the Unites States Navy

A number of loads onboard a ship are very important for survival of the ship and crew during combat and emergencies. These loads are designated **vital loads** and must be provided with primary and alternate sources of power. Some of the vital loads have automatic bus transfer switches (ABT) which switch to the alternate source automatically on loss of the primary source. Others use manual bus transfer switches (MBT). Examples of vital loads include:

Collective Protection System Class W Ventilation	ABT
Emergency Communications	MBT
Emergency Lighting	ABT
Fire Pumps	ABT
AFFF Pumps	ABT
Interior Communications	ABT
Machinery Space Circle W Ventilation	MBT
Steering Gear Auxiliaries	ABT
Surface Search Radar	MBT
VHF Bridge-to-Bridge Radio	MBT
Vital Propulsion Auxiliaries	MBT and ABT
Auxiliaries to support generator prime movers	MBT

From a naval architectural viewpoint, the placement of electric generators requires a number of compromises. Placing the heavy generators as low as possible is beneficial for hydrostatic stability purposes. The lower the generator however, the more volume is required for intake and exhaust ducting. Gas turbine generators are lighter than diesel generators, but require greater volumes of air. Furthermore, design requirements exist for separating 50 % of the installed capacity by two watertight bulkheads and installing a minimum of three generators. Generally, weight can be minimized by using the smallest number of generators (three). However, if four generators are used, the generators can be located in two machinery spaces instead of three. By using only one set of intake and exhaust ducts, volume for ductwork can also be reduced. Since most recent ships have had weight constraints placed on them by Congress, the minimum number of generators have been used.[7]

7 A very simple cost model for warships assigns a cost per ton of different components of a ship. With this in mind Congress has in the past placed constraints on the weight of ships in order to keep costs down.

Enclaving is a concept for arranging ships which involves locating all the equipment required for a given combat system within the same general area of the ship. If a ship is completely divided into a number of enclaves, one enclave can be damaged by enemy ordnance while the others remain functional and capable of continuing the engagement. To work properly, this concept requires the enclaving of sources of distributed services (such as electricity, cooling water, fire fighting water and dry air). Presently, enclaving has not been incorporated in any warship design but its use has been proposed for a number of new designs[8]. If enclaves are ever used, they will have a significant impact on the type, size, number, and location of electric generators. In some enclaves it may not even be possible to locate a conventional generator. Alternate generating or storage devices such as fuel cells or batteries may be used.

2.4 Integrated Electric Drive

Most modern warships mechanically couple the main propulsion prime movers with the propeller shaft. The mechanical power train is very efficient but imposes constraints on machinery arrangement and adversely impacts survivability. The prime mover is usually very heavy and must be located near the center of the ship to prevent excess trim. Shafting must therefore penetrate a number of watertight boundaries and maintain precise alignment over a great distance. The long length and precision requirements of the shafting make it very vulnerable to weapon induced damage. While electric propulsion eliminates many of the survivability and arrangement constraints of the mechanical system, the propulsion system must be carefully designed to ensure overall plant efficiency is not degraded by the extra power conversion losses in converting to and from electric power. Designed properly, an electric drive system can achieve the survivability and arrangeability benefits without suffering from a lower propulsion plant efficiency.

Integrated electric drive interconnects the generation of power for propulsion with the generation of ship service electric power. The propulsion plant for U.S. warships typically averages between 30 and 37.5 MW per shaft. The capacity is sized to provide enough power to propel the ship at a desired maximum speed. Most ships however, do not operate for extended periods of time at maximum speed. Operating at half maximum speed requires only about 20 percent of the installed power and quarter maximum speed requires only 2 or

8 Enclaving requires a greater redundancy of equipment which results in the ship becoming larger and more expensive. Since most ship designs are cost constrained, enclaving provisions are often deleted to reduce the per unit price of the warships.

3 percent. Thus a 28 knot frigate with a 30 MW plant could go 7 knots using less than 1 MW of power and 14 knots with about 6 MW of power. If the propulsion plant consists of two 15 MW generators, one generator could easily supply all the required power for both propulsion and ship service at the normal operating speeds of 12 to 15 knots. This has the potential of reducing the fuel consumption of warships under normal operating conditions by improving the overall efficiency of the power plant even though the efficiency of the power transmission system is lower. By careful selection of generator number and size, one can tune the overall efficiency of a plant for optimization at several different speeds. In the U.S. Navy, optimizing plant efficiency for 20 knots is beneficial since this is the speed used to calculate the amount of fuel carried by the ship.[9]

In a typical integrated electric drive scheme, the propulsion prime movers are connected to both a propulsion generator and to a ship service generator (PDSS or Propulsion Derived Ship Service). The speed of the generator is set to optimize efficiency of the prime mover at the given power loading. Consequently, cycloconverters are used to convert the power to either 60 Hz for ship service, or to whatever frequency the propulsion motors require. Usually, an additional diesel or gas turbine ship service generator is included to provide power in port or during emergencies. Figure 2.4-1 shows a typical PDSS design for a two shaft frigate sized ship.

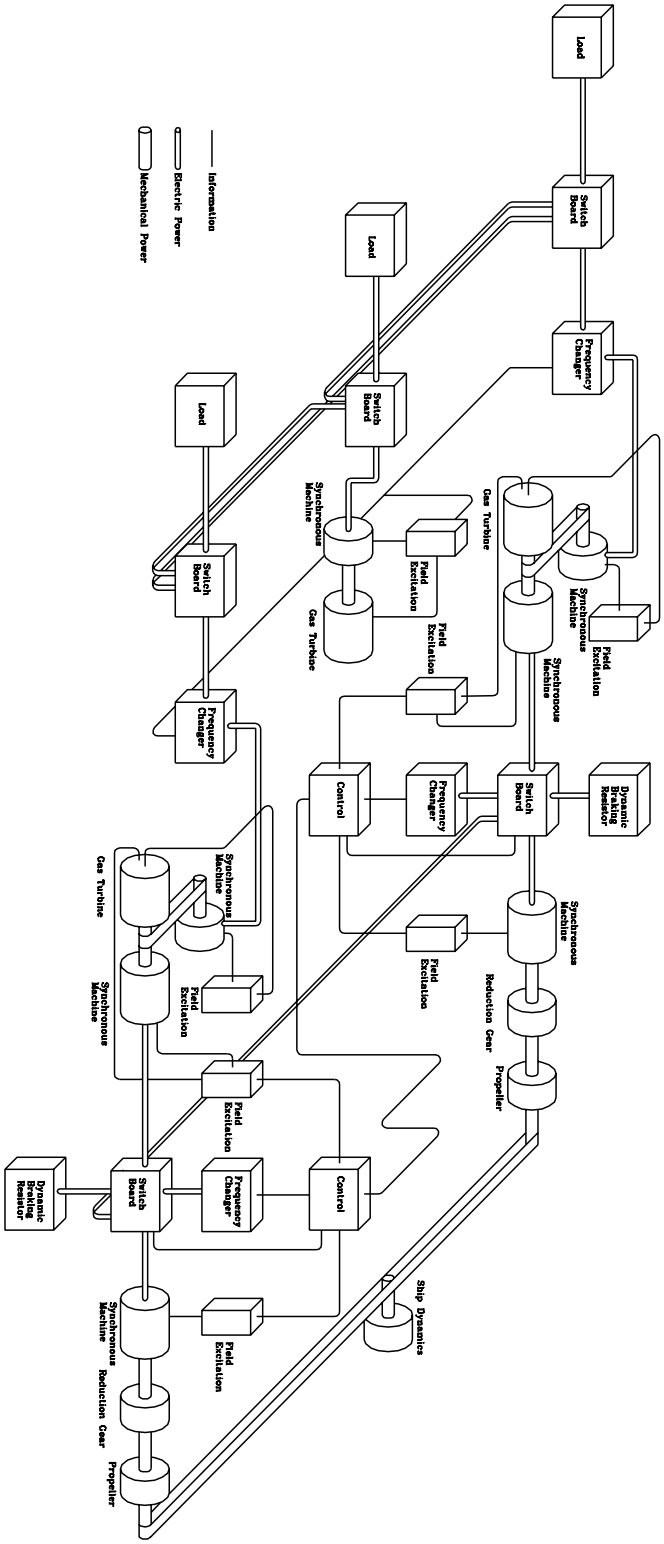
Figure 2.4-1 emphasizes the need to model mechanical dynamics and control information signals. The control signals can couple the dynamics of different devices within the system and must therefore be carefully modelled. The control signals can also destroy such properties as diagonal dominance which makes analysis of commercial power systems much easier.

One of the features of an electric drive system which may be exploited in the future is the ability to divert all of the propulsion power capacity from propulsion to some sort of high power combat system. Weapons such as rail guns and high energy lasers may become possible. These types of weapons would be safer for the ship since the requirement to store large amounts of chemical explosives for propellent charges would be reduced. Energy to move projectiles would be stored in the form of relatively inert fuel oil instead of highly

---

9 Most other navies use 18 knots which allows for combined plants such as CODOG where a diesel engine is used for cruising and a gas turbine for high speed. Unfortunately, the size requirement for a diesel capable of propelling a ship at 20 knots is prohibitive and results in U.S. warships only using gas turbines and carrying much more fuel.

Figure 2.4-1 Integrated Electric Drive



explosive chemical propellents. Switching large amounts of electric power onboard ships presents a number of technical challenges both in the design of physical equipment and also in attempts to accurately simulate the phenomena. The effect of pulse loads on the electric system is not a trivial simulation problem.

Chapter 3 Framework

Conducting time domain simulations of systems of nonlinear lumped parameter models characterizing shipboard electric power systems requires an organized approach to developing device models as well as network equations. The major contribution of this thesis is the development of a simulation environment having the following properties:

1. An object oriented approach to developing the mathematical description of devices independent of the manner in which the variables are represented.
2. An organized method for generating system equations for interconnecting device models into subsystems and systems.
3. An algorithm for solving the system equations and variables by identifying smaller blocks of equations and variables which can be sequentially solved. The algorithm develops the concept of the device structural jacobian matrix and the system structural jacobian matrix.
4. The ability to use a wide range of methods to describe variable waveforms. In particular, describing waveforms through vectors of coefficients of polynomial series, orthogonal function series, and data series are stressed.
5. The ability to solve the system of equations by employing either the Newton-Raphson Method or Waveform Relaxation. The Newton-Raphson method is modified to improve convergence properties through the use of continuation methods.

This chapter is organized into five parts. The first part defines the device which is the fundamental building block of the system simulation. The second part shows how to interconnect several device models into systems and subsystems. The third part defines the waveform as a vector of coefficients to approximate waveforms over time intervals. The fourth part details the actual procedure for conducting a simulation. The fifth and final part details some finer points which should be considered when constructing models.

3.1 Device Description

A **Device Description** is an organized manner for describing the characteristics of a physical component. This description includes definitions of variables which interface with other components in a system, variables called states which allow for information storage, and constitutive relations describing the device behavior.

3.1.1 Interface Variables

The **interface variables** are defined as either **potential variables** or **flow variables** depending on their interaction with the interface variables of other devices within a system or subsystem. **Systems** and **subsystems** are constructed by grouping the interface variables of one or more devices into sets called **nodes** and applying network equations determined by the types of variables attached to the nodes.

All **potential variables** attached to a node are equated to a potential value associated with the node. Physical quantities which can be classified as potentials include voltages, signal levels, rotational speeds, deflections, and pressures. All potentials are referenced to **0**. All potential variables connected to the same node must be defined with respect to the same system wide reference level. In other words, **0** must mean the same thing for all of the potentials attached to a given node.

The sum of all **flow variables** attached to a node is equated to zero. Physical quantities analogous to flow variables include currents, power flows, torques, forces and mass flow rates.

3.1.2 Terminals

Terminals provide a mechanism for organizing the interface variables of a device. In general, there are two types of terminals: **Normal Terminals** and **Information Terminals**.

A **normal terminal** has associated with it a flow variable and a potential variable. Its electrical analog is one of the wiring terminals on an electrical device. A mechanical analog is the rotating shaft coupling of a gearbox. The equations for exchanging energy between devices can be generated through the list of normal terminals connected together at a given node.

An **information terminal** has associated with it only a potential variable. The potential variable is used to convey knowledge between devices without transferring energy. Set points, meter readings, and control signals are all examples of energyless data which can be conveyed through information terminals.

All normal terminals have an associated **KCL Group number**. A KCL group is the smallest subset of a device's terminals such that the sum of the flow variables within the subset is identically zero for at least one of the possible dynamic configurations of the device. Normal terminals which can not be associated with a KCL group are given a group number of **0**. The remaining terminals are assigned the group number of their parent KCL group.

The KCL Group number is used to detect possible reference frame problems within a simulation network. A given electrical circuit problem for example, must have at least one normal terminal with a **0** group node within a given independent system to ensure the set of system KCL equations is not singular. Normally this terminal is associated with a one terminal device with an export potential and import flow which is used to specify the value of a given reference node potential. This Reference Frame Check is discussed in greater detail in section 3.2.4.

Some devices may have variable numbers of KCL Groups depending on the operating point of the device. A simple model of a two terminal switch for example, would have 1 KCL group when the switch is closed (the sum of the currents entering the switch is identically zero) and 2 KCL groups when the switch is open (both flow variables are identically zero). For the purpose of defining the device, the *worst* case in terms of creating singular systems should be used. In the switch example, each terminal should have their own KCL group number for a total of two KCL groups.

3.1.3 Variable Direction: Import and Export Variables

The Interface variables can further be classified by whether they are a resource (**Import**) or product (**Export**) of the device description. A device description can be considered a means for generating export variables based on the values of the import variables, states, parameters, continuation parameter, and time.

An **import variable** is taken as input by the device description. An import variable can be any interface variable associated with either normal or information terminals. To

ensure a consistent set of equations when several devices are connected together in a system, the total number of import variables associated with normal terminals must equal the number of normal terminals

An **export variable** is explicitly defined and considered a product of the device description. An export variable can be any interface variable associated with either normal or information terminals. To ensure a consistent set of equations when several devices are connected together in a system, the total number of export variables associated with normal terminals must equal the number of normal terminals.

3.1.4 States

States are variables whose values are stored for a given time for later use. States can be used for example, to store the constant of integration for a dynamic equation. States can also be used to store the operating mode for a given device. In general, if the value of a given variable depends on the previous value of another variable, that other variable is a state.

3.1.5 Parameters

Parameters are constants which specify characteristics of the device or in other words, customizes a given device description to represent a given physical device. A model of a resistor for example, includes a parameter for resistance. This precludes the requirement to develop a model for every resistor value. We only need construct a generic resistor model instead of a 10K resistor model, a 22K resistor model, etc.

3.1.6 Constitutive Equations

The **constitutive equations** are a consistent set of equations for specifying the values of the states and export variables. In general, the number of constitutive equations needed is equal to the number of normal terminals plus the number of export variables associated with information terminals. The total number of import variables associated with normal terminals and the total number of export variables associated with normal terminals must independently equal the number of normal terminals. There is no constraint on the number of import variables associated with information terminals.

3.1.7 Device Jacobian Matrices

A **Device Jacobian Matrix** provides the sensitivities (partial derivatives) of the export variables with respect to the import variables. This implies there is a given ordering of both the import $\boldsymbol{X_I}$ and export $\boldsymbol{X_E}$ variables:

$$X_I = \begin{bmatrix} X_{I1} \\ X_{I2} \\ X_{I3} \\ . \\ . \\ . \\ X_{In_t} \end{bmatrix}$$

$$X_E = \begin{bmatrix} X_{E1} \\ X_{E2} \\ X_{E3} \\ . \\ . \\ . \\ X_{En_t} \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial X_{E1}}{\partial X_{I1}} & \frac{\partial X_{E1}}{\partial X_{I2}} & \frac{\partial X_{E1}}{\partial X_{I3}} & \cdots & \frac{\partial X_{E1}}{\partial X_{In_t}} \\ \frac{\partial X_{E2}}{\partial X_{I1}} & \frac{\partial X_{E2}}{\partial X_{I2}} & \frac{\partial X_{E2}}{\partial X_{I3}} & \cdots & \frac{\partial X_{E2}}{\partial X_{In_t}} \\ \frac{\partial X_{E3}}{\partial X_{I1}} & \frac{\partial X_{E3}}{\partial X_{I2}} & \frac{\partial X_{E3}}{\partial X_{I3}} & \cdots & \frac{\partial X_{E3}}{\partial X_{In_t}} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ \frac{\partial X_{En_t}}{\partial X_{I1}} & \frac{\partial X_{En_t}}{\partial X_{I2}} & \frac{\partial X_{En_t}}{\partial X_{I3}} & \cdots & \frac{\partial X_{En_t}}{\partial X_{In_t}} \end{bmatrix}$$

The Device Jacobian Matrix is used to generate a consistent set of import variables which simultaneously satisfy the device constitutive equations along with constraints imposed by the connections of terminals to nodes. From the device point of view however, the Jacobian matrix is merely a product that must be computed.

Up to this point, we have not discussed the manner in which the variables are described. If the variables are represented by real numbers, then each element of the Jacobian is also a real number. If instead the variables are represented by vectors, then the Jacobian elements will be matrices.

3.1.8 Device Structural Jacobian Matrix

The **Device Structural Jacobian Matrix** describes the properties of the elements of the device Jacobian matrix for a given type of variable representation without actually providing any values. The following codes can be used to describe the properties of the matrix elements of the device Jacobian matrix:

Code	Type of Matrix
0	Zero Matrix (all elements are always zero)
I	Identity Matrix (always the identity matrix)
D	Diagonal Matrix (always a linear main diagonal matrix)
L	Linear Matrix (The elements are always constant)
A	Nonlinear AC Matrix (see Note 3.1.8-1)
N	Nonlinear Matrix (The elements may not be constants)
U	Unknown (The dependence is unknown (treat as nonlinear))

Note 3.1.8-1: An AC Matrix is one for which the constant component of the export variable depends only on the constant component of the import variable. The other components of the export variable can not depend on the constant component of the import variable but are not restricted in any other way.

The device structural Jacobian matrix is useful in developing the algorithm for generating a consistent set of import variables without having to deal directly with the potentially much larger device Jacobian matrices. If an iterative solution scheme is used to

develop the consistent set of import variables, the device structural Jacobian matrix indicates directly which matrix elements must be recalculated for each iteration. (Only the nonlinear and unknown elements have values which change between iterations)

3.1.9 Continuation Parameter

A system containing one or more nonlinear devices may be difficult to solve with an iterative method. The region of convergence around the solution may be so small as to make the probability of success for choosing a starting point for the iterative scheme almost zero. One method for enlarging the region of convergence is through the use of a **continuation parameter** which varies from **0** to **1**. When the continuation parameter has value **1**, the export variables are developed using the normal nonlinear constitutive equations. When the continuation parameter has value **0** however, the export variables are developed using a linear set of constitutive equations. As the continuation parameter increases from **0** to **1**, the export variables traverse a continuous path from the linear solution to the nonlinear solution. One common method for generating such a dependence on a continuation parameter α is:

$$F(X,\alpha) = \alpha F_n(X) + (1-\alpha)F_l(X)$$

where $\boldsymbol{F_n(X)}$ is the nonlinear function for generating the export variables, $\boldsymbol{F_l(X)}$ is the linear function approximation, and $F(X,\alpha)$ is the function for determining the export variables for intermediate values of α. Section 3.4.2 describes in detail continuation parameters in relation to the Newton-Raphson method.

3.1.10 Discontinuity Time Prediction

If the variables are described as a waveform over a given time interval $\boldsymbol{[t_0,t_1]}$ knowledge of the time of discontinuities can prove useful to the algorithm which generates the consistent set of import variables. The accuracy of a vector description of a waveform often deteriorates greatly if there is a discontinuity during the time interval. Varying $\boldsymbol{t_1}$ such that it falls on a discontinuity will often improve the accuracy of the waveform representation. For this reason, each device has the opportunity to recommend a recalculation time for the current interval. Normally, the system would use the minimum recommended recalculation time offered by any of the devices to recompute the time interval.

3.2 Network Description

A **network** is composed of a system of devices and **subsystems** whose terminals are interconnected at **nodes**. The **network** is a closed system having no terminals defined for any of its nodes. A **subsystem** is a system having terminals defined for at least one of its nodes and therefore can not be solved independently of other devices or subsystems.

3.2.1 Nodes

A **node** connects together one or more terminals from one or more devices. The nodal connections are the means by which devices are combined to form systems (both networks and subsystems). The nodes provide the association of device import and export variables with system variables through **nodal equations**. Each node is assigned a **serial number** for identifying it from the other nodes. There are two types of Nodes: **Normal Nodes** and **Information Nodes**.

3.2.1.1 Normal Nodes

A **Normal Node** has at least one normal terminal attached to it. Information terminals can be associated with the node as long as none of the information terminal potentials are defined as an export variable. A normal node has associated with it a node potential as well as a Kirchhoff Current Law (KCL) equation. The number of normal nodes is designated by $\boldsymbol{n_n}$.

In a subsystem, a normal node can also have associated with it a terminal for connecting with other subsystems and devices. This terminal can be either a normal terminal having an associated terminal potential and flow variable or an information terminal having only an export potential. (*import* and *export* refer here to the direction relative to the defining subsystem which is opposite to the normal definition which is relative to the components of the subsystem). The total number of normal node normal terminals defined for a subsystem is designated $\boldsymbol{n_{ntn}}$. For any given subystem the number of normal node terminal export variables and import variables must both independently equal $\boldsymbol{n_{ntn}}$. The total number of normal node information terminals is designated $\boldsymbol{n_{nti}}$.

3.2.1.2 Information Nodes

An **Information Node** has only information terminals attached to it. Furthermore, one and only one of the terminal potential variables must be an export variable. Only a node potential is associated with an information node. Information nodes work in the

same manner as hooking up stereo componenents: you can hook up as many inputs (import variables) as you want to any given output (export variable), but should never hook up two or more outputs together. The number of information nodes is designated by $\boldsymbol{n_i}$.

As an option for subsystems, an information node can have associated with it an information terminal for connecting with other subsystems. Since the meanings of *import* and *export* are once again reversed for this terminal, no other export potentials from other devices or subsystems may be attached to the node if the information terminal potential is an import variable. If the information terminal potential is an export variable, exactly one other export potential from other devices or subsystems may be attached to the node. The total number of information node information terminals is designated $\boldsymbol{n_{iti}}$.

3.2.2 System Variables

System variables comprise the minimum set of variables from which all of the device import and export variables can be derived from. The set of system variables is composed of node potentials as well as all device import flow variables and normal node normal terminal export flow variables. For a subsystem, the node terminal import variables are assumed to be provided by the encompassing system or subsystem and are not considered system variables.

3.2.2.1 Node Potentials

All of the node potentials of the normal and information nodes are system variables which must be solved for. Hence there are a total of $\boldsymbol{n_p = n_n + n_i}$ node potentials.

3.2.2.2 System Flow Variables

All of the Import Flow Variables of the various devices making up the system as well as the export flow variables of the normal node terminals are system variables. The number of system flow variables is designated by $\boldsymbol{n_f}$.

3.2.3 System Equations

3.2.3.1 Kirchhoff Current Law Equations

Kirchhoff's current law states the sum of the flow variables entering a node is equal to zero. For a given normal node or normal terminal node, this law is expressed by

generating a list of the terminals of the various devices and subsystems attached to the node. The number of Kirchhoff Current Law equations is equal to the number of normal nodes n_n.

$$f_j() = \sum_{i=1}^{n_t} I_{ji} = 0$$

where

$f_j()$	KCL Equation for node ***j*** (Should Equal Zero)
n_t	Number of normal terminals attached to node
I_{ji}	Flow Variable associated with ***i***th normal terminal attached to node ***j***

3.2.3.2 Potential Difference Equations

A **Potential Difference Equation** is created for each of the export potential variables of the various devices and other subsystems and for each of the import potential variables of the node terminals. This equation merely states the difference between the node potential and the potential variable is zero. This equation is expressed by generating a list of the terminals of the various devices and subsystems attached to the node having an export potential variable. Since one and only one export information potential can be assigned to an information node and can never be attached to a normal node, the number of potential equations due to export information potentials is simply n_i. The requirement for a device to have equal number of import and export variables associated with normal terminals forces the number of export normal potentials to be n_f. Hence the total number of potential equations is $n_v = n_i + n_f$.

$$f_{ji}() = V_j - V_{ji} = 0$$

where

$f_{ji}()$	Potential Difference Equation for node ***j*** export potential variable ***i*** (Should Equal Zero)
V_j	Node ***j*** Potential
V_{ji}	***i***th export potential variable associated with node ***j***.

3.2.3.3 R_{min} and G_{min}

One method for preventing linear dependences among the system equations is to modify the equations to include an extra term corresponding to either a small conductance G_{min} to the ground potential for KCL equations or a small series resistance R_{min} for the potential difference equations. The KCL equation is now given by:

$$f_j() = G_{\min} V_j + \sum_{i=1}^{n_t} I_{ji} = 0$$

The potential difference equation is similarly modified:

$$f_{ji}() = V_j - V_{ji} - R_{min} I_{ji} = 0$$

The goal in using G_{min} and R_{min} is to reduce the condition number of the **system Jacobian matrix** to the point where the system can reliably be solved (A singular matrix has an infinite condition number). G_{min} and R_{min} can also add fictitious dynamics to the system and thereby lead the simulation to produce incorrect results. Hence if used, G_{min} and R_{min} should be large enough to bring the condition number down to a reasonable level, but small enough to prevent their inclusion from having appreciable effect on the simulation results.

In general, the use of G_{min} and R_{min} should be avoided for these reasons:

1. G_{min} and R_{min} are fictitious elements. If either is significant, they should be explicitly included as a device.

2. The indiscriminite use of G_{min} and R_{min} adds to the complexity of the system and decreases the degree to which the system can be reduced into smaller blocks. In other words the inclusion of G_{min} and R_{min} may greatly increase the computation time.

G_{min} and R_{min} are included in WAVESIM for these reasons

1. G_{min} and R_{min} can be selectively specified for individual nodes. If a simulation fails to converge for one reason or another, G_{min} and R_{min} can be employed to find the part of the system experiencing difficulties. G_{min} and R_{min} are excellent debugging tools.

2. Since $\boldsymbol{G_{min}}$ effectively connects the node to the ground potential, $\boldsymbol{G_{min}}$ can be used to ensure all of the nodes have the same potential reference and ensure there are no linear dependent KCL equations.

3.2.4 Reference Frame Testing

If a given set of a system's normal nodes can be found such that all terminals attached to any of its nodes have nonzero KCL groups and such that if a terminal is attached to one of the set's nodes, then all of remaining terminals of the parent KCL group are also attached to one of the nodes of the set, then there exists the possibility of a singular system due to the linear dependence of the KCL equations for the set of normal nodes.

If $\boldsymbol{G_{min}}$ is non-zero for a node, it should be considered a terminal with a **0** KCL Group. If $\boldsymbol{G_{min}}$ is zero, it should be ignored.

Testing for a possible singular system can be accomplished with the following algorithm:

1. Set all the normal node **`circuit_group_indicators`** to **0**.
   Set the **`circuit_group_counter`** to **0**
   Set the **`circuit_group_singular_flag`** to **0**

2. Start with the first normal node having a **0** **`circuit_group_indicator`**
   If none can be found then algorithm is complete.
   Increment **`circuit_group_counter`**.

3. Change the **`circuit_group_indicator`** of the node to the
   `circuit_group_counter`.

4. For each terminal attached to the node:
   4a. If the KCL group number is zero, set the
   `circuit_group_singular_flag` to **1.**
   4b. If the KCL group number is nonzero, loop through each normal
   terminal of the device. If the terminal belongs to the same
   KCL group and the node the terminal is attached to has a
   0 **`circuit_group_indicator`**, then set the node
   `circuit_group_indicator` to the negative
   of the **`circuit_group_counter`**.

5. Search all of the nodes for a negative **`circuit_group_indicator`**
   If none can be found and the **`circuit_group_singular_flag`** is zero
   Warn user that a singular system may exist with the group nodes.
   If none can be found then go to step 2
   If one is found, then go to step 3

Setting a proper reference for each such set of system nodes can be accomplished by attaching to one of the nodes a one terminal device having the following characteristics:

3.2.4.1 Reference Device

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Grp) Type
Ref	$\boldsymbol{V}$ (export)	$\boldsymbol{I}$ (import)	(0) Normal

Parameters

$\boldsymbol{V_{Ref}}$ Reference Potential Level

Equations

$$\boldsymbol{V = V_{Ref}}$$

Device Structural Jacobian

$$J_{DS} = [0]$$

Device Jacobian

$$J_{D} = [0]$$

Notes

Most conventional circuit simulations define a reference node for which a potential is defined and the KCL equation is not written. Adding this reference device to a node effectively converts that node to a reference node in the usual senses. While it is true that the KCL equation and an additional Potential Difference equation are still written for this reference node, each is part of a one element block. The potential difference equation can be solved before the simulation starts since it does not depend on any of the system variables. The flow variable on the other hand, only appears in the KCL equation of the one node and thus can be solved after all the other system variables have been found. In fact, the flow variable should normally equal zero if the rest of the circuit is indeed linearly dependent.

As a convenience to the user, WAVESIM automatically attaches a reference device with $\boldsymbol{V_{ref} = 0}$ to the node with serial number **0** if that node is used.

3.2.5 System Reduction

The previous sections detail a method for generating a full set of system variables and system equations. The total number of system variable equals $n_{sf} = n_n + n_i + n_f$ which also equals the number of system equations. For even a small system the algebraic order n_{sf} can become quite large. For this reason, elminating system variables and equations through system reduction is desirable. The primary tool for performing system reduction is the **system structural Jacobian**.

3.2.5.1 System Structural Jacobian

The **System Structural Jacobian** facilitates the reduction of the algebraic order of the system by showing the nature of the dependence of system equations to each of the system variables. The System Structural Jacobian is constructed by combining elements of the device structural Jacobian matrices according to the arithmetic of structural Jacobian elements. The types of elements in the system structural Jacobian is given by:

Code	Type of Matrix
0	Zero Matrix (all elements are always zero)
I	Identity Matrix (always the identity matrix)
D	Diagonal Matrix (always a linear main diagonal matrix)
L	Linear Matrix (The elements are always constant)
A	Nonlinear AC Matrix (see Note 3.2.5.1-1)
N	Nonlinear Matrix (The elements may not be constants)
U	Unknown (The dependence is unknown (treat as nonlinear))

Note 3.2.5.1-1: An AC Matrix is one for which the constant component of the export variable depends only on the constant component of the import variable. The other components of the export variable can not depend on the constant component of the import variable but are not restricted in any other way.

The addition and subtraction operators for the structural Jacobian elements is a function of the manner in which the system variables are represented. For all of the methods used in this thesis, the following definitions apply:

$$\boldsymbol{I + I = D}$$

$$\boldsymbol{I - I = 0}$$

$$\boldsymbol{I \pm 0 = I}$$

$$\boldsymbol{-I \pm 0 = D}$$

$$\boldsymbol{-I - I = D}$$

$$\boldsymbol{\pm n \pm m = \pm m \pm n = n \ (n \geq m,\ n \neq I)}$$

$$\boldsymbol{U > N > A > L > D > I > 0}$$

Note, the Identity Code **I**, is not strictly necessary and if eliminated simplifies the addition and subtraction operators to:

$$\boldsymbol{\pm n \pm m = \pm m \pm n = n \ (n \geq m)}$$

Before the system structural jacobain can be constructed, the system variables and equations must be ordered. The first $\boldsymbol{n_p}$ variables are the node potentials of the normal and information nodes arranged in the order of the node serial numbers. The next $\boldsymbol{n_f}$ variables are the import flow variables ordered first by device then by device terminal. The first $\boldsymbol{n_n}$ equations conform to the Kirchhoff Current Law equations for the normal nodes arranged in order of the node serial numbers. The remaining $\boldsymbol{n_v}$ equations are the potential equations for the export potentials ordered first by the node serial number they are attached to, then by the order of the devices attached to the node, and finally by the order of the terminals in the device.

The system structural Jacobian is constructed in two parts after being initialized to contain only **0**. First, a Kirchhoff Current Law equation is generated for each normal node. The normal terminals of the normal nodes are examined one at a time. If the flow variable is an import variable, it is also a system variable and an ***I*** is added to the corresponding element of the system Jacobian matrix. If the flow variable is an export variable, its corresponding row of the device structural Jacobian matrix is extracted. The columns of the device structural matrix row correspond to the device import variables. All of the device import variables can be associated to either a node potential (one of the first $\boldsymbol{n_p}$ columns of the system structural Jacobian) or to one of the remaining $\boldsymbol{n_f}$ import flow variable columns. Hence it is quite easy to locate to which column each element of the device structural Jacobian row must be added. If $\boldsymbol{G_{min}}$ is non-zero for the node, a ***D*** code is added to the column corresponding to the node potential.

The remaining $\boldsymbol{n_v}$ rows of the system structural Jacobian matrix are constructed by examining each node one at a time. If the node has an export potential associated with it. An $\boldsymbol{I}$ is added to the corresponding node potential column and potential equation row element (unless of course the node is a reference node and does not have a column associated with its potential). The row of the device structural Jacobian matrix corresponding to the export potential is then extracted. In exactly the same manner as described above for the export flow variables, the columns of the system structural Jacobian matrix are correlated to the columns of the device structural Jacobian matrix. Once correlated, the elements of the device structural Jacobian row are subtracted from the appropriate elements of the system structural Jacobian matrix. If $\boldsymbol{R_{min}}$ is non-zero for the node and the terminal having the export potential has an import flow variable, then a $\boldsymbol{D}$ is added to the column corresponding to the import flow flow variable. If $\boldsymbol{R_{min}}$ is non-zero for the node and the terminal having the export potential has an export flow variable, then a $\boldsymbol{D}$ is multiplied by the elements of the corresponding row of the device structural Jacobian matrix before being added to the corresponding column in the system structural Jacobian matrix.

Once the structural Jacobian matrix has been constructed it can be examined to ensure there are no glaring problems such as a row or column containing only **0** elements. If a row or column contains only **0** elements, the system is ill-posed and can not be solved.

3.2.5.2 Blocks

The primary reason for constructing the system structural Jacobian matrix is to break down the system of equations and system variables into smaller **blocks** which can be sequentially solved instead of solving the entire system at once. A **block** $\boldsymbol{B_i}$ is defined as $\boldsymbol{n_{bi}}$ system variables and n_{bi} equations which only depend on system variables of the present block and previous blocks in the sequence. A block of size $\boldsymbol{n_{bi}}$ is identified by finding $\boldsymbol{n_{bi}}$ rows in the system structural Jacobian matrix that have not already been allocated to a block and have exactly $\boldsymbol{n_{bi}}$ columns containing non-**0** elements. Of the many combinations of blocks which can be found for a system, the best combination contains the largest number of small blocks. Here is an algorithm for finding the blocks:

1. Create a list for each row containing the number of unallocated non-**0** entries in that row. (Initially all the rows and columns are unallocated)

2. Examine the list for rows having only 1 unallocated non-**0** entries. Create a block for each of these rows and their associated columns. Mark the rows and columns as allocated.

3. Update the list of unallocated non-**0** entries in each row.

4. Continue steps 2 and 3 until no more single rows can be allocated.

5. Examine the list for two rows only having unallocated non-**0** entries in the same two columns. Create a block for each pair of rows and their associated columns. Mark the rows and columns as allocated.

6. Update the list of unallocated non-**0** entries in each row.

7. Repeat steps 2-6 until no more single row and double row blocks can be identified.

8. Examine the list for three rows only having unallocated non-**0** entries in the same three columns. Create a block for each set of three rows and their associated columns. Mark the rows and columns as allocated.

9. Update the list of unallocated non-**0** entries in each row.

10. Repeat steps 2-9 until no more blocks of up to size 3 can be identified.

11. Continue the above algorithm until all of the rows and columns have been allocated. Remember it is necessary to go back and attempt to identify smaller sized blocks after discovering a larger block since the removal of a column could allow the identification of a new smaller block.

The order of identifying blocks is very important because they must be solved in the same order. Each block contains the same number of system variables and system equations. The equations only depend on system variables determined from the present and previous blocks. Hence the simulation problem becomes an issue of solving sequences of relatively small systems of equations described by blocks.

3.2.6 Reduced System

The reduced system consists of the sequence of blocks which when solved, provide the solution for all the system variables. Solving each of the blocks can be done a number of ways. Most schemes start with an initial guess for the system variables and generate corrections to the guesses until all of the system equations for that block are satisfied.

Generating the corrections is normally done through the use of a block Jacobian matrix which can be constructed in much the same manner as the system structural Jacobian. If the block structural Jacobian does not contain any ***A, N*** or ***U*** elements, the block Jacobian can be inverted and multiplied by the system equation errors to provide the required corrections. If there are any nonlinearities, this scheme can be performed several times until the system equation errors are close to zero. This method is commonly referred to as the Newton-Raphson method and if the initial guess is close enough to the solution, the method converges quadratically. This method is described in much more detail in section 3.4.1.

Relaxation techniques can also be used to calculate the system variables. Relaxation techniques start with an initial guess for all of the system variables and update each variable one at a time by solving a single system equation by assuming all of the other variables are constant. Typically, one system equation is assigned the task of solving for a particular system variable. With careful thought as to the assignment of variables to equations, it is often possible for such a system to converge to a solution. Common relaxation techniques are the Gauss-Seidel and Gauss-Jacobi methods.

3.3 Waveforms

Up to this point, the development of the simulation structure has been independent of the manner in which variables are actually described. The simplest and most commonly used method for representing variables is through a single real number representing the value of a variable at a specific time. For static simulations where the problem is to obtain the steady state solution for the system, this method works very well. Appendix C and Appendix D demonstrate this procedure for the classic load flow problems. For dynamic simulations however, some knowledge as to the time history of the variables is needed to calculate derivatives and integrals. A dynamic simulation is implemented as a series of static simulations where the dynamics are represented by functions of the time increment and state variables. The various integration techniques for this type of simulation differ only in the interpolation scheme used to approximate the variables between successive static simulations. The time increment between static simulations must be carefully controlled to ensure the interpolation scheme has enough accuracy for numerical stability. Integration in this manner requires careful control of the time increment to ensure the interpolation scheme is accurate enough to ensure numerical stability along with an accurate solution.

Another approach to representing variables is the **waveform**. This method employs a vector of coefficients to continuously describe the time domain value of the variable over some time interval $[t_0\ t_1]$. The **type** of the waveform determines how the coefficients are interpreted to generate the time domain values. Possible types include Data Series, Fourier Series, Legendre Series, Polynomial Series and Legendre Series. The principal advantages of using waveforms over discrete points include:

1. Interpolation is not generally required to determine intermediate points. The value of a variable can readably be determined for any time.

2. The numerical stability of Integration and Differentiation techniques do not have to depend on the time step control since integration and differentiation become waveform operators on an equal level to all other operators. Time step control becomes only an issue of numerical accuracy and not of numerical stability.

3. Certain operations may be easier to perform with one waveform type. The ability to efficiently convert a waveform from one type to another type and back again allows one to use the most efficient waveform type in the calculations of a given operator.

3.3.1 Waveform Definition

A waveform approximates the instantaneous value of a variable over some time interval. The elements of information contained within a waveform must as a minimum include:

1. The name of the waveform
2. The beginning and ending times of the interval (t_0 t_1)
3. An Array of Coefficients representing the waveform (c_i)
4. The number of coefficients in the Coefficient Array (n)
5. A waveform type indicator.

The **waveform type indicator** identifies how the coefficients should be interpreted when operations are performed on the waveform. Here is an example of a C structure defining a Waveform:

```
typedef struct Waveform
 {
   char   *name;  /* character string of the name
                     of the variable */
   double    t0;  /* time of the beginning of the interval */
   double    t1;  /* time of the end of the interval */
   void      *c;  /* array of coefficients */
   long       n;  /* number of elements in the array */
   long    type;  /* waveform type indicator */
   long version;  /* Version of this waveform */

   struct Waveform *next;  /* pointer for forward
                              linked lists */
   struct Waveform *last;  /* pointer for backwards
                              linked lists */
   struct Jacobian *jnum;  /* pointer to linked list of
                              jacobians where this waveform
                              is the numerator */
   struct Jacobian *jden;  /* pointer to linked list of
                              jacobians where this waveform
                              is the denominator */
 }
WAVEFORM;
```

The above definition also includes the following optional information:

6. A Version Number to record a change in the waveform's properties.
7. An Address Pointer to the waveform representing the previous time interval.
8. An Address Pointer to the waveform representing the following time interval.
9. An Address Pointer to a linked list of Jacobian Structures.

The waveform address pointers allow one to construct a **linked list** of waveforms to describe the time history of a variable over a number of time intervals. The **Jacobian** structure as well as the **version number** will be described in section 3.3.3.

Note the waveform coefficients are declared to be of type **void**. This is done to allow for the coefficients to be abstract data representations in themselves. Normally the waveform coefficients would be double precision floating point numbers, but it should also be possible to incorporate other types of data. It may be advantageous for example, to represent the coefficients with complex numbers. In this case, each element in the coefficient array would be a structure holding double precision floating point numbers corresponding to the real and imaginary parts (Or magnitude and phase angle) of the complex number.

3.3.2 Waveform Operators

Waveform Operators are functions which act on waveform arguments to generate new waveforms, or provide some information about the waveform arguments. The types of functions can be broken down into several groups:

1. Arithmetic Operators
2. Trigonometric/Exponential Operators
3. Switching Operators
4. Integral/Differential Operators
5. Waveform Content
6. Special Functions

3.3.2.1 Arithmetic Operators

The arithmetic operators are the customary addition, subtraction, multiplication, division, and assignment operators usually associated with floating point arithmetic. The assignment operator is a bit more complex since it must incorporate waveform type and number of coefficient conversions.

3.3.2.2 Trigonometric/Exponential Operators

The Trigonometric/Exponential operators include most of the transcendental functions used in engineering. Examples include sine, cosine, tangent, logarithms, exponentials, as well as the inverse functions. Error handling can become quite complex since several of these operators may be undefined at one or more points within the argument waveform. These operators are usually handled by converting the arguments to a series of data points, performing the operation point by point, and then converting back to the appropriate waveform type.

3.3.2.3 Switching Operators

Switching Operators are operators producing waveforms which themselves or one of their derivatives are discontinuous. Examples include the absolute value function, the sign function and the step function. The typical method for calculating these functions is

to determine the discontinuity points and use integration to create a characteristic function series solution (e.g. Legendre Series or Chebyshev Series) for the result. The series solution is then converted to the appropriate waveform type.

3.3.2.4 Integral/Differential Operators

One of the key advantages of using waveforms in dynamic simulations is that integration and differentiation become very simple operators where the stability of a numerical integration scheme is generally not an issue. For many waveform types, the integration operator is a linear matrix operation with bounded coefficients. If the argument waveform has bounded coefficients, the returned waveform will also be bounded. Of course, numerical stability does not assure numerical accuracy. Because the integration operator typically generates some truncation error, the returned waveform can still contain considerable errors.

3.3.2.5 Waveform Content

The significance of the Truncation Error of a waveform can be estimated by calculating the **waveform content** of its higher order term. The waveform content of a term is defined as the magnitude of a coefficient divided by the square root of the sum of the squares of all the coefficients. Normally, one expects the higher order terms of an orthogonal series representation to progressively have smaller and smaller waveform contents. Hence if the last few terms have values below a preset threshold, the truncation error can normally be assumed negligible.

Accurate truncation error estimation is still a difficult and currently unsolved research topic. The waveform content method is a practical method but should not be taken as the last word on the subject.

3.3.2.6 Special Operators

Several special operators unique to waveforms should also be developed. One very useful operator returns the time of zero crossing of the waveform. Another returns the value and time of every local minimum and maximum of a waveform.

The **smoothing operator** is one method for reducing the waveform content of higher order coefficients. A waveform is smoothed by returning the local average of the waveform over some prespecified time increment. Smoothing eliminates discontinuities

in a waveform and its derivatives. Since discontinuities tend to amplify the waveform content of the higher order terms, removing the discontinuities should reduce the higher order term waveform content.

3.3.3 Jacobians

A **Jacobian** matrix contains the partial derivatives of the coefficients of one waveform with respect to another waveform. Here is a sample C structure to define a Jacobian:

```
typedef struct Jacobian
 {
   struct Waveform *num;   /* address of waveform in the
                              numerator of the partial
                              derivatives */
   struct Waveform *den;   /* address of waveform in the
                              denominator of the partial
                              derivatives */
   long version;          /* Version number of the
                             jacobian matrix */
   long num_version;      /* Version nbr of
                             numerator Waveform */
   long den_version;      /* Version nbr of
                             denominator Waveform */
   void **j;              /* array of jacobian elements
                                  The first row index is for
                                  an array of pointers whose
                                  elements are arrays with
                                  the colum index */
   char sj;               /* Structural Jacobian Code */

   struct Jacobian *next;  /* address for linked list
                              of Jacobians */
 }
JACOBIAN;
```

Jacobians are used in the process of solving simultaneous systems of waveform equations through relaxation methods or through the Newton-Raphson Method. The purpose of **num_version** and **den_version** is to record which versions of the numerator and denominator waveforms the jacobian was calculated for. The element **version** is used when several jacobians are combined and it is necessary to determine whether the combined matrix must be recalculated.

In general, all operations defined for a waveform should also generate the jacobian of the results with respect to the arguments. Through the use of the chain rule, the jacobian matrix of the export variables of a device with respect to the device import variables can be determined.

The **structural jacobian code** indicates the dependence and structure of the jacobian matrix. Here is a list of the codes:

Code	Type of Matrix
0	Zero Matrix (all elements are always zero)
I	Identity Matrix (always the identity matrix)
D	Diagonal Matrix (always a linear main diagonal matrix)
L	Linear Matrix (The elements are always constant)
N	Nonlinear Matrix (The elements may not be constants)
U	Unknown (The dependence is unknown (treat as nonlinear))

The structural jacobian code along with the version numbers determines whether or not a jacobian matrix needs to be recalcuated. If the structural jacobian is of type **0**, **I**, **D**, or **L** then the jacobian need not be reconstructed if the there is a version mismatch between the waveform version and the jacobian version. If the structural jacobian of type **N** or **U**, and there is a mismatch between the version numbers of the jacobian and the waveforms, then the jacobian elements must be recalculated. After every recalculation, the version numbers are updated. In this manner, only jacobian matrices with changing coefficients are ever recalculated.

Technically, the structural jacobian codes depend on the waveform type used. In this thesis however, all of the waveform types produce the same structural jacobian codes.

3.3.3.1 Jacobian Operators

Several operators for jacobian objects will prove useful in developing a simulation environment. These operators include:

1. Addition and Subtraction
2. Identity and Zero Jacobian generators
3. Multiplication by a constant
4. Multiplication of two jacobians
5. Multiplication of a jacobian by a waveform
6. Inverting a jacobian

If the waveform is described by an array of double precision floating point numbers, the Jacobian coefficients can also be defined to be an array of double precision floating point numbers. In this case, the above operations employ standard matrix manipulations.

3.3.4 Waveform Examples

While the possibilities of waveform definitions is endless, this thesis will concentrate on the following waveform types:

Waveform Type	Code
Undefined	0
Data Series	1
Fourier Series	2
Legendre Series	3
Polynomials	4
Matlab Polynomials	5
Chebyshev Series	6

The code in the above table refers to the value of element **`type`** in the **WAVEFORM** structure. Appendix E describes these waveforms and their arithmetic in great detail.

3.4 Conducting the Simulation

Once the physical system has been specified by device descriptions and network equations, the solution for all of the system variables can be determined in several ways. The method used in this thesis is the Newton-Raphson method with continuation parameters.

3.4.1 Basic Newton-Raphson Algorithm

The Newton-Raphson method solves a system of nonlinear equations $\boldsymbol{F(x,u) = 0}$, $F() \in \Re^n$, for the system variables $x \in \Re^n$ with system input variables $u \in \Re^m$by first linearizing the system of equations about a given guess for the solution $\boldsymbol{x^k}$ then solving the linear system to produce a new guess $\boldsymbol{x^{k+1}}$. This procedure is repeated until $\boldsymbol{F(x^k,u) = 0}$ is satisfied within a given tolerance. The sequence of points $\boldsymbol{x^k}$ starting with $\boldsymbol{k = 0}$ is called the solution trajectory for $\boldsymbol{x^0}$. A converging solution trajectory eventually converges to a solution while a diverging solution trajectory does not.

$\boldsymbol{F(x,u)}$ is linearized by taking the Taylor series expansion about the point $\boldsymbol{x^k}$:

$$F(x,u) = F(x^k,u) + J(x^k,u)x_\Delta + O(x \cdot x) = 0$$

$$x = x^k + x_\Delta$$

where the Jacobian matrix $\boldsymbol{J(x_k,u)}$ is defined by:

$$J(x^0,u) = \frac{\partial F(x^0,u)}{\partial x}$$

Assuming the error $\boldsymbol{O(x \bullet x)}$ is negligible and the Jacobian can be inverted, the correction x_Δ for a given guess $\boldsymbol{x^k}$ is given by the linear approximation:

$$x_\Delta = -J^{-1}(x^k,u)x^k$$

The correction is applied to $\boldsymbol{x^k}$ to produce $\boldsymbol{x^{k+1}}$, the value of $\boldsymbol{x}$ for the next iteration:

$$x^{k+1} = x^k + x_\Delta$$

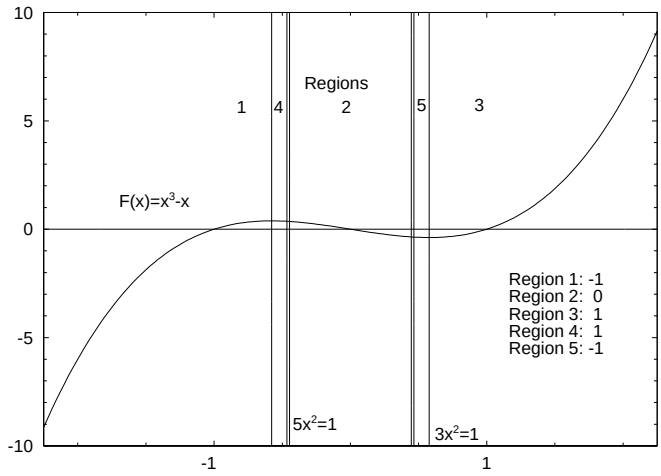
Around each solution of $\boldsymbol{F(x,u) = 0}$ for which the Newton-Raphson method reliably converges, a region exists such that if a trajectory enters that region, it will never leave and eventually converge to the solution. The size of this local convergence region depends on the nonlinearity of the system. For purely linear systems, this region encompasses the

entire $\Re^n$ space. If the intitial guess falls within the local convergence region, the Newton-Raphson method will by definition converge. If the initial guess falls outside the local convergence region, one of several things can happen. First, the solution trajectory could enter the local convergence region of a solution and converge on a solution. Second, the Newton-Raphson method could fail due to a singular Jacobian. Third, the trajectory could diverge and tend to infinity. Fourth, the trajectory could become cyclic where $\boldsymbol{x^{k+q} = x^k}$ for $\boldsymbol{k}$ sufficiently large enough. Finally, the trajectory could enter a chaotic region in which there is no solution but from which the trajectory never leaves and is not cyclic.

As an example, define $\boldsymbol{F(x,u)}$ to be the following 1×1 system:

$$F(x,u) = x^3 - x$$

Figure 3.4.1-1: $\boldsymbol{F(x,u) = x^3-x}$



The Jacobian matrix is:

$$J = [3x^2 - 1]$$

The recursion formula for $\boldsymbol{x^{k+1}}$ is given by:

$$x^{k+1} = x^k - \frac{(x^k)^3 - x^k}{3(x^k)^2 - 1}$$

There are three solutions for this system and their local convergence regions are given by:

Root	Local Convergence Region
$x_1 = -1$	$-\infty < x^0 < -\sqrt{\frac{1}{3}}$
$x_2 = 0$	$-\sqrt{\frac{1}{5}} < x^0 < \sqrt{\frac{1}{5}}$
$x_3 = +1$	$\sqrt{\frac{1}{3}} < x^0 < \infty$

In two other regions, the solution trajectory jumps to one of the local convergence regions after one iteration:

Root	Convergence Region
$x_1 = -1$	$0.46560 < x^0 < \sqrt{\frac{1}{3}} = 0.57735$
$x_3 = +1$	$-\sqrt{\frac{1}{3}} = -0.57735 < x^0 < -0.46560$

In two other regions, the solution trajectory may jump to one of the local convergence regions after several iterations or fail to converge:

Variable Behavior Region
$\sqrt{\frac{1}{5}} = 0.44721 < x^0 < 0.46560$
$-0.46560 < x^0 < -\sqrt{\frac{1}{5}} = -0.44721$

On the boundaries for the above regions, the Newton-Raphson method fails:

x^0	Failure Mode
$\pm\sqrt{\frac{1}{3}}$	Singular Jacobian
± 0.46560	Singular Jacobian
$\pm\sqrt{\frac{1}{5}}$	Cyclic Trajectory

In the above analysis, no constraints were made in the speed of convergence or on the size of ***x***. If $|x^k| \gg 1$ the speed of convergence will be very slow since $x^{k+1} \approx \frac{2}{3}x^k$ and the number of iterations ***l*** required will be about:

$$l - k \approx \frac{\log(|x^k|)}{\log(1.5)} \approx 5.68\log(|x^k|)$$

Furthermore, most machines have a limit as to the largest number which can be represented. If an iteration causes ***x*** to exceed this number in magnitude, a floating point overflow error will typically be generated. This phenomena is known as Newton Overflow and has the effect of reducing the size of the convergence regions. For example, if ***x*** is known to be bounded by the interval **[-10 10]**, then $\boldsymbol{x^0}$ should be restricted to the following regions:

Root	Convergence Region
$\boldsymbol{x_1}$ **= -1**	$-10 < x^0 < -0.58904$ $0.46560 < x^0 < 0.56675$
$\boldsymbol{x_2}$ **= 0**	$-\sqrt{\frac{1}{5}} = -0.44721 < x^0 < \sqrt{\frac{1}{5}} = 0.44721$
$\boldsymbol{x_3}$ **= +1**	$-0.56675 < x^0 < -0.46560$ $0.58904 < x^0 < 10.0$

3.4.2 Continuation Methods with Newton-Raphson

The previous discussion indicates the need for careful selection of the initial guess $\boldsymbol{x}^0$. The use of a continuation parameter in so called homotopy methods is one of the many ways for attempting to generate $\boldsymbol{x}^0$ within the convergence region of the desired solution. In general, a function $\boldsymbol{H(x,u,\alpha) = 0}$ is generated such that $\boldsymbol{H(x,u,1) = F(x,u)}$ and $\boldsymbol{H(x,u,0) = G(x,u)}$ where $\boldsymbol{G(x,u)}$ is a linear function in $\boldsymbol{x}$. One common method of creating $\boldsymbol{H(x,u,\alpha)}$ is:

$$H(x,u,\alpha) = \alpha F(x,u) + (1-\alpha)G(x,u)$$

The problem now is to develop the linear function $\boldsymbol{G(x,u)}$. There are several approaches which can be taken for each row $\boldsymbol{G_i(x,u)}$:

1. Linearize about a known operating point. This is equivelent to providing an initial guess for each of the variables and using the Newton-Raphson method directly.

2. Use a least squares fit of a linear system over a known operating region of $\boldsymbol{F_i(x,u)}$.

3. Select $\boldsymbol{G_i(x,u)}$ such that the solution for $\boldsymbol{H(x,u,0) = 0}$ is most likely to be within the convergence region of $\boldsymbol{F(x,u)}$.

Once $\boldsymbol{H(x,u,\alpha)}$ has been constructed, it can be used in several ways:

1. Start with α**=0** and obtain a solution to the linear system, then progressively increment alpha by small amounts and solve the nonlinear system until α**=1.** The rational is to employ the unbounded local region of convergence of the linear system to move the initial guess into the local region of convergence for the next nonlinear system formed by incrementing α. As α is incremented, the solution for the previous value of α is assumed to be within the local region of convergence for the present value of alpha. Appendix B demonstrates this may not always happen due to bifurcations of solutions as α is incremented.

2. Start with α**=1** and attempt to obtain a solution to the nonlinear solution. If the trajectory has not converged after $\boldsymbol{n_{max}}$ iterations, decrement α and attempt to find a solution. Progressively decrement α until a solution is obtained, then increment α using the solution of the previous value for α for the initial guess. This procedure assumes the local convergence region for a given solution will increase as α is

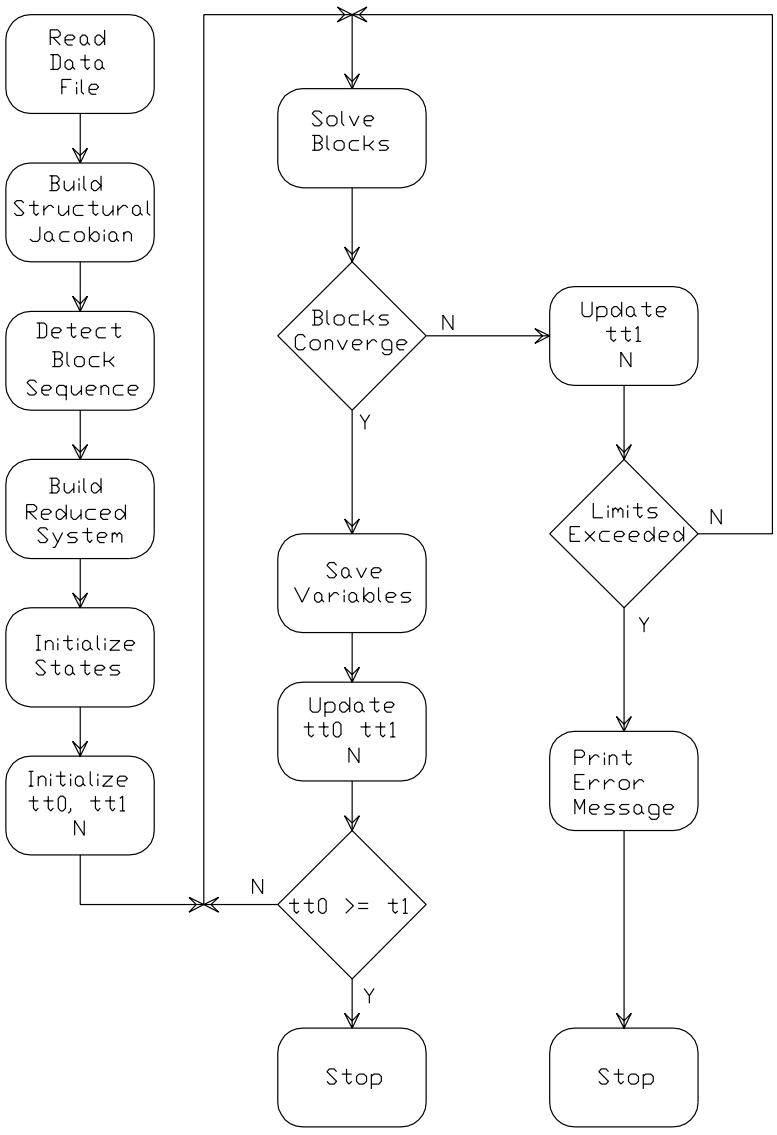
decremented. Eventually the local convergence region should grow large enough to encompass even a poor guess for the solution. This procedure has the advantage over the previous method in that it may avoid bifurcations which occur between **0** and the minimum value for α used. However, the number of iterations for α may be larger.

Note that the value for $\boldsymbol{n}_{max}$ as well as the convergence criteria may be a function of α. There is no reason to obtain a highly accurate solution for intermediate values of α since the only purpose is to move the initial guess for the next α iteration into the new local region of convergence. Only when α=**1** should the convergence criteria be enforced for obtaining a highly accurate solution.

3.4.3 Simulation Algorithm

The simulation algorithm employed by WAVESIM is conducted totally within the MATLAB environment and is composed of four parts. The first part initializes all of the simulation parameters. The second part performs the time increment control and has embedded with in it the third part which is the sequential solving of each of the blocks. The final part is composed mostly of plotting and storing the results of the simulation.

Figure 3.4.3-1: Simulation Flowchart



3.4.3.1 System Initialization

A number of parameters and arrays need initialization before the simulation can commence. These parameters and arrays are:

n	Initial number of waveform coefficients
N	Actual number of waveform coefficients used
wtype	Waveform type indicator
t0	Beginning time of simulation
t1	Ending time of simulation
sb_n_min	Minimum number of coefficients to use
sb_n_max	Maximum number of coefficients to use
sb_n_data	Number of points per waveform for plots
sb_dt_init	Initial time increment
sb_dt_optimum	Optimum time increment
sb_dt_min	Minimum time increment
sb_dt_max	Maximum time increment
sb_dt_ave	Minimum time of interest (Averaging interval)

Break Points are user specified times for which waveform interval boundaries are forced to occur. Break Points are completely optional and their inclusion is up to the system modeler.

sb_bp	Array of Break Points
sb_bp_nbr	Number of break points
sys_node_serial	Array of Node Serial Numbers
sys_node_name	Array of Node Names
sb_alpha_init	Initial Value of continuation parameter alpha for nonlinear blocks
sb_dalpha_init	Initial Value of alpha increment
sb_dalpha_min	Minimum alpha increment
sb_dalpha_max	Maximum alpha increment
sys_Gmin	Array of Gmin values for all of the nodes
sys_Rmin	Array of Rmin values for all of the nodes

The index for **sys_Gmin** and **sys_Rmin** are the node numbers of the nodes they apply to.

sb_check_eqn_err	= 0 for don't check equation error = 1 for checking equation error
sb_check_var_err	= 0 for don't check max variable correction = 1 for checking max variable correction
sys_kcl_err	Array of maximum KCL errors for all nodes
sys_pot_err	Array of maximum Potential Differences for all nodes
sys_nd_err	Array of max corrections to Node Potentials for all nodes
sys_fv_err	Array of max corrections to Flow Variables for all nodes
sb_i_kcl_err	Multiplier for maximum KCL error for alpha less than 1
sb_i_pot_err	Multiplier for maximum Potential Difference for alpha less than 1
sb_i_nd_err	Multiplier for max correction to node potential for alpha less than 1
sb_i_fv_err	Multiplier for max correction to flow variable for alpha less than 1

The index for the above eight arrays are the node numbers of the nodes they apply to.

sb_maxcnt	Maximum number of iterations for alpha = 1
sb_i_maxcnt	Maximum number of iterations for alpha < 1
sb_div_start_cnt	Number of iterations to skip before checking for divergence
sb_div_max_cnt	Maximum number of diverging iterations before assume system is diverging
sb_i_div_err	Multiplier of errors for ignoring diverging check
sb_max_wc	Maximum waveform content of a waveform
sb_nbr_wc	Number of coefficients to apply waveform content to
sb_mult_wc	Multiplier to **sb_max_wc** for decrementing **N**
sys_pot_scale	Array of Scaling factors for node potentials
sys_flow_scale	Array of Scaling factors for flows attached to nodes

The index for **sys_pot_scale** and **sys_flow_scale** are the node numbers of the nodes they apply to.

dev_par_*name*	Device parameter arrays: ***name*** is the device name
dev_s0_*name*	Device state initial value array: ***name*** is the device name

ivar_nd_*nbr*	Initial guesses for node potentials: ***nbr*** is the node serial number
ivart_nd_*nbr*	Waveform type for initial guess ***nbr*** is the node serial number
ivar_fv_*name*	Initial guess for flow variables: ***name*** is the variable name
ivart_fv_*name*	Waveform type for initial guess ***name*** is the variable name
his_t	Matrix of time increment end points First row is beginning of intervals Second row is end of intervals Columns are waveform interval index
his_N	Vector of number of coefficients in waveforms for each waveform interval
his_col	The waveform interval index. After simulation this equals the number of columns in history arrays
his_nd_*nbr*	Matrix of Node Potential waveforms. Each column corresponds to the waveform for the node potential over a given waveform interval. ***nbr*** is the node serial number
his_fv_*name*	Matrix of Import Flow Variable waveforms. Each column corresponds to the waveform for the import flow variable over a given waveform interval. ***name*** is the variable name
his_s_*name*	Matrix of Device ***name*** state values. The first column corresponds to the initial state values with subsequent columns corresponding to the state values at the end of waveform intervals. Note this matrix has 1 more column than all the other history arrays.
blk_*nbr*_nrow	Number of rows in block ***nbr***
blk_*nbr*_ncol	Number of columns in block ***nbr***
blk_*nbr*_row_sys	Cross Reference of Block ***nbr*** rows to System Rows
blk_*nbr*_col_sys	Cross Reference of Block ***nbr*** columns to System Columns

blk_*nbr*_linear_flag	= 0 if block ***nbr*** is nonlinear = 1 if block ***nbr*** is linear

Time Increment Initialization

ddt	Actual time increment
tt0	Beginning of current waveform interval
tt1	End of current waveform interval

ddt, **tt0**, and **tt1** are initialized according to the following equations:

ddt = sb_dt_init

tt0 = t0

tt1 = minimum of:

t0 + ddt

t1

sb_bp(1)

cnt_tot	Set to zero: Total number of Jacobian inverses
his_flops	Number of floating point operations used

3.4.3.2 Time Loop

Truncation Error Control

The simulation time interval between **t0** and **t1** may be divided into a number of waveform intervals to improve the truncation error of the system variable waveforms. In general, truncation error can be reduced by either increasing **N** or by decreasing the waveform interval **tt1 - tt0**. Within WAVESIM, the general strategy for dealing with too large of a truncation error is to increase the number of coefficients **N** if the waveform interval is less than **sb_dt_optimum** and shorten the waveform time interval if greater than **sb_dt_optimum**. In general, the strategy is to minimimize **N** while maximizing the waveform interval subject to the constraint that the truncation error is within tolerances. Finding the optimum combination of waveform intervals and number of coefficients is not obvious and much work remains for developing better algorithms.

3.4.3.2.1 Time Loop iteration initialization

The simulation time loop continues as long as **tt0 < t1.** The beginning of each iteration begins with the definition of the following arrays:

tt = [tt0 tt1 sb_dt_ave]
ii = Identity Matrix of size **N**
zz = Zero Matrix of size **N**×**N**

Variable Initial Guesses

Next, initial guesses are provided for all system variables (**var_nd_*nbr*** and **var_fv_*name***) by converting the waveforms **ivar_nd_*nbr*** of type **ivart_nd_*nbr*** and waveforms **ivar_fv_*name*** of type **ivart_fv_*name*** into waveforms of type **wtype** and size **N**.

In the present incarnation of WAVESIM, the same waveform is used as the initial guess for all waveform time intervals regardless of the values for **tt0** and **tt1**. Normally, a constant value is specified. A better method would allow the user to specify an actual guess as to the waveform history as a function of time. The time loop iteration initialization would then have the responsibility of converting the waveform data as provided by the user into a waveform of type **wtype** and size **N** over the interval between **tt0** and **tt1**. Providing an initial guess for the waveform history of all the variables would allow for example, a linear model of a system be run first to generate the initial guess for a nonlinear model of the same system. Convergence of the nonlinear system

should be greatly accelerated for many systems. Parameter sensitivity studies would also be greatly accelerated if the parameter variations are not expected to cause major changes in system performance.

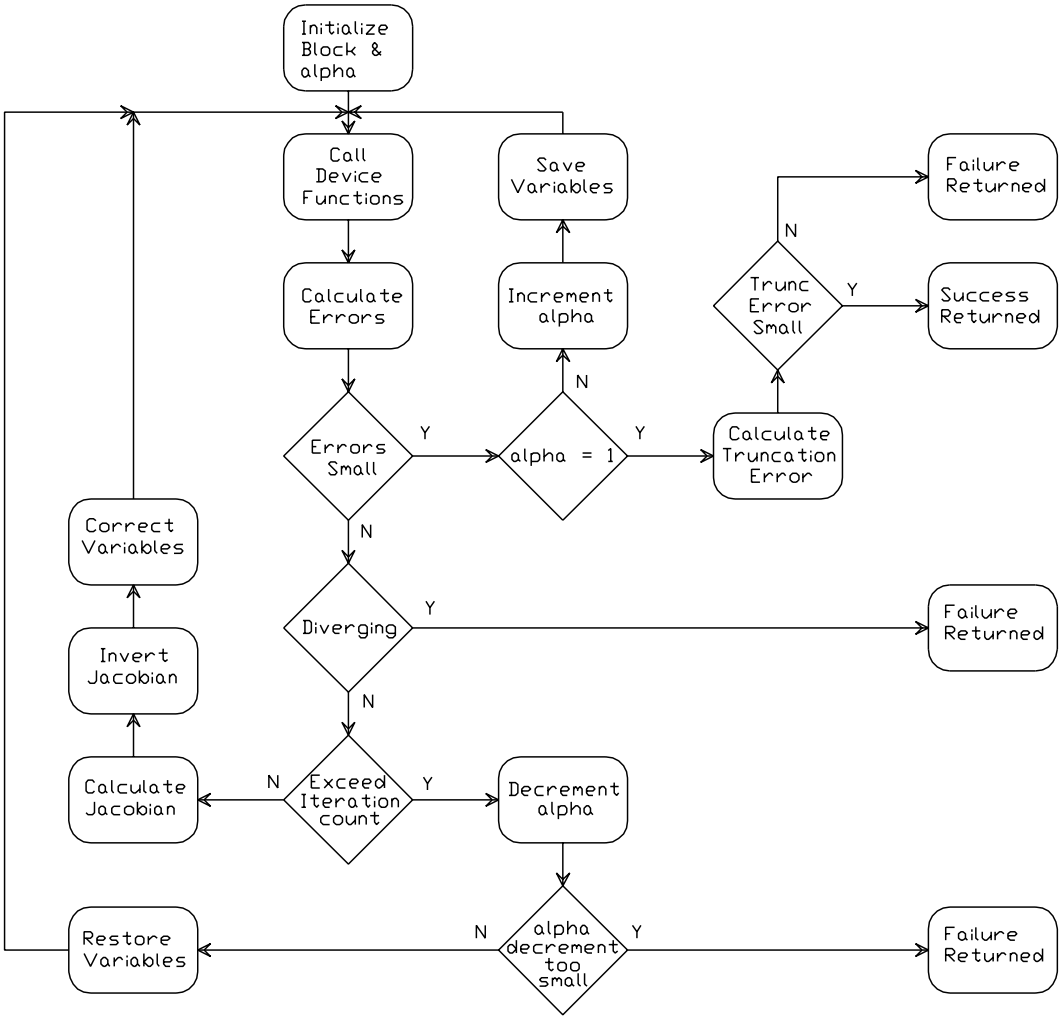
Failure Flags

Two final variables, **converge_failure** and **fatal_error** are initialized to zero. **converge_failure** is set to one by a block if convergence failed for that block or if one of the block waveforms has too large of a harmonic content. Convergence could fail if the number of iterations exceeded the maximum allowed and the alpha increment is smaller than the minimum allowed. **converge_failure** is used to indicate the following blocks should not be solved because previous blocks could not be solved. **fatal_error** is set to one if convergence cannot be obtained even when **N** is equal to or greater than the maximum value **sb_n_max** and the time increment is equal to or smaller than the minimum value **sb_dt_min**. If **fatal_error** is set, the simulation fails.

3.4.3.2.2 Solving the Blocks

The blocks are solved sequentially in the order of their detection in the system reduction procedure. If **converge_failure** is nonzero, a previous block could not be solved for the given time increment and number of coefficients. For this reason, a block is not solved if **converge_failure** is nonzero.

Figure 3.4.3-2: Solving the Block



3.4.3.2.2.1 Block Initialization

Each block requires the initialization of several arrays and variables before the block can be solved:

blk_*nbr*_max_eqnerr	Array of maximum errors for the block equations
blk_*nbr*_max_varcor	Array of maximum variable corrections for the block variables
blk_*nbr*_imax_eqnerr	Array of multipliers to **blk_*nbr*_max_eqnerr** for alpha < 1
blk_*nbr*_imax_varcor	Array of multipliers to **blk_*nbr*_max_varcor** for alpha < 1
blk_*nbr*_cnt	Number of iterations (initialized to **0**)
blk_*nbr*_cnt_div	Number of diverging iterations (initialized to **0**)
blk_*nbr*_alpha	Block continuation parameter. = 1 if linear block = **sb_alpha_init** if nonlinear block
blk_*nbr*_dalpha	Block continuation parameter increment = **sb_dalpha_init**
good_alpha	Last value of alpha for which block converged. Initialized to -1
good_var_nd_*nbr*	Last value of node ***nbr*** potential for which block converged. Initialized to **var_nd_*nbr***
good_var_fv_*name*	Last value of import flow ***name*** for which block converged. Initialized to **var_fv_*name***
blk_*nbr*_trec	Recommended recalculation time for block Initialized to **tt1**
blk_*nbr*_ivc	Array of indexes in block variable array for which the variable correction was greater than allowed. Initialized to an empty array.
div_cnt	Number of diverging iterations, set to **0**
div_err	Maximum relative error of previous iteration Initially set to **0**.

3.4.3.2.2.2 Continuation Parameter Loop

The block continuation parameter loop continues as long as **blk_*nbr*_alpha ≤ 1.** Within this loop, the following procedures occur:

1. Import Variables for all associated devices specified
2. Device Objects called to generate
   A. Export Variables
   B. Device Jacobian Matrix
   C. State values at time **tt1**
   D. Recommended recalculation time
3. Block recalculation time calculated
4. KCL and Potential Difference Equation Errors calculated
5. Errors Scaled and compared to maximum limits
   if good, solution saved and **blk_*nbr*_alpha** incremented as necessary.
6. Iterations counted and compared to maximum limit
   blk_*nbr*_alpha decremented and variables reset as necessary.
7. Block Jacobian Matrix assembled and scaled
8. Variable Corrections Calculated
9. System variables corrected

3.4.3.2.2.2.1 Device Import Variable specification

The matrix **dev_i_*name*** is generated for each device ***name*** where the columns are the waveform coefficients for each of the device import variables. Each column of the **dev_i_*name*** matrix is one of the system variables, hence all are available.

3.4.3.2.2.2.2 Call Device Objects

Each of the device objects associated with the block is provided with the following information:

wtype	Waveform type
dev_i_*name*	Device ***name*** import variable matrix

dev_par_*name* Device ***name*** parameter array

dev_s0_*name* Device ***name*** state initial value **tt0** array

tt Time structure

blk_*nbr*_alpha Block ***nbr*** Continuation Parameter

From this information, each of the device objects generates the following

dev_e_*name* Device ***name*** export variable matrix

dev_j_*name* Device ***name*** jacobian matrix

dev_s1_*name* Device ***name*** state final value **tt1** array

dev_tr_*name* Device ***name*** recommended recalculation structure
= **[nt1 ntt]** where
nt1 = recommended **tt1** for present interval
or set to **tt1** if no recommendation
ntt = recommended **tt1** for next interval
or set to **tt0** if no recommendation

3.4.3.2.2.2.3 Recommended Recalculation Time

The block recommended recalculation time **blk_*nbr*_trec** is set to the minimum value of all the **nt1** values from all of the devices associated with the block. If convergence fails **blk_*nbr*_trec** is used to generate a new value for **tt1**.

Similarly, **blk_*nbr*_ntrec** is set to the minimum value of all the **ntt** values greater than **tt1** from all of the devices associated with the block. For a successful convergence, **blk_*nbr*_ntrec** is used to help generate a new value for **tt1** for the next waveform interval.

3.4.3.2.2.2.4 Equation Errors

For each of the node ***nd*** KCL equations associated with block ***nbr***, an error variable **blk_*nbr*_kcl_*nd*** is generated by adding the flow variables of the attached terminals to the flow through **Gmin**. Likewise, for each of the export potential ***name*** Potential Difference equations associated with block ***nbr***, an error variable **blk_*nbr*_pot_*name*** is generated by subtracting from the node potential waveform, the waveform of the export potential as well as the contribution from **Rmin**

$$\mathbf{blk\_}nbr\mathbf{\_kcl\_}nd = \Sigma\ \mathbf{dev\_e\_}name\mathbf{(:,}col\mathbf{)} + \Sigma\ \mathbf{var\_fv\_}vname + \mathbf{var\_nd\_}nd \times \mathbf{Gmin}$$

blk_*nbr*_pot_*vname* = var_nd_*nd* - dev_e_*name*(:,*col*) -
dev_*x*_*name*(:,*col*) × Rmin

where

nbr	Block Number
nd	Node Serial Number
name	Device name
vname	Variable name
(:,*col*)	The appropriate column from the matrix
x	Either **e** or **i** depending on associated flow variable being an export or import variable

The KCL equation errors are multiplied by the appropriate flow variable scaling factor from the **sys_flow_scale** array while the Potential Difference equation errors are multiplied by the appropriate potential scaling factor from the **sys_pot_scale** array. Once scaled, the error vectors are assembled into a block error vector **blk_*nbr*_err.**

3.4.3.2.2.2.5 Error Criteria Check

Applying Error Criteria

If **blk_*nbr*_alpha** ≥ **1** then **blk_*nbr*_ier** is filled with the indexes of the rows of **blk_*nbr*_err** which are greater in magnitude than the corresponding rows of **blk_*nbr*_max_eqnerr**. In the same manner, **blk_*nbr*_rel_err** is set equal to the absolute value of **blk_*nbr*_err** divided by **blk_*nbr*_max_eqnerr**.

If **blk_*nbr*_alpha < 1** then **blk_*nbr*_ier** is filled with the indexes of the rows of **blk_*nbr*_err** which are greater in magnitude than the corresponding rows of **blk_*nbr*_imax_eqnerr**. Similarly, **blk_*nbr*_rel_err** is set equal to the absolute value of **blk_*nbr*_err** divided by **blk_*nbr*_imax_eqnerr**.

Divergence Check

On the first iteration for a given value **blk_*nbr*_alpha**, **div_cnt** is initialized to **0**. For the first **sb_div_start_cnt - 1** iterations, **div_err** is set to the maximum value of **blk_*nbr*_rel_err**. On subsequent iterations, if the maximum value of **blk_*nbr*_rel_err** is smaller than **div_err** then **div_cnt** is reset to **0**, otherwise **div_cnt** is incremented. In any case **div_err** is set to the maximum value of **blk_*nbr*_rel_err**. If **div_cnt** ≥ **sb_div_max_cnt** then the algorithm assumes the

block is diverging for the given value of **blk_*nbr*_alpha**. The failure to converge condition is indicated by setting **blk_*nbr*_cnt = maxcnt**: either **sb_i_maxcnt** if **blk_*nbr*_alpha < 1** or **sb_maxcnt** if **blk_*nbr*_alpha** ≥ **1**.

Block Convergence Success

If **blk_*nbr*_ier** is the empty set or **sb_check_eqn_err** is **0**, and **blk_*nbr*_alpha** ≥ **1** and **blk_*nbr*_ivc** is the empty set, then the block solving algorithm has been completed and the continuation parameter loop is broken. The algorithm proceeds to checking the truncation error for the system variables associated with the block.

Increment Continuation Parameter

If **blk_*nbr*_ier** is the empty set or **sb_check_eqn_err** is **0**, and **blk_*nbr*_alpha < 1** and **blk_*nbr*_ivc** is the empty set, then it is time to increment the continuation parameter **blk_*nbr*_alpha**. First however, the current value of all the variables associated with the block are copied into **good_var_nd_*nd*** or **good_var_fv_*name***. **blk_*nbr*_alpha** is copied into **good_alpha.** The variables and continuation parameter must be saved because it may be necessary to restore the variables if the block fails to converge with the next continuation parameter value. **blk_*nbr*_alpha** is then set equal to the minimum of **1** and **blk_*nbr*_alpha + blk_*nbr*_dalpha** and the continuation parameter loop is repeated.

Iteration Count: Decrement Continuation Parameter

If the error is still too large, corrections to the system variables associated with the block must be generated. But first, the number of iterations **blk_*nbr*_cnt** must be incremented and compared to the maximum allowed **maxcnt**: either **sb_i_maxcnt** if **blk_*nbr*_alpha < 1** or **sb_maxcnt** if **blk_*nbr*_alpha** ≥ **1**. If the limit has been exceeded, and one of the devices has recommended a value for **blk_*nbr*_trec** less than **tt1**, then **converge_failure** is set to **1** and attempts to solve the block cease. If the limit has been exceeded and **blk_*nbr*_trec** equals **tt1**, the block is recalculated with a decremented **blk_*nbr*_alpha** which is set to the maximum of:

$$\begin{matrix} \texttt{(blk\_}nbr\texttt{\_alpha + good\_alpha) / 2} \\ \texttt{blk\_}nbr\texttt{\_alpha - blk\_}nbr\texttt{\_dalpha} \\ \texttt{0} \end{matrix}$$

If **blk_*nbr*_alpha** has been decremented, the system variables associated with the block must be reset to the values stored in either **good_var_nd_*nd*** or **good_var_fv_*name***.

Block Convergence Failure

If **blk_*nbr*_alpha - good_alpha < sb_dalpha_min** then the block has failed to converge and nothing more can be done on the block level. The variable **trec** is set equal to **blk_*nbr*_trec** and the **converge_failure** flag is set to **1**. This is a signal to the system to not solve any more blocks and either adjust the value of **tt1** or adjust the number of coefficients **N** before trying to solve the system again.

3.4.3.2.2.2.6 Assemble Jacobian

Jacobian Construction

If the error is too large, but the maximum number of iterations **maxcnt** has not been exceeded, the block jacobian matrix must be calculated. The block jacobian matrix **blk_*nbr*_j** is constructed in the same manner as the system structural jacobian was previously constructed with the exception that now the variables and equations are only those which are part of the block and the matrix elements are submatrices instead of structural jacobian codes.

Jacobian Scaling

Once the block jacobian has been assembled, it is scaled by dividing each of the columns by the appropriate element of either the **sys_flow_scale** (if the column corresponds to an import flow variable) or **sys_pot_scale** (if the column corresponds to a node potential) vectors. Likewise, rows of the block jacobian are multiplied by the appropriate element of either the **sys_flow_scale** (if the row corresponds to a KCL equation) or **sys_pot_scale** (if the row corresponds to a Potential Difference equation) vectors. Scaling is performed to normalize all of the variables and hopefully improve the accuracy of the numerical computations required for solving the variable corrections.

Correction Vector Calculation

The variable correction vector **blk_*nbr*_dlta** is generated by solving the matrix equation:

blk_*nbr*_j blk_*nbr*_dlta = blk_*nbr*_err

The most direct method (and one of the least numerically efficient method) of calculating **blk_*nbr*_dlta** is to invert **blk_*nbr*_j** and multiply by **blk_*nbr*_err**. Relaxation methods and Gaussian elimination with back substitution are other means to the same end.

Singular Jacobian

If **blk_*nbr*_j** is singular, **blk_*nbr*_dlta** can not be calculated and in the present incarnation of WAVESIM, the simulation fails. Future versions should include an algorithm for attempting to recover from the singular jacobian.

3.4.3.2.2.2.7 Correct Variables

Each of the system variables associated with the block are corrected by subtracting the appropriate rows of **blk_*nbr*_dlta** divided by the corresponding element of the scaling factor vectors (**sys_pot_scale** or **sys_flow_scale**).

3.4.3.2.2.2.8 Variable Correction Criteria

If the block is nonlinear (**blk_*nbr*_linear_flag == 0**) and the variable correction flag is set (**sb_check_var_err == 1**) then **blk_*nbr*_ivc** contains the indexes of **blk_*nbr*_dlta** which exceed in magnitude **blk_*nbr*_imax_varcor** if **blk_*nbr*_alpha < 1** or **blk_*nbr*_max_varcor** if **blk_*nbr*_alpha** ≥ **1**. If **blk_*nbr*_ivc** is not empty, then one of the variable corrections was too large and another iteration is necessary. In any case, the continuation parameter loop is repeated.

3.4.3.2.2.3 Truncation Error Control

Once a block has been solved, a truncation error check must be performed on each of the associated system variables. The truncation error is assumed negligible if the *waveform content* of the last **sb_nbr_wc** coefficients of each waveform is less than the limit specifed by **sb_max_wc**. If all the system variables have negligible truncation error, block ***nbr*** has been solved and the next block is processed. If the truncation error of any of the variables is too large, **converge_failure** is set to **1** to indicate the block has not been solved.

3.4.3.2.2.3 Time Step Control: Successful Convergence

If all the blocks successfully obtained a solution then the variable **converge_failure** will equal **0**. The task now is to save all of the variables in the history arrays, update **tt0** and **tt1**, update **N**, and update the device states.

Update History Variables

The history variables are extended by one column. The variable **his_col** is incremented and is the column index for all but the state arrays. In particular:

```
his_t(1,his_col) = tt0
his_t(2,his_col) = tt1

his_N(1,his_col) = N

his_nd_nbr(1:N,his_col) = var_nd_nbr
his_fv_name(1:N,his_col) = var_fv_name

his_s_name(:,his_col+1) = dev_s1_name
dev_s0_name = dev_s1_name
```

where **(1:N,his_col)** refers to the first **N** rows of column **his_col** and **(:,his_col+1)** refers to all the rows of column **his_col + 1**.

Update Time Interval and Number of Coefficients

The time interval is updated by:

```
tt0 = tt1
```

If **tt0** ≥**tt1** then the simulation has successfully completed and the time loop is exited. Otherwise must update **tt1** as well. Initially:

```
tt1 = tt1 + ddt
```

Next, check if a break point (element of **sb_bp**) exists between **tt0** and **tt1**. If such a break point exists, set **tt1** equal to the earliest break point after **tt0.**

Since reducing **N** is normally beneficial, if **tt1 - tt0 > sb_dt_optimum** and **N > sb_N_min** the algorithm assumes the waveforms are well behaved and decrementing **N** (as long as **N > sb_n_min**) is appropriate.

Since the series converged for the previous time increment, setting **ddt** equal to the minimum of **2**×**ddt** and **sb_dt_max** allows the system to increase the next time interval.

Plot Intermediate Results

Before proceeding to solve the system over the updated time interval, WAVESIM creates a plot of the system variables over the previous time interval.

3.4.3.2.4 Time Step Control: Unsuccessful Convergence

Fatal Error

If one of the blocks failed to converge, **tt1 - tt2** ≤ **sb_dt_min**, and **N** ≥ **sb_n_max** then the simulation has failed completely and can not proceed further. In this case, the simulation comes to a halt prematurely.

Recommended Recalculation Time

If one of the blocks failed to converge and **trec < tt1**,then **tt1 = trec** and the time loop is repeated.

Time Increment / Number of Coefficient Control

If one of the blocks failed to converge and **trec** ≥ **tt1**, **tt1 - tt0** ≤ **sb_dt_optimum** and **N < sb_n_max**, **N** is incremented in an attempt to improve convergence. To improve convergence if **tt1 - tt0 > sb_dt_optimum** or **N** ≥ **sb_n_max**, the time interval is halved by setting **tt1 = (tt1 + tt0)/2.0**. Halving **ddt** is also prudent as long as **ddt** ≥ **sb_dt_min**. Once **ddt** and **N** have been updated, the time loop is continued.

3.4.3.3 Simulation Wrap-up

Once the simulation has completed, the variables stored in the history arrays are plotted and saved as the user desires. If the operator desires, the device state variables can be used as the initial conditions for a following simulation or saved in file for future simulations.

3.5 Device Modelling Techniques

The previous sections described the method WAVESIM uses to generate a mathematical system of equations and variables for interconnecting a number of different devices. Up to now a device has been treated as a *black box* characterized by its definition, initialization, variables which must be provided to it as resources and variables which are generated by it as products. As a review, here are properties of the *black box*:

Definition (`device.def`)

Name of Device Type

Number of Parameters
Names of Parameters
Default Values of Parameters

Number of States
Names of States
Default Values of State Initial Conditions

Number of terminals
Terminal Definitions
- Terminal Name
- Terminal Type (normal or information)
- Flow Variable Type (import or export)
- Potential Variable Type (import or export)
- Terminal KCL Group Number

Device Structural Jacobian

Initialization (WAVESIM input file)

Name of Device
Name of defining Device Type
Parameter Values
State Initial Conditions
Assignment of terminals to nodes

Resources (Arguments of MATLAB *device*.m file)

Waveform type
Import Variable Waveforms
Parameter Values
Value of states at beginning of time interval
Time Structure
    Beginning time of Interval
    Ending time of interval
    Minimum time interval of interest
Continuation Parameter

Products (Products of MATLAB *device*.m file)

Export Variable Waveforms
Device Jacobian Matrix
Value of states at end of time interval
Recommended Time Structure
    Recommended Recalculation Time this interval
    Recommended ending time of next interval.

While these specifications are the hard requirements for developing a new device type, they are not very constraining and it is possible to generate very inefficient and unworkable devices. The following sections are meant as guidance for developing new device types.

3.5.1 Import and Export Variable definitions

One of the first tasks in designing a new object is determining which variables should be import variables and which should be export variables. The requirement is simply that the total number of export variables associated with normal terminals must equal the total number of import variables associated with normal terminals. To minimize the number of system equations however, one should usually try to define flow variables as export variables and potential variables as import variables.

The constitutive equations defining a device may preclude defining all the flow variables as export variables. An ideal voltage source of magnitude V_S for example, has the following constitutive equations:

$$V_1 = V_2 + V_S$$
$$I_1 = - I_2$$

Clearly, this set of equations can not be reorganized to specify both currents (flows) explicitly. In this case potential V_1 and flow I_1 are export variables and potential V_2 and flow I_2 are import variables.

3.5.2 Interface Variable Units

When developing devices, a consistent convention for interface variable units is required. Flows are usually referenced such that positive flow into a terminal with a positive potential refers to power dissipated by the device. This definition is clear if the flow corresponds to currents or forces, but is less clear for torques. For rotating shafts where torques are the flow variable and rotational speed the potential, the positive direction for speed is in the *normal operating* direction while the direction for torque is determined by the power dissipation rule. A motor connected to a propeller would normally have associated a positive rotational speed and a negative torque. The propeller would have a positive rotational speed and a positive torque associated with its interaction with the motor along with a positive forward speed and negative force associated with its interaction with the ship dynamics. The ship dynamics model would have an associated positive force and positive forward speed.

Many power system simulations go through great effort to normalize all variables by dividing by device base quantities to improve numerical accuracy. The models are all expressed in a Per Unit (PU) basis where the base quantities are machine ratings. The problems occur when several devices with different base quantities are combined. The system variables must all be scaled appropriately to ensure the elements of the system equations are all in the same units. Keeping the bases consistent requires much effort and is very prone to error.

In WAVESIM, physical quantities using the metric system (SI) are recommended for all interface variables. Strict use of the metric system ensures the proper quantities are added and subtracted on the systems level. Individual devices may then scale the interface variables by their own base quantites for internal calculations. Likewise, each node of the system can have a scaling factor assigned to it for both flow and potential variables. In this manner, the beneficial aspects of the per unit system can be retained with little confusion as to ensuring consistent base quantities.

Metric System (SI)

Length	meters
Time	seconds
Mass	kilograms
Voltage	volts
Current	amperes
Force	newtons
Angle	radians
Speed	meters/second
Rotational Speed	radians/second
Torque	newton-meters

3.5.3 Potential References

The node potentials are all referenced to an arbitrary value called **0**. The reference frame for this level is a property of the device definition, but must be consistent with the reference frame for other device definitions to which the device may be connected. Following are suggested reference points:

Electrical Voltage	Volts above Ground Potential
Mechanical Angle	Radians relative to the positive vertical
Mechanical Rotational Speed	Radians per Second relative to stationary
Mechanical Displacement	System Dependent
Mechanical Speed	meters per second relative to stationary

If mechanical rotational speeds or mechanical speeds are specified, but the actual angle is required within the device calculations, the speed can be integrated. If more than one device requires the integration of the speed, then the system modeller must ensure the state initial conditions corresponding to the angle or displacement is consistent for all devices.

If an absolute reference cannot be established for a device, two terminals can be defined such that all constitutive relations depend only on the difference between the two terminal potentials. This relative definition of potentials is commonly used for modelling circuit elements. An ideal transformer for example, is a four terminal device with the following constitutive equations:

$$\boldsymbol{V_{1p} = n(V_{2p} - V_{2m}) + V_{1m}}$$
$$\boldsymbol{i_{1m} = - i_{2m} / n}$$
$$\boldsymbol{i_{1p} = - i_{1m}}$$
$$\boldsymbol{i_{2p} = - i_{2m}}$$

Note that $\boldsymbol{V_{ip}}$ is defined relative to $\boldsymbol{V_{1m}}$ and is a function of $\boldsymbol{(V_{2p} - V_{2m})}$. None of the export flow variables is a function of the absolute value of any of the potentials.

3.5.4 Discontinuity Control

One of the difficulties with using vectors of orthogonal series coefficients to represent waveforms is the poor truncation error performance when approximating discontinuous variables or variables having discontinuous derivatives. These discontinuities are usually a function of either time or the zero crossing of one of the variables. In any case, the time of the discontinuity is often easily determined by the device object. If the frequency of the discontinuities is low enough, it would be prudent for the device to specify the earliest discontinuity of the interval as a recommended recalculation time.

If the discontinuity is a function of a waveform zero crossing, special care must be taken to ensure the device does not continously estimate the zero crossing to be within a small increment of **`tt0`** or **`tt1`** and force the time loop to iterate **`tt1`** around the discontinuity. One way around this problem is for the device to move or remove any discontinuities within **`sb_dt_ave`** of either **`tt0`** or **`tt1`** in any of its export variables. If **`sb_dt_ave`** is small enough, then moving the discontinuity should not affect the accuracy of the simulation very much yet still prevent the system time loop from *hunting* for the discontinuity by varying **`tt1`**.

If many discontinuities occur in an export variable more frequently than **`sb_dt_ave`**, then the export variable should be smoothed. The smoothing operation calculates the local average of a waveform over the interval **`[t-sb_dt_ave,t+sb_dt_ave]`**. In this manner, the higher order terms of the export variable are attenuated and the waveform is more likely to pass the truncation error test.

3.5.5 Consistent Initial Conditions

Most simulation environments require the user to specify the initial values for all the states at time **`t0`**. In this regard WAVESIM is no different. Unfortunately, determining a consistent set of initial conditions which meet some definition of *normal operating conditions* is not an easy task for either a system modeler or a computer program. First of all, the concept of a *normal operating condition*, is not always easy to describe mathematically. Furthermore, even if a definition for *normal operating condition*, can be made, there is often much difficulty in determing that condition.

An ideal solution would be for each device to calculate its own initial conditions during the first time increment. If a device is capable of determining an initial condition based only on its parameters and the values of its import variables, then the following technique can be used:

1. Define a state called **`IC`**, always initialize it to **0**.
2. Define Sufficient Parameters to determine the *normal operating condition*.
3. Within the constitutive equations, have a check for the initial value of **`IC`** equalling zero. If **`IC = 0`** at the beginning of the interval then use the equations for the *normal operationg condition* to determine the initial values of the other states. Otherwise use the initial values of the other states as passed to the device. In any case, the final value for the state **`IC`** should be set to **1**.

This method for determining the initial conditions is well suited for determining the initial conditions of the states of rotating machines. Essentially, a load flow is conducted in the first time increment to determine the initial state values.

3.5.6 Waveform type conversion

Performing the calculations for the constitutive equations for certain devices may be easier to accomplish in one waveform over other waveforms. Converting the import variables to a fixed waveform type is permissible and at times desirable. As long as the export variables are converted back to the proper type and the jacobians reflect the waveform conversions, all should work out well.

If the export variables depend on higher order terms of intermediate calculations, converting the import variables to waveforms of a length longer than `N` and performing all of the intermediate calculations using this longer length before truncating back to `N` when generating the export variables may be desirable in avoiding excessive truncation errors.

Chapter 4 WAVESIM

4.1 Basic Description

WAVESIM, a simulation program written in the C programming language, demonstrates the algorithms discussed in detail in Chapter 3 for simulating systems of nonlinear lumped parameter models representing the electro-mechanical components comprising an Integrated Electric Drive system. The general characteristics of WAVESIM are:

1. System and Simulation Parameters specified in a text **Input File**.
2. Device Definitions are in text file **`device.def`**.
3. WAVESIM Performs following 4 tasks:
   A. Reads in Device Definitions and initializes simulation.
   B. Reads Input File and determines devices and nodes of system.
   C. Builds and reduces system into a sequence of blocks.
   D. Writes a MATLAB **script file** for conducting the simulation.
4. The actual Simulation is conducted in MATLAB.
5. Supported Waveform types are:
   A. Data Series.
   B. Fourier Series.
   C. Legendre Series.
   D. Polynomial Expansions.
   E. Chebyshev Series.
6. Waveform operators are MATLAB functions defined in **M-files**.
7. Device Constitutive Equations are detailed in MATLAB functions defined in M-files.
8. The present Incarnation of WAVESIM has these limitations:
   A. Subsystems have not been implemented.
   B. System and Device Structural Jacobians must be time independent.
   C. Newton-Raphson is the only equation solving method used.
      Relaxation Techniques have not been implemented.

MATLAB was chosen as the environment for conducting the simulation for the following reasons:

1. MATLAB is ideally suited for treating vectors and matrices as abstract data types.
2. MATLAB has built in plotting routines.
3. The ability to create MATLAB **M-files** which when invoked, execute a long series of commands called a **script**. M-files can also be used to create new MATLAB functions.
4. MATLAB has many built in functions for analysing matrix properties.
5. Since WAVESIM is an algorithm demonstration program, speed is not of primary concern. Interest in determining if the algorithms work is of higher interest than optimizing for speed.

4.2 Running WAVESIM

Under either the UNIX operating system or IBM DOS, WAVESIM is executed by entering at the commmand prompt:

```
athena% wavesim file.in
```

where **file.in** is an optional entry for the file name of the input file. WAVESIM will attempt to read in the **device.def** file and if successful, will display the following header:

```
WAVESIM
Revision 2.0     <>     April 1991

(C) Copyright 1990,1991 by Norbert H. Doerry
```

If WAVESIM encountered errors when reading **device.def**, an error message is printed and the program terminates.

If **file.in** was not specified on the command line, the user is prompted for a file name:

```
Enter WAVESIM INPUT file name :
```

If instead of a file name **q** is entered, WAVESIM terminates execution. A directory listing can be obtained by entering a **?** followed optionally by a file specification (operating system dependent).

Under normal execution of WAVESIM, there is no further interaction with the user. WAVESIM automatically creates an output file having the same base filename as **file.in** but having **.m** as an extension (i.e. **file.in** becomes **file.m**).

NOTE: Do not create input files with **.m** extensions as these files will be overwritten by WAVESIM. Also avoid using file names which are valid MATLAB functions.

WAVESIM provides extensive support for providing the user with feedback through the use of the **DEBUG** command. Most of the major routines in WAVESIM have a debug option for displaying the results of calculations internal to WAVESIM.

If errors are found reading either **`device.def`** or the input file, WAVESIM displays an error message which includes the file name and the line number within the file. WAVESIM attempts to continue reading an input file even if errors are detected but will only create an output file if no errors are encountered.

4.3 Input File Specification

The Input File describes the system topology, defines the device parameters, and specifies simulation paramaters. The basic characteristics of the file are:

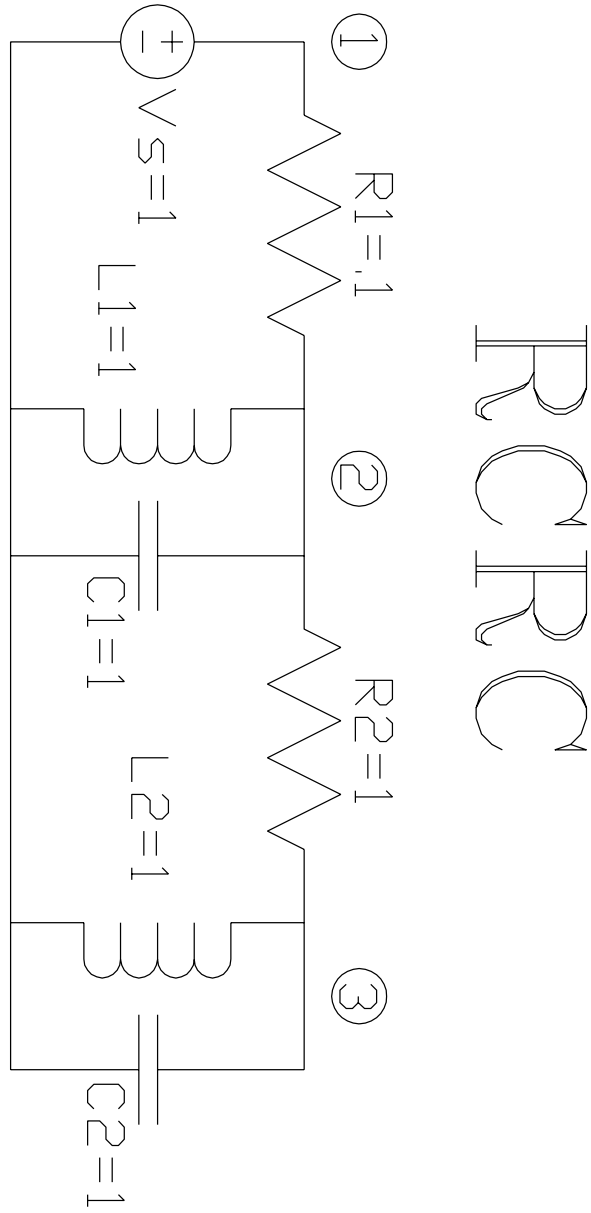
1. ASC II text files.
2. Lines beginning with **%**, **#** or **!** are ignored. Empty lines are ignored as well.
3. Data lines can be continued on the following line if the last characters in the line are **...** or ****.
4. Commands all begin with a **key-word**. Key-words are case insensitive and usually can be truncated to three letters unless a conflict with another key-word exists.
5. Commands and their arguments may be separated by either spaces or tabs.
6. The contents of other files can be incorporated by using the **INCLUDE** command.
7. Single Line Commands have data arguments entered on only one line.
8. Multiple Line Commands consist of groups of subordinate commands. The group must end with a line beginning with the key-word **END**.

Here is a summary of the Commands available :

DEBUG	**Print Debug Information**
DEFAULT	**Default System Parameter Initialization**
DEVICE	**Device Specification**
INCLUDE	**Include another file**
NODE	**Node Parameter Specification**
TIME	**Time Increment Control**

Example Input File

```
%
% rcrc.in
%
debug
     build_system_identify
     build_system_blocks
     find_block
     END
%
device VDC_SOURCE Vs
     TERMINAL  1 1
     TERMINAL  2 0
     PARAMETER   VS 1.0
     END
%
device RESISTOR R1
     TERMINAL  1 1
     TERMINAL  2 2
     PARAMETER   R  0.1
     END
%
device RESISTOR R2
     TERMINAL  1 2
     TERMINAL  2 3
     PARAMETER   R  1.0
     END
%
device INDUCTOR L1
     TERMINAL  1 2
     TERMINAL  2 0
     PARAMETER   L  1.0
     END
%
device INDUCTOR L2
     TERMINAL  1 3
     TERMINAL  2 0
     PARAMTER    L  1.0
     END
%
device CAPACITOR C1
     TERMINAL  1 2
     TERMINAL  2 0
     PARAMETER   C  1.0
     END
%
device CAPACITOR C2
     TERMINAL  1 3
     TERMINAL  2 0
     PARAMETER   C  1.0
     END
%
```



Example Input File (continued)

```
%
node 1
     scale potential 1.0
     scale flow      1.0
     error kcl      5e-3
     error pot      5e-3
     end
%
default
     Gmin  0
     Rmin  0

     check both

     error eqn kcl   5e-3
     error eqn pot   5e-3
     error var node  5e-3
     error var flow  5e-3
     error mult kcl  10.0
     error mult pot  10.0
     error mult node 10.0
     error mult flow 10.0

     scale potential 1.0
     scale flow      1.0

     max count 10
     max int count 6

     alpha init 1.0
     alpha inc init .25
     alpha inc min .05

     diverge start 3
     diverge max cnt 2
     diverge error mult 10.0

     waveform content max .005
     waveform content nbr 2

     wtype 3

     nbr coef 7
     nbr coef min 6
     nbr coef max 14
     nbr data 20

     END
%
time
     dt min  0.025
     dt max  5.0
     dt opt  0.250
     dt init 1.0
     dt ave  0.0
     start   0.0
     finish 20.0
     END
%
```

4.3.1 DEBUG

If **DEBUG** is specified without any arguments, the command is interpreted as a multi-line command. Each of the following lines should contain the name of one of the subroutines listed below. If the key-word **OFF** follows the subroutine name, the debug flag for that subroutine is turned off. Otherwise, the debug flag for the specified routine is turned on. The last line of the group should begin with the key-word **END**.

If **DEBUG** is specified with arguments, the command is interpreted as a single-line command and the arguments should consist of one of the subroutines listed below and optionally, the key-word **OFF**. A single line command does **not** have an **END** keyword associated with it.

Here is a list of subroutines for which debug flags have been defined (Note: The subroutine names are case sensitive)

init_devices
read_device_def
read_file
read_file_device
read_file_default
read_file_node
read_file_time
read_file_debug
build_system
build_system_identify
build_system_structural_jacobian
build_system_blocks
find_block
print_system_identify
write_file

4.3.2 DEFAULT

If **DEFAULT** is specified without any arguments, the command is interpreted as a multi-line command. Each of the following lines should contain one of the subordinate commands listed below. The last line of the group should begin with the key-word **END**.

If **DEFAULT** is specified with arguments, the command is interpreted as a single-line command and the arguments should consist of one of the subordinate commands listed below. A single line command does **not** have an **END** keyword associated with it.

Here is a summary of the **DEFAULT** subordinate commands:

ALPHA	**Continuation Parameter Control**
CHECK	**Error Checking Flags**
DIVERGE	**Divergence Test Control**
ERROR	**Default Error Levels**
GMIN	**Default Node Leakage Conductance**
MAX	**Maximum Iteration Counts**
NBR	**Number of Coefficients Control**
RMIN	**Default Node Series Resistances**
SCALE	**Default Variable Scaling Factors**
WAVEFORM CONTENT	**Waveform Content Limits**
WTYPE	**Waveform Type**

4.3.2.1 DEFAULT: ALPHA

The **ALPHA** subordinate command specifies the parameters needed to control the continuation parameter for nonlinear blocks.

Command	Description	MATLAB Variable
ALPHA INIT *Value*	Continuation Parameter Initial Value	**sb_alpha_init**
ALPHA INC INIT *Value*	Continuation Parameter Initial Increment	**sb_dalpha_init**
ALPHA INC MIN *Value*	Minimum Continuation Parameter Increment	**sb_dalpha_min**

For a nonlinear block, the continuation parameter is initialized to the **ALPHA INIT** value. The initial increment for the continuation parameter is specified by **ALPHA INC INIT**. If the block fails to converge, the continuation parameter is progressively decremented until the block converges or if convergence fails due to the difference between the last value of the continuation parameter that converged and the present value of the continuation parameter being less than **ALPHA INC MIN**. If the block converges, the continuation parameter is incremented by **ALPHA INC INIT** until it equals **1**.

4.3.2.2 DEFAULT: CHECK

The **CHECK** subordinate command determines for nonlinear blocks, whether the equation error, the variable correction magnitude, or both should be used for the convergence criteria.

Command	Description	MATLAB Variable
CHECK EQN	Check only Equation Errors	**sb_check_eqn_err = 1** **sb_check_var_err = 0**
CHECK VAR	Check only Variable Corrections	**sb_check_eqn_err = 0** **sb_check_var_err = 1**
CHECK BOTH	Check both Equation Errors and Variable Corrections	**sb_check_eqn_err = 1** **sb_check_var_err = 1**

4.3.2.3 DEFAULT: DIVERGE

The **DIVERGE** subordinate command specifies when and how to check a nonlinear block for divergence. After **DIVERGE START** iterations, if the largest relative error increases for **DIVERGE MAX CNT** iterations and the relative error is at least **DIVERGE ERROR MULT** then the block is assumed to be diverging and the **converge_failure** flag is set.

Command	Description	MATLAB Variable
DIVERGE START *Value*	Number of iterations to wait before testing for divergence	**sb_div_start_cnt**
DIVERGE MAX CNT *Value*	Number of iterations to allow relative error to increase befor concluding divergence	**sb_div_max_cnt**
DIVERGE ERR MULT *Value*	Value of relative error below which to ignore divergence iteration count	**sb_i_div_err**

4.3.2.4 DEFAULT: ERROR

The **ERROR** subordinate command determines for nonlinear blocks, the default maximum equation errors and variable corrections which are permissible. These default values can be overridden for a specific node with the **NODE** command. When the conituation parameter equals **1**, **ERROR EQN KCL** is the maximum error for the node KCL eqution and **ERROR EQN POT** is the maximum error for the potential difference equations. Likewise when the continuation parameter equals **1**, **ERROR VAR NODE** is the maximum correction to a node potential and **ERROR VAR FLOW** is the maximum correction to an import flow variable. The **ERROR MULT** subordinate commands are multipliers to the above limits for continuation parameters less than **1**.

Command	Description	MATLAB Variable
ERROR EQN KCL *Value*	Default KCL Equation Maximum Error	**sys_kcl_err**[1]
ERROR EQN POT *Value*	Default Potential Difference Maximum Error	**sys_var_err**[1]
ERROR VAR NODE *Value*	Default Maximum correction to Node Potentials	**sys_nd_err**[1]
ERROR VAR FLOW *Value*	Default Maximum correction to Import Flow Variables	**sys_fv_err**[1]
ERROR MULT KCL *Value*	Multiplier to **ERROR EQN KCL** when continuation parameter < 1	**sb_i_kcl_err**
ERROR MULT POT *Value*	Multiplier to **ERROR EQN POT** when continuation parameter < 1	**sb_i_pot_err**
ERROR MULT NODE *Value*	Multiplier to **ERROR VAR NODE** when continuation parameter < 1	**sb_i_nd_err**
ERROR MULT FLOW *Value*	Multiplier to **ERROR VAR FLOW** when continuation parameter < 1	**sb_i_fv_err**

Note 1: **sys_*xxx*_err** are actually arrays containing for each equation or variable, either the default value specified here or the overriding value specified in the **NODE** command.

4.3.2.5 DEFAULT: GMIN

The **GMIN** subordinate command defines the default value for G_{min}. G_{min} is used to modify the KCL equations to help prevent singular systems. G_{min} should normally be set to **0** unless a singularity problem exists. The value for G_{min} can be overridden for a particular node through the **NODE** command.

Command	Description	MATLAB Variable
GMIN ***Value***	Leakage Conductance to **0** Potential	**sys_Gmin**[1]

Note 1: **sys_Gmin** is actually an array containing the value for G_{min} for each node: either the default value specified here or the overriding value specified in the **NODE** command.

4.3.2.6 DEFAULT: MAX

The **MAX** subordinate command determines the maximum number of Newton-Raphson iterations for a nonlinear block before the continuation parameter is decremented. **MAX COUNT** specifies the maximum number of iterations when the continuation parameter equals **1** while **MAX INT COUNT** specifies the maximum number of iterations when the continuation parameter is less than **1**.

Command	Description	MATLAB Variable
MAX COUNT ***Value***	Maximum number of Iterations when the continuation parameter equals 1	**sb_maxcnt**
MAX INT COUNT ***Value***	Maximum number of Iterations when the continuation parameter is less than 1	**sb_i_maxcnt**

4.3.2.7 DEFAULT: NBR

The **NBR** subordinate command controls the number of coefficients the waveforms will have. **NBR COEF** specifies the initial number of coefficients to use. **NBR COEF MIN** is the minimum number of coefficients to use while **NBR COEF MAX** is the maximum number of coefficients. **NBR DATA** is the number of data points per waveform used when generating plots.

Command	Description	MATLAB Variable
NBR COEF *Value*	Initial number of coefficinets	**n**
NBR COEF MIN *Value*	Minimum number of coefficients	**sb_n_min**
NBR COEF MAX *Value*	Maximum number of coefficients	**sb_n_max**
NBR DATA *Value*	Number of points per waveform to use in plots.	**sb_n_data**

4.3.2.8 DEFAULT: RMIN

The **RMIN** subordinate command defines the default value for R_{min}. R_{min} is used to modify the Potential Difference equations to help prevent singular systems. R_{min} should normally be set to **0** unless a singularity problem exists. The value for R_{min} can be overridden for a particular node through the **NODE** command.

Command	Description	MATLAB Variable
RMIN *Value*	Series Resistance for Export Potentials	**sys_Rmin**[1]

Note 1: **sys_Rmin** is actually an array containing the value for R_{min} for each node: either the default value specified here or the overriding value specified in the **NODE** command.

4.3.2.9 DEFAULT: SCALE

The **SCALE** subordinate command specifies the default scaling parameters for the potential and flow variables. The default scaling parameters can be overridden for a particular node through the **NODE** command.

Command	Description	MATLAB Variable
SCALE POTENTIAL ***Value***	Default scaling factor for Potentials	**sys_pot_scale**[1]
SCALE FLOW ***Value***	Default scaling factor for Flow Variables	**sys_flow_scale**[1]

Note 1: **sys_pot_scale** and **sys_flow_scale** are actually arrays containing the scaling factors for each node: either the default values specified here or the overriding values specified in the **NODE** command.

4.3.2.10 DEFAULT: WAVEFORM CONTENT

The **WAVEFORM CONTENT** subordinate command controls the maximum allowable truncation error by specifying the maximum waveform content **WAVE CONT MAX** for the last **WAVE CONT NBR** coefficients of a waveform.

Command	Description	MATLAB Variable
WAVE CONT MAX ***Value***	Maximum Waveform Content	**sb_max_hh**
WAVE CONT NBR ***Value***	Number of Coefficients to apply maximum to.	**sb_nbr_hh**

4.3.2.11 DEFAULT: WTYPE

The **WTYPE** subordinate command specifies the waveform type to use in the simulation

Command	Description	MATLAB Variable
WTYPE ***Value***	Waveform Type Indicator 1 Data Series 2 Fourier Series 3 Legencre Series 4 Polynomials 5 MATLAB Polynomials 6 Chebyshev Series	**wtype**

4.3.3 DEVICE

DEVICE is always a multi-line command. The command must be entered in the following format:

DEVICE *Device_Type Name*

where:

Device_Type	Device Type Name from **device.def** file.
Name	Name of this particular device.

The subordinate commands for the **DEVICE** command are:

TERMINAL	Assign Terminals to Nodes (mandatory).
PARAMETER	Assign Parameter Values (optional).
STATE	Assign State Initial Conditions (optional).

All of the terminals as defined in the **device.def** must be assigned to a node. If the parameters or states are not assigned values, the default values specified in **device.def** are used.

The last line of the command group must begin with the key-word **END**

4.3.3.1 DEVICE: TERMINAL

The **TERMINAL** subordinate command assigns a terminal to a node and must be entered in the following format:

TERMINAL *Terminal_Name Node_Nbr*

where:

Terminal_Name	Terminal Name from **device.def** file.
Node_Nbr	Serial Number of Node this terminal is attached to.

All of the terminals as defined in **device.def** must be attached to a node of the system.

4.3.3.2 DEVICE: PARAMETER

The **PARAMETER** subordinate command assigns a value to a parameter of the device. If the parameter is a single value as defined in **device.def** then the parameter command must be of the following format:

PARAMETER *Parameter_Name Value*

where:

Parameter_Name	Parameter Name from **device.def** file.
Value	Parameter Value.

If the parameter is a matrix as defined in **device.def** then the parameter command must be of the following format:

PARAMETER *Parameter_Name* MATRIX
matrix_values
END

where:

Parameter_Name	Parameter Name from **device.def** file.
matrix_values	Parameter matrix. The number of rows and columns of the matrix must be the same as specified in the **device.def** file. Rows are entered one line at a time with columns separated by spaces.

If a parameter as defined in **device.def** is not assigned a value, then the default values specified in **device.def** is used.

4.3.3.3 DEVICE: STATE

The **STATE** subordinate command assigns an initial value to a state of the device and must be entered in the following format:

STATE *State_Name Value*

where:

State_Name	State Name from **device.def** file.
Value	Initial value of state.

If a state as defined in **device.def** is not assigned a value, then the default values specified in **device.def** is used.

4.3.4 INCLUDE

INCLUDE is always a single-line command. The command must be entered in the following format:

INCLUDE *File_Name*

where:

File_Name	Name of the file to include.

The contents of the included file are inserted at the location of the **INCLUDE** command.

4.3.5 NODE

NODE is always a multi-line command. The command must be entered in the following format:

NODE *Node_Nbr*

where:

Node_Nbr	Serial Number of the node.

The subordinate commands for the **NODE** command are:

ERROR	Node Error Levels
GMIN	Specify node G_{min} value
NAME	Assign a name to the node
RMIN	Specify node R_{min} value
SCALE	Specify node scaling factors

The last line of the command group must begin with the key-word **END**

4.3.5.1 NODE: ERROR

The **ERROR** subordinate command determines for nonlinear blocks, the maximum equation errors and variable corrections which are permissible. These values override the default values. When the conituation parameter equals **1**, **ERROR EQN KCL** is the maximum error for the node KCL eqution and **ERROR EQN POT** is the maximum error for the potential difference equations. Likewise when the continuation parameter equals **1**, **ERROR VAR NODE** is the maximum correction the node potential and **ERROR VAR FLOW** is the maximum correction to an import flow variable. The **ERROR MULT** subordinate commands of the **DEFAULT** command are multipliers to the above limits for continuation parameters less than **1**.

Command	Description	MATLAB Variable
ERROR EQN KCL *Value*	Maximum KCL Equation Error	**sys_kcl_err**[1]
ERROR EQN POT *Value*	Maximum Potential Difference Error	**sys_var_err**[1]
ERROR VAR NODE *Value*	Maximum correction to Node Potential	**sys_nd_err**[1]
ERROR VAR FLOW *Value*	Maximum correction to Import Flow Variables attached to this node	**sys_fv_err**[1]

Note 1: **sys_*xxx*_err** are actually arrays containing for each equation or variable, either the default value or the overriding value specified here.

4.3.5.2 NODE: GMIN

The **GMIN** subordinate command defines the value for G_{min}. G_{min} is used to modify the KCL equation to help prevent singular systems. G_{min} should normally be set to **0** unless a singularity problem exists. The value for G_{min} overrides the default value.

Command	Description	MATLAB Variable
GMIN *Value*	Leakage Conductance to **0** Potential	**sys_Gmin**[1]

Note 1: **sys_Gmin** is actually an array containing the value for G_{min} for each node: either the default value or the overriding value specified here.

4.3.5.3 NODE: NAME

The **NAME** subordinate command

Command	Description	MATLAB Variable
NAME ***Node_Name***	Name of the Node	**sys_node_name**[1]

The node name is only used to associate the node serial number to a more understandable label. The node name is optional and does not affect computation in any way.

Note 1: **sys_node_name** is actually an array containing the names of all the nodes.

4.3.5.4 NODE: RMIN

The **RMIN** subordinate command defines the node value for R_{min}. R_{min} is used to modify the Potential Difference equations to help prevent singular systems. R_{min} should normally be set to **0** unless a singularity problem exists. The value for R_{min} overrides the default value.

Command	Description	MATLAB Variable
RMIN ***Value***	Series Resistance for Export Potentials	**sys_Rmin**[1]

Note 1: **sys_Rmin** is actually an array containing the value for R_{min} for each node: either the default value or the overriding value specified here.

4.3.5.5 NODE: SCALE

The **SCALE** subordinate command specifies the node scaling parameters for the potential and flow variables. The scaling parameters override the default values.

Command	Description	MATLAB Variable
SCALE POTENTIAL ***Value***	Scaling factor for Node Potential	**sys_pot_scale**[1]
SCALE FLOW ***Value***	Node scaling factor for Flow Variables	**sys_flow_scale**[1]

Note 1: **sys_pot_scale** and **sys_flow_scale** are actually arrays containing the scaling factors for each node: either the default values or the overriding values specified here.

4.3.6 TIME

If **TIME** is specified without any arguments, the command is interpreted as a multi-line command. Each of the following lines should contain one of the subordinate commands listed below. The last line for the section should begin with the key-word **END**.

If **TIME** is specified with arguments, the arguments should consist of one of the subordinate commands listed below. A single line command does **not** have an **END** keyword associated with it.

The subordinate commands for the **TIME** command are:

BREAK	Insert Break Point
DT	Time Increment Control
FINISH	Ending Time of Simulation
START	Starting Time of Simulation

4.3.6.1 TIME: BREAK

The **BREAK** subordinate command inserts a simulation break point which forces a waveform boundary to occur at the designated time. Bracketing intervals in which a discontinuity will occur with breakpoints can reduce the computational effort required by WAVESIM.

Command	Description	MATLAB Variable
BREAK ***Time***	Break Point time	**sb_bp**[1] **sb_bp_nbr**

Note 1: **sb_bp** is actually an array of break points in chronological order. **sb_bp_nbr** is the number of break points.

4.3.6.2 TIME: DT

The **DT** subordinate command controls the waveform interval.

Command	Description	MATLAB Variable
DT MIN *Value*	Minimum Waveform Increment	**sb_dt_min**
DT MAX *Value*	Maximum Waveform Increment	**sb_dt_max**
DT OPTIMUM *Value*	Optimum Waveform Increment	**sb_dt_optimum**
DT INITIAL *Value*	Initial Waveform Increment	**sb_dt_init**
DT AVE *Value*	Minimum Time Interval of Interest (Averaging Interval)	**sb_dt_ave**

If the time interval is less than **DT OPTIMUM**, the number of coefficients is less then the maximum and a block does not converge, the number of coefficients is increased for the next iteration. Otherwise, if the block does not converge the time interval is reduced.

DT AVE is the minimum time interval of interest and is used by devices to smooth their export waveforms or to move discontinuity boundaries.

4.3.6.3 TIME: FINISH

The **FINISH** subordinate command specifies the ending time of the simulation.

Command	Description	MATLAB Variable
TIME FINISH *Value*	Ending Time of Simulation	**t1**

4.3.6.4 TIME: START

The **START** subordinate command specifies the beginning time of the simulation.

Command	Description	MATLAB Variable
TIME START *Value*	Starting Time of Simulation	**t0**

4.4 Device Definition File Specification

The device definition file **`device.def`** contains the definitions of the device types which can be specified in a WAVESIM Input File. The basic characteristics of the file are:

1. ASC II text files
2. Lines beginning with **`%`**, **`#`** or **`!`** are ignored. Empty lines are ignored as well.
3. Data lines can be continued on the following line if the last characters in the line are **`...`** or **`\`**.
4. Commands all begin with a **key-word**. Key-words are case insensitive and usually can be truncated to three letters unless a conflict with another key-word exists.
5. Commands and their arguments may be separated by either spaces or tabs.
6. The contents of other files can be incorporated by using the **`INCLUDE`** command.
7. Single Line Commands have data arguments entered on only one line.
8. Multiple Line Commands consist of groups of subordinate commands. The group must end with a line beginning with the key-word **`END`**.

Example `device.def` File

```
%
%  device.def
%
% debug init_devices
debug read_device_def
%
device RESISTOR
     Terminal 1 Pot    V1 Import
     Terminal 1 Flow   I1 Export 1
     Terminal 2 Pot    V2 Import
     Terminal 2 Flow   I2 Export 1
     Parameter  R 1e-15
     Function   resistor
     Structural Jacobian All
          DD
          DD
     end
end
%
device INDUCTOR
     Terminal 1 Pot    V1 Import
     Terminal 1 Flow   I1 Export 1
     Terminal 2 Pot    V2 Import
     Terminal 2 Flow   I2 Export 1
     Parameter  L 1e-15
     Function   inductor
     Structural Jacobian All
          LL
          LL
     end
end
%
device CAPACITOR
     Terminal 1 Pot    V1 Export
     Terminal 1 Flow   I1 Export 1
     Terminal 2 Pot    V2 Import
     Terminal 2 Flow   I2 Import 1
     Parameter  C 1e-15
     Function   capacitor
     Structural Jacobian All
          IL
          0D
     end
end
%
device VDC_SOURCE
     Terminal 1 Pot    V1 Export
     Terminal 1 Flow   I1 Export 1
     Terminal 2 Pot    V2 Import
     Terminal 2 Flow   I2 Import 1
     Parameter  VS 1.0
     Function   vdc_src
     Structural Jacobian All
          I0
          0D
     end
end
%
device REFERENCE
     Terminal Gnd Pot  V0 Export
     Terminal Gnd Flow I0 Import 0
     Parameter  Vref 0.0
     Structural Jacobian ALL
          0
     end
end
%
% include load flow definitions
include loadflow.def
%
% include rotating machinery IED Models
include powersys.def
%
% include other circuit elements
include circ_elm.def
```

4.4.1 DEBUG

The **DEBUG** command is always a single-line command and results in the display of debug information for a specified routine during the execution of WAVESIM.

Command	Description
DEBUG init_devices	Print Info on Initial System Parameters
DEBUG read_device_def	Print Info on what is read from **device.def**

4.4.2 DEVICE

DEVICE is always a multi-line command. The command must be entered in the following format:

DEVICE *Device_Type*

where:

Device_Type	Device Type Name (must be unique)

The Device Type Name is used to correlate a given device in an input file with the properties of the device as specified here. The subordinate commands for **DEVICE** are:

TERMINAL	Specify Terminal Variable Properties
PARAMETER	Specify Parameters
STATE	Specify States
FUNCTION	Specify MATLAB function
STRUCTURAL JACOBIAN	Specify Structural Jacobian

The last line of the command group must begin with the key-word **END**

4.4.2.1 DEVICE: TERMINAL

The **TERMINAL** subordinate command defines the properties of the variables associated with a terminal. If the Terminal is a normal terminal, both the flow and potential variables need definitions. Flow variables also require a KCL group number ***KCL*** which corresponds to the group of terminals for which KCL can be written internally to the device. If the flow variable does not belong to a KCL group, its value should be **0**. Variable are **IMPORT** is they are a resource to the device and are **EXPORT** if they are a product of the device. The total number of export variables associated with normal nodes must equal the total number of import variables associated with normal nodes.

Normal Node potentials are defined by either

TERMINAL *Terminal_Name* POTENTIAL *Variable_Name* EXPORT

or

TERMINAL *Terminal_Name* POTENTIAL *Variable_Name* IMPORT

Normal Node flows are defined by either

TERMINAL *Terminal_Name* FLOW *Variable_Name* EXPORT *KCL*

or

TERMINAL *Terminal_Name* FLOW *Variable_Name* IMPORT *KCL*

Information Node potentials are defined by either

TERMINAL *Terminal_Name* INFORMATION *Variable_Name* EXPORT

or

TERMINAL *Terminal_Name* INFORMATION *Variable_Name* IMPORT

Where

Terminal_Name	One word name for Terminal
Variable_Name	One word name for Variable
KCL	KCL Group Number (0 if none)

4.4.2.2 DEVICE: PARAMETER

The **PARAMETER** subordinate command defines the parameters of the device and optionally, declares the default values for the parameters. Parameters can either be single valued or a matrix. A single valued parameter is defined by:

PARAMETER *Parameter_Name Default_Value*

where

Parameter_Name	One word name for Parameter
Default_Value	Optional Default Value for Parameter

Matrix parameters for which which no default values are provided are defined by:

PARAMETER *Parameter_Name* MATRIX *Nbr_Row Nbr_Col*

where

Nbr_Row	Number of rows in Matrix 0 should never be used -1 indicates variable dimensioned
Nbr_Col	Number of columns in Matrix 0 should never be used -1 indicates variable dimensioned

Matrix parameters for which which default values are provided are defined by:

PARAMETER *Parameter_Name* MATRIX *Nbr_Row Nbr_Col*
DEFAULT
Default_Matrix
END

where

Default_Matrix	Default Matrix values, Each matrix row should be entered one line at a time with columns separated by spaces.

4.4.2.3 DEVICE: STATE

The **STATE** subordinate command defines the states of the device and optionally, declares the default initial values for the states. States are defined by:

STATE *State_Name Default_Value*

where

State_Name	One word name for State
Default_Value	Optional Default Initial Value for State

4.4.2.4 DEVICE: FUNCTION

The **FUNCTION** subordinate command is mandatory and defines the MATLAB function which defines the device constitutive equations. The MATLAB function is specified by:

FUNCTION *MATLAB_Function*

where

MATLAB_Function	MATLAB Function name

4.4.2.5 DEVICE: STRUCTURAL JACOBIAN

The **STRUCTURAL JACOBIAN** subordinate command defines the structural jacobian matrix of the device for 1 or for all of the waveform types. The structural jacobian for all waveform types is specified by:

```
STRUCTURAL JACOBIAN ALL
    Structural_Jacobian
    END
```

where

Structural_Jacobian	Structural Jacobian Matrix. The Rows correspond to Export Variables ordered according to the order of definition. Similarly, the Columns correspond to Import Variables ordered according to the order of definition. The elements are Structural Jacobian Codes detailed below.

The structural jacobian for one particular waveform type is specified by:

```
STRUCTURAL JACOBIAN Waveform_Type
    Structural_Jacobian
    END
```

where

Waveform_Type	Waveform Type Code the structural jacobian is defined for

Structural Jacobian Codes

Code	Type of Matrix
0	Zero Matrix (all elements are always zero)
I	Identity Matrix (always the identity matrix)
D	Diagonal Matrix (always a linear main diagonal matrix)
L	Linear Matrix (The elements are always constant)
A	Nonlinear AC Matrix (see Note 1)
N	Nonlinear Matrix (The elements may not be constants)
U	Unknown (The dependence is unknown (treat as nonlinear))

Note 1: An AC Matrix is one for which the constant component of the export variable depends only on the constant component of the import variable. The other components of the export variable can not depend on the constant component of the import variable but are not restricted in any other way.

Waveform Type Codes

Waveform Type	Code
Undefined	0
Data Series	1
Fourier Series	2
Legendre Series	3
Polynomials	4
Matlab Polynomials	5
Chebyshev Series	6

4.4.3 INCLUDE

INCLUDE is always a single-line command. The command must be entered in the following format:

INCLUDE *File_Name*

where:

File_Name Name of the file to include.

The contents of the included file are inserted at the location of the **INCLUDE** command.

4.5 Adding devices

Adding New devices to WAVESIM requires the creation of a MATLAB M-file defining the device constitutive equations and the addition of an entry in the **`device.def`** file.

4.5.1 MATLAB M-FILE

Creating a MATLAB M-FILE for generating a new device requires adherence to a strict function argument list format. The following header indicates the format required by WAVESIM:

```
function [e,jacob,s1,tt1]=function[1](stype,i,par,mpar1[2],mpar2[2],s0,tt,alpha)
%
%  FUNCTION
%
%  VERSION 2.5 of 19 April 1991
%  (C) Copyright 1990, 1991  by Norbert H. Doerry
%
%  [e , jacob, s1, tt1] = function(stype,i,par,s0,tt,alpha)
%
%  FUNCTION creates the values and jacobian matrix for a FUNCTION
%
%
%    stype = 1 data points
%          = 2 fourier series
%          = 3 legendre series
%          = 4 polynomial
%          = 5 MATLAB Polynomials
%          = 6 chebyshev series
%
%    i     = [i1 i2 ...] where
%              i1, i2, ... are column vectors of import variables
%
%    par   = [p1 p2 ...] where
%                               p1  = parameter_1
%                               p2  = parameter_2
%                               ...
%
%    mpar1 = matrix parameter parameter_M1
%    mpar2 = matrix paraemter parameter_M2
%
%    s0    = [S0_1 S0_2 ...] where
%                               S0_1  = state_1 value at t0
%                               S0_2  = state_2 value at t0
%                               ...
%
%    tt    = [t0 t1 dtave] where
%                  t0     = initial time of the interval
%                  t1     = final time of the interval
%                 dtave   = averaging increment
%
%    alpha = continuation parameter
%
```

```
%    e     = [e1 e2 ...] where
%              e1, e2, ... are column vectors of export variables
%
%    jacob = Jacobian matrix of e with respect to i
%
%    s1    = [S1_1 S1_2 ...] where
%                               S1_1  = state_1 value at t1
%                               S1_2  = state_2 value at t1
%                               ...
%
%
%    tt1   = [nt1 ntt] where
%                    nt1  = recommended recomputation time this interval
%                    ntt  = recommended ending time next interval
%
```

Note 1: ***function*** is the name of the function defining the device. The MATLAB M-FILE should be called ***function.m***.

Note 2: ***mpar1***, ***mpar2***, etc. are only specified if the device as defined in **device.def** has matrix parameters.

Example MATLAB M-File

```
function [e , jacob , s1, tt1] = resistor(stype,i,par,s0,tt,alpha)
%
%  RESISTOR
%
%  VERSION 1.6 of 25 February 1991
%  (C) Copyright 1990,1991 by Norbert H. Doerry
%
%  [e , jacob, s1, tt1] = resistor(stype,i,par,s0,tt,alpha)
%
%  resistor creates the values and jacobian matrix for a resistor
%
%
%    stype = 1 data points
%          = 2 fourier series
%          = 3 legendre series
%          = 4 polynomial
%
%    i     = [v1 v2] where v1 and v2 are column vectors
%    par   = [R]     where R is the resistance
%    s0    = []
%    tt    = [t0 t1 dt]
%              t0    = initial time of the interval
%              t1    = final time of the interval
%              dt    = averaging time interval
%    alpha = continuation parameter
%
%    e     = [i1 i2] where i1 and i2 are column vectors
%    jacob = Jacobian matrix of e with respect to i
%    s1    = []
%    tt1   = [nt1 ntt] where
%              nt1 recommended recomputation time this interval
%              ntt recommended ending time next interval
%

%    structural jacobian
%
%         D D
%         D D
%
n = size(i);
n(2) = [];
t0 = tt(1);
t1 = tt(2);
dt = tt(3);
%
tt1 = [t1 t0];
%
R = par(1);
%
s1 = [];
%
v1 = i(:,1);
v2 = i(:,2);
%
i1 = (v1 - v2) / R;
i2 = - i1;
%
%
e = [i1 i2];
%
%
ii = eye(n);
%
jacob = [ ii / R -ii / R ; - ii / R ii / R ];
%
```

4.5.2 `device.def` File

A **`DEVICE`** entry must be made in the **`device.def`** file as described in a previous section. Here is an example of the entry made for the resistor:

```
%
DEVICE RESISTOR
%
     TERMINAL 1 POTENTIAL V1 IMPORT
     TERMINAL 1 FLOW      II EXPORT 1
     TERMINAL 2 POTENTIAL V2 IMPORT
     TERMINAL 2 FLOW      I2 EXPORT 1
     PARAMETER R 1e-15
     FUNCTION resistor
     STRUCTURAL JACOBIAN ALL
          DD
          DD
     END
END
```

Note: a device can be have multiple entries in **`device.def`** to reflect different default state initial values and default parameter values. For example, one may desire to create a model of a 1000 ohm resistor:

```
%
DEVICE 1K_RESISTOR
%
     TERMINAL 1 POTENTIAL V1 IMPORT
     TERMINAL 1 FLOW      II EXPORT 1
     TERMINAL 2 POTENTIAL V2 IMPORT
     TERMINAL 2 FLOW      I2 EXPORT 1
     PARAMETER R 1000
     FUNCTION resistor
     STRUCTURAL JACOBIAN ALL
          DD
          DD
     END
END
```

In this manner, one can develop devices which reflect the specific operating parameter of a particular model. A Gas Turbine model for example, could be called **`GT_501K-17`** and have all the parameters prespecified for an Allison 501K-17 Gas Turbine.

4.6 Adding Waveform Types

Adding a new waveform type requires:

1. Assignment of a waveform type code.
2. Writing MATLAB M-File functions for converting to and from the other waveform types.
3. Modification of **`wconvert.m`**
4. Writing MATLAB M-FILE functions for accomplishing the waveform operations required by the devices
5. Modificaiton of ***`wfunction`*`.m`** files

4.6.1 Conversion M-Files

Here is an example of a conversion M-File for converting a Legendre Series into a Polynomial:

```
function [poly, jacob] = leg_poly(leg,n)
%
% [poly, jacob] = leg_poly(leg,n)
%
% Norbert H. Doerry
% Revision 1.1  21 November 1990
%
% LEG_POLY converts a Legendre Series to a Normal Polynomial
%    leg   = vector of Legendre Series Coefficients in ascending
%               order
%    n     = size of polynomial array to create
%
%    poly  = answer
%    jacob = partial derivative of poly wrt leg
%

nl = size(leg);
nl(2) = [];
%
if n <= 0
 n2 = nl;
else
 n2 = n;
end
%
% build the jacobian
%
jacob = zeros(n2,nl);
%
if nl > n2
 nn = n2;
else
 nn = nl;
end
%
for i=1:nn
  jacob(1:i,i) = legendre(i-1);
  end
%
poly = jacob * leg;
%
%
%
```

Note that a jacobian matrix must be specified for each result with respect to each argument. Here is the current version of **`wconvert.m`** which is the normal method for accessing the conversion routines:

```
function [w2,jacob] = wconvert(w1,n,s1,s2)
%
%  WCONVERT
%
%  VERSION 1.3 of 26 March 1991
%  (C) Copyright 1990.1991 by Norbert H. Doerry
%
%  [w2,jacob] = wconvert(w1,n,s1,s2)
%
%
%  WCONVERT converts a waveform of one type to another type and
%       also returns the jacobian of the conversion:
%
%       w     = input waveform
%       n     = number of points in output waveform
%
%       s1    = type of input waveform
%                 = 1 data points
%                 = 2 fourier series
%                 = 3 legendre series
%                 = 4 polynomial
%                 = 5 for matlab polynomial (not implemented yet)
%                 = 6 chebyshev series
%
%       s2    = type of output waveform
%
%


n1 = size(w1);
n1(2) = [];
if n < 1
  n2 = n1;
else
  n2 = n;
end
%
%
if s1 < 1 | s1 > 6
  'Illegal waveform type'
  s1
  return;
end
if s2 < 1 | s2 > 6
  'Illegal waveform type'
  s2
  return;
end
if s1 == 5 | s2 == 5
  'Illegal waveform type'
  s2
  return;
end
%
```

```
%
if s1 == 1
  if s2 == 1
     [w2, jacob] = datadata(w1,n2);
     return;
  elseif s2 == 2
     [w2, jacob] = datafour(w1,n2);
     return;
  elseif s2 == 3
     [w2, jacob] = data_leg(w1,n2);
     return;
  elseif s2 == 4
     [w2, jacob] = datapoly(w1,n2);
     return;
  else
     [w2, jacob] = datacheb(w1,n2);
     return;
  end

%
elseif s1 == 2
  if s2 == 1
     [w2, jacob] = fourdata(w1,n2);
     return;
  elseif s2 == 2
     [w2, jacob] = fourfour(w1,n2);
     return;
  elseif s2 == 3
     [w2, jacob] = four_leg(w1,n2);
     return;
  elseif s2 == 4
     [w2, jacob] = fourpoly(w1,n2);
     return;
  else
     [w3, j3]    = fourpoly(w1,n2);
     [w2, jj]    = polycheb(w3,n2);
     jacob = jj * j3;
     return;
  end

%
elseif s1 == 3
  if s2 == 1
     [w2, jacob] = leg_data(w1,n2);
     return;
  elseif s2 == 2
     [w2, jacob] = leg_four(w1,n2);
     return;
  elseif s2 == 3
     [w2, jacob] = leg_leg(w1,n2);
     return;
  elseif s2 == 4
     [w2, jacob] = leg_poly(w1,n2);
     return;
  else
     [w2, jacob] = leg_cheb(w1,n2);
     return;
  end

%
elseif  s1 == 4
  if s2 == 1
     [w2, jacob] = polydata(w1,n2);
     return;
  elseif s2 == 2
     [w2, jacob] = polyfour(w1,n2);
     return;
  elseif s2 == 3
     [w2, jacob] = poly_leg(w1,n2);
     return;
  elseif s2 == 4
     [w2, jacob] = polypoly(w1,n2);
     return;
  else
     [w2, jacob] = polycheb(w1,n2);
     return;
  end
```

```
else
  if s2 == 1
     [w2,jacob] = chebdata(w1,n2);
     return;
  elseif s2 == 2
     [w3,j3]    = chebpoly(w1,0);
     [w2,jj]    = polyfour(w3,n2);
     jacob = jj * j3;
     return;
  elseif s2 == 3
     [w2,jacob] = cheb_leg(w1,n2);
     return;
  elseif s2 == 4
     [w2,jacob] = chebpoly(w1,n2);
     return;
  else
     [w2,jacob] = leg_leg(w1,n2);
     return;
   end
end
```

4.6.2 Waveform Functions

Waveform functions are defined in a similar manner to the conversion files. Here is an example of a MATLAB M-File for integrating a polynomial:

```
function [p2 , jacob] = poly_int(p1,n,c)
%
%
%  [p2 , jacob] = poly_int(p1,n,c)
%
% Norbert H. Doerry
% Revision 1.0 of 6 December 1990
%
%    p1 = input polynomial
%    n  = number of points in p2
%    c  = integration constant
%
%    p2 = integeral of p1
%    jacob = partial of p2 with respect to p1
%
%
%    POLY_INT integrates a given polynomial and converts the
%    result to another polynomial of size n
%

%
n1 = size(p1);
n1(2) = [];
%
% Error checking
%
if n < 2
  n2 = n1;
else
  n2 = n;
end
%
if n1 < 1
  p2 = [];
  return;
  end
%
% calculate the integration matrix
%
j1 = zeros(n1+1,n1);
%
for i = 1 : n1
  j1(i+1,i) = 1 / i;
end
%
% xx is (-1)^(n-1)
%
xx = ones(1,n2);
for i = 2 : n2
  xx(1,i) = - xx(1,i-1);
end

%
%  calculate the indefinite integral of the polynomial
%
pi1 = j1 * p1;
%
% convert to a polynomial of the right size
%
[p2,j2] = polypoly(pi1,n2);
j3 = j2 * j1;
%
% convert to a definite integral by adding the constant of
%    integration and subtracting the value of the indefinite
%    integral at x = -1
%
jacob = j3 - [xx * j3 ; zeros(n2-1,n1)];
p2(1,1) = p2(1,1) - xx * j3 * p1 + c(1,1);
```

Normally, a user would call **w_int** to integrate a waveform:

```
function [w2 , jacob] = w_int(w1,n,s1,c)
%
% W_INT
%
% Version 1.2 of 19 April 1991
% (C) Copyright 1991 by Norbert H. Doerry
%
%  [w2, jacob] = w_int(w1,n,s1,c)
%
%  W_INT integrates a waveform and returns the result into a waveform of
%    the same type but of possible different length
%
%    w1  = input waveform
%     n  = number of points in output waveform
%    s1  = type of waveform
%        = 1 data points
%        = 2 fourier series
%        = 3 legendre series
%        = 4 polynomial
%        = 5 for matlab polynomial
%        = 6 for chebyshev series
%
%     c  = constant of integration
%
%    w2  =  waveform which is integral of w1
%    jacob = jacobian of w2 with respect to w1

n1 = size(w1);
n1(2) = [];
if n < 1
  n2 = n1;
else
  n2 = n;
end
%
% check for illegal waveform type
%
if s1 < 1 | s1 > 6
  'Illegal waveform type'
  s1
  return;
  end

if s1 == 1
  [w2 , jacob] = data_int(w1,n2,c);
%
elseif s1 == 2
  [w2 , jacob] = four_int(w1,n2,c);
%
elseif s1 == 3
  [w3, j3] = leg_poly(w1,n1);
  [w4, j4] = poly_int(w3,n2,c);
  [w2, j2] = poly_leg(w4,n2);
  jacob = j2 * j4 * j3;
%
elseif s1 == 4
  [w2 , jacob] = poly_int(w1,n2,c);
%
elseif s1 == 5
  [w3, j3] = mplypoly(w1,n1);
  [w4, j4] = poly_int(w3,n2,c);
  [w2, j2] = polymply(w4,n2);
  jacob = j2 * j4 * j3;
%
elseif s1 == 6
  [w3, j3] = chebpoly(w1,n1);
  [w4, j4] = poly_int(w3,n2,c);
  [w2, j2] = polycheb(w4,n2);
  jacob = j2 * j4 * j3;
%
else
  'error'
end
```

Notice how the waveform conversion routines are used to implement waveform operations which have not yet been defined for a given waveform type.

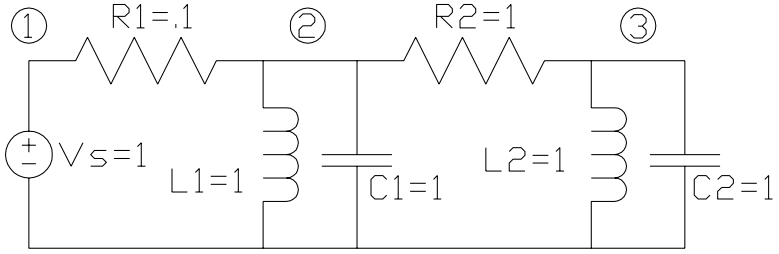
Chapter 5 Simulation Results

As a demonstration of the capabilities of WAVESIM, the results of three simulations are presented here. While these simulations are relatively simple, they include the important features of more difficult simulations, yet are not so complicated as to be unverifiable. The first simulation of a simple electical circuit containing only linear devices verifies the ability of WAVESIM to construct a viable system and limit truncation error by controlling the waveform interval and number of coefficients. The second simulation increases the complexity by including a nonlinear device and provides a good test of the Newton-Raphson solver. The third and final simulation demonstrates the use of a continuation parameter to improve the region of convergence of the simulation.

5.1 Linear Electrical Circuit

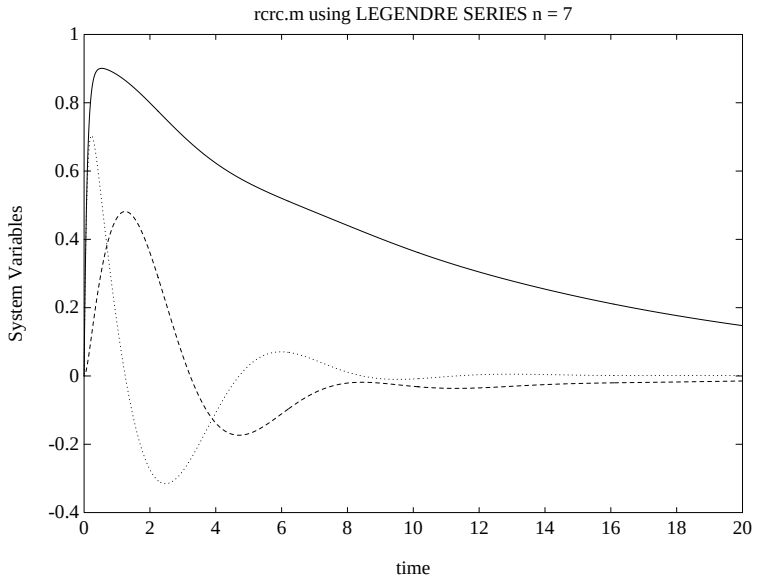
To demonstrate WAVESIM's ability to solve linear circuit problems, the circuit shown in figure 5.1-1 was simulated. Initially, both capacitors have zero charge and the inductor currents are zero as well. The transients of the capacitor voltages and current are shown in figure 5.1-2. The simulation was conducted using Legendre Series with the 20 second simulation time split up among 23 intervals. Eleven intervals were rejected due to excessive truncation error.

Figure 5.1-1: Linear Electrical Circuit: Schematic



The results shown in figure 5.1-2 are identical (to working precision) to an analytic solution of the circuit.

Figure 5.1-2: Linear Electrical Circuit: Simulation Results



The input file specifying the system is given by:

Input File for Linear Electrical Circuit

```
%
% rcrc.in
%
device VDC_SOURCE Vs
     TERMINAL   1    1
     TERMINAL   2    0
     PARAMETER VS    1.0
     END

%
device RESISTOR   R1
     TERMINAL   1   1
     TERMINAL   2   2
     PARAMETER  R   0.1
     END

%
device RESISTOR   R2
     TERMINAL   1   2
     TERMINAL   2   3
     PARAMETER  R   1.0
     END

%
device INDUCTOR   L1
     TERMINAL   1    2
     TERMINAL   2    0
     PARAMETER  L    1.0
     END

%
device INDUCTOR   L2
     TERMINAL   1     3
     TERMINAL   2     0
     PARAMETER  L   1.0
     END

%
device CAPACITOR  C1
     TERMINAL   1     2
     TERMINAL   2     0
     PARAMETER  C   1.0
     END

%
device CAPACITOR  C2
     TERMINAL   1     3
     TERMINAL   2     0
     PARAMETER  C   1.0
     END
%

%
default
     Gmin 0
     Rmin 0
     rimport NO
     check both
```

```
     error eqn kcl   5e-3
     error eqn pot   5e-3
     error var node  5e-3
     error var flow  5e-3
     error mult kcl  10.0
     error mult pot  10.0
     error mult node 10.0
     error mult flow 10.0
     max count 10
     max int count 6
     alpha init 1.0
     alpha inc init .25
     alpha inc min  .05
     alpha inc max  .50
     diverge start 3
     diverge max cnt 2
     diverge error mult 10.0
     waveform content max .001
     waveform content nbr 2
     range max .005
     scale potential 1.0
     scale flow      1.0
     stype 3
     nbr coef 7
     nbr coef min 6
     nbr coef max 14
     nbr data 20
        END
%

%
time
     dt min .01
     dt max 2.0
     dt opt .250
     dt init .5
     dt ave 0.0
     start   0.0
     finish 20.0
     END

%
plot
     potential R1 2
     node 3
     flow C2 1
     END
```

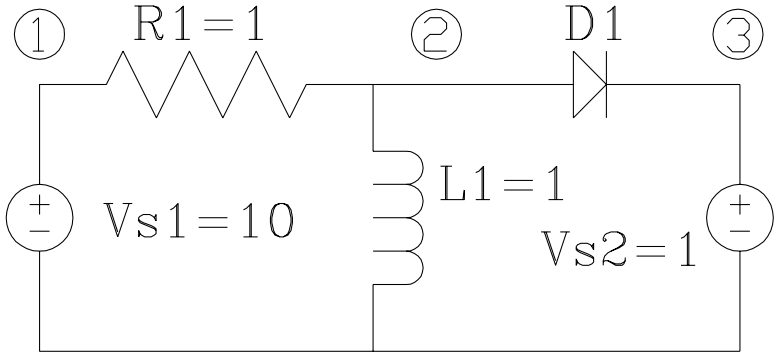
Figure 5.1-3 shows the time increment and number of coefficients used for each of the intervals. Once the transients start to decay, the number of coefficients are decreased to the minimum allowed.

Figure 5.1-3 Truncation Error Control		
Interval	Ending Time (sec)	Number of Coefficients
1	0.25	7
2	0.50	7
3	0.75	7
4	1.25	7
5	2.25	7
6	3.25	6
7	4.25	6
8	4.75	6
9	5.25	6
10	6.25	6
11	7.25	6
12	8.25	6
13	8.75	6
14	9.25	6
15	10.25	6
16	11.25	6
17	12.25	6
18	14.25	6
19	15.25	6
20	16.25	6
21	17.25	6
22	18.25	6
23	20.00	6

5.2 Nonlinear Electrical Circuit

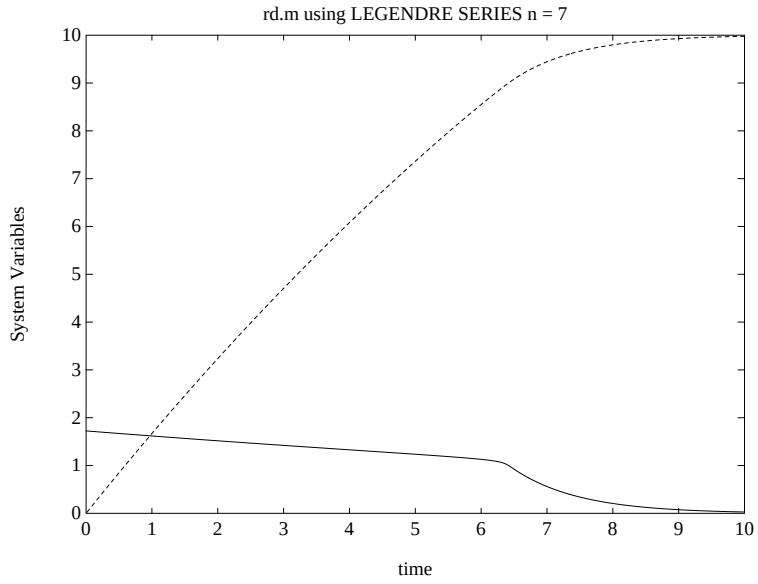
To demonstrate WAVESIM's ability to solve nonlinear circuit problems, the circuit shown in figure 5.2-1 was simulated. Initially, the inductor current is zero. As the inductor current builds up, its voltage is clamped by the diode to one diode drop above 1 volt and its current ramps up almost linearly. When the inductor voltage falls far enough to turn the diode off, the current and voltage both show a normal exponential transient behavior. Figure 5.2-2 shows the inductor voltage and current as a function of time.

Figure 5.2-1 Nonlinear Electrical Circuit: Schematic



The results shown in figure 5.2-2 were calculated using Legendre Series over seven time intervals. Five additional intervals were rejected due to excessive truncation error. These results match closely an analytic solution to the circuit.

Figure 5.2-2 Nonlinear Electrical Circuit: Simulation Results



The input file specifying the system is given by:

Input File for Nonlinear Electrical Circuit

```
%
% rd.in
%
device VDC_SOURCE Vs1
     TERMINAL   1  1
     TERMINAL   2  0
     PARAMETER  VS 10.0
        END
%

device RESISTOR   R1
     TERMINAL   1    1
     TERMINAL   2    2
     PARAMETER  R  1.0
     END
%

device INDUCTOR   L1
     TERMINAL   1    2
     TERMINAL   2    0
     PARAMETER  L  1.0
     END
%

device DIODE1      D1
     TERMINAL   1    2
     TERMINAL   2    3
     END
%
```

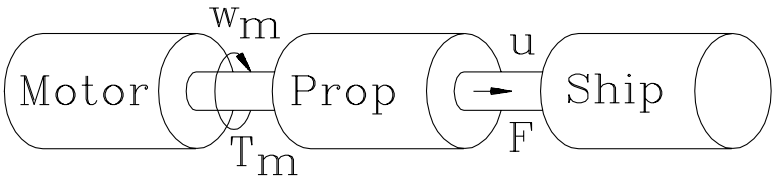
```
device VDC_SOURCE Vs2
     TERMINAL   1    3
     TERMINAL   2    0
     PARAMETER  VS 1.0
     END

%
default
        Gmin 0
        Rmin 0
        rimport NO
        check both
        error eqn kcl   5e-3
        error eqn pot   5e-3
        error var node  5e-3
        error var flow  5e-3
        error mult kcl  10.0
        error mult pot  10.0
        error mult node 10.0
        error mult flow 10.0
        max count 10
        max int count 6
        alpha init 1.0
        alpha inc init .25
        alpha inc min  .05
        alpha inc max  .50
        diverge start 3
        diverge max cnt 2
        diverge error mult 10.0
        waveform content max .001
        waveform content nbr 2
        range max .005
        scale potential 1.0
        scale flow      1.0
        wtype 3
        nbr coef 7
        nbr coef min 7
        nbr coef max 10
        nbr data 20
        END
%

time
        dt min .01
        dt max 5.0
        dt opt .250
        dt init 5
        dt ave  .01
        start   0.0
        finish 10.0
        END

%
plot
     potential R1 2
     flow L1 1
     END
```

Figure 5.2-3 Truncation Error Control		
Interval	Ending Time (sec)	Number of Coefficients
1	5.00	7
2	6.25	7
3	6.4063	7
4	6.7188	7
5	7.3438	7
6	8.5938	7
7	10.0	7

5.3 Nonlinear Mechanical System

To demonstrate the ability of WAVESIM to use continuation parameters in simulating nonlinear mechanical systems, a mechanical power train was modelled. The acceleration characteristic of a ship was determined for a propeller rotating at a constant speed. Figure 5.3-1 shows a schematic diagram of the system.

Figure 5.3-1 Nonlinear Mechanical System: Schematic



The propeller model is described in Appendix F-8 while the ship dynamics model is described in Appendix F-9. Figure 5.3-2 shows the parametric curves used for $\boldsymbol{C_T()}$ and $\boldsymbol{C_Q()}$. This data is for a three bladed propeller with an expanded area ratio of .5 and an H/D ratio of .6 [81].

Figure 5.3-3 shows the Residual drag coefficient used in the ship dynamics model. This data is from the Taylor Standard Series for a hull with beam to draft ratio of 3.0, Prismatic Coefficient (C_p) of .68, and Volumetric coefficient of 0.002. The Frictional Drag Coeffient was calculated using the standard ITTC Line:

$$C_f(R_e) = \frac{.075}{(\log_{10}(R_e) - 2)^2}$$

Figure 5.3-2 Nonlinear Mechanical System: Propeller Characteristics

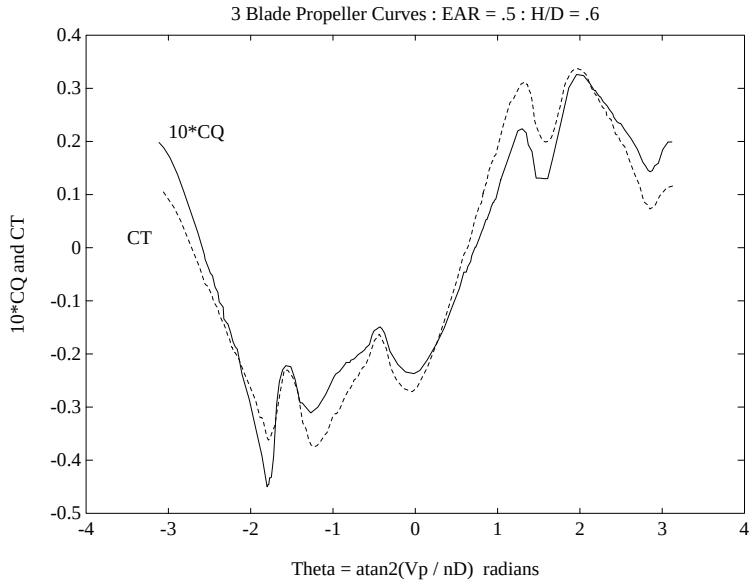
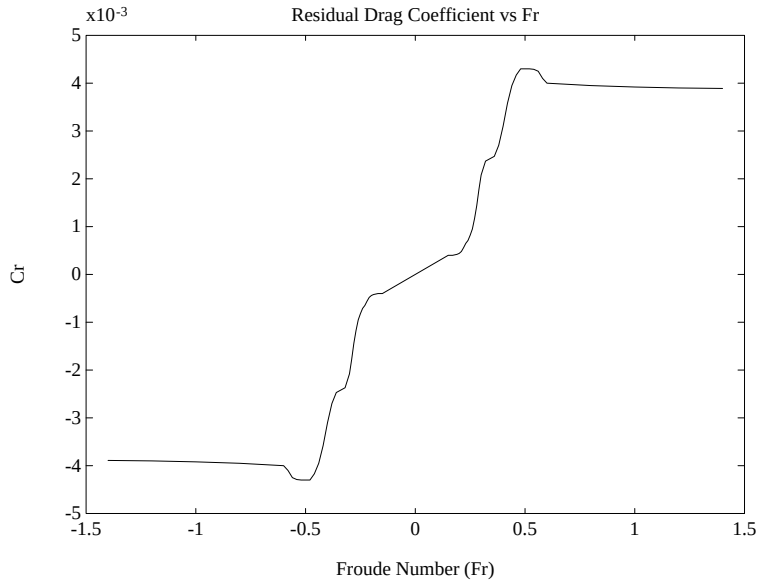


Figure 5.3-3 Nonlinear Mechanical System: Drag Coefficient



The input file specifying the system is given by:

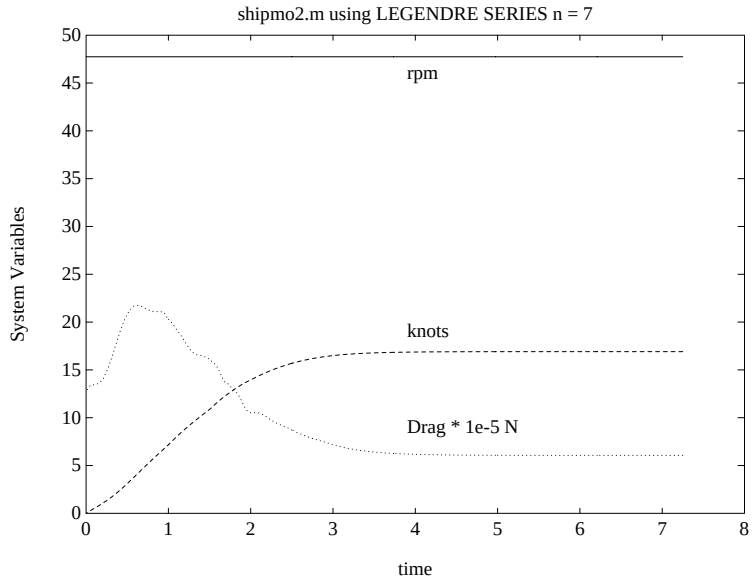
Input File for Nonlinear Mechanical System

```
%
% shipmo1.in
%
% Model of a prime mover attached to prop going to the sea
%
device NODE_REF GT1
  TERMINAL 1 1
  PARAMETER Vref 5.0
  END
%
device PROP1 prop
  TERMINAL SHAFT 1
  TERMINAL WATER 2
  PARAMETER D  10.0
  PARAMETER w  0.0
end

%
% parameters are rough ones from Sue B Gail
%
device SHIPDYN1 ship
  TERMINAL WATER 2
  PARAMETER L 100
  PARAMETER A 3300
  PARAMETER M 15000000
  PARAMETER Madd 1.05
  PARAMETER Ca 0.0004
  STATE Us 0
end

%
Node 1
  SCALE POTENTIAL 1
  SCALE FLOW      1e-4
  NAME Propeller_Shaft
  Gmin 1
  END
Node 2
  SCALE POTENTIAL 1
  SCALE FLOW      1e-4
  NAME Hydrodynamic_U_force
  END

%
default
        Gmin 0
        Rmin 0
        rimport NO
        check eqn
        error eqn kcl   1e-2
        error eqn pot   1e-2
        error var node  1e-2
        error var flow  1e-2
        error mult kcl  10.0
        error mult pot  10.0
        error mult node 10.0
        error mult flow 10.0
        max count 10
```

```
        max int count 6
        alpha init 0.5
        alpha inc init .25
        alpha inc min  .05
        alpha inc max  .50
        diverge start 3
        diverge max cnt 2
        diverge error mult 10.0
        waveform content max .005
        waveform content nbr 2
        range max .005
        scale potential 1.0
        scale flow      1.0
        stype 3
        nbr coef 7
        nbr coef min 6
        nbr coef max 10
        nbr data 20
        END

%
time
        dt min .25
        dt max 5.0
        dt opt 1.0
        dt init 2.0
        dt ave 0.0
        start   0.0
        finish  7.25
        END

%
plot
% node 1 converted to RPM
% node 2 converted to Knots (more or less)
%
        node  1  9.5492966
        node  2  1.8
        flow ship WATER 1e-5
        flow prop SHAFT 1e-5
        END
```

The results of the simulation using Legendre Series are shown in figure 5.3-4. The simulation was broken into 24 intervals shown in figure 5.3-5. An additional 25 intervals were rejected due to excessive truncation error. For each iteration, the continuation parameter was initially set to 0.5. This value helped assure the initial value of 0 was within the convergence region of the nonlinear blocks. While 0.5 was suitable for most iterations, several required the continuation parameter be decremented further to achieve convergence.

Figure 5.3-4 Nonlinear Mechanical System: Simulation Results



As expected, the force on the propeller is greatest during the acceleration of the ship. As the ship accelerates, the increased forward velocity on the ship results in smaller torques and forces. In reality it is doubtful the motor would be capable of maintaining a constant RPM during the acceleration phase.

Figure 5.3-5 Truncation Error Control		
Interval	Ending Time (sec)	Number of Coefficients
1	1	7
2	1.5	8
3	1.75	10
4	2.25	10
5	2.5	10
6	2.75	10
7	3.0	10
8	3.25	10
9	3.5	10
10	3.75	10
11	4.0	10
12	4.25	10
13	4.5	10
14	4.75	10
15	5.0	10
16	5.25	10
17	5.5	10
18	5.75	10
19	6.0	10
20	6.25	10
21	6.5	10
22	6.75	10
23	7	10
24	7.25	10

Chapter 6 Conclusions

In its present form, WAVESIM is ideally suited for testing numerical algorithms. While it is capable of simulating large systems, the interpretive nature of MATLAB is not numerically efficient enough for serious simulations. Careful development of a simulation environment based on the techniques explored in WAVESIM should prove effective in solving tightly coupled multirate systems of lumped parameter models.

The simulation environment described in this thesis should be considered a framework for future developments. Many improvements are possible and desirable. In particular, the following areas need further attention:

Truncation Error Control

The present method for controlling truncation error is heuristic and should be examined for improvement. Truncation Error propagation should be examined and given a theoretical basis.

Discontinuity Time Prediction

The accuracy of the methods used in WAVESIM depend partly on the ability to predict discontinuities and force them to occur on time interval boundaries. The methods used in current models are crude and should be replaced with more robust and accurate methods.

Stability Analysis

WAVESIM presently does not perform any stability analysis. Since WAVESIM abandons the standard state space representation of the system, determining the eigenstructure of the system is not easy. A stability measure based on the characteristics of individual devices would fit well with the structure of WAVESIM and would be quite useful in the design of distributed controls.

Smoothing Operation

The smoothing operator for removing the effects of high frequency discontinuities needs to be examined to improve its efficiency. How long to make the smoothing interval is a question which has not been satisfactorily answered.

Partitioning and Relaxation

The approach WAVESIM uses for developing the set of system equations is ideally suited for use with relaxation methods if the system is weakly coupled. An extension to the structural Jacobian to include matrix norms would greatly simply the task of partitioning the system into a set of weakly coupled blocks which internally are strongly coupled. Each individual block would be solved using Newton-Raphson with the system solved using a relaxation technique. Unfortunately, the process of constructing a system matrix of norms from device matrix norms is presently not possible because arithmetic operators for matrix norms have not been identified.

Overall, WAVESIM has been very successful in developing the algorithms for building systems in terms of device functions, treating waveforms as an abstract data type, and employing the structural Jacobian matrix to reduce the system into a sequence of smaller blocks. Much work remains, but the foundation of a waveform based simulator capable of handling tightly coupled multirate simulation problems is contained within WAVESIM.

References

Chapter One: Introduction

[1] Antognetti, P., D. O. Pederson, and H. de Man, **Computer Design Aids for VLSI Circuits**, Martinus Nijhoff Publishers, 1986.

[2] Carlsen, K., E. H. Lenfest, J. J. LaForest, **MANTRAP Machine and Network Transients Program**, 1976 Power Industry Computer Applications Conference, pp. 144-150.

[3] Casey, John P., **AC Electric Drive Machinery Design**, Presented at the 1990 Chesapeake Marine Engineering Symposium, The Society of naval Architects and Marine Engineers, Arlington, VA, March 14, 1990.

[4] Hatchtel, G. D., et. al., **The Sparse Tableau Approach to Network Analysis and Design**, IEEE Transactions on Circuit Theory, Vol. CT-18, Jan. 1971, pp. 101-108.

[5] Hindmarsh, Alan C., **LSODE and LSODI, Two new Intitial Value Ordinary Differential Equation Solvers**, Lawrence Livermore Laboratory.

[6] Ho, C.W., et. al., **The Modified Nodal Approach to Network Analysis**, IEEE Transactions on Circuits and Systems, vol. CT-18, Jan 1971, pp. 101-108.

[7] Hultgren, Kenneth J, **VSCF Cycloconverter Power Equipment A Versatile Technology for Wide Range PDSS**. Presented at the 1990 Destroyer, Cruiser & Firgate Technology Symposium, The American Society of Naval Engineers, Biloxi, MS, September 27, 1990.

[8] Ilic-Spong, Marija and John Zaborsky, **A Different Approach to Load Flow**, IEEE Transactions on PAS, Jan 1982, pp. 168-179.

[9] Larsen, E. V. and W. W. Price, **MANSTAB/POSSIM Power System Dynamic Analysis Programs - A New Approach Combining Nonlinear Simulation and Linearized State-Space/Frequency Domain Capabilities**, 1977 Power Industry Computer Applications Conference, pp. 350-357.

[10] Luini, James F., Richard P. Shulz, and Anne E. Turner, **A Digital Computer Program for Analyzing Long Term Dynamic Response of Power Systems**, EPRI Research Project 90-7.

[11] Meyer, W. Scott, **Machine Translation of an Electromagnetic Transients Program (EMTP) Among Different Digital Computer Systems**, 1977 Power Industry Computer Applications Conference, pp. 272-277.

[12] Mitchell and Gauthier Associates, **Advanced Continuous Simulation Language (ACSL) User Guide / Reference Manual**, Concord, MA 1975.

[13] Prasad, N. R., and R. D. Dunlop, **Three Phase Simulation of the Dynamic Interaction Between Synchronous Generators and Power Systems Using the Continuous Systems Modelling Program (CSMP III)**, IEEE Transactions of Power Apparatus and Systems, Vol PAS-94 No. 3, May/June 1975, pp. 1042-1049.

[14] Stevenson, William, **Elements of Power System Analysis**, McGraw-Hill Book Company, 1962.

[15] Weeks, W.T., et al., **Algorithms for ASTAP - A Network Analysis Program**, IEEE Trans. on CT., Nov. 1973, pp. 628-634.

[16] White, J., A. S. Vincentelli, F. Odeh, and A. Ruehli, **Waveform Relaxation: Theory and Practice**, Transactions of the Society for Computer Simulation, Volume 2, Number 1, 1985, pp. 95-132.

Chapter Two: Shipboard Electric Systems

[17] Department of Defense, **Interface Standard for Shipboard Systems, Section 300A, Electric Power, Alternating Current (Metric)**, MIL-STD-1399(NAVY) 13 October 1987.

[18] Departemnt of the Navy, **General Specifications for Ships of the United States Navy, Section 300, General Requirements for Electric Plant**, Naval Sea Systems Command, 1987.

[19] Department of the Navy, **General Specifications for Ships of the United States Navy, Section 320, General Requirements for Electric Power Distribution Systems,** Naval Sea Systems Command, 1987.

[20] Ballard, Michael A, **Impacts of Electric Propulsion Systems on Submarine Design**, Naval Engineer and Electrical Engineering and Computer Science Master's thesis, MIT, 1989

[21] Davis, James Clinton, **A Comparative Study of Various Electric Propulsion Systems and Their Impact on a Nominal Ship Design**, Naval Engineer and Electrical Engineering and Computer Science Master's thesis, MIT, 1987.

[22] Doerry, Norbert H., **Shipboard Electrical Distribution Systems**, Term paper for MIT course 6.683, 22 May 1989.

[23] Graham, C., R. Hamly, J. Reed, **Simplified Math Model for the Design of Naval Frigates**, MIT Department of Ocean Engineering, September 1986.

[24] Jolliff, James V., Dr., and Dr. David L. Greene, **Advanced Integrated Electric Propulsion A Reality of the Eighties**, Naval Engineers Journal, April 1982, pp. 232-252.

Chapter Three: Framework

[25] Abelson, Harold, **The Bifurcation Interpreter: A step towards the automatic analysis of dynamical systems**, A.I. Memo 1174, September 1989, MIT.

[26] Abelson, Harold, Micahel Eisenberg, Mathew Halfant, Jacob Katzenelson, Elisha Sacks, Gerald Jay Sussman, Jack Wisdom, and Ken Yip, **Intelligence in Scientific Computing**, A. I. Memo 1094, November 1988, MIT.

[27] Akimoto, Yoshiakira, Hideo Tanaka, Hiromi Ogi, Hisao Taoka and Toshiaki Sakaguchi, **Distributed Simulator for Power System Analysis using a Hypercube Computer**, Proceedings of the Tenth Power Systems Computation Conference, Graz, Austria, 19-24 August 1990. pp. 742-749.

[28] Braham, Rafik and James O. Hamblen, **Simulation of Large Integrated Circuits**, Transactions of teh Society for Computer Simulation, Volume 5, Number 4, 1988, pp. 243-263.

[29] Braess, D., and E. Grebe, **A Numerical Analysis of Load-Flow Calculation Methods**, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-100, No. 7, July 1981.

[30] Brennan, K. E, S. L. Campbell and L. R. Petzold, **Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations**, Elsevier Science Publishing Co., Inc., New York, 1989.

[31] Calovic, M. A. and V. C. Strezoski, **Calculation of Steady-State Load Flows Incorporating System Control Effects and Consumer Self-Regulation CHaracteristics**, Electrical Power & Energy Systems, Vol 3, No 2, April 1981, pp. 65-74.

[32] Chao, Kwong-Shu, Dan-Kai Liu, and Ching-Tsai Pan, **A Systematic Search Method for Obtaining Multiple Solutions of Simultaneous Nonlinear Equations**, IEEE Transactions on Circuits and Systems, Vol CAS-22, No 9. September 1975, pp. 748-753

[33] Chua, Leon O., and Niantsu N. Wang, **On the Application of Degree Theory to the Analysis of Resistive Nonlinear Networks**, Circuit Theory and Applications, Vol 5, 1977, pp. 35-68.

[34] Chua, L. O., and A. Ushida, **A Switching-Parameter Algorithm for Finding Multiple Solutions of Nonlinear Resistive Circuits**, Circuit Theory and Applications, Vol 4, 1976, pp. 215-239.

[35] Concordia, C. and S. Ihara, **Load Representation in Power Systems Stability Studies**, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-101, No. 4, April 1982.

[36] Crandall, Stephen H., **Engineering Analysis, A Survey of Numerical Procedures**, Robert E. Krieger Publishing Company, Malabar, Florida, 1986 Reprint Edition

[37] Crow, M. L., M. D. Ilic, and J. K. White, **Convergence Properties of the Waveform Relaxation Method as Applied to Electric Power Systems**, ISCAS 89.

[38] Crow, M. L., **Systems of Differential/Algebraic Equations with Applications to Power System Transient Stability Analysis**, Doctoral Thesis, The University of Illinois at Urbana-Champaign, August 1989.

[39] Eisenberg, Michael, **Descriptive Simulation: Combining Symbolic and Numerical Methods in the Analysis of Chemical Reaction Mechanisms**, A.I Memo No. 1171, MIT, September 1989.

[40] Elmquist, Hilding, **Manipulation of Continuous Models Based on Equations to Assignment Statements**, Simulation of Systems 1979, L. Dekker, G. Savastano, and G. C. Vansteenkiste (eds), North-Holland Publishing Company, 1980.

[41] Fröberg, Carl-Eric, **Numerical Mathematics, Theory and Computer Applications**, The Benjamin / Cummings Publishing Company, Inc. 1985.

[42] Gear, C. W., **Efficient Step Size Control for Output and Discontinuities**, Transactions of the Society for Computer Simulation, Volume 1, Number 1, 1984, pp. 27-31.

[43] Gear, C. W., and L. R. Petzold, **ODE Methods for the Solution of Differential/Algebraic Systems**, SIAM J. Numer.Anal., Vol 21, No 4, August 1984, pp. 716-728.

[44] Gear, C. W., **Simultaneous Numerical Solution of Differential-Algebraic Equations**, IEEE Transactions on Circuit Theory, Vol CT-18, No 1, January 1971.

[45] Golub, Gene H. and Charles F. Van Loan, **Matrix Computations**, The Johns Hopkins University Press, Baltimore, MD, 1985.

[46] Hildebrand, F. B., **Introduction to Numerical Analysis**, Dover Publications, Inc., New York, 1987 Edition.

[47] Hirobe, H. Doi, M. Goto, Y. Kokai, S. Yokokawa, and T. Suzuki, **Development of a Large-Scale Analytical Simulator for Studying Power Systems**, Proceedings of the Tenth Power Systems Computation Conference, Graz, Austria, 19-24 August 1990, pp. 750-757.

[48] Howe, R. M., X. A Ye and B. H. Li, **An Improved Method for Simulation of Dynamic Systems with Discontinuous Nonlinearities**, Transactions of the Society for Computer Simulation, Volume 1, Number 1, 1984, pp. 33-47.

[49] Huang, G., and W. Ongsakul, **A New Relaxation Algorithm for Power Flow Analysis**, IEEE CH2869-8/90/0000-1276, 1990.

[50] Ilic-Spong, Marija, I. Norman Katz, Huizhu Dai, and John Zaborsky, **Block Diagonal Dominance for Systems of Nonlinear Equations with Application to Load Flow Calculations in Power Systems**, Mathematical Modelling, Vol 5, 1984, pp. 275-297.

[51] Ilic-Spong, M., M. L. Crow, and M. A. Pai, **Transient Stability Simulation by Waveform Relaxation Methods**, IEEE Transactions on Power Systems, Vol. PWRS-2, No. 4, November 1987.

[52] Jennings, Alan, **Matrix Computation for Engineers and Scientists**, John Wiley and Sons, 1985.

[53] Karnopp, Dean, **General Method for Including Rapidly Switched Devices in Dynamics System Simulation Models**, Transactions of the Society for Computer Simulation, Volume 2, Number 1, 1985, pp. 155-168.

[54] Klos, A., and J. Bialek, **Homeostasis as a Tool for Power System Simulation,** Proceedings of the Tenth Power Systems Computation Conference, Graz, Austria, 19-24 August 1990, pp. 701-708.

[55] Kundert, Kenneth S., **Sparse Matrix Techniques**, Circuit Analysis, Simulation and Design, A. E. Ruehli, (editor), Elsevier Science Publishers B. V. (North-Holland), 1986.

[56] Mattson, Sven Erik, **On modelling and differential/algebraic systems**, Simulation, January 1989, pp 24-32.

[57] Neyer, Andreas, Felix F. Wu, and Karl Imhof, **Object-Oriented Programming for Flexible Software: Example of a Load Flow**, IEEE 88 SM 733-8, presented at the IEEE/PES 1988 Summer Meeting, Portland, Oregon, July 24 - 29, 1988.

[58] Ortega, J. M. and W. C. Rheinboldt, **Iterative Solution of Nonlinear Equations in Several Variables**, Academic Press, Inc.

[59] Palusinski, Olgierd, **Simulation of Dynamics Systems Using Multirate Integration Techniques**, Transactions of the Society for Computer Simulation, Volume 2, Number 4, 1986, pp. 257-273.

[60] Palusinski, Olgierd A., and Tomas K. Simacek, **Continuous Expansion in Integration of Partitioned Dynamic Systems**, Transactions of the Society for Computer Simulation, Volume 2, Number 1, 1985, pp. 11-25.

[61] Petzold, Linda, **Differential/Algebraic Equations are not ODE's**, SIAM J. SCI. STAT. COMPUT., Vol 3, No 3, September 1982, pp. 367-385.

[62] Pottle, Christopher, **Comprehensive Active Network Analysis by Digital Computer - A State-Space Approach**, Reprinted from Proc. Third Ann. Allerton Conf. Circuits and Systems Thoery, 659-668, in Computer-Aided Circuit Design: Simulation and Optimization, S. W. Direktor, editor, Dowden, Hutchinson & Ross, Inc., Stroudsburg, PA, 1973, pp. 29-38.

[63] Pottle, Christopher, **A "Textbook" Computerized State-Space Network Analysis Algorithm**, Reprinted from IEEE Trans Circuit Theory, CT-16 566-568 (1969) in Computer-Aided Circuit Design: Simulation and Optimization, S. W. Direktor, editor, Dowden, Hutchinson & Ross, Inc., Stroudsburg, PA, 1973, pp. 39-41.

[64] Rabinowitz, Philip, **Numerical Methods for Nonlinear Algebraic Equations**, Gordon and Breach Science Publishers, London, 1970.

[65] Sangiovanni-Vincentelli, A. L., **Circuit Simulation**, Computer Design Aids for VLSI Circuits, P. Antognetti, D. O. Pederson, and H. De Man, (eds), Martinus Nijhoff Publishers, 1986.

[66] Sasson, Albert M., Carlos Trevino, and Florencio Aboytes, **Improved Newton's Load Flow Through a Minimization Technique**, Paper 71 TP 18-PWR presented at IEEE Winter Power Meeting, New York, January 31 - February 5, 1971.

[67] Sincovec, Richard F., Albert M. Erisman, Elizabeth L. Yip, and Michael A. Epton, **Analysis of Descriptor Systems Using Numerical Algorithms**, IEEE Transactions on Automatic Control, Vol AC-26, No 1, February 1981.

[68] Singh, Sudarshan Pal and Ruey-Wen Liu, **Existence of State Equation Representation of Linear Large-Scale Dynamical Systems**, IEEE Transactions on Circuit Theory, Vol CT-20, No 3, May 1973.

[69] Somuah, Clement B., and Syed M. Islam, **Multi-Time Step Solution of Stiff Differential Systems: Application to Power System**, International Journal of Modelling and Simulation, Vol. 9, No. 2, 1989, pp. 53-58.

[70] Tellegen, B. D. H., **A General Network Theorem, with Applications**, Philip Research Laboratory Report 259-269, 1952.

[71] Thomas, Robert J., Robert D. Barnard, and Jerome Meisel, **The Generation of Quasi Steady-State Load-Flow Trajectories and Multiple Singular Point Solutions**, Paper 71 TP 111-PWR presented at IEEE Winter Power Meeting, New York, January 31 - February 5, 1971.

[72] Trajkovic, Ljiljana, Robert C. Melville, and San-Chin Fang, **Passivity and No-Gain Properties Establish Global Convergence of a Homotopy Method for DC Operating Points**, IEEE CH2868-8/90/0000-0914, 1990.

[73] Vandenberghe, Lieven and Joos Vandewalle, **A Globally Convergent Algorithm for Solving a Broad Class of Nonlinear Resistive Circuits**, IEEE CH2868-8/90/0000-0403, 1990.

[74] Vandenberghe, L. and J. Vandewalle, **Variable Dimension Algorithms for Solving Nonlinear Resistive Circuits**, European Conference on Circuit Theory and Design 5-8 September 1989. pp. 385-389.

[75] Vidyasagar, Mathukumalli, **On the Well-Posedness of Large-Scale Interconnected Systems**, IEEE Transactions on Automatic Control, Vol AC-25, No 3, June 1980.

[76] Wamser, Robert J., and Ilya W. Slutsker, **Power Flow Solution by the Newton-Raphson Method in Transient Stability Studies**, IEEE Transactions on Power Apparatus and Systems, Vol PAS-103, No. 8, August 1984, pp. 2299-2306.

[77] Wasynczuk, O. and R. A. DeCarlo, **The Component Connection Model and Structure Preserving Model Order Reduction**, International Federaton of Automatic Control, Vol 17, No 4, 1981, pp. 619-626.

[78] White, Jacob K. and Alberto Sangiovanni-Vincentelli, **Relaxation Techniques for the Simulation of VLSI Circuits**, Kluwer Academic Publishers, Boston, 1987.

Chapter Four: WAVESIM

[79] Kernighan, Brian W., and Dennis M. Ritchie, **The C Programming Language**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[80] Moler, Cleve, John Little and Steve Bangert, **PC-MATLAB for MS-DOS Personal Computer**, The MathWorks, Inc., Sherborn, MA 01770. Version 3.2-PC, June 8, 1987.

Appendices:

[81] Baker, D. W., and C. L. Patterson, **Representation of Propeller Thrust and Torque Characteristics for Simulations, Appendix C Data for 18 Propellers**, Naval Ship Research and Development Center, Washington, D.C., document MEL 202/67, March 1970.

[82] Basler Electric Company, **Instruction Manual for Voltage Regulator Models SR4A & SR8A**, Puplication 9 0177 00 990, Revision B of July 1986.

[83] Bose, B. K., **Power Electronics and AC Drives**, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[84] Dalton, R. C., **Turbine Generator Simulations for DD-692 Class 450 KW Machine and SSN-637 Class 2000 KW Machine**, Naval Ship System Engineering Station, Philadelphia (NAVSSES Project C-267), March 1984.

[85] Doerry, Norbert H., "Shipboard Electrical Generator Simulation", Term paper for 6.238, MIT, May 1988.

[86] Doerry, Norbert H., **Computer Simulation of Shipboard Electrical Distribution Systems**, Naval Engineer and Master of Science in Electrical Engineering and Computer Science thesis, MIT, May 1989.

[87] Fitzgerald, A. E., Charles Kingsley Jr., and Stephen D. Umans, **Electric Machinery**, 4th Edition, McGraw-Hill, Inc., 1983.

[88] Hildebrand, F. B., **Advanced Calculus for Applications**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1976.

[89] IEEE Power Engineering Society, "Dynamic Models for Steam and Hydro Turbines in Power System Studies", IEEE Committee Report, Paper T 73 089-0, 1973.

[90] Kirtley, J. L., **Synchronous Machine Dynamic Models,** LEES Technical Report TR-87-008, Massachusetts Institute of Technology, June 5, 1987.

[91] Krause, Paul C., **Analysis of Electric Machinery**, McGraw-Hill, Inc., 1986.

[92] Krause, Paul C., "Final Report on Modeling and Simulating of an Electric Drive System for Advanced Warships", for David Taylor Research Center, under Contract No. N61533-88-M-0746, P. C. Krause and Associates, Inc., West Lafayette, Indiana, May 20, 1988 Revision.

[93] Krause, Paul C., "Final Report on Modeling and Simulation of an Electric Drive System for Surface Combatant", for DTNSRDC Contract No. N6153387M2776, P. C. Krause and Associates, Inc., West Lafayette, Indiana, September 1987.

[94] Krause, Paul C., "Final Report on Modeling and Simulating High Speed / High Frequency SSTG Set for Post SSN 21 Submarines", for David Taylor Research Center, Contract No. N61533-89-M-2105, P. C. Krause and Associates, Inc., West Lafayette, Indiana, September 22, 1989.

[95] Krause, Paul C., "FInal Report on Modeling and Simulating Propulsion Derived Ship Service for DDG51, Flight III", for David Taylor Research Center, Contract No. N61533-88-M-2769, P. C. Krause and Associates, Inc., West Lafayette, Indiana, January 1989.

[96] Krause, Paul C., "Modeling of Shipboard Electric Power Distribution System", SBIR Phase I Final Report, P. C. Krause and Associates, Inc., West Lafayette, Indiana, July 1988.

[97] Miniovich, I. Ya., **Investigation of Hydrodynamic Characteristics of Screw Propellers Under Conditions of Reversing and Calculation Methods for Backing of Ships**, Translated for U.S Navy Bureau of Ships in 1960 by Royer & Roger, Inc., Washington DC. from Transactions of the A. N. Krylov Central Scientific Research Institute, Issue No. 122, 1958.

[98] Press, William H., Brian P. Flannery, Saul A. Teukolsky, William T. Vettering, **Numerical Recipes in C: The Art of Scientific Computing**, Cambridge University Press, New York, 1988.

[99] Rowen, W.I., "Simplified Mathematical Representations of Heavy-Duty Gas Turbines", Journal of Engineering for Power, October 1983, Vol 105, pages 865-869.

[100] Sarma, Mulukutla, **Synchronous Machines (Their Theory, Stability, and Excitation Systems)**, Gordon and Breach Science Publishers, New York, 1979.

[101] Sauer, P. W., and M. A. Pai, **Course Guide and Notes for Power System Dynamics and Stability**, Department of Electrical Engineering and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana Ill.

[102] Velez-Reyes, Miguel and George C. Verghese, **Developing Reduced Order Electrical Mahcine Models Using Participation Factors**, 12th IMACS World Congress, Paris, July 1988.

[103] Woodson, Herbert H. and James R. Melcher, **Electromechanical Dynamics**, John Wiley and Sons, 1968.

[104] Woodward Governor Company, **Electrical Generating Systems: Synchronizing and Methods of Controlling Output**, Manual 25104B, 1985.

[105] Woodward Governor Company, **The Control of Prime Mover Speed, Part I, The Controlled System**, Manual 25031, 1981.

[106] Woodward Governor Company, **The Control of Prime Mover Speed, Part II, Speed Governor Fundamentals**, Manual 25031, 1981.

[107] Woodward Governor Company, **The Control of Prime Mover Speed, Part III, Parallel Operation of Alternators**, Manual 25031, 1981.

[108] Woodward Governor Company, **The Control of Prime Mover Speed, Part IV-A, Mathematical Analysis**, 1981.

[109] Woodward Governor Company, **9900-326 System, 9900-323 Electronic Governor Control, 9900-322 Interface Panel includes 9900-305 Temperature Sensor, Part 2, Theory of Operation and Calibration Procedures for Allison 501K-17 Gas-Turbine Engines**, Manual 83029 Part 2B.

[110] Woodward Governor Company, **2301 Load Sharing & Speed Control**, Manual 82406H., 1978.

[111] Woodward Governor Company, **2301A Electronic Load Sharing and Speed Controls 9905 Series, UL Listed E 97763, Installation Operation and Calibration**, Manual 82389G, 1987.

[112] Woodward Governor Company, **EG-M Control Box**, Manual 37705G, 1964.

[113] Woodward Governor Company, **EG-A Control Box**, Manual 37706N, 1970.

[114] Woodward Governor Company, **Master Frequency Trimmer For 2301A Multiple Engine System**, Manual 82397, 1987.

[115] Woodward Governor Company, **SPM Digital Synchronizer**, Bulletin 82714A, 1976.

[116] Woodward Governor Company, **Digital Speed Matching Synchronizer (DSM)**, Manual 85100E, 1986.

[117] Woodward Governor Company, **SPM Digital Synchronizer**, Manual 82708A, 1972.

[118] Woodward Governor Company, **Precise Frequency Control**, Manual 82497B, 1984

Appendix A: Glossary

Block — A subset of a system's equations and variables which must be solved simultaneously. Blocks are organized into a sequence where variables determined in a previous block may be used in following blocks.

Continuation Parameter — A technique to enlarge the convergence region of a nonlinear system by using the solution to a linear system as the initial guess for the solution of another system which is a combination of the linear and nonlinear systems. The process is repeated with each iteration increasing the nonlinear portion until the solution to the nonlinear system is determined. The continuation parameter determines the relative proportion of the nonlinear system: 0 for the linear system and 1 for the desired nonlinear system.

Device — A device is a mathematical model of a physical piece of equipment comprising a system. Devices interact with one another through interface variables which are associated with other device interface variables through terminals connected at nodes. The equations describing a given device type are specified in the device definition. A given instance of a device also has associated parameters and nodal connections.

`device.def` — A file for describing a device definition. Each device type has an entry describing the device type name, terminals, states, parameters, structural Jacobian, and MATLAB M-File containing the constitutive equations.

Device Jacobian — A matrix whose elements are the partial derivatives of the export variables of a device with respect to the device import variables.

Device Structural Jacobian Matrix — A matrix describing the dependence of a device's export variable with respect to the import variables. The dependence is specified by a matrix whose elements are a code indicating if the dependence is zero, identity, diagonal, linear, or nonlinear.

Export Variable — An interface variable (either a potential or flow variable) of a device which is explicitly defined by the device constitutive equations. A device takes import variables as input and produces export variables.

Flow Variable An interface variable (either an export or import variable) associated with a normal terminal of a device which corresponds to a quantity satisfying Kirchhoff's Current Law at nodes. Examples of flow variables are currents, forces, and torques.

G_{min} A modification to the KCL equations at a node corresponding to the insertion of a conductance to the 0 potential. Used to prevent singular systems.

Import Variable An interface variable (either a potential or flow variable) of a device which is implicitly defined by the device constitutive equations. A device takes import variables as input and produces export variables.

Information Terminal A terminal of a device having only a potential associated with it. Used to convey energyless information between devices.

Interface Variable Variables through which devices communicate energy and information transfer to other devices. Interface variables are associated with terminals, can be classified as either flow or potential variables and can be classified as either import or export variables.

KCL Kirchhoff's Current Law which states the sum of the flow variables attached to a node is zero.

KCL Equation Equates the sum of the flow variables attached to a node to zero.

KCL Group Number If a subset of a device's flow variables add to zero by definition, then the elements of such a subset have a device-unique nonzero group number. Flow variables which do not belong to such a subset have a 0 KCL Group Number.
KCL Group Numbers are used to determine possible singular systems due to linear dependence of system KCL equations.

MATLAB M-File Text files of MATLAB commands for creating new MATLAB functions or executing *scripts*.

Newton-Raphson Method An iterative technique for solving systems of nonlinear equations which uses the Jacobian Matrix to generate corrections to the system variables.

Node — A connection point for connecting terminals of one or more devices. If at least one normal terminal is attached, the node is a normal node and a system KCL equation is written to equate the sum of all the attached flow variables to zero. If only information terminals are attached to a node, the node is an information node. All nodes have an associated node potential.

Normal Terminal — A terminal having both a potential and flow variable. Used to simulate energy transfer between devices.

Parameter — A variable which does not change throughout the simulation. Usually refers to machine ratings, resistances, time constants, etc.

Potential Difference Equation — Each export potential variable in a system has an associated potential difference equation equating to zero the difference between the potential of the node to which the variable is attached and the value of the export potential.

Potential Variable — An interface variable (either an import or export variable) associated with either a normal or information node. All of the potential variables attached to the same node are equal to the potential of the node.

R_{min} — A modification to a potential difference equation corresponding to the insertion of a series resistance. Used to prevent singular systems.

Smoothing Operator — A waveform operator for removing the high order waveform content of its argument by returning a waveform which is the convolution of the argument with a square pulse. The returned waveform is effectively the local average of the argument waveform. The smoothing operator removes unnecessary detail from a waveform and improves the representation of the desired properties of the waveform with fewer coefficients.

State — A state is a device variable whose value is retained between waveform intervals. Constants of integration and device operating modes are the most common uses of states.

Structural Jacobian Code A code indicating the nature of an element of a Jacobian matrix:

- 0 Zero Matrix
- I Identity Matrix
- D Diagonal Matrix
- L Linear Matrix
- A Type A nonlinear Matrix
- N Nonlinear Matrix
- U Unknown

Subsystem A subset of the devices of a system which are grouped together and solved independently of other devices and subsystems. Subsystems have not been implemented in WAVESIM.

System A group of devices and subsystems and the nodes interconnected them.

System Jacobian A matrix containing the partial derivatives of the system equations with respect to the system variables.

System Structural Jacobian Matrix A matrix describing the dependence of a system's equations with respect to the system variables. The dependence is specified by a matrix whose elements are a code indicating if the dependence is zero, identity, diagonal, linear, or nonlinear.

System Variable The set of system variables is composed of all the node potentials and all the device import flow variables.

Terminal A modelling analogy to a physical attachment point on a device. Normal terminals have an associated flow and potential variable and are used to model the transfer of energy into and out of a device. Information terminals have only a potential variable and are used to convey information between devices.

Waveform A representation of a variable over a given time interval consisting of a vector of coefficients and a waveform type indicator for specifying how the coefficients should be interpreted. Common waveform types are Legendre Series, Chebyshev Series, Polynomials and Data Points.

Waveform Content The magnitude of a coefficient of a waveform divided by the square root of the sum of all the waveform coefficients. The Waveform Contents of the higher order coefficients are used to determine if the truncation error is negligible.

Waveform Type Indicator An indicator specifying how the coefficients of the waveform vector should be interpreted. Common waveform types are Legendre Series, Chebyshev Series, Polynomials and Data Points.

WAVESIM A numerical algorithm development program incorporating the systematic treatment of waveforms as a data type, the terminal description of devices, and the use of structural Jacobians in system reduction.

Appendix B: Continuation Parameter Pitfalls

If used properly, continuation parameters can help enlargen the region of convergence of an iterative scheme. This section will show how continuation parameters can fail due to bifurcations of solutions.

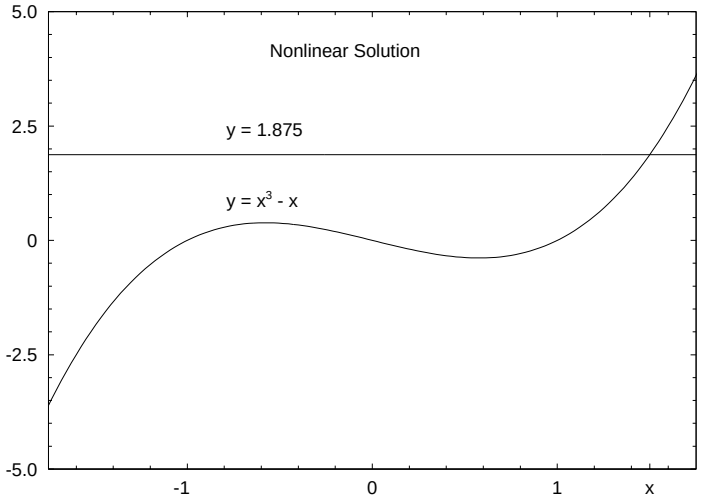
Take for example, the following system of two equations and two unknowns ***F*(*x*,*y*) = 0**:

$$F_1(x,y) = y - (x^3 - x) = 0$$

$$F_2(x,y) = y - (Mx + B) = 0$$

Initially, set ***M* = 0** and ***B* = 1.875**. From the following figure, it is obvious the solution is the intersection of the two curves and falls at the point **(1.5,1.875)**.

Figure B-1: Solution to $y = x^3 - x$ and $y = 1.875$



To solve this system with a continuation parameter, we create a new function ***H*(*x*,*y*,α)** which is formed by combining ***F*(*x*,*y*)** with a linear system ***G*(*x*,*y*):**

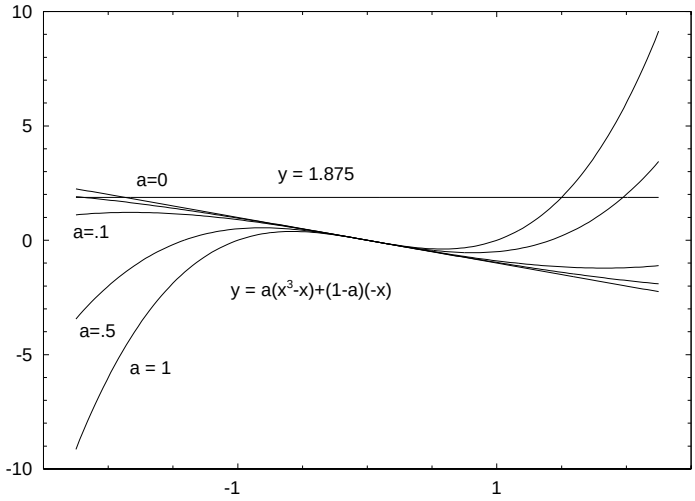
$$G_1(x,y) = y - (mx + b) = 0$$

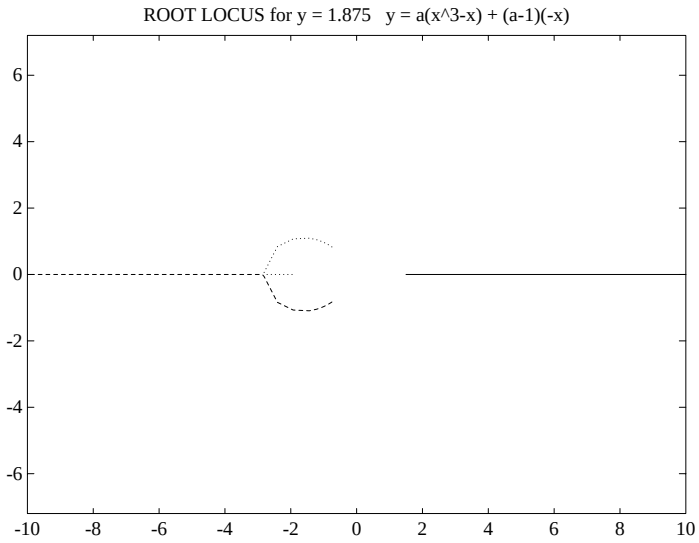
$$G_2(x,y) = y - (Mx + B) = 0$$

$$H(x, y, \alpha) = \alpha F(x, y) + (1 - \alpha) G(x, y) = 0$$

The modeller now has the choice of selecting ***m*** and ***b***. A natural choice would be a linearization about a given point. If we linearize about ***x* = 0**, the values are ***m* = -1** and ***b* = 0**. The following figure shows the results of this selection:

Figure B-2: Continuation Method for *m*=-1 *b*=0 *M*=0 *B*=1.875



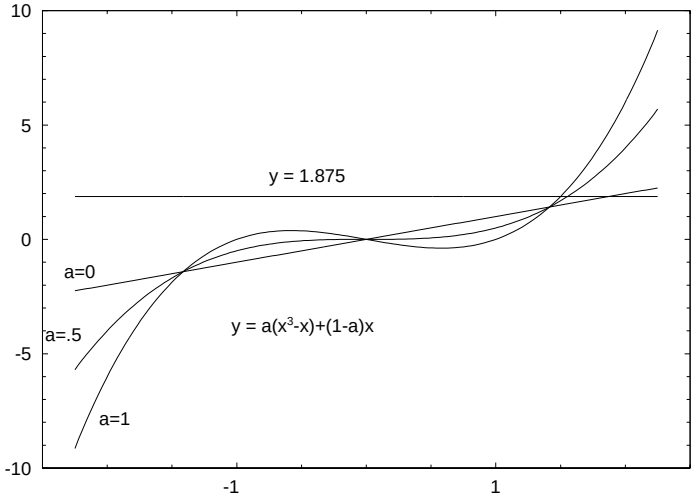


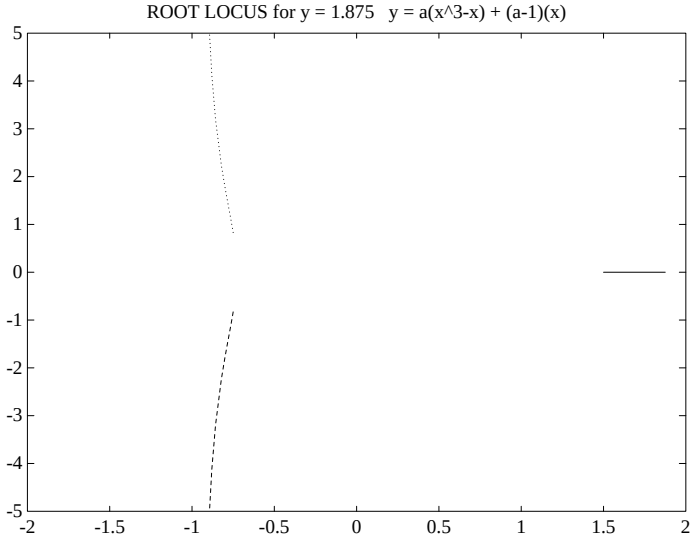
α	***x* Root Locus Points** $y = \alpha(x^3 - x) + (1 - \alpha)(-x)$ $y = 1.875$		
0			**-1.8750**
0.0100	10.8310	-8.8820	**-1.9490**
0.0200	7.8686	-5.8222	**-2.0464**
0.0400	5.7569	-3.2569	**-2.5000**
0.0421	5.6274	-2.8637	**-2.7637**
0.0422	**5.6214**	-2.8107 - 0.0613i	-2.8107 + 0.0613i
0.0600	**4.8125**	-2.4063 - 0.8387i	-2.4063 + 0.8387i
0.1000	**3.8553**	-1.9277 - 1.0712i	-1.9277 + 1.0712i
0.2000	**2.8743**	-1.4372 - 1.0937i	-1.4372 + 1.0937i
0.4000	**2.1609**	-1.0804 - 1.0010i	-1.0804 + 1.0010i
0.5000	**1.9746**	-0.9873 - 0.9614i	-0.9873 + 0.9614i
0.7000	**1.7263**	-0.8632 - 0.8981i	-0.8632 + 0.8981i
1.0000	**1.5000**	-0.7500 - 0.8292i	-0.7500 + 0.8292i

Note the solution for $\alpha = 0$ is **(-1.875,1.875)** which is not very close to the desired solution for $\alpha = 1$. Furthermore, as α increases slightly, it actually becomes slightly more negative until the nonlinear curve no longer intersects the linear equation in the left hand plane. At this point, the solution has a discontinuity and jumps into the right hand plane with a value for ***x*** much larger than the solution. The root locus for **x** as α goes from **0** to **1** clearly shows this. Hence for this selection of ***m*** and ***b***, the use of the continuation parameter makes the job of solving the system tougher instead of easier.

If we choose different values for ***m*** and ***b***, the situtation may change. Say for example, we set ***m* = 1** and ***b* = 0**. This selection appears to work well as can be seen with the following figure:

Figure B-3: Continuation Method for *m*=1 *b*=0 *M*=0 *B*=1.875

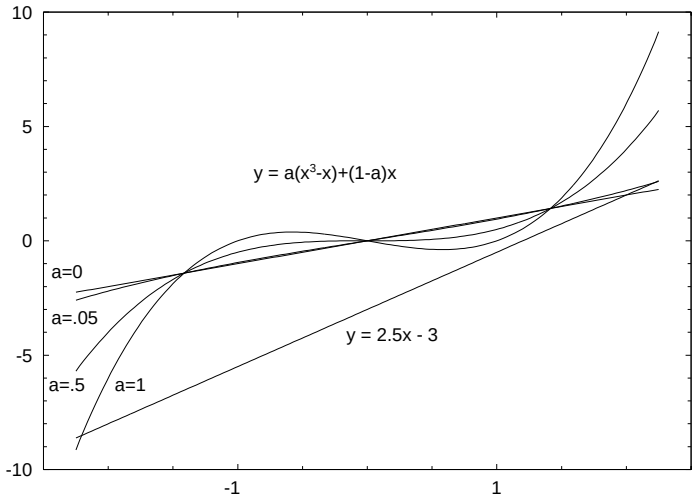


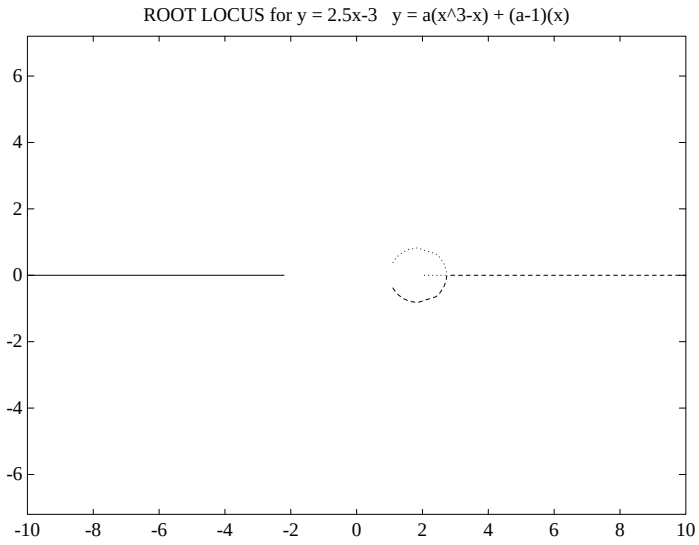


α	***x* Root Locus Points** $y = \alpha(x^3 - x) + (1 - \alpha)x$ $y = 1.875$		
0	**1.8750**		
0.0400	**1.7891**	-0.8945 - 5.0399i	-0.8945 + 5.0399i
0.2000	**1.6440**	-0.8220 - 2.2421i	-0.8220 + 2.2421i
0.5000	**1.5536**	-0.7768 - 1.3455i	-0.7768 + 1.3455i
1.0000	**1.5000**	-0.7500 - 0.8292i	-0.7500 + 0.8292i

The solution for $\alpha = 0$ is close to the solution and as α increases, it rapidly converges on the desired solution **(1.5,1.875)**. We should not rejoice however, because even this selection can fail for other choices for ***M*** and ***B***. For example, if ***M* = 2.5** and ***B* = -3**, the following figure demonstrates a discontinuity in the solution path:

Figure B-4: Continuation Method for *m*=1 *b*=0 *M*=2.5 *B*=-3



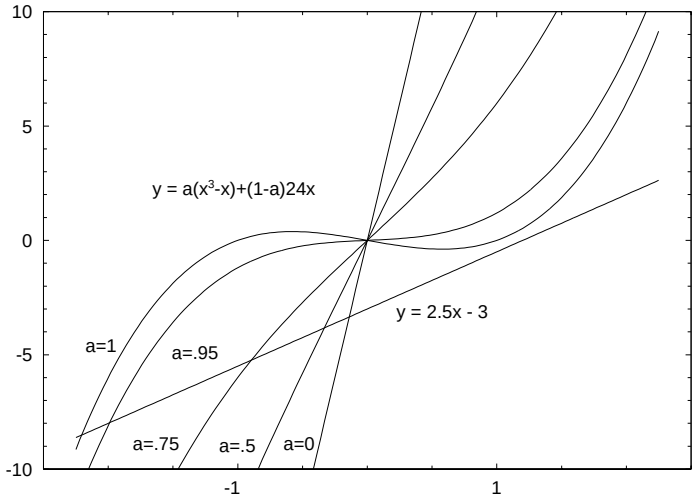


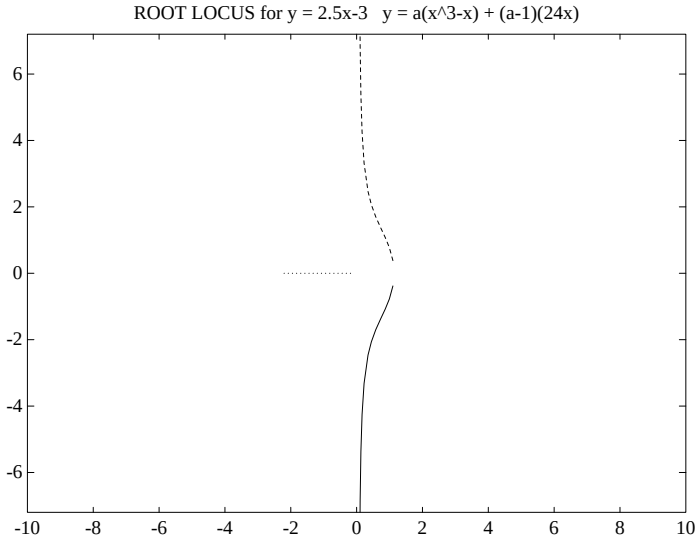
α	***x* Root Locus Points** $y = \alpha(x^3 - x) + (1 - \alpha)x$ $y = 2.5y - 3$		
0			**2.0000**
0.0100	-13.2173	11.1887	**2.0286**
0.0200	-9.6223	7.5604	**2.0619**
0.0400	-7.0779	4.9274	**2.1505**
0.0735	-5.4659	2.7706	**2.6953**
0.0736	**-5.4628**	2.7314 - 0.0326i	2.7314 + 0.0326i
0.0800	**-5.2778**	2.6389 - 0.3761i	2.6389 + 0.3761i
0.1000	**-4.8192**	2.4096 - 0.6471i	2.4096 + 0.6471i
0.3000	**-3.1844**	1.5922 - 0.7780i	1.5922 + 0.7780i
0.5000	**-2.6891**	1.3445 - 0.6507i	1.3445 + 0.6507i
1.0000	**-2.2047**	1.1024 - 0.3815i	1.1024 + 0.3815i

The bottom line is that it may not be possible to develop a transformation function whose solution vector is always continuous. Any information known as to the region where the probable operating point lies should be used in directing the solution to that region. For this example, if ***x*** is known to be constrained to the interval **[-5 5]** and ***M*** is known to be less than **17.75** ($.75 \bullet 5^2 - 1$ is the slope of the line tangent to $\boldsymbol{y = x^3 - x}$ and passing through (5,120)) then ***y*** is also constrained to the interval **[-120 120]**. If we use as our linearizing function the line connecting **(-5,-120)** and **(5,120)** it is clear the root locus will also remain within the constraints for any value of ***M*** or ***B*** meeting the constraints at $\alpha = 1$:

$$y = \alpha(x^3 - x) + (1 - \alpha)24x$$

Figure B-5: Continuatin method for *m*=24 *b*=0 *M*=2.5 *B*=-3





α	***x* Root Locus Points** $y = \alpha(x^3 - x) + (1 - \alpha)24x$ $y = 2.5y - 3$		
0	0.0789 -13.7847i	0.0789 +13.7847i	**-0.1395**
0.1000	0.0789 -13.7847i	0.0789 +13.7847i	**-0.1579**
0.2000	0.0909 - 9.0843i	0.0909 + 9.0843i	**-0.1817**
0.3000	0.1070 - 6.8338i	0.1070 + 6.8338i	**-0.2141**
0.4000	0.1301 - 5.3666i	0.1301 + 5.3666i	**-0.2603**
0.5000	0.1657 - 4.2523i	0.1657 + 4.2523i	**-0.3313**
0.6000	0.2265 - 3.3147i	0.2265 + 3.3147i	**-0.4530**
0.7500	0.4476 - 2.0659i	0.4476 + 2.0659i	**-0.8952**
0.8500	0.7291 - 1.3744i	0.7291 + 1.3744i	**-1.4582**
0.9500	0.9945 - 0.7737i	0.9945 + 0.7737i	**-1.9890**
1.0000	1.1024 - 0.3815i	1.1024 + 0.3815i	**-2.2047**

For this example it is actually quit easy to determine if a bifurcation will occur. At a bifurcation, the ***x*** root locus points satisfy the following relationship:

$$(x-c)^2(x-d)=0$$

$$x^3+(-d-2c)x^2+(c^2+2cd)x+(-c^2d)$$

where ***c*** is the multiple root whose paths will deviate from the real ***x***-axis and ***d*** is the root staying on the ***x***-axis. Now the actual equation defining the roots is given by:

$$\alpha x^3+((1-\alpha)m-\alpha-M)x+(1-\alpha)b-B=0$$

Equating terms we get:

$$-d-2c=0$$

$$c^2+2cd=\frac{1}{\alpha}((1-\alpha)m-\alpha-M)$$

$$-c^2d=\frac{1}{\alpha}((1-\alpha)b-B)$$

solving this system for ***c***, ***d***, and α, we get:

$$d=-2c$$

$$c=\left(\frac{(1-\alpha)b-B}{2\alpha}\right)^{\frac{1}{3}}$$

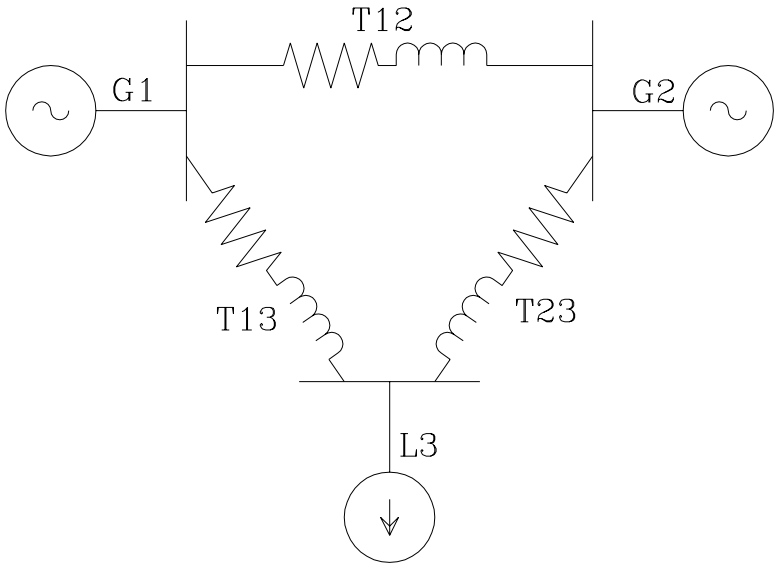
$$(1-\alpha)m-\alpha-M+3\alpha\left(\frac{(1-\alpha)b-B}{2\alpha}\right)^{\frac{2}{3}}=0$$

If one of the solutions for α is a real number in the interval **[0,1]**, then there will be a bifurcation and possibly a discontinuity in the path. If two of the roots approach from $+\infty$ and $-\infty$ along the real axis and a solution for α exists in the interval **[0,1]**, then there will definitely be a discontinuity in the solution path. If two roots appraoch from off the real axis, combine at the bifurcation point, then travel in the ***+x*** and ***-x*** directions, there will be three real solutions for ***x*** and the solution path will converge onto one of them. If there is no real solution for α in the interval **[0,1]**, then there will be no bifurcation, no discontinuity in the solution path and the solution will be unique.

Appendix C: Load Flow Example

The method for building systems can be applied to static simulations for determining equilibrium points of systems. The traditional load flow is representative of this type of problem. Figure C-1 shows a three bus load flow example consisting of four device types: PV Generator, VD Generator (Slack Bus), PQ Load, and a transmission line.

Figure C-1: 3 Bus Load Flow Example



C-1: Device Definitions

C-1.1: PV Generator

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ	$\boldsymbol{V}$ (export)	$\boldsymbol{Q}$ (import)	(0) Normal
DP	$\boldsymbol{D}$ (import)	$\boldsymbol{P}$ (export)	(0) Normal

The import $\boldsymbol{X_{imp}}$ and export $\boldsymbol{X_{exp}}$ vectors are defined by:

$$X_{imp} = \begin{bmatrix} Q \\ D \end{bmatrix}$$

$$X_{\exp} = \begin{bmatrix} V \\ P \end{bmatrix}$$

Parameters

$\boldsymbol{P_G}$	Scheduled Generator Power
$\boldsymbol{V_G}$	Scheduled Generator Voltage

Equations

$$\boldsymbol{V = V_G}$$
$$\boldsymbol{P = -P_G}$$

Device Structural Jacobian

The device structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Device Jacobian

The device jacobian is given by:

$$J_D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

C-1.2: VD Generator (Slack Bus)

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ	$\boldsymbol{V}$ (export)	$\boldsymbol{Q}$ (import)	(0) Normal
DP	$\boldsymbol{D}$ (export)	$\boldsymbol{P}$ (import)	(0) Normal

The import $\boldsymbol{X_{imp}}$ and export $\boldsymbol{X_{exp}}$ vectors are defined by:

$$X_{imp} = \begin{bmatrix} Q \\ P \end{bmatrix}$$

$$X_{\exp} = \begin{bmatrix} V \\ D \end{bmatrix}$$

Parameters

$\boldsymbol{V_G}$	Scheduled Generator Voltage
$\boldsymbol{D_G}$	Scheduled Generator Angle

Equations

$$\boldsymbol{V = V_G}$$
$$\boldsymbol{D = D_G}$$

Device Structural Jacobian

The device structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Device Jacobian

The device jacobian is given by:

$$J_D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

C-1.3: PQ Load

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ	$\boldsymbol{V}$ (import)	$\boldsymbol{Q}$ (export)	(0) Normal
DP	$\boldsymbol{D}$ (import)	$\boldsymbol{P}$ (export)	(0) Normal

The import $\boldsymbol{X_{imp}}$ and export $\boldsymbol{X_{exp}}$ vectors are defined by:

$$X_{imp} = \begin{bmatrix} V \\ D \end{bmatrix}$$

$$X_{\exp} = \begin{bmatrix} Q \\ P \end{bmatrix}$$

Parameters

$\boldsymbol{P_L}$	Scheduled Load Real Power
$\boldsymbol{Q_L}$	Scheduled Load Reactive Power

Equations

$$\boldsymbol{P = P_L}$$
$$\boldsymbol{Q = Q_L}$$

Device Structural Jacobian

The device structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Device Jacobian

The device jacobian is given by:

$$J_D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

C-1.4: Transmission Line

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ1	$\boldsymbol{V_1}$ (import)	$\boldsymbol{Q_1}$ (export)	(0) Normal
DP1	$\boldsymbol{D_1}$ (import)	$\boldsymbol{P_1}$ (export)	(0) Normal
VQ2	$\boldsymbol{V_2}$ (import)	$\boldsymbol{Q_2}$ (export)	(0) Normal
DP2	$\boldsymbol{D_2}$ (import)	$\boldsymbol{P_2}$ (export)	(0) Normal

The import $\boldsymbol{X_{imp}}$ and export $\boldsymbol{X_{exp}}$ vectors are defined by:

$$X_{imp} = \begin{bmatrix} V_1 \\ D_1 \\ V_2 \\ D_2 \end{bmatrix}$$

$$X_{\exp} = \begin{bmatrix} Q_1 \\ P_1 \\ Q_2 \\ P_2 \end{bmatrix}$$

Parameters

$\boldsymbol{R}$	Transmission Line resistance
$\boldsymbol{X}$	Transmission Line reactance

Equations

Obtain Y:

$$A = \frac{R}{R^2 + X^2}$$

$$B = -\frac{X}{R^2 + X^2}$$

$$Y = \sqrt{A^2 + B^2}$$

$$D_Y = \text{atan2}(B, A)$$

Calculate Side one current

$$I_{1R} = V_1 Y \cos(D_1 + D_Y) - V_2 Y \cos(D_2 + D_Y)$$

$$I_{1I} = V_1 Y \sin(D_1 + D_Y) - V_2 Y \sin(D_2 + D_Y)$$

Calculate Side two current

$$I_{2R} = -I_{1R}$$

$$I_{2I} = -I_{1I}$$

Calculate real and imaginary parts of the voltages

$$V_{1R} = V_1 \cos(D_1)$$

$$V_{1I} = V_1 \sin(D_1)$$

$$V_{2R} = V_2 \cos(D_2)$$

$$V_{2I} = V_2 \sin(D_2)$$

Calculate the export variables (Powers)

$$P_1 = V_{1R} I_{1R} + V_{1I} I_{1I}$$

$$Q_1 = -V_{1R} I_{1I} + V_{1I} I_{1R}$$

$$P_2 = V_{2R} I_{2R} + V_{2I} I_{2I}$$

$$Q_2 = -V_{2R} I_{2I} + V_{2I} I_{2R}$$

Device Structural Jacobian

$$J_{DS} = \begin{bmatrix} N & N & N & N \\ N & N & N & N \\ N & N & N & N \\ N & N & N & N \end{bmatrix}$$

Device Jacobian

Calculate the Partial derivatives of the voltages with respect to the import variables:

$$\frac{\partial V_{1R}}{\partial X_{imp}} = [\cos(D_1) \quad -V_1 \sin(D_1) \quad 0 \quad 0]$$

$$\frac{\partial V_{1I}}{\partial X_{imp}} = [\sin(D_1) \quad V_1 \cos(D_1) \quad 0 \quad 0]$$

$$\frac{\partial V_{2R}}{\partial X_{imp}} = [0 \quad 0 \quad \cos(D_2) \quad -V_2 \sin(D_2)]$$

$$\frac{\partial V_{2I}}{\partial X_{imp}} = [0 \quad 0 \quad \sin(D_2) \quad V_2 \cos(D_2)]$$

Calculate the partials of the currents with respect to the import variables:

$$\frac{\partial I_{1R}}{\partial X_{imp}} = [Y \cos(D_1 + D_Y) \quad -YV_1 \sin(D_1 + D_Y) \quad -Y \cos(D_2 + D_Y) \quad YV_2 \sin(D_2 + D_Y)]$$

$$\frac{\partial I_{1I}}{\partial X_{imp}} = [Y \sin(D_1 + D_Y) \quad YV_1 \cos(D_1 + D_Y) \quad -Y \sin(D_2 + D_Y) \quad -YV_2 \cos(D_2 + D_Y)]$$

$$\frac{\partial I_{2R}}{\partial X_{imp}} = -\frac{\partial I_{1R}}{\partial X_{imp}}$$

$$\frac{\partial I_{2I}}{\partial X_{imp}} = -\frac{\partial I_{1I}}{\partial X_{imp}}$$

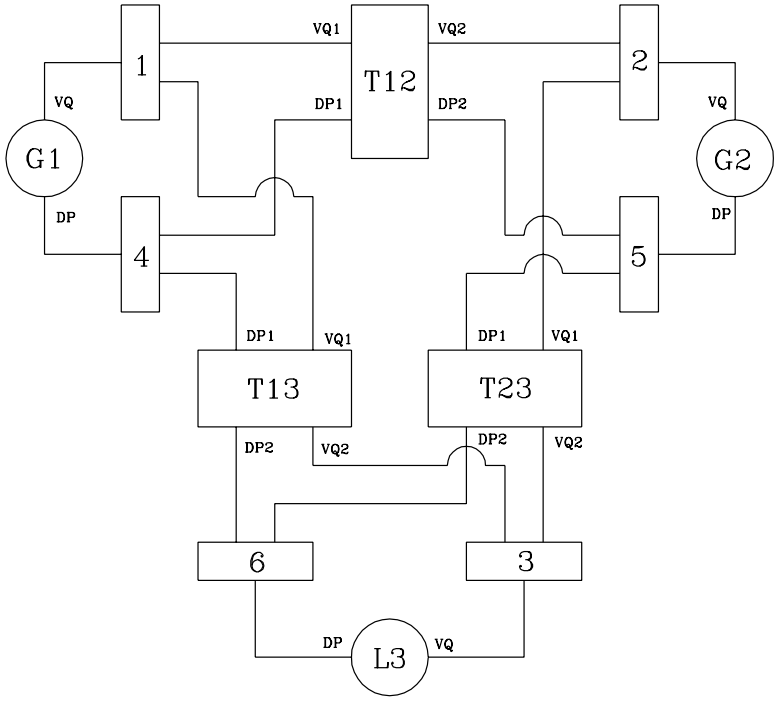
Calculate the jacobian matrix

$$J_D = \begin{bmatrix} -I_{1I} & I_{1R} & V_{1I} & -V_{1R} & 0 & 0 & 0 & 0 \\ I_{1R} & I_{1I} & V_{1R} & V_{1I} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -I_{2I} & I_{2R} & V_{2I} & -V_{2R} \\ 0 & 0 & 0 & 0 & I_{2R} & I_{2I} & V_{2R} & V_{2I} \end{bmatrix} \begin{bmatrix} \frac{\partial V_{1R}}{\partial X_{imp}} \\ \frac{\partial V_{1I}}{\partial X_{imp}} \\ \frac{\partial I_{1R}}{\partial X_{imp}} \\ \frac{\partial I_{1I}}{\partial X_{imp}} \\ \frac{\partial V_{2R}}{\partial X_{imp}} \\ \frac{\partial V_{2I}}{\partial X_{imp}} \\ \frac{\partial I_{2R}}{\partial X_{imp}} \\ \frac{\partial I_{2I}}{\partial X_{imp}} \end{bmatrix}$$

C-2: Network Description

Figure C-2-1 details the device interconnections of the 3 Bus system shown in Figure C-1.

Figure C-2-1: 3 Bus Loadflow Block Diagram



C-2.1: Variable Labeling Convention

For this example, the following convention will be used for labeling variables and functions:

Device Terminal Variables: $x_{aa\_bb\_cdf}$

aa Device Name
bb Variable Name

c ***n*** = normal terminal
i = information terminal

d ***i*** = import variable
e = export variable

f ***p*** = potential variable
f = flow variable

Device import variable vector x_{aa_i}

aa Device name

Device export variable defining function $x_{aa\_bb\_cdf} = f_{aa\_bb\_cdf}(x_{aa\_i})$

aa Device Name
bb Variable Name

c ***n*** = normal terminal
i = information terminal

d ***e*** = export variable

f ***p*** = potential variable
f = flow variable

Device Jacobian J_{aa}

Device Jacobian Element $J_{aa\_bb\_gg}$

aa Device Name
bb Export Variable Name
gg Import Variable Name

System Variables: Node Potentials V_n

n Node Serial Number

System Variables: Flow Variables I_{aa_bb}

aa Device Name
bb Variable Name

System Equation: KCL $g_n()$

n Serial number of node KCL is applied to

System Equation: Potentials $g_{n\_aa\_bb}()$

n Serial number of node
aa Device Name
bb Export Potential Variable Name

System Jacobian Element: **KCL vs Node Potential $J_{sys\_n\_m}$**

System Jacobian Element: KCL vs Import Flow $J_{sys\_n\_aa\_bb}$

System Jacobian Element: Potential Eqn vs Node Potential $J_{sys\_cc\_dd\_m}$

System Jacobian Element: Potential Eqn vs Import flow $J_{sys\_cc\_dd\_aa\_bb}$

n Serial number of KCL node
m Serial number of Node Potential
aa Flow Variable Device Name
bb Flow Variable Device Variable Name
cc Potential Equation Potential Device Name
dd Potential Equation Potential Variable Name

C-2.2: Network Specification

Now that the variable labeling convention has been addressed, it is time to define the devices and the network interconnecting them.

PD Generator G1

Terminal	Potential Variable	Flow Variable	Node
VQ	$\boldsymbol{x_{G1\_V\_nep}}$	$\boldsymbol{x_{G1\_Q\_nif}}$	1
DP	$\boldsymbol{x_{G1\_D\_nep}}$	$\boldsymbol{x_{G1\_P\_nif}}$	4

Parameters

$\boldsymbol{V_G}$	1.0 PU
$\boldsymbol{D_G}$	0.0 RAD

Import Vector:

$$x_{G1\_i} = \begin{bmatrix} x_{G1\_Q\_nif} \\ x_{G1\_P\_nif} \end{bmatrix} = \begin{bmatrix} I_{G1\_Q} \\ I_{G1\_P} \end{bmatrix}$$

PV Generator G2

Terminal	Potential Variable	Flow Variable	Node
VQ	$\boldsymbol{x_{G2\_V\_nep}}$	$\boldsymbol{x_{G2\_Q\_nif}}$	2
DP	$\boldsymbol{x_{G2\_D\_nip}}$	$\boldsymbol{x_{G2\_P\_nef}}$	5

Parameters

$\boldsymbol{P_G}$	0.5 PU
$\boldsymbol{V_G}$	1.05 PU

Import Vector:

$$x_{G2\_i} = \begin{bmatrix} x_{G2\_Q\_nif} \\ x_{G2\_D\_nip} \end{bmatrix} = \begin{bmatrix} I_{G2\_Q} \\ V_5 \end{bmatrix}$$

PQ Load L3

Terminal	Potential Variable	Flow Variable	Node
VQ	$\boldsymbol{x_{L3\_V\_nip}}$	$\boldsymbol{x_{L3\_Q\_nef}}$	3
DP	$\boldsymbol{x_{L3\_D\_nip}}$	$\boldsymbol{x_{L3\_P\_nef}}$	6

Parameters

$\boldsymbol{P_L}$	0.6 PU
$\boldsymbol{Q_L}$	0.3 PU

Import Vector:

$$x_{L3\_i} = \begin{bmatrix} x_{L3\_V\_nip} \\ x_{L3\_D\_nip} \end{bmatrix} = \begin{bmatrix} V_3 \\ V_6 \end{bmatrix}$$

Transmission Line T12

Terminal	Potential Variable	Flow Variable	Node
VQ1	$\boldsymbol{x_{T12\_V1\_nip}}$	$\boldsymbol{x_{T12\_Q1\_nef}}$	1
DP1	$\boldsymbol{x_{T12\_D1\_nip}}$	$\boldsymbol{x_{T12\_P1\_nef}}$	4
VQ2	$\boldsymbol{x_{T12\_V2\_nip}}$	$\boldsymbol{x_{T12\_Q2\_nef}}$	2
DP2	$\boldsymbol{x_{T12\_D2\_nip}}$	$\boldsymbol{x_{T12\_P2\_nef}}$	5

Parameters

$\boldsymbol{R}$	0.15 PU
$\boldsymbol{X}$	0.60 PU

Import Vector:

$$x_{T12\_i} = \begin{bmatrix} x_{T12\_V1\_nip} \\ x_{T12\_D1\_nip} \\ x_{T12\_V2\_nip} \\ x_{T12\_D2\_nip} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_4 \\ V_2 \\ V_5 \end{bmatrix}$$

Transmission Line T13

Terminal	Potential Variable	Flow Variable	Node
VQ1	$x_{T13\_V1\_nip}$	$x_{T13\_Q1\_nef}$	1
DP1	$x_{T13\_D1\_nip}$	$x_{T13\_P1\_nef}$	4
VQ2	$x_{T13\_V2\_nip}$	$x_{T13\_Q2\_nef}$	3
DP2	$x_{T13\_D2\_nip}$	$x_{T13\_P2\_nef}$	6

Parameters

R	0.05 PU
X	0.20 PU

Import Vector:

$$x_{T13\_i} = \begin{bmatrix} x_{T13\_V1\_nip} \\ x_{T13\_D1\_nip} \\ x_{T13\_V2\_nip} \\ x_{T13\_D2\_nip} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_4 \\ V_3 \\ V_6 \end{bmatrix}$$

Transmission Line T23

Terminal	Potential Variable	Flow Variable	Node
VQ1	$x_{T23\_V1\_nip}$	$x_{T23\_Q1\_nef}$	2
DP1	$x_{T23\_D1\_nip}$	$x_{T23\_P1\_nef}$	5
VQ2	$x_{T23\_V2\_nip}$	$x_{T23\_Q2\_nef}$	3
DP2	$x_{T23\_D2\_nip}$	$x_{T23\_P2\_nef}$	6

Parameters

R	0.10 PU
X	0.40 PU

Import Vector:

$$x_{T23\_i} = \begin{bmatrix} x_{T23\_V1\_nip} \\ x_{T23\_D1\_nip} \\ x_{T23\_V2\_nip} \\ x_{T23\_D2\_nip} \end{bmatrix} = \begin{bmatrix} V_2 \\ V_5 \\ V_3 \\ V_6 \end{bmatrix}$$

C-2.3: System Variables and Equations

There are nine system variable and equations associated with this example. There are the six node potentials plus three import flow variables ordered in the following manner:

$$x_{sys} = [V_1 \quad V_2 \quad V_3 \quad V_4 \quad V_5 \quad V_6 \quad I_{G1\_Q} \quad I_{G1\_P} \quad I_{G2\_Q}]^T$$

The nine system equations are composed of six Kirchhoff Current Law equations and three potential equations:

$$g_1(x_{sys}) = I_{G1\_Q} + x_{T12\_Q1\_nef} + x_{T13\_Q1\_nef}$$

$$g_2(x_{sys}) = I_{G2\_Q} + x_{T12\_Q2\_nef} + x_{T23\_Q1\_nef}$$

$$g_3(x_{sys}) = x_{L3\_Q\_nef} + x_{T13\_Q2\_nef} + x_{T23\_Q2\_nef}$$

$$g_4(x_{sys}) = I_{G1\_P} + x_{T12\_P1\_nef} + x_{T13\_P1\_nef}$$

$$g_5(x_{sys}) = x_{G2\_P\_nef} + x_{T12\_P2\_nef} + x_{T23\_P1\_nef}$$

$$g_6(x_{sys}) = x_{L3\_P\_nef} + x_{T13\_P2\_nef} + x_{T23\_P2\_nef}$$

$$g_{1\_G1\_V}(x_{sys}) = V_1 - x_{G1\_V\_nep}$$

$$g_{2\_G2\_V}(x_{sys}) = V_2 - x_{G2\_V\_nep}$$

$$g_{4\_G1\_D}(x_{sys}) = V_4 - x_{G1\_D\_nep}$$

C-2.4: System Structural Jacobian Matrix

The equations for generating the system jacobian matrix are given by:

$$\boldsymbol{g_1(x_{sys})}$$

$$J_{sys\_1\_1} = J_{T12\_Q1\_V1} + J_{T13\_Q1\_V1} = N + N$$

$$J_{sys\_1\_2} = J_{T12\_Q1\_V2} = N$$

$$J_{sys\_1\_3} = J_{T13\_Q1\_V2} = N$$

$$J_{sys\_1\_4} = J_{T12\_Q1\_D1} + J_{T13\_Q1\_D1} = N + N$$

$$J_{sys\_1\_5} = J_{T12\_Q1\_D2} = N$$

$$J_{sys\_1\_6} = J_{T13\_Q1\_D2} = N$$

$$J_{sys\_1\_G1\_Q} = I$$

$$\boldsymbol{g_2(x_{sys})}$$

$$J_{sys\_2\_1} = J_{T12\_Q2\_V1} = N$$

$$J_{sys\_2\_2} = J_{T12\_Q2\_V2} + J_{T23\_Q1\_V1} = N + N$$

$$J_{sys\_2\_3} = J_{T23\_Q1\_V2} = N$$

$$J_{sys\_2\_4} = J_{T12\_Q2\_D1} = N$$

$$J_{sys\_2\_5} = J_{T12\_Q2\_D2} + J_{T23\_Q1\_D1} = N + N$$

$$J_{sys\_2\_6} = J_{T23\_Q1\_D2} = N$$

$$J_{sys\_2\_G2\_Q} = I$$

$$\boldsymbol{g_3(x_{sys})}$$

$$J_{sys\_3\_1} = J_{T13\_Q2\_V1} = N$$

$$J_{sys\_3\_2} = J_{T23\_Q2\_V1} = N$$

$$J_{sys\_3\_3} = J_{L3\_Q\_V} + J_{T13\_Q2\_V2} + J_{T23\_Q2\_V2} = 0 + N + N$$

$$J_{sys\_3\_4} = J_{T13\_Q2\_D1} = N$$

$$J_{sys\_3\_5} = J_{T23\_Q2\_D1} = N$$

$$J_{sys\_3\_6} = J_{L3\_Q\_D} + J_{T13\_Q2\_D2} + J_{T23\_Q2\_D2} = 0 + N + N$$

$$\boldsymbol{g_4(x_{sys})}$$

$$J_{sys\_4\_1} = J_{T12\_P1\_V1} + J_{T13\_P1\_V1} = N + N$$

$$J_{sys\_4\_2} = J_{T12\_P1\_V2} = N$$

$$J_{sys\_4\_3} = J_{T13\_P1\_V2} = N$$

$$J_{sys\_4\_4} = J_{T12\_P1\_D1} + J_{T13\_P1\_D1} = N + N$$

$$J_{sys\_4\_5} = J_{T12\_P1\_D2} = N$$

$$J_{sys\_4\_6} = J_{T13\_P1\_D2} = N$$

$$J_{sys\_4\_G1\_P} = I$$

$$\boldsymbol{g_5(x_{sys})}$$

$$J_{sys\_5\_1} = J_{T12\_P2\_V1} = N$$

$$J_{sys\_5\_2} = J_{T12\_P2\_V2} + J_{T23\_P1\_V1} = N + N$$

$$J_{sys\_5\_3} = J_{T23\_P1\_V2} = N$$

$$J_{sys\_5\_4} = J_{T12\_P2\_D1} = N$$

$$J_{sys\_5\_5} = J_{G2\_P\_D} + J_{T12\_P2\_D2} + J_{T23\_P1\_D1} = 0 + N + N$$

$$J_{sys\_5\_6} = J_{T23\_P1\_D2} = N$$

$$J_{sys\_G2\_Q} = J_{G2\_P\_Q} = 0$$

$$\boldsymbol{g_6(x_{sys})}$$

$$J_{sys\_6\_1} = J_{T13\_P2\_V1} = N$$

$$J_{sys\_6\_2} = J_{T23\_P2\_V1} = N$$

$$J_{sys\_6\_3} = J_{L3\_P\_V} + J_{T13\_P2\_V2} + J_{T23\_P2\_V2} = 0 + N + N$$

$$J_{sys\_6\_4} = J_{T13\_P2\_D1} = N$$

$$J_{sys\_6\_5} = J_{T23\_P2\_D1} = N$$

$$J_{sys\_6\_6} = J_{L3\_P\_D} + J_{T13\_P2\_D2} + J_{T23\_P2\_D2} = 0 + N + N$$

$$\boldsymbol{g_{1\_G1\_V}(x_{sys})}$$

$$J_{sys\_G1\_V\_1} = I$$

$$J_{sys\_G1\_V\_G1\_Q} = -J_{G1\_V\_Q} = 0$$

$$J_{sys\_G1\_V\_G1\_P} = -J_{G1\_V\_P} = 0$$

$$\boldsymbol{g_{2\_G2\_V}(x_{sys})}$$

$$J_{sys\_G2\_V\_2} = I$$

$$J_{sys\_G2\_V\_5} = -J_{G2\_V\_D} = 0$$

$$J_{sys\_G2\_V\_G2\_Q} = -J_{G2\_V\_Q} = 0$$

$$\boldsymbol{g_{4\_G1\_D}(x_{sys})}$$

$$J_{sys\_G1\_D\_4} = I$$

$$J_{sys\_G1\_D\_G1\_Q} = -J_{G1\_D\_Q} = 0$$

$$J_{sys\_G1\_D\_G1\_P} = -J_{G1\_D\_P} = 0$$

By applying the rules of Structural Jacobian element arithmetic on the system equations, we can generate the following system Structural Jacobian:

$$J_{ss} = \begin{bmatrix} N & N & N & N & N & N & I & 0 & 0 \\ N & N & N & N & N & N & 0 & 0 & I \\ N & N & N & N & N & N & 0 & 0 & 0 \\ N & N & N & N & N & N & 0 & I & 0 \\ N & N & N & N & N & N & 0 & 0 & 0 \\ N & N & N & N & N & N & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Close inspection of this matrix reveals seven blocks: Six 1×1 element blocks and one 3×3 element block:

Block 1

System Row: 7

System Column: 1

System Variable: $\boldsymbol{V_1}$

Equations:

$$g_{1\_G1\_V}(x_{sys}) = V_1 - x_{G1\_V\_nep}$$

Structural Jacobian:

$$J_{B1} = [I]$$

Block 2

System Row: 8

System Columns: 2

System Variable: $\boldsymbol{V_2}$

Equations:

$$g_{2\_G2\_V}(x_{sys}) = V_2 - x_{G2\_V\_nep}$$

Structural Jacobian:

$$J_{B2} = [I]$$

Block 3

System Row: 9

System Column: 4

System Variable: $\boldsymbol{V_4}$

Equations:

$$g_{4\_G1\_D}(x_{sys}) = V_4 - x_{G1\_D\_nep}$$

Structural Jacobian:

$$J_{B3} = [I]$$

Block 4

System Rows: 3 5 6

System Columns: 3 5 6

System Variables: $\boldsymbol{V_3}$ $\boldsymbol{V_5}$ $\boldsymbol{V_6}$

Equations:

$$g_3(x_{sys}) = x_{L3\_Q\_nef} + x_{T13\_Q2\_nef} + x_{T23\_Q2\_nef}$$

$$g_5(x_{sys}) = x_{G2\_P\_nef} + x_{T12\_P2\_nef} + x_{T23\_P1\_nef}$$

$$g_6(x_{sys}) = x_{L3\_P\_nef} + x_{T13\_P2\_nef} + x_{T23\_P2\_nef}$$

Structural Jacobian:

$$J_{B4} = \begin{bmatrix} N & N & N \\ N & N & N \\ N & N & N \end{bmatrix}$$

Block 5

System Row: 1

System Column: 7

System Variable: $\boldsymbol{I_{G1\_Q}}$

Equations:

$$g_1(x_{sys}) = I_{G1\_Q} + x_{T12\_Q1\_nef} + x_{T13\_Q1\_nef}$$

Structural Jacobian:

$$J_{B5} = [I]$$

Block 6

System Row: 2

System Column: 8

System Variable: $\boldsymbol{I_{G1\_P}}$

Equations:

$$g_2(x_{sys}) = I_{G2\_Q} + x_{T12\_Q2\_nef} + x_{T23\_Q1\_nef}$$

Structural Jacobian:

$$J_{B6} = [I]$$

Block 7

System Row: 4

System Column: 9

System Variable: $\boldsymbol{I_{G2\_Q}}$

Equations:

$$g_4(x_{sys}) = I_{G1\_P} + x_{T12\_P1\_nef} + x_{T13\_P1\_nef}$$

Structural Jacobian:

$$J_{B7} = [I]$$

C-2.5: Solving the System

Applying the equations for the first three blocks yields:

$$V_1 = f_{G1\_V\_nep}\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = 1.0 \quad \text{PU}$$

$$V_2 = f_{G2\_V\_nep}\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = 1.05 \quad \text{PU}$$

$$V_4 = f_{G1\_D\_nep}\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = 0.0 \quad \text{PU}$$

Now the following system of three equations for the fourth block must be solved:

$$x_{error} = \begin{bmatrix} g_3(x_{B4}, x_{pre}) \\ g_5(x_{B4}, x_{pre}) \\ g_6(x_{B4}, x_{pre}) \end{bmatrix} = 0$$

Where:

$$x_{B4} = \begin{bmatrix} V_3 \\ V_5 \\ V_6 \end{bmatrix}$$

$$x_{pre} = \begin{bmatrix} V_1 \\ V_2 \\ V_4 \end{bmatrix}$$

Starting with the intial guess of $\mathbf{[1\ 0\ 0]^T}$ for $\boldsymbol{x_{B4}}$ we obtain the following error vector and jacobian matrix:

$$x^0_{error} = \begin{bmatrix} 0.1824 \\ -0.4485 \\ 0.5706 \end{bmatrix}$$

$$J^0_{B4} = \begin{bmatrix} 6.9412 & 0.6176 & -1.7941 \\ -0.6176 & 4.1176 & -2.4706 \\ 1.7253 & -2.4706 & 7.1765 \end{bmatrix}$$

Inverting the Jacobian and multiplying by the error vector results in the following correction vector for $\boldsymbol{x_{B4}}$:

$$x_{\Delta}^{0} = \begin{bmatrix} 0.0441 \\ -0.0769 \\ 0.0424 \end{bmatrix}$$

$$x_{B4}^{1} = x_{B4}^{0} - x_{\Delta}^{0}$$

By repeating the Newton-Raphson iterations several more times, the following table can be constructed:

Iteration	V_3	V_5	V_6	$g_3()$	$g_5()$	$g_6()$
0	1.0000	0.0000	0.0000	0.1824	-0.4485	0.5706
1	0.9559	0.0769	-0.0424	0.0289	-0.0084	0.0311
2	0.9502	0.0762	-0.0463	2.284e-4	-0.448e-4	2.208e-4
3	0.9502	0.0761	-0.0464	1.370e-8	-0.214e-8	1.240e-8

From these results, the final three blocks can easily be solved:

$$I_{G1\_Q} = -f_{T12\_Q1\_nef}(x_{T12\_i}) - f_{T13\_Q1\_nef}(x_{T13\_i})$$

$$\boldsymbol{I_{G1\_Q}} \textbf{ = -0.1451 PU}$$

$$I_{G1\_P} = -f_{T12\_P1\_nef}(x_{T12\_i}) - f_{T13\_P1\_nef}(x_{T13\_i})$$

$$\boldsymbol{I_{G1\_P}} \textbf{ = -0.1233 PU}$$

$$I_{G2\_Q} = -f_{T12\_Q2\_nef}(x_{T12\_i}) - f_{T23\_Q1\_nef}(x_{T23\_i})$$

$$\boldsymbol{I_{G2\_Q}} \textbf{ = -0.2483 PU}$$

C-3: Summary of Results

Bus 1

Bus Voltage Magnitude	1.0 PU	
Bus Voltage Angle	0.0 rad	
G1 Real/Reactive Power	-0.1233 PU	-0.1451 PU
T12 Real/Reactive Power	-0.1437 PU	-0.0423 PU
T13 Real/Reactive Power	0.2671 PU	0.1874 PU

Bus 2

Bus Voltage Magnitude	1.0500 PU	
Bus Voltage Angle	0.0761 rad	
G2 Real/Reactive Power	-0.5 PU	-0.2483 PU
T12 Real/Reactive Power	0.1471 PU	0.0558 PU
T23 Real/Reactive Power	0.3529 PU	0.1925 PU

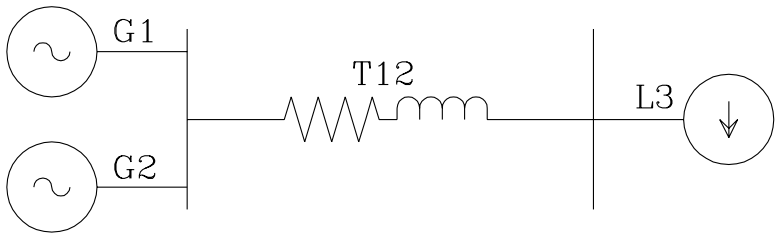
Bus 3

Bus Voltage Magnitude	0.9502 PU	
Bus Voltage Angle	-0.0464 rad	
L3 Real/Reactive Power	0.6000 PU	0.3000 PU
T13 Real/Reactive Power	-0.2617 PU	-0.1661 PU
T23 Real/Reactive Power	-0.3383 PU	-0.1339 PU

Appendix D: Modified Load Flow Example

Appendix C demonstrated how a system can be built and solved for a conventional load flow problem. This example demonstrates how control signals such as real and reactive power sharing signals can be incorporated in the load flow solution. In particular, this example connects two parallel generators to a load via a transmission line. A conventional load flow fails for this example because the generator bus voltage magnitude is overdetermined and there is no relationship for sharing reactive power. In this example, information variables are used to force each generator to be proportionally loaded and have the same power angle.

Figure D-1: Parallel Generator Load Flow Example



D-1: Device Definitions

In addition to the transmission line and PQ load defined in Appendix C, two more devices must be defined: A slack bus generator incorporating the load sharing information, and a PQ generator employing the load sharing.

D-1.1: VDS Generator (Slack Bus)

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ	***V*** (export)	***Q*** (import)	(0) Normal
DP	***D*** (export)	***P*** (import)	(0) Normal
p	***p*** (export)		Information
q	***q*** (export)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} Q \\ P \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} V \\ D \\ p \\ q \end{bmatrix}$$

Parameters

$\boldsymbol{V_G}$	Scheduled Generator Voltage
$\boldsymbol{D_G}$	Scheduled Generator Angle
$\boldsymbol{P_B}$	Scheduled Generator Power Base

Equations

$$V = V_G$$

$$D = D_G$$

$$p = -\frac{P}{P_B}$$

$$q = \frac{Q}{P}$$

Device Structural Jacobian

The device structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & L \\ N & N \end{bmatrix}$$

Device Jacobian

The device jacobian is given by:

$$J_D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{P_B} \\ \frac{1}{P} & -\frac{Q}{P^2} \end{bmatrix}$$

D-1.2: PQS Generator

Interface Variables

Terminal	Potential Variable	Flow Variable	(KCL Group) Type
VQ	***V*** (import)	***Q*** (export)	(0) Normal
DP	***D*** (import)	***P*** (export)	(0) Normal
p	***p*** (import)		Information
q	***q*** (import)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} V \\ D \\ p \\ q \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} Q \\ P \end{bmatrix}$$

Parameters

$\boldsymbol{P_B}$ Scheduled Generator Power Base

Equations

$$P = -P_B p$$

$$Q = -P_B pq$$

Device Structural Jacobian

The device structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} 0 & 0 & N & N \\ 0 & 0 & L & 0 \end{bmatrix}$$

Device Jacobian

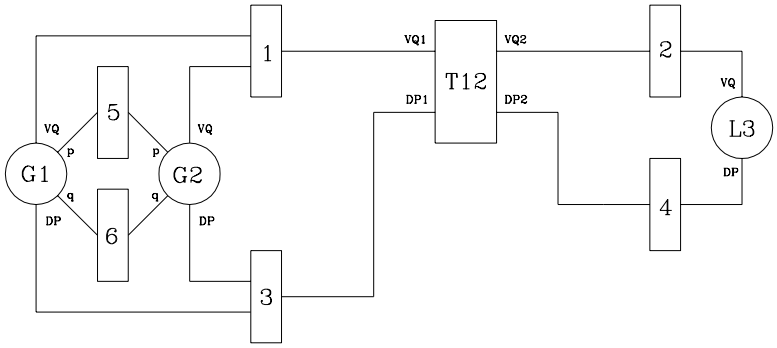
The device jacobian is given by:

$$J_D = \begin{bmatrix} 0 & 0 & -P_B q & -P_B p \\ 0 & 0 & -P_B & 0 \end{bmatrix}$$

D-2: Network Description

Figure D-2-1 is a block diagram of the system represented in Figure D-1:

Figure D-2-1: Parallel Generator Example Block Diagram



D-2.1: Network Specification

Using the same variable labeling convention as in Appendix C, the devices and network are specified by:

VDS Generator G1

Terminal	Potential Variable	Flow Variable	Node
VQ	$x_{G1\_V\_nep}$	$x_{G1\_Q\_nif}$	1
DP	$x_{G1\_D\_nep}$	$x_{G1\_P\_nif}$	3
p	$x_{G1\_p\_iep}$		5
q	$x_{G1\_q\_iep}$		6

Parameters:

V_G	1.05 PU
D_G	0.00 rad
P_B	1.00 PU

Import Vector:

$$x_{G1\_i} = \begin{bmatrix} x_{G1\_Q\_nif} \\ x_{G1\_P\_nif} \end{bmatrix} = \begin{bmatrix} I_{G1\_Q} \\ I_{G1\_P} \end{bmatrix}$$

PQS Generator G2

Terminal	Potential Variable	Flow Variable	Node
VQ	$x_{G2\_V\_nip}$	$x_{G1\_Q\_nef}$	1
DP	$x_{G2\_D\_nip}$	$x_{G1\_P\_nef}$	3
p	$x_{G2\_p\_iip}$		5
q	$x_{G2\_q\_iip}$		6

Parameters:

P_B	0.50 PU

Import Vector:

$$x_{G2\_i} = \begin{bmatrix} x_{G2\_V\_nip} \\ x_{G2\_D\_nip} \\ x_{G2\_p\_iip} \\ x_{G2\_q\_iip} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_3 \\ V_5 \\ V_6 \end{bmatrix}$$

Transmission Line T12

Terminal	Potential Variable	Flow Variable	Node
VQ1	$x_{T12\_V1\_nip}$	$x_{T12\_Q1\_nef}$	1
DP1	$x_{T12\_D1\_nip}$	$x_{T12\_P1\_nef}$	3
VQ2	$x_{T12\_V2\_nip}$	$x_{T12\_Q2\_nef}$	2
DP2	$x_{T12\_D2\_nip}$	$x_{T12\_P2\_nef}$	4

Parameters:

R	0.05 PU
X	0.20 PU

Import Vector:

$$x_{T12\_i} = \begin{bmatrix} x_{T12\_V1\_nip} \\ x_{T12\_D1\_nip} \\ x_{T12\_V2\_nip} \\ x_{T12\_D2\_nip} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_3 \\ V_2 \\ V_4 \end{bmatrix}$$

PQ Load L3

Terminal	Potential Variable	Flow Variable	Node
VQ	$\boldsymbol{x_{L3\_V\_nip}}$	$\boldsymbol{x_{L3\_Q\_nef}}$	2
DP	$\boldsymbol{x_{L3\_D\_nip}}$	$\boldsymbol{x_{L3\_P\_nef}}$	4

Parameters:

$\boldsymbol{P_L}$	0.60 PU
$\boldsymbol{Q_L}$	0.10 PU

Import Vector:

$$x_{L3\_i} = \begin{bmatrix} x_{L3\_V\_nip} \\ x_{L3\_D\_nip} \end{bmatrix} = \begin{bmatrix} V_2 \\ V_4 \end{bmatrix}$$

D-2.2: System Variables and Equations

There are eight system variables and equations associated with this example. There are the six node potentials plus two import flow variables ordered in the following manner:

$$x_{sys} = [V_1 \quad V_2 \quad V_3 \quad V_4 \quad V_5 \quad V_6 \quad I_{G1\_Q} \quad I_{G1\_P}]^T$$

The eight system equations are composed of four Kirchhoff Current Law equations and four potential equations:

$$g_1(x_{sys}) = I_{G1\_Q} + x_{G2\_Q\_nef} + x_{T12\_Q1\_nef}$$

$$g_2(x_{sys}) = x_{L3\_Q\_nef} + x_{T12\_Q2\_nef}$$

$$g_3(x_{sys}) = I_{G1\_P} + x_{G2\_P\_nef} + x_{T12\_P1\_nef}$$

$$g_4(x_{sys}) = x_{L3\_P\_nef} + x_{T12\_P2\_nef}$$

$$g_{1\_G1\_V}(x_{sys}) = V_1 - x_{G1\_V\_nep}$$

$$g_{3\_G1\_D}(x_{sys}) = V_3 - x_{G1\_D\_nep}$$

$$g_{5\_G1\_p}(x_{sys}) = V_5 - x_{G1\_p\_iep}$$

$$g_{6\_G1\_q}(x_{sys}) = V_6 - x_{G1\_q\_iep}$$

D-2.3: System Structural Jacobian Matrix

Using the device structural jacobian matrices along with the system equations, the following system structural jacobian can be created:

$$J_{SS} = \begin{bmatrix} N & N & N & N & N & N & I & 0 \\ N & N & N & N & 0 & 0 & 0 & 0 \\ N & N & N & N & L & 0 & 0 & I \\ N & N & N & N & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & L \\ 0 & 0 & 0 & 0 & 0 & I & N & N \end{bmatrix}$$

Applying the system reduction algorithms, five blocks can be identified: two 1×1 element blocks and three 2×2 element blocks:

Block 1

System Row: 5

System Column: 1

System Variable: $\boldsymbol{V_1}$

Equation:

$$g_{1\_G1\_V}(x_{sys}) = V_1 - x_{G1\_V\_nep}$$

Structural Jacobian:

$$J_{B1} = [I]$$

Block 2

System Row: 6
System Column: 3
System Variable: $\boldsymbol{V_3}$
Equation:

$$g_{3\_G1\_D}(x_{sys}) = V_3 - x_{G1\_D\_nep}$$

Structural Jacobian:

$$J_{B2} = [I]$$

Block 3

System Rows: 2 4
System Columns: 2 4
System Variables: $\boldsymbol{V_2}$ $\boldsymbol{V_4}$
Equations:

$$g_2(x_{sys}) = x_{L3\_Q\_nef} + x_{T12\_Q2\_nef}$$

$$g_4(x_{sys}) = x_{L3\_P\_nef} + x_{T12\_P2\_nef}$$

Structural Jacobian:

$$J_{B3} = \begin{bmatrix} N & N \\ N & N \end{bmatrix}$$

Block 4

System Rows: 3 7

System Columns: 5 8

System Variables: $\boldsymbol{V_5}$ $\boldsymbol{I_{G1\_P}}$

Equations:

$$g_3(x_{sys}) = I_{G1\_P} + x_{G2\_P\_nef} + x_{T12\_P1\_nef}$$

$$g_{5\_G1\_p}(x_{sys}) = V_5 - x_{G1\_p\_iep}$$

Structural Jacobian:

$$J_{B4} = \begin{bmatrix} L & I \\ I & L \end{bmatrix}$$

Block 5

System Rows: 1 8

System Columns: 6 7

System Variables: $\boldsymbol{V_6}$ $\boldsymbol{I_{G1\_Q}}$

Equations:

$$g_1(x_{sys}) = I_{G1\_Q} + x_{G2\_Q\_nef} + x_{T12\_Q1\_nef}$$

$$g_{6\_G1\_q}(x_{sys}) = V_6 - x_{G1\_q\_iep}$$

Structural Jacobian:

$$J_{B5} = \begin{bmatrix} N & I \\ I & N \end{bmatrix}$$

D-2.4: Solving the System

Applying the equations for the first two blocks yields:

$$V_1 = f_{G1\_V\_nep}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = 1.05 \quad \text{PU}$$

$$V_3 = f_{G1\_D\_nep}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = 0.0 \quad \text{rad}$$

The remaining blocks are systems of 2×2 equations and unknowns. Blocks 3 and 5 are nonlinear and must be solved iteratively. Block 4 is linear block requiring only one iteration:

Block 3:

n	V_2	V_4	$g_2()$	$g_4()$
0	1.0000	0.0000	-0.1353	0.5414
1	1.0000	-0.1095	0.0293	0.0085
2	0.9933	-0.1105	2.017e-4	0.841e-4
3	0.9933	-0.1105	1.046e-8	0.569e-8

Block 4:

n	V_5	I_{G1_P}	$g_3()$	$g_{G1\_p}()$
0	1.0000	1.0000	1.1188	2.0000
1	0.4125	-0.4125	0.0000	0.0000

Block 5:

n	V_6	I_{G1_Q}	$g_1()$	$g_{G1\_q}()$
0	0.0000	0.0000	0.1750	0.0000
1	0.2828	-0.1167	0.000e-8	-0.000e-8

D-3: Summary of Results

	Bus 1	
Bus Voltage Magnitude	1.05 PU	
Bus Voltage Angle	0.00 rad	
G1 Real/Reactive Power	-0.4125 PU	-0.1167 PU
G2 Real/Reactive Power	-0.2063 PU	-0.0583 PU
T12 Real/Reactive Power	0.6188 PU	0.1750 PU
	Bus 2	
Bus Voltage Magnitude	0.9933 PU	
Bus Voltage Angle	-0.1105 rad	
L3 Real/Reactive Power	0.6000 PU	0.1000 PU
T12 Real/Reactive Power	-0.6000 PU	-0.1000 PU
	Information Node 5 (*p*)	
Magnitude	0.4125	
	Information Node 6 (*q*)	
Magnitude	0.2828	

Appendix E: Waveform Examples

E-1 Examples of Waveform Types

While the possibilities of waveform definitions is endless, this thesis will concentrate on the following waveform types:

Waveform Type	Code
Undefined	0
Data Series	1
Fourier Series	2
Legendre Series	3
Polynomials	4
Matlab Polynomials	5
Chebyshev Series	6

The code in the above table refers to the value of the **`type`** element in the **`WAVEFORM`** structure.

E-1.1 Data Series

A data series consists of ***n*** equally spaced samples of the waveform stored in an array of double precision floating point numbers. The first coefficient is associated with the value of the waveform at the beginning of the time interval and the last coefficient is associated with the value of the waveform at the end of the time interval. Each element of the array is given by:

$$c_i = f(t_i)$$

$$t_i = t_0 + \frac{i-1}{n-1}(t_1 - t_0)$$

$$i = 1, 2, \ldots, n$$

A data series representation is primarily used for plotting the time history of variables and for calculating waveform operators which would prove difficult with other waveform types.

E-1.2 Polynomial Expansion

A polynomial expansion consists of ***n*** coefficients of a polynomial representation of the waveform normalized over the interval **[-1 1]**.

$$f(x) = \sum_{i=1}^{n} c_i x^{i-1}$$

$$x = -1 + 2\frac{t - t_0}{t_1 - t_0}$$

Polynomial expansions are useful for evaluating switching operators described above.

A *Matlab* Polynomial expansion is expressed in descending order:

$$f(x) = \sum_{i=1}^{n} c_i x^{n-i}$$

E-1.3 Orthogonal Function Series

Orthogonal Function Series can be an excellent means for representing waveforms. In an orthogonal series representation, the value of the coefficient of a given order of the characteristic function is independent of the number of terms in the orthogonal series. This means truncating an orthogonal series by eliminating higher order coefficients will still result in the best possible fit with the remaining coefficients.

In general, an orthogonal series representation is of the form:

$$f(x) = \sum_{i=1}^{n} c_i F_{i-1}(x)$$

$$x \in [x_0 \quad x_1]$$

where $\boldsymbol{F_i(x)}$ is the *i*th order characteristic function of the orthogonal function series with respect to the weighting function $\boldsymbol{r(x)}$. These characteristic functions observe the following property:

$$\int_{x_0}^{x_1} r(x)F_m(x)F_n(x)dx = 0 \quad \text{for} \quad m \neq n$$

$$\int_{x_0}^{x_1} r(x)F_m(x)F_m(x)dx = G(m)$$

With this property, the coefficients $\boldsymbol{c_i}$ of the series can be found:

$$c_i = \frac{1}{G(i-1)} \int_{x_0}^{x_1} r(x)f(x)F_{i-1}(x)dx$$

E-1.3.1 Fourier Series

Perhaps the most widely used orthogonal function series is the Fourier Series. Unfortunately, the Fourier Series is unsuitable for dynamic simulations. To see why, one need only look at the manner in which a function is expressed in a Fourier Series:

$$f(x) = A_0 + \sum_{i=1}^{n} A_i \cos(i\pi x) + \sum_{i=1}^{n} B_i \sin(i\pi x)$$

$$x = -1 + 2\frac{t - t_0}{t_1 - t_0}$$

Notice that at $\boldsymbol{x = 1}$ and $\boldsymbol{x = -1}$ $\boldsymbol{\sin(i\pi x) = 0}$ and $\boldsymbol{\cos(i\pi x) = (-1)^i}$. Consequently **f(1) = f(-1)**. In other words, the starting value and ending value of any waveform represented by a fourier series is forced to be identical. In dynamic simulations however, we often have equations of the form:

$$\frac{dy}{dt} = \mathrm{f}(z, t)$$

This equation is normally evaluated by integration:

$$y = y_0 + \int_{t_0}^{t} \mathrm{f}(z, \tau)d\tau$$

where y_0 is the value of the waveform y at the beginning of the interval. If y is represented by a Fourier Series, then y evaluated at the end of the interval will also be y_0. In other words, while the value of a state variable may change within the interior of a time interval, at the boundaries, the value is constrained to be a constant independent of the length of the time interval. This constraint is artificial and not a property of real physical systems.

E-1.3.2 Legendre Series

Legendre Series use legendre polynomials to form the basis of an orthogonal function series over the interval **[-1 1]**. Legendre polynomials $L_i(x)$ of order i are defined by the following equations:

$$L_i(x) = \frac{u_i(x)}{u_i(1)} \quad \text{for} \quad i \quad even$$

$$L_i(x) = \frac{v_i(x)}{v_i(1)} \quad \text{for} \quad i \quad odd$$

$$u_i(x) = 1 - \frac{i(i+1)}{2!}x^2 + \frac{i(i-2)(i+1)(i+3)}{4!}x^4 - \frac{i(i-2)(i-4)(i+1)(i+3)(i+5)}{6!}x^6 + \ldots$$

$$v_i(x) = x - \frac{(i-1)(i+2)}{3!}x^3 + \frac{(i-1)(i-3)(i+2)(i+4)}{5!}x^5 -$$

$$\frac{(i-1)(i-3)(i-5)(i+2)(i+4)(i+6)}{7!}x^7 + \ldots$$

The first six Legendre polynomials are readily found to be:

$$L_0(x) = 1$$

$$L_1(x) = x$$

$$L_2(x) = \frac{1}{2}(3x^2 - 1)$$

$$L_3(x) = \frac{1}{2}(5x^3 - 3x)$$

$$L_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$$

$$L_5(x) = \frac{1}{8}(63x^5 - 70x^3 + 15x)$$

Legendre Series also obey the following recursion formula

$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x)$$

An *n*th order legendre series representation of a waveform is given by:

$$f(x) = \sum_{i=1}^{n} c_i L_{i-1}(x)$$

where:

$$x_0 = -1$$

$$x_1 = 1$$

$$r(x) = 1$$

$$F_i(x) = L_i(x)$$

$$G(i) = \frac{2}{2i+1}$$

The time interval **[t_0 t_1]** can be mapped to the interval **[-1 1]** with the following transformation:

$$x = -1 + 2\frac{t - t_0}{t_1 - t_0}$$

The coefficients $\boldsymbol{c_i}$ can be found by integration:

$$c_i = \frac{2i-1}{2}\int_{-1}^{1} f(x)L_{i-1}(x)dx$$

E-1.3.3 Chebyshev Series

Chebyshev Series use Chebyshev polynomials to form the basis of an orthogonal function series over the interval **[-1 1]**. Chebyshev polynomials $\boldsymbol{T_i(x)}$ of order $\boldsymbol{i}$ are defined by the following equations:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x) \quad \text{for} \quad i \geq 1$$

The following three Chebyshev polynomials are given by:

$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

An ***n***th order Chebyshev Series representation of a waveform is given by:

$$f(x) = \sum_{i=1}^{n} c_i T_{i-1}(x)$$

where

$$x_0 = -1$$

$$x_1 = 1$$

the weighting function ***r*(*x*)** is given by:

$$r(x) = \frac{1}{\sqrt{1 - x^2}}$$

and:

$$F_i(x) = T_i(x)$$

$$G(0) = \pi$$

$$G(m) = \frac{\pi}{2} \quad \text{for} \quad m > 0$$

E-2 Waveform Conversions

This section describes how to convert a waveform consisting of a vector of coefficients of order n_1 to a waveform of possibly a different type composed of a vector of coefficients of another order n_2. In all cases, the conversion is a linear matrix operator. Hence for given values of n_1 and n_2, the conversion matrix need only be calculated once.

From here on, $L_i(x)$ refers to a vector containing the polynomial coefficients of the ith order Legendre Polynomial. $L_i(x_j)$ refers to the ith order Legendre Polynomial evaluated at x_j. Likewise, $T_i(x)$ refers to a vector containing the polynomial coefficients of the ith order Chebyshev Polynomial. $T_i(x_j)$ refers to the ith order Chebyshev Polynomial evaluated at x_j.

E-2.1 Legendre Series

E-2.1.1 Legendre Series to Data Series

Converting a Legendre Series of order n_1 to a data series of order n_2 requires the construction of the following matrix:

$$A_{LD} = \begin{bmatrix} 1 & L_1(x_0) & L_2(x_0) & \dots & L_{n_1-1}(x_0) \\ 1 & L_1(x_1) & L_2(x_1) & \dots & L_{n_1-1}(x_1) \\ 1 & L_1(x_2) & L_2(x_2) & \dots & L_{n_1-1}(x_2) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & L_1(x_{n_2-1}) & L_2(x_{n_2-1}) & \dots & L_{n_1-1}(x_{n_2-1}) \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_2 - 1}$$

If C_l is the vector of the Legendre Series coefficients and C_d is the vector of data series points, the following relation holds:

$$C_d = A_{LD} C_l$$

E-2.1.2 Legendre Series to Legendre Series

Converting a Legendre Series of order n_1 to order n_2 requires only the truncation of terms if $n_1 > n_2$ or the insertion of zeros in the higher order terms if $n_1 < n_2$.

E-2.1.3 Legendre Series to Chebyshev Series

Converting a Legendre Series of order $\boldsymbol{n_1}$ to a Chebyshev Series of order $\boldsymbol{n_2}$ first requires the truncation or padding with zeros of the Legendre series to order $\boldsymbol{n_2}$. The resulting Legendre Series should then be multiplied by the following upper trianglular matrix:

$$A_{LT} = A_T^{-1} A_L$$

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \ldots \quad L_{n_2-1}(x)]$$

$$A_T = [T_0(x) \quad T_1(x) \quad T_2(x) \quad \ldots \quad T_{n_2-1}(x)]$$

where $\boldsymbol{L_i(x)}$ is a vector of order $\boldsymbol{n_2}$ holding the polynomial coefficients of the *i*th order Legendre Polynomial and $\boldsymbol{T_i(x)}$ is a vector of order $\boldsymbol{n_2}$ holding the polynomial coefficients of the *i*th order Chebyshev Polynomial.

E-2.1.4 Legendre Series to Polynomial Expansion

Converting a Legendre Series of order $\boldsymbol{n_1}$ to a polynomial expansion of order $\boldsymbol{n_2}$ first requires the truncation or padding with zeros of the Legendre series to order $\boldsymbol{n_2}$. The resulting Legendre Series vector should then be multiplied by the following upper triangular matrix:

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \ldots \quad L_{n_2-1}(x)]$$

where $\boldsymbol{L_i(x)}$ is a vector of order $\boldsymbol{n_2}$ holding the polynomial coefficients of the *i*th order Legendre Polynomial.

E-2.2 Chebyshev Series

E-2.2.1 Chebyshev Series to Data Series

Converting a Chebyshev Series of order $\boldsymbol{n_1}$ to a data series of order $\boldsymbol{n_2}$ requires the construction of the following matrix:

$$A_{TD} = \begin{bmatrix} 1 & T_1(x_0) & T_2(x_0) & \dots & T_{n_1-1}(x_0) \\ 1 & T_1(x_1) & T_2(x_1) & \dots & T_{n_1-1}(x_1) \\ 1 & T_1(x_2) & T_2(x_2) & \dots & T_{n_1-1}(x_2) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & T_1(x_{n_2-1}) & T_2(x_{n_2-1}) & \dots & T_{n_1-1}(x_{n_2-1}) \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_2 - 1}$$

If $\boldsymbol{C_t}$ is the vector of the Chebyshev Series coefficients and $\boldsymbol{C_d}$ is the vector of data series points, the following relation holds:

$$C_d = A_{TD} C_t$$

E-2.2.2 Chebyshev Series to Legendre Series

Converting a Chebyshev Series of order $\boldsymbol{n_1}$ to a Legendre Series of order $\boldsymbol{n_2}$ first requires the truncation or padding with zeros of the Chebyshev series to order $\boldsymbol{n_2}$. The resulting Chebyshev Series should then be multiplied by the following upper trianglular matrix:

$$A_{LT} = A_L^{-1} A_T$$

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \dots \quad L_{n_2-1}(x)]$$

$$A_T = [T_0(x) \quad T_1(x) \quad T_2(x) \quad \dots \quad T_{n_2-1}(x)]$$

where $\boldsymbol{L_i(x)}$ is a vector of order $\boldsymbol{n_2}$ holding the polynomial coefficients of the *i*th order Legendre Polynomial and $\boldsymbol{T_i(x)}$ is a vector of order $\boldsymbol{n_2}$ holding the polynomial coefficients of the *i*th order Chebyshev Polynomial.

E-2.2.3 Chebyshev Series to Chebyshev Series

Converting a Chebyshev Series of order n_1 to order n_2 requires only the truncation of terms if $n_1 > n_2$ or the insertion of zeros in the higher order terms if $n_1 < n_2$.

E-2.2.4 Chebyshev Series to Polynomial Expansion

Converting a Chebyshev Series of order n_1 to a polynomial expansion of order n_2 first requires the truncation or padding with zeros of the Chebyshev series to order n_2. The resulting Chebyshev Series vector should then be multiplied by the following upper triangular matrix:

$$A_T = [T_0(x) \quad T_1(x) \quad T_2(x) \quad \dots \quad T_{n_2-1}(x)]$$

where $T_i(x)$ is a vector of order n_2 holding the polynomial coefficients of the ith order Chebyshev Polynomial.

E-2.3 Polynomial Expansion

E-2.3.1 Polynomial Expansion to Data Series

Converting a polynomial expansion of order $\boldsymbol{n_1}$ to a data series of order $\boldsymbol{n_2}$ requires the construction of the following matrix:

$$A_{PD} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n_1-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n_1-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n_1-1} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & x_{n_2-1} & x_{n_2-1}^2 & \dots & x_{n_2-1}^{n_1-1} \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_2 - 1}$$

If $\boldsymbol{C_p}$ is the vector of the polynomial coefficients and $\boldsymbol{C_d}$ is the vector of data series points, the following relation holds:

$$C_d = A_{PD} C_p$$

E-2.3.2 Polynomial Expansion to Legendre Series

Converting a polynomial expansion of order $\boldsymbol{n_1}$ to a Legendre Series of order $\boldsymbol{n_2}$ requires first converting to a Legendre series of order $\boldsymbol{n_1}$ then converting the Legendre Series to order $\boldsymbol{n_2}$. Recall that the matrix for converting from a Legendre Series to a Polynomial is upper triangular. Hence one only needs to use backward substitution to solve for the Legendre Series coefficients:

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \dots \quad L_{n_2-1}(x)]$$

$$C_p = A_L C_l$$

E-2.3.3 Polynomial Expansion to Chebyshev Series

Converting a polynomial expansion of order $\boldsymbol{n_1}$ to a Chebyshev Series of order $\boldsymbol{n_2}$ requires first converting to a Chebyshev series of order $\boldsymbol{n_1}$ then converting the Chebyshev

Series to order $\boldsymbol{n_2}$. Recall that the matrix for converting from a Chebyshev Series to a Polynomial is upper triangular. Hence one only needs to use backward substitution to solve for the Chebyshev Series coefficients:

$$A_T = [T_0(x) \quad T_1(x) \quad T_2(x) \quad \ldots \quad T_{n_2-1}(x)]$$

$$C_p = A_T C_t$$

E-2.3.4 Polynomial Expansion to Polynomial Expansion

Converting a polynomial expansion to another polynomial expansion of higher order only requires setting the higher order terms to zero. Converting to a lower number of terms requires more effort. The best method is to convert to an orthogonal function series, truncate, and convert back. Since all of these operations are linear matrix operations, the conversion matrix need only be calculated once. For this conversion, either the Legendre Series or the Chebyshev series would be appropriate since the type conversions to and from the series solution does not add any truncation error (The truncation error is solely due to the truncation of the Legendre Series or Chebyshev series and not due to the conversions).

E-2.4 Data Series

E-2.4.1 Data Series to Data Series

There are many methods for converting a data series to another data series with a different number of coefficients. Two common interpolation schemes for performing this conversion are linear interpolation and cubic splines. These methods can be found in many numerical methods textbooks and will not be described here.

E-2.4.2 Data Series to Legendre Series

If $\boldsymbol{n_1 \geq n_2}$, a Data Series can be converted to a Legendre Series by taking the pseudo-inverse of the matrix converting a Legendre Series to a Data Series. If $\boldsymbol{n_1 < n_2}$, the Data Series can be converted in a similar manner to a Legendre Series of order $\boldsymbol{n_1}$ padded with zeros to order $\boldsymbol{n_2}$.

$$A_{LD} = \begin{bmatrix} 1 & L_1(x_0) & L_2(x_0) & \dots & L_{n_2-1}(x_0) \\ 1 & L_1(x_1) & L_2(x_1) & \dots & L_{n_2-1}(x_1) \\ 1 & L_1(x_2) & L_2(x_2) & \dots & L_{n_2-1}(x_2) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & L_1(x_{n_1-1}) & L_2(x_{n_1-1}) & \dots & L_{n_2-1}(x_{n_1-1}) \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_1 - 1}$$

If $\boldsymbol{C_d}$ is the vector of data series points and $\boldsymbol{C_l}$ is the vector of Legendre Series Coefficients, the following relation holds:

$$C_d = A_{LD} C_l$$

$$C_l = (A_{LD}^T A_{LD})^{-1} A_{LD}^T C_d$$

E-2.4.3 Data Series to Chebyshev Series

Converting a Data Series to a Chebyshev Series can be done in the same manner as the conversion to a Legendre Series:

$$A_{TD} = \begin{bmatrix} 1 & T_1(x_0) & T_2(x_0) & \dots & T_{n_2-1}(x_0) \\ 1 & T_1(x_1) & T_2(x_1) & \dots & T_{n_2-1}(x_1) \\ 1 & T_1(x_2) & T_2(x_2) & \dots & T_{n_2-1}(x_2) \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & T_1(x_{n_1-1}) & T_2(x_{n_1-1}) & \dots & T_{n_2-1}(x_{n_1-1}) \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_1 - 1}$$

If $\boldsymbol{C_d}$ is the vector of data series points and $\boldsymbol{C_c}$ is the vector of Chebyshev Series Coefficients, the following relation holds:

$$C_d = A_{TD} C_c$$

$$C_c = (A_{TD}^T A_{TD})^{-1} A_{TD}^T C_d$$

E-2.4.4 Data Series to Polynomial Expansion

Converting a data series to a polynomial expansion of equal or less order using a least squares fit is a straight forward process. If the number of points in the data series $\boldsymbol{n_1}$ is equal to or less than the number of points in the polynomial $\boldsymbol{n_2}$, the resulting polynomial will pass through each point of the data series. If larger, the polynomial will not necessarily pass through all of the data series points, but will be a least square approximation.

For $\boldsymbol{n_1 \le n_2}$:

For this case, the problem is to solve for the coefficients of the polynomial $\boldsymbol{c_{pi}}$ for $\boldsymbol{i \le n_1}$. For the higher coefficients ($\boldsymbol{i > n_1}$), $\boldsymbol{c_{pi} = 0}$. In the following discussion, let $\boldsymbol{C_d}$ be the vector of $\boldsymbol{n_1}$ data series coefficients and $\boldsymbol{C_p}$ be the vector containing the first $\boldsymbol{n_1}$ polynomial coefficients. Define the $\boldsymbol{n_1 \times n_1}$ matrix $\boldsymbol{A}$ as follows:

$$A_{PD} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n_1-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n_1-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n_1-1} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & x_{n_1-1} & x_{n_1-1}^2 & \dots & x_{n_1-1}^{n_1-1} \end{bmatrix}$$

$$x_i = -1 + 2\frac{i}{n_1 - 1}$$

Matrix $\boldsymbol{A_{PD}}$ is square, clearly has rank $\boldsymbol{n_1}$, and therefore is invertible. Consequently solving for $\boldsymbol{C_p}$ is straight forward:

$$A_{PD} C_p = C_d$$

$$C_p = A_{PD}^{-1} C_d$$

For $\boldsymbol{n_1 > n_2}$:

If the number of data points is greater than the number of polynomials, the number of columns in the $\boldsymbol{A_{PD}}$ matrix described above would have $\boldsymbol{n_2}$ columns and $\boldsymbol{n_1}$ rows. $\boldsymbol{A_{PD}}$ would clearly not be invertible. The pseudo-inverse of $\boldsymbol{A_{PD}}$ can be calculated and provides the least squares fit of the data series:

$$C_p = (A_{PD}^T A_{PD})^{-1} A_{PD}^T C_d$$

E-3 Waveform Arithmetic

This section describes how to perform addition, subtraction, multiplication, and division on the various types of waveforms.

E-3.1 Data Series

Performing waveform arithmetic on data series is very easy. The waveforms are converted to the proper size and then added, subtracted, multiplied or divided element by element.

E-3.2 Polynomials

E-3.2.1 Addition/Subtraction

Adding or subtracting two polynomial waveforms simply entails converting the two waveforms to the proper length and adding or subtracting element by element.

E-3.2.2 Multiplication

Multiplying polynomial waveform $\boldsymbol{W}$ of size $\boldsymbol{n_w}$ and $\boldsymbol{Y}$ of size $\boldsymbol{n_y}$ together to get polynomial $\boldsymbol{Z}$ of size $\boldsymbol{n_w + n_y - 1}$ can be accomplished by constructing the following matrix of size $\boldsymbol{n_w + n_y - 1 \times n_w}$:

$$M_p = \begin{bmatrix} Y_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ Y_2 & Y_1 & 0 & 0 & \dots & 0 & 0 \\ Y_3 & Y_2 & Y_1 & 0 & \dots & 0 & 0 \\ Y_4 & Y_3 & Y_2 & Y_1 & \dots & 0 & 0 \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ 0 & 0 & 0 & 0 & \dots & Y_{n_y} & Y_{n_y-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & Y_{n_y} \end{bmatrix}$$

$$Z = M_p W$$

$\boldsymbol{Z}$ can now be truncated or padded with zeros to convert it to the proper length. The truncation of a polynomial is discussed in section E-2.3.4.

E-3.2.3 Division

Dividing two polynomial expansions can be difficult, particularly if the denominator polynomial has one or more zero crossings. In general, there is no simple method for performing the division, although the recursion process described in this section will work. Define the problem to be:

$$\sum_{i=1}^{n_y} Y_i x^{i-1} = \frac{\sum_{j=1}^{n_b} B_j x^{j-1}}{\sum_{k=1}^{n_c} C_k x^{k-1}}$$

There are two parts to the problem. The first task is to use synthetic division until the numerator of the remainder is of size $\boldsymbol{n_c}$ **-1** or less. The second task is to convert the remaining fraction into another polynomial expansion by a process similar to synthetic division, but proceeding from the constant term and working up in order.

Synthetic division is the process of dividing one polynomial by another until the remainder is of order 1 less than the denominator.

$$\frac{\sum_{l=1}^{n_d} d_l x^{l-1}}{\sum_{k=1}^{n_c} c_k x^{k-1}} = \frac{d_{n_d}}{c_{n_c}} x^{n_d - n_c} + \frac{\sum_{l=1}^{n_d-1} r_l x^{l-1}}{\sum_{k=1}^{n_c} c_k x^{k-1}}$$

$$r_l = d_l - \frac{d_{n_d}}{c_{n_c}} c_l$$

$$y_{f(n_d - n_c + 1)} = \frac{d_{n_d}}{c_{n_c}}$$

Initially, $\boldsymbol{d_i}$ is set equal to $\boldsymbol{b_i}$. After the first iteration, $\boldsymbol{d_i}$ is set equal to the remainder $\boldsymbol{r_l}$. The process is repeated until $\boldsymbol{n_d = n_c - 1}$. At this point, the direction of the division is reversed and we get:

$$\frac{\sum_{l=1}^{n_d} d_l x^{l-1}}{\sum_{k=1}^{n_c} c_k x^{k-1}} = \frac{d_1}{c_1} + x \frac{\sum_{l=1}^{n_d-1} r_l x^{l-1}}{\sum_{k=1}^{n_c} c_k x^{k-1}}$$

$$r_{l-1} = d_l - \frac{d_1}{c_1} c_l$$

In this manner, we can express the remaining fraction as another polynomial expansion. The actual values for $\boldsymbol{Y_i}$ are equal to the sum of the components from the forward and backwards synthetic division.

Note that if the denominator has a zero over the interval **[-1,1]**, the backwards synthetic division will result in a diverging series.

E-3.3 Legendre Series

E-3.3.1 Addition/Subtraction

Adding or subtracting two Legendre Series waveforms simply entails converting the two waveforms to the proper length and adding or subtracting element by element.

E-3.3.2 Multiplication

Multiplying two Legendre Series together can be accomplished in two ways. The first way is to convert the Legendre Series to polynomial expansions, multiply the two together, then convert the product to the Legendre Series of the proper size. The second method uses the recursion formula for the Legendre series to assist in the process:

To multiply Legendre Series $\boldsymbol{Y}$ of size $\boldsymbol{n_y}$ by the Legendre Series $\boldsymbol{W}$ of size $\boldsymbol{n_w}$ to obtain the Legendre Series $\boldsymbol{Z}$ of size $\boldsymbol{n_z = n_y + n_w - 1}$, $\boldsymbol{Y}$ must first be converted to a polynomial expansion $\boldsymbol{Y_p}$ of size $\boldsymbol{n_y}$:

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \ldots \quad L_{n_y-1}(x)]$$

$$Y_p = A_L Y$$

The recursion formula for the Legendre Series is given by:

$$xL_i(x) = \left(\frac{i}{2i+1} \right) L_{i-1}(x) + \left(\frac{i+1}{2i+1} \right) L_{i+1}(x)$$

which can be translated into the following matrix for multiplying a given Legendre Series of size $\boldsymbol{n_z}$ by $\boldsymbol{x}$:

$$A_{XL} = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & \frac{2}{5} & 0 & \dots & 0 & 0 \\ 0 & \frac{2}{3} & 0 & \frac{3}{7} & \dots & 0 & 0 \\ 0 & 0 & \frac{3}{5} & 0 & \dots & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \dots & \frac{n_z - 2}{2(n_z - 2) + 1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \frac{n_z - 1}{2(n_z - 1) + 1} \\ 0 & 0 & 0 & 0 & \dots & \frac{n_z - 1}{2(n_z - 2) + 1} & 0 \end{bmatrix}$$

If we define the vector $\boldsymbol{Y_{p1}}$ to be $\boldsymbol{Y_p}$ padded with zeros such that it is of size $\boldsymbol{n_z}$, we can define the following $\boldsymbol{n_z} \times \boldsymbol{n_w}$ matrix:

$$A_{mpl} = \begin{bmatrix} Y_{p1} & A_{XL}Y_{p1} & A_{XL}A_{XL}Y_{p1} & \dots & A_{XL}^{n_w - 1}Y_{p1} \end{bmatrix}$$

The final $\boldsymbol{n_z} \times \boldsymbol{n_w}$ multiplication matrix $\boldsymbol{A_{mll}}$ can now be found:

$$A_{mll} = A_{mpl}A_L$$

$$Z = A_{mll}W$$

Of course $\boldsymbol{Z}$ may have to be truncated or padded with zeros to convert it to the desired length.

E-3.3.3 Division

There is no straight forward method for dividing two legendre series. The easiest way appears to be converting to polynomial expansions, performing the divison, then converting back to the legendre series.

E-3.4 Chebyshev Series

E-3.4.1 Addition/Subtraction

Adding or subtracting two Chebyshev Series waveforms simply entails converting the two waveforms to the proper length and adding or subtracting element by element.

E-3.4.2 Multiplication

Multiplying two Chebyshev Series together can be accomplished in two ways. The first way is to convert the Chebyshev Series to polynomial expansions, multiply the two together, then convert the product to the Chebyshev Series of the proper size. The second and preferred method uses an alternate definition of a Chebyshev Polynomial to assist in the process:

$$T_n(x) = \cos(n \cos^{-1}(x))$$

$$T_{-n}(x) = T_n(x)$$

From this definition, the product of two Chebyshev Polynomials can easily be derived:

$$T_n(x)T_m(x) = \cos(n \cos^{-1}(x)) \cos(m \cos^{-1}(x))$$

$$T_n(x)T_m(x) = \frac{1}{2}(\cos((n+m)\cos^{-1}(x)) + \cos((n-m)\cos^{-1}(x)))$$

$$T_n(x)T_m(x) = \frac{1}{2}(T_{n+m}(x) + T_{n-m}(x))$$

To multiply Chebyshev Series $\boldsymbol{Y}$ of size $\boldsymbol{n_y}$ by the Chebyshev Series $\boldsymbol{W}$ of size $\boldsymbol{n_w}$ to obtain the Chebyshev Series $\boldsymbol{Z}$ of size $\boldsymbol{n_z = n_y + n_w - 1}$, three $\boldsymbol{n_z \times n_w}$ matrices should first be constructed:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ . \\ . \\ . \\ Y_{n_y} \end{bmatrix}$$

$$A_{mt1} = \begin{bmatrix} Y_1 & 0 & 0 & \dots & 0 & 0 \\ Y_2 & Y_1 & 0 & \dots & 0 & 0 \\ Y_3 & Y_2 & Y_1 & \dots & 0 & 0 \\ . & . & . & & . & . \\ . & . & . & & . & . \\ . & . & . & & . & . \\ 0 & 0 & 0 & \dots & Y_{n_y} & Y_{n_y-1} \\ 0 & 0 & 0 & \dots & 0 & Y_{n_y} \end{bmatrix}$$

$$A_{mt2} = \begin{bmatrix} Y_1 & Y_2 & Y_3 & \dots \\ Y_2 & Y_3 & Y_4 & \dots \\ Y_3 & Y_4 & Y_5 & \dots \\ . & . & . & \\ . & . & . & \\ . & . & . & \end{bmatrix}$$

$$A_{mt3} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & Y_1 & Y_2 & Y_3 & \dots \\ 0 & 0 & Y_1 & Y_2 & \dots \\ 0 & 0 & 0 & Y_1 & \dots \\ . & . & . & . & \\ . & . & . & . & \\ . & . & . & . & \end{bmatrix}$$

The final $\boldsymbol{n_z \times n_w}$ multiplication matrix $\boldsymbol{A_{mtt}}$ is given by:

$$A_{mtt} = \frac{1}{2}(A_{mt1} + A_{mt2} + A_{mt3})$$

$$Z = A_{mtt} W$$

Of course ***Z*** may have to be truncated or padded with zeros to convert it to the desired length.

E-3.4.3 Division

There is no straight forward method for dividing two chebyshev series. The easiest way appears to be converting to polynomial expansions, performing the divison, then converting back to the chebyshev series.

E-4 Waveform Functions

E-4.1 Data Series

E-4.1.1 Trigonometric and Exponential Functions

All trigonometric and exponential functions can be performed point by point on the data series coefficients.

E-4.1.2 Integration and Differentiation

There are a number of techniques for integrating or differentiating Data Series. All are by their nature approximations and can suffer from numerical instability problems associated with conventional simulations. One simple method of integration employs the trapezoidal rule:

$$S_{DD} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ \frac{h}{2} & \frac{h}{2} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ \frac{h}{2} & h & \frac{h}{2} & 0 & 0 & \dots & 0 & 0 & 0 \\ \frac{h}{2} & h & h & \frac{h}{2} & 0 & \dots & 0 & 0 & 0 \\ \frac{h}{2} & h & h & h & \frac{h}{2} & \dots & 0 & 0 & 0 \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ \frac{h}{2} & h & h & h & h & \dots & \frac{h}{2} & 0 & 0 \\ \frac{h}{2} & h & h & h & h & \dots & h & \frac{h}{2} & 0 \\ \frac{h}{2} & h & h & h & h & \dots & h & h & \frac{h}{2} \end{bmatrix}$$

$$h = \frac{2}{n-1}$$

With this matrix, the integral equation:

$$Y = Y_0 + \int_{\tau=-1}^{x} W d\tau$$

Becomes the matrix operation:

$$Y = S_{DD} W + Y_0$$

The vector ***Y*** may be now converted to a different length if so desired.

Differentiating a Data series can be done in a number of ways. The secant method can be easily implemented with the following matrix:

$$A_{DD1} = \frac{1}{h}\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & \dots & 0 & 0 & 0 \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ . & . & . & . & . & & . & . & . \\ 0 & 0 & 0 & 0 & 0 & \dots & -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix}$$

$$h = \frac{1}{n-1}$$

Another approach is to choose a differentiation matrix such that it is consistent with the integration matrix. Consistency is defined by the following matrix equation:

$$S_{DD} A_{DD2} = M$$

where

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ -1 & 1 & 0 & 0 & \dots \\ -1 & 0 & 1 & 0 & \dots \\ -1 & 0 & 0 & 1 & \dots \\ . & . & . & . & \\ . & . & . & . & \\ . & . & . & . & \end{bmatrix}$$

The ***M*** matrix reflects the fact that differentiating a data series will destroy the subsequent constant of integration. Since $\boldsymbol{S_{DD}}$ is generally singular, only its pseudo-inverse can be taken:

$$D_{DD2} = (S_{DD}^T S_{DD})^{-1} S_{DD}^T M$$

This matrix actually has a very simple construction:

$$D_{DD2} = \frac{1}{h}\begin{bmatrix} d_{11} & \frac{2n-3}{n} & -\frac{2n-5}{n} & \frac{2n-7}{n} & -\frac{2n-9}{n} & \dots \\ -\frac{1}{n} & d_{22} & \frac{2n-5}{n} & -\frac{2n-7}{n} & \frac{2n-9}{n} & \dots \\ \frac{1}{n} & -\frac{3}{n} & d_{33} & \frac{2n-7}{n} & -\frac{2n-9}{n} & \dots \\ -\frac{1}{n} & \frac{3}{n} & -\frac{5}{n} & d_{44} & \frac{2n-9}{n} & \dots \\ \frac{1}{n} & -\frac{3}{n} & \frac{5}{n} & -\frac{7}{n} & d_{55} & \dots \\ . & . & . & . & & \\ . & . & . & . & & \\ . & . & . & . & & \end{bmatrix}$$

$\boldsymbol{d_{xx}}$ is equal to the the negative sum of all the other terms in the ***x***th row. As a consequence, all of the row sums of $\boldsymbol{D_{DD2}}$ are equal to zero and $\boldsymbol{D_{DD2}}$ is singular.

E-4.1.3 Switching Functions

All switching functions can be performed point by point on the data series coefficients.

E-4.1.4 Waveform Smoothing

There are times when it may be desirable to remove high spectral content features of a waveform. One way to do this is to replace the value at each point in the time domain by the average of the waveform over some interval **[x - Δ,x + Δ]**. This can be accomplished by defining the following:

$$n_\Delta = \text{int}\left(\frac{\Delta(n-1)}{2} \right)$$

where **int(*x*)** is the integer nearest ***x***.

$$A_{smth} = \begin{bmatrix} \frac{1}{n_\Delta} & \frac{1}{n_\Delta} & \frac{1}{n_\Delta} & \cdots & \frac{1}{n_\Delta} & 0 & 0 & \cdots \\ \frac{1}{n_\Delta+1} & \frac{1}{n_\Delta+1} & \frac{1}{n_\Delta+1} & \cdots & \frac{1}{n_\Delta+1} & \frac{1}{n_\Delta+1} & 0 & \cdots \\ \frac{1}{n_\Delta+2} & \frac{1}{n_\Delta+2} & \frac{1}{n_\Delta+2} & \cdots & \frac{1}{n_\Delta+2} & \frac{1}{n_\Delta+2} & \frac{1}{n_\Delta+2} & \cdots \\ . & . & . & & . & . & . & \\ . & . & . & & . & . & . & \\ . & . & . & & . & . & . & \\ \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \cdots & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \cdots \\ 0 & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \cdots & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \frac{1}{2n_\Delta} & \cdots \\ . & . & . & & . & . & . & \\ . & . & . & & . & . & . & \\ . & . & . & & . & . & . & \end{bmatrix}$$

Multiplying a data series by $\boldsymbol{A_{smth}}$ will return a smoothed version of the data series.

E-4.2 Polynomial Expansion

E-4.2.1 Trigonometric and Exponential Functions

There is often no direct way of evaluating a trigonometric or expontential function of a polynomial expansion. Instead, the function is performed on a data series converted from the argument polynomial. The resulting polynomial is then reconverted back into a polynomial.

E-4.2.2 Integration and Differentiation

Integrating a polynomial $\boldsymbol{Y}$ of size $\boldsymbol{n_y}$ results in another polynomial $\boldsymbol{Z}$ of size $\boldsymbol{n_z = n_y + 1}$. The $\boldsymbol{n_z \times n_y}$ integration matrix $\boldsymbol{S_{DP}}$ is given by:

$$S_{DP} = \begin{bmatrix} -1 & \frac{1}{2} & -\frac{1}{3} & \frac{1}{4} & \cdots & \frac{(-1)^{n_y-1}}{n_y-1} & \frac{(-1)^{n_y}}{n_y} \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{n_y-1} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{n_y} \end{bmatrix}$$

The integral is evaluated by:

$$Z = S_{DP}Y + Z_0$$

Of course, $\boldsymbol{Z}$ may be converted to a polynomial of a different size if desired.

Differentiating a polynomial $\boldsymbol{Y}$ of size $\boldsymbol{n_y}$ results in another polynomial $\boldsymbol{Z}$ of size $\boldsymbol{n_z = n_y - 1}$. The $\boldsymbol{n_z \times n_y}$ differentiation matrix $\boldsymbol{A_{DP}}$ is given by:

$$A_{DP} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 2 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 3 & \dots & 0 & 0 \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ 0 & 0 & 0 & 0 & \dots & n_y - 2 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & n_y - 1 \end{bmatrix}$$

The Differential is evaluated by:

$$Z = A_{DP} Y$$

E-4.2.3 Switching Functions

Switching functions are those which produce a Polynomial waveform ***Y*** which is composed of ***m*** pieces of other Polynomial waveforms. Let $\boldsymbol{f_j}$ be the Polynmomial representation of the ***j***th piece of ***Y***. Let $\boldsymbol{x_0(j)}$ be the ***x*** coordinate of the ending point of the ***j***th piece where $\boldsymbol{x_0(0) = -1}$ and $\boldsymbol{x_0(m) = 1}$.

Define $\boldsymbol{Y_l}$ to be the Legendre Series representation of ***Y*** :

$$Y_l = \begin{bmatrix} Y_{l1} \\ Y_{l2} \\ Y_{l3} \\ . \\ . \\ . \\ Y_{ln} \end{bmatrix}$$

Then using the orthogonality property of the Legendre Series:

$$Y_{li} = \sum_{j=1}^{m} \int_{x_0(j-1)}^{x_0(j)} \left(\frac{2i-1}{2} \right) f_j(x) L_i(x) dx$$

Now define the following row vector:

$$l(x_0(j)) = [L_0(x_0(j)) \quad L_1(x_0(j)) \quad L_2(x_0(j)) \quad \dots \quad L_{n-1}(x_0(j)) \quad L_n(x_0(j))]$$

With $\boldsymbol{S_{DL}}$ as defined in section E-4.3.2 and $\boldsymbol{A_{mpl}()}$ as defined in section E-3.3.2 the solution for $\boldsymbol{Y_l}$ can easily be found:

$$G_l = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \dots & 0 \\ 0 & \frac{3}{2} & 0 & \dots & 0 \\ 0 & 0 & \frac{5}{2} & \dots & 0 \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 0 & 0 & 0 & \dots & \frac{2n-1}{2} \end{bmatrix}$$

$$Y_l^T = \sum_{j=1}^{m} (l(x_0(j)) - l(x_0(j-1)))S_{DL}A_{mpl}(f_j)G_l$$

Now we need only convert $\boldsymbol{Y_l}$ to a polynomial exansion $\boldsymbol{Y}$:

$$Y = A_L Y_l$$

where

$$A_L = [L_0(x) \quad L_1(x) \quad L_2(x) \quad \dots \quad L_{n_y-1}(x)]$$

E-4.2.4 Waveform Smoothing

There are times when it may be desirable to remove high spectral content features of a waveform. One way to do this is to replace the value at each point in the time domain by the average of the waveform over some interval **[x - Δ,x + Δ]**. This can be expressed by the following integral:

$$\sum_{i=1}^{n} Y_i x^{i-1} = \frac{1}{2\Delta} \int_{x-\Delta}^{x+\Delta} \sum_{j=1}^{n} W_j \tau^{j-1} d\tau$$

The only problem with the above equation is near the boundaries $\boldsymbol{x = -1}$ and $\boldsymbol{x = 1}$ where the integration interval has the possibility of crossing the boundaries and including

within the average a section of the polynomial outside the defining interval **[-1,1]**. Hence the smoothed polynomial should be composed of the following three segments (assuming $\Delta \le 1$):

$$-1 \le x < -1 + \Delta$$

$$\sum_{i=1}^{n} Y_{ai} x^{i-1} = \frac{1}{x + \Delta + 1} \int_{-1}^{x+\Delta} \sum_{j=1}^{n} W_j \tau^{j-1} d\tau$$

$$-1 + \Delta \le x \le 1 - \Delta$$

$$\sum_{i=1}^{n} Y_{bi} x^{i-1} = \frac{1}{2\Delta} \int_{x-\Delta}^{x+\Delta} \sum_{j=1}^{n} W_j \tau^{j-1} d\tau$$

$$1 - \Delta < x \le 1$$

$$\sum_{i=1}^{n} Y_{ci} x^{i-1} = \frac{1}{1 - x + \Delta} \int_{x-\Delta}^{1} \sum_{j=1}^{n} W_j \tau^{j-1} d\tau$$

Note, if $1 < \Delta < 2$ then the interval boundaries are given by:

$$[-1 \quad , \quad 1 - \Delta]$$

$$[1 - \Delta \quad , \quad -1 + \Delta]$$

$$[-1 + \Delta \quad , \quad 1]$$

If $\Delta > 2$ then there is only one interval and the average of the waveform is returned:

$$Y_1 = \frac{1}{2} \int_{-1}^{1} \sum_{i=1}^{n} W_i x^{i-1} dx$$

$$Y_i = 0 \quad \text{for} \quad i > 1$$

For $\Delta < 2$ evaluating the integrals require the definition of shifting a waveform left or right by Δ. This can be done by constructing the following binonomial matrix:

$$B_{\exp} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \dots \\ 0 & 1 & 2 & 3 & 4 & \dots \\ 0 & 0 & 1 & 3 & 6 & \dots \\ 0 & 0 & 0 & 1 & 4 & \dots \\ 0 & 0 & 0 & 0 & 1 & \dots \\ . & . & . & . & . & \\ . & . & . & . & . & \\ . & . & . & . & . & \end{bmatrix}$$

This matrix can be generated by the following recursion formula:

$$B_{\exp}(1, j) = 1$$

$$B_{\exp}(2{:}n, 1) = 0$$

$$B_{\exp}(i, j) = B_{\exp}(i, j-1) + B_{\exp}(i-1, j-1)$$

Multiplying $\boldsymbol{B}_{\mathbf{exp}}$ element by element by the following matrix B_Δ will give us the transformation matrix $\boldsymbol{B}_{\boldsymbol{shft}}$ for shifting a waveform left by Δ.

$$B_\Delta = \begin{bmatrix} 1 & \Delta & \Delta^2 & \Delta^3 & \Delta^4 & \dots \\ 0 & 1 & \Delta & \Delta^2 & \Delta^3 & \dots \\ 0 & 0 & 1 & \Delta & \Delta^2 & \dots \\ 0 & 0 & 0 & 1 & \Delta & \dots \\ 0 & 0 & 0 & 0 & 1 & \dots \\ . & . & . & . & . & \\ . & . & . & . & . & \\ . & . & . & . & . & \end{bmatrix}$$

$$\sum_{i=1}^{n} M_i x^{i-1} = \sum_{j=1}^{n} N_j (x + \Delta)^{j-1}$$

$$M = B_{shft} N$$

The tools are now all present. $\boldsymbol{W}$ can be integrated using the integration matrix $\boldsymbol{S}_{\boldsymbol{DP}}$ defined in section E-4.2.2. The limits of integration for the three segments can be applied by either using $\boldsymbol{B}_{\boldsymbol{shft}}$ for the limits involving $\boldsymbol{x}$, or by direct evaluation for those limits not involving $\boldsymbol{x}$. Dividing by the averaging interval comes next. For the first and third

intervals, the methods outlined in section E-3.2.3 can be used to divide a polynomial by another polynomial. Finally, the procedure for generating Switching Functions described in section E-4.2.3 can be used to generate the coefficients for the solution $\boldsymbol{Y}$.

E-4.3 Legendre Series

E-4.3.1 Trigonometric and Exponential Functions

There is often no direct way of evaluating a trigonometric or expontential function of a Legendre Series. Instead, the function is performed on a data series converted from the argument Legendre Series. The resulting polynomial is then reconverted back into a Legendre Series.

E-4.3.2 Integration and Differentiation

Differentiating a Legendre Series can be done easily by differentiating the recursion formula for the Legendre Series. Recall:

$$(i+1)L_{(i+1)}(x) = (2i+1)xL_i(x) - iL_{(i-1)}(x)$$

Differentiating:

$$\frac{dL_{i+1}(x)}{dx} = \left(\frac{2i+1}{i+1}\right)\left(x\frac{dL_i(x)}{dx} + L_i(x)\right) - \left(\frac{i}{i+1}\right)\frac{dL_{i-1}(x)}{dx}$$

where:

$$\frac{dL_0(x)}{dx} = 0$$

$$\frac{dL_1(x)}{dx} = 1 = L_0(x)$$

The goal is to generate the following $\boldsymbol{n} \times \boldsymbol{n}$ matrix:

$$A_{DL} = \left[\begin{array}{ccccc} \frac{dL_0(x)}{dx} & \frac{dL_1(x)}{dx} & \frac{dL_2(x)}{dx} & \cdots & \frac{dL_{n-1}(x)}{dx} \end{array}\right]$$

The columns of $\boldsymbol{A_{DL}}$ can be solved recursively once we define the matrix $\boldsymbol{A_{XL}}$ for multiplying a Legendre Series Vector by $\boldsymbol{x}$.

$$xL_i(x) = \left(\frac{i}{2i+1}\right)L_{i-1}(x) + \left(\frac{i+1}{2i+1}\right)L_{i+1}(x)$$

$$A_{XL} = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & \frac{2}{5} & 0 & \dots & 0 & 0 \\ 0 & \frac{2}{3} & 0 & \frac{3}{7} & \dots & 0 & 0 \\ 0 & 0 & \frac{3}{5} & 0 & \dots & 0 & 0 \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ 0 & 0 & 0 & 0 & \dots & \frac{n-2}{2(n-2)+1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \frac{n-1}{2(n-1)+1} \\ 0 & 0 & 0 & 0 & \dots & \frac{n-1}{2(n-2)+1} & 0 \end{bmatrix}$$

Note that the last row of $\boldsymbol{A_{XL}}$ has been elminated to make the matrix square. This will not cause any problems since in the recursion formula, the last coefficient of the vector multiplying $\boldsymbol{A_{XL}}$ is always zero.

Let $\boldsymbol{A_{DL}(:,j)}$ represent the $\boldsymbol{j}$th column of $\boldsymbol{A_{DL}}$. Let $\boldsymbol{I}$ be the $\boldsymbol{n} \times \boldsymbol{n}$ identity matrix. The recursion formula states:

$$A_{DL}(:, i+2) = \left(\frac{2i+1}{i+1} \right) (A_{XL} A_{DL}(:, i+1) + I(:, i+1)) - \left(\frac{i}{i+1} \right) A_{DL}(:, i)$$

$$1 \le i \le n-2$$

Once $\boldsymbol{A_{DL}}$ is constructed, it can be used to calculate derivatives. Let $\boldsymbol{W}$ and $\boldsymbol{Y}$ be vectors of Legendre Series coefficients of size $\boldsymbol{n}$. Then the following statements are identical:

$$Y = \frac{dW}{dx}$$

$$Y = A_{DL} W$$

Of course, the $\boldsymbol{n}$th coefficient of $\boldsymbol{Y}$ will always be zero since the $\boldsymbol{n}$th row (as well as the first column) of $\boldsymbol{A_{DL}}$ will always be populated with zeros.

Integration is a bit more complex. In general, the problem is to solve the following equation:

$$Y = Y_0 + \int_{\tau=-1}^{x} W d\tau$$

First, the $\boldsymbol{n+1} \times \boldsymbol{n}$ indefinite integral matrix $\boldsymbol{S_{IL}}$ should be found. The easiest way of generating $\boldsymbol{S_{IL}}$ begins by adding an additional column to $\boldsymbol{A_{DL}}$ using the same recursion formula to form the $\boldsymbol{n} \times \boldsymbol{n+1}$ matrix $\boldsymbol{A_{DL1}}$. $\boldsymbol{S_{IL}}$ is simply the pseudo-inverse of $\boldsymbol{A_{DL1}}$:

$$S_{IL} = (A_{DL1}^T A_{DL1})^{-1} A_{DL1}^T$$

The next step is to evaluate the integral at $\boldsymbol{x = -1}$. This can be done by multplying the following row vector by $\boldsymbol{S_{IL}}$:

$$X_{-1} = [1 \quad -1 \quad 1 \quad -1 \quad \ldots \quad (-1)^{n-1}]$$

$$S_{-1} = X_{-1} S_{IL}$$

The first row of $\boldsymbol{S_{IL}}$ contains all zeros. If this row is replaced by $\boldsymbol{-S_{-1}}$ and the resulting matrix called $\boldsymbol{S_{DL}}$, we have all the pieces for calculating the integral of a Legendre Series:

$$Y = S_{DL} W + Y_0$$

Of course, the vector $\boldsymbol{Y}$ may have to be truncated or padded with zeros as required.

E-4.3.3 Switching Functions

Switching functions are those which produce a Legendre Series waveform $\boldsymbol{Y}$ which is composed of $\boldsymbol{m}$ pieces of other Legendre Series waveforms. Let $\boldsymbol{f_j}$ be the legendre series representation of the $\boldsymbol{j}$th piece of $\boldsymbol{Y}$. Let $\boldsymbol{x_0(j)}$ be the $\boldsymbol{x}$ coordinate of the ending point of the $\boldsymbol{j}$th piece where $\boldsymbol{x_0(0) = -1}$ and $\boldsymbol{x_0(m) = 1}$.

Let:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ . \\ . \\ . \\ Y_n \end{bmatrix}$$

Then using the orthogonality property of the Legendre Series:

$$Y_i = \sum_{j=1}^{m} \int_{x_0(j-1)}^{x_0(j)} \left(\frac{2i-1}{2} \right) f_j(x) L_i(x) dx$$

Now define the following row vector:

$$l(x_0(j)) = [L_0(x_0(j)) \quad L_1(x_0(j)) \quad L_2(x_0(j)) \quad \ldots \quad L_{n-1}(x_0(j)) \quad L_n(x_0(j))]$$

With $\boldsymbol{S_{DL}}$ as defined in section E-4.3.2 and $\boldsymbol{A_{mll}()}$ as defined in section E-3.3.2 the solution for $\boldsymbol{Y}$ can easily be found:

$$G_l = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \ldots & 0 \\ 0 & \frac{3}{2} & 0 & \ldots & 0 \\ 0 & 0 & \frac{5}{2} & \ldots & 0 \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 0 & 0 & 0 & \ldots & \frac{2n-1}{2} \end{bmatrix}$$

$$Y^T = \sum_{j=1}^{m} (l(x_0(j)) - l(x_0(j-1))) S_{DL} A_{mll}(f_j) G_l$$

E-4.3.4 Waveform Smoothing

There is no obvious method for performing waveform smoothing in the Legendre Series spectral domain. Instead, the waveform should be converted to a polynomial expansion and the methods of section E-4.2.4 employed.

E-4.4 Chebyshev Series

E-4.4.1 Trigonometric and Exponential Functions

There is often no direct way of evaluating a trigonometric or expontential function of a Chebyshev Series. Instead, the function is performed on a data series converted from the argument Chebyshev Series. The resulting polynomial is then reconverted back into a Chebyshev Series.

E-4.4.2 Integration and Differentiation

Differentiating a Chebyshev Series can be done easily by differentiating the recursion formula for the Chebyshev Polynomials. Recall:

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x)$$

Differentiating:

$$\frac{dT_{i+1}(x)}{dx} = 2x\frac{dT_i(x)}{dx} + 2T_i(x) - \frac{dT_{i-1}(x)}{dx}$$

where

$$\frac{dT_0(x)}{dx} = 0$$

$$\frac{dT_1(x)}{dx} = 1$$

The goal is to generate the following $\boldsymbol{n} \times \boldsymbol{n}$ matrix:

$$A_{DT} = \left[\frac{dT_0(x)}{dx} \quad \frac{dT_1(x)}{dx} \quad \frac{dT_2(x)}{dx} \quad \cdots \quad \frac{dT_{n-1}(x)}{dx} \right]$$

The columns of $\boldsymbol{A_{DT}}$ can be solved recursively once we define matrix $\boldsymbol{A_{XT}}$ for multiplying a Chebyshev Series vector by $\boldsymbol{x}$.

$$xT_i(x) = \frac{1}{2}(T_{i-1}(x) + T_{i+1}(x))$$

$$A_{XT} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & \frac{1}{2} & 0 & \dots & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & \dots & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \dots & 0 & 0 \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{2} & 0 \end{bmatrix}$$

Note that the last row of $\boldsymbol{A_{XT}}$ has been eliminated to make the matrix square. This will not cause any problems since in the recursion formula which follows, the last coefficient of the vector multiplying $\boldsymbol{A_{XT}}$ is always zero.

Let $\boldsymbol{A_{DT}(:,j)}$ represent the ***j***the column of $\boldsymbol{A_{DT}}$. Let $\boldsymbol{I}$ be the $\boldsymbol{n \times n}$ identity matrix. The recursion formula states:

$$A_{DT}(:, i+2) = 2A_{XT}A_{DT}(:, i+1) + 2I(:, i+1) - A_{DT}(:, i)$$

$$i \le i \le n-2$$

Once $\boldsymbol{A_{DT}}$ has been constructed, it can be used to calculate derivatives. Let $\boldsymbol{W}$ and $\boldsymbol{Y}$ be vectors of Chebyshev Series coefficient of size $\boldsymbol{n}$. Then the following statements are identical:

$$Y = \frac{dW}{dx}$$

$$Y = A_{DT}W$$

Of course the ***n***th coefficient of $\boldsymbol{Y}$ will always be zero since the ***n***th row (as well as the first column) of $\boldsymbol{A_{DT}}$ will always be populated with zeros.

Integration is a bit more complex. In general, the problem is to solve the following equation:

$$Y = Y_0 + \int_{\tau=-1}^{x} W d\tau$$

First, the $\boldsymbol{n+1} \times \boldsymbol{n}$ indefinite integral matrix $\boldsymbol{S_{IT}}$ should be found. The easiest way of generating $\boldsymbol{S_{IT}}$ begins by adding an additional column to $\boldsymbol{A_{DT}}$ using the same recursion formula to form the $\boldsymbol{n} \times \boldsymbol{n+1}$ matrix $\boldsymbol{A_{DT1}}$. $\boldsymbol{S_{IT}}$ is simply the pseudo-inverse of $\boldsymbol{A_{DT1}}$:

$$S_{IT} = (A_{DT1}^T A_{DT1})^{-1} A_{DT1}^T$$

The next step is to evaluate the integral at $\boldsymbol{x = -1}$. This can be done by multplying the following row vector by $\boldsymbol{S_{IL}}$:

$$X_{-1} = [1 \quad -1 \quad 1 \quad -1 \quad \ldots \quad (-1)^{n-1}]$$

$$S_{-1} = X_{-1} S_{IT}$$

The first row of $\boldsymbol{S_{IT}}$ contains all zeros. If this row is replaced by $\boldsymbol{-S_{-1}}$ and the resulting matrix called $\boldsymbol{S_{DT}}$, we have all the pieces for calculating the integral of a Chebyshev Series:

$$Y = S_{DT} W + Y_0$$

Of course, the vector $\boldsymbol{Y}$ may have to be truncated or padded with zeros as required.

E-4.4.3 Switching Functions

Switching functions for Chebyshev Series can not be evaluated as easily as the switching functions for Legendre Series due to the weighting function $\boldsymbol{r(x)}$ for the Chebyshev Polynomials. Recall:

$$c_1 = \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx$$

$$c_m = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_{m-1}(x)}{\sqrt{1-x^2}} dx$$

The situation is not hopeless due to the following integral equations:

$$\int \frac{dx}{\sqrt{1-x^2}} = \sin^{-1}(x)$$

$$\int \frac{x}{\sqrt{1-x^2}} dx = -\sqrt{1-x^2}$$

$$\int \frac{x^{2m}}{\sqrt{1-x^2}} dx = \frac{(2m)!}{(m!)^2}\left[-\sqrt{1-x^2} \sum_{r=1}^{m} \frac{r!(r-1)!}{2^{2m-2r+1}(2r)!} x^{2r-1} + \frac{\sin^{-1}(x)}{2^{2m}} \right]$$

$$\int \frac{x^{2m+1}}{\sqrt{1-x^2}} dx = -\sqrt{1-x^2} \sum_{r=1}^{m} \frac{(2r)!(m!)^2}{(2m+1)!(r!)^2} r^{m-r} x^{2r}$$

Thus if $\boldsymbol{f_j}$ is the Chebyshev Series representation of the ***j***th piece of Chebyshev Series waveform ***Y*** and $\boldsymbol{x_0(j)}$ is the ***x*** coordinate of the ending point of the ***j***th piece, then we can state the following:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ . \\ . \\ . \\ Y_n \end{bmatrix}$$

$$x_0(0) = -1$$

$$x_0(m) = 1$$

$$Y_1 = \frac{1}{\pi} \sum_{j=1}^{m} \int_{x_0(j-1)}^{x_0(j)} \frac{f_j(x)}{\sqrt{1-x^2}} dx$$

$$Y_m = \frac{2}{\pi} \sum_{j=1}^{m} \int_{x_0(j-1)}^{x_0(j)} \frac{f_j(x) T_{m-1}(x)}{\sqrt{1-x^2}} dx$$

The process should now be clear:

1. Convert $\boldsymbol{f_j(x)}$ to a polynomial representation $\boldsymbol{f_{pj}(x)}$

2. Multiply $f_{pj}(x)$ by the polynomial representation for $T_i(x)$ and call the resulting polynomial $f_{ij}(x)$.

3. Use the above integral equations to evaluate at $x = x_0(j)$ and $x = x_0(j\text{-}1)$ the integral of $f_{ij}(x)$ term by term to form the jth component of Y_i called Y_{ji}.

4. Sum up Y_{ji} over j to produce Y_i.

While the above process will produce the *correct* values for Y_i, the following method is much easier to calculate and produces nearly identical results:

1. Convert $f_j(x)$ to a Legendre Series representation $f_{lj}(x)$

2. Calculate the Legendre Series Representation Y_l of Y with the methods of section E-4.3.3.

3. Convert Y_l to the Chebyshev Series Representation Y.

E-4.4.4 Waveform Smoothing

There is no obvious method for performing waveform smoothing in the Chebyshev Series spectral domain. Instead, the waveform should be converted to a polynomial expansion and the methods of section E-4.2.4 employed.

Appendix F: Model Development

The following electrical power system models have been develped in support of WAVESIM:

Three Phase Synchronous Generator

Voltage Regulator

Prime Mover

Three Phase Switch

Transmission Line

Constant Impedance Loads

Reduction Gear

Propeller

Ship Dynamics

Pulse Generator

Induction Motor

F-1 3 Phase Synchronous Machine Model

Two models are presented for simulating a three phase synchronous model. The first expresses the voltages and currents in terms of a rotating reference frame (dq0) rotating at the base frequency. This model is suitable for studies where the voltages and currents are balanced, nearly sinusoidal, and near the base frequency. For fast transients or unbalanced operations, the actual instantaneous values for the voltages and currents should be used (abc frame). Both models are very similar in that the terminal values are transformed to a rotating reference frame alligned with the rotor of the machine (Park's Transformation)

F-1.1 DQ0 Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Stator Direct	$\boldsymbol{V_D}$ (import)	$\boldsymbol{I_D}$ (export)	(1) Normal
Stator Quadrature	$\boldsymbol{V_Q}$ (import)	$\boldsymbol{I_Q}$ (export)	(1) Normal
Stator Zero Sequence	$\boldsymbol{V_0}$ (import)	$\boldsymbol{I_0}$ (export)	(1) Normal
Mechanical	ω_m (import)	$\boldsymbol{T_m}$ (export)	(0) Normal
Field Voltage	$\boldsymbol{V_{FD}}$ (import)		Information
Stator D-axis Current	$\boldsymbol{I_{DI}}$ (export)		Information
Stator Q-axis Current	$\boldsymbol{I_{QI}}$ (export)		Information
Stator 0-axis Current	$\boldsymbol{I_{0I}}$ (export)		Information
Field Current	$\boldsymbol{I_{FI}}$ (export)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} V_D \\ V_Q \\ V_0 \\ V_{FD} \\ \omega_m \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_D \\ I_Q \\ I_0 \\ I_F \\ T_m \\ I_{DI} \\ I_{QI} \\ I_{0I} \end{bmatrix}$$

Parameters

$\boldsymbol{x_d}$	Synchronous Reactance (PU)
$\boldsymbol{x_q}$	Negative Sequence Reactance (PU)
x_d'	Transient Reactance (PU)
x_d''	D-axis Subtransient Reactance (PU)
x_q''	Q-axis Subtransient Reactance (PU)
$\boldsymbol{x_{al}}$	Armature Leakage Reactance (PU)
T_{do}'	Transient Open Circuit Time Constant (seconds)
T_{do}''	D-axis Subtransient OC Time Constant (seconds)
T_{qo}''	Q-axis Subtransient OC Time Constant (seconds)
T_{ad}	Armature Time Constant (sec)
$\boldsymbol{H}$	Inertia Constant (sec)
$\boldsymbol{p_p}$	Pole Pairs
$\boldsymbol{i_{fnl}}$	Field Current for no load rated voltage (amps)
ω_{bs}	Base System Frequency (rad/sec)
Θ_{bs}	Base System Angle (radians)
$\boldsymbol{V_{SB}}$	Base System Voltage (volts)
$\boldsymbol{P_{SB}}$	Base System Power (watts)
$\boldsymbol{V_{MS}}$	Base Machine Voltage (volts)
$\boldsymbol{P_{MB}}$	Base Machine Power (watts)

States

Θ_S	rotor angle wrt to synchronous frame (rad)
ψ_{dS}	D-axis flux-linkage (PU)
ψ_{qS}	Q-axis flux-linkage (PU)
e_{qS}''	Q-axis voltage behind subtransient reactance (PU)
e_{dS}''	D-axis voltage behind subtransient reactance (PU)
e_{qS}'	Q-axis voltage behind transient reactance (PU)

Equations

Constant Definitions

Base Quantities

$$I_{SB} = \frac{2}{3}\frac{P_{SB}}{V_{SB}}$$

$$T_{SB} = \frac{P_{SB}}{\omega_{SB}}$$

$$I_{MB} = \frac{2}{3}\frac{P_{MB}}{V_{MB}}$$

$$T_{MB} = \frac{p_p P_{MB}}{\omega_{bs}}$$

$$I_{fB} = I_{fnl}(x_d - x_{al})$$

$$V_{fB} = \frac{P_{MB}}{I_{fB}}$$

Other Constants

$$x_{ad} = x_d - x_{al}$$

$$x_f = \frac{x_{ad}^2}{x_d - x_d'}$$

$$r_f = \frac{x_f}{\omega_{bs} T_{do}'}$$

$$x_{kd} = \frac{x_{ad}^2}{x_d - x_d''}$$

$$\alpha = \frac{x_d - x_d''}{x_d' - x_d''}$$

Angle Calculations

$$\Theta = \int (\omega_{bs} - \omega_m p_p) dt + \Theta_S$$

$$\Theta = S(\omega_{bs} - \omega_m p_p) + \Theta_{S0}$$

$$C_\Theta = \cos(\Theta)$$

$$S_\Theta = \sin(\Theta)$$

Variable Rotation and Scaling

$$v = \begin{bmatrix} v_d \\ v_q \\ v_0 \\ v_{fd} \end{bmatrix}$$

$$V = \begin{bmatrix} V_D \\ V_Q \\ V_0 \\ V_{FD} \end{bmatrix}$$

$$R_v = \begin{bmatrix} \frac{V_{SB}}{V_{MB}} M(C_\Theta) & -\frac{V_{SB}}{V_{MB}} M(S_\Theta) & 0 & 0 \\ \frac{V_{SB}}{V_{MB}} M(S_\Theta) & \frac{V_{SB}}{V_{MB}} M(C_\Theta) & 0 & 0 \\ 0 & 0 & \frac{V_{SB}}{V_{MB}} & 0 \\ 0 & 0 & 0 & \frac{V_{SB}}{V_{fB}} \end{bmatrix}$$

$$v = R_v V$$

Solving the electrical dynamical equations

The five electrical dynamical equations must be solved simultaneously. Since the Integration Matrix and Multiplication Matrix are linear matrices, the entire problem becomes a linear process. Hence the system of equations can be represented by a matrix equation.

First define the integration and multiplication matrices

$$\int x(t)dt = Sx + x_S \quad ; \quad S \in \Re^{n\times n} \quad x_S \in \Re^n$$

$$x \cdot y = M(y)x \quad ; \quad M \in \Re^{n\times n}$$

Now we define the system of equations

$$A = \begin{bmatrix} I + \frac{S}{T_{ad}} & -SM(\omega_m p_p) & -\frac{S}{T_{ad}} & 0 & 0 \\ SM(\omega_m p_p) & I + \frac{S}{T_{aq}} & 0 & \frac{S}{T_{aq}} & 0 \\ -S\frac{x_d' - x_d''}{T_{do}'' x_d''} & 0 & I + S\frac{x_d'}{T_{do}'' x_d''} & 0 & -\frac{S}{T_{do}''} \\ 0 & S\frac{x_q - x_q''}{T_{qo}'' x_q''} & 0 & I + S\frac{x_q}{T_{qo}'' x_q''} & 0 \\ 0 & 0 & -S\frac{\alpha - 1}{T_{do}'} & 0 & I + S\frac{\alpha}{T_{do}'} \end{bmatrix}$$

$$x = \begin{bmatrix} \psi_d \\ \psi_q \\ e_q'' \\ e_d'' \\ e_q' \end{bmatrix}$$

$$B_v = \begin{bmatrix} \omega_{bs} & 0 & 0 & 0 \\ 0 & \omega_{bs} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \dfrac{x_{ad}}{r_f} \end{bmatrix}$$

$$B_S = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}$$

$$s_0 = \begin{bmatrix} \Psi_{dS0} \\ \Psi_{qS0} \\ e_{qS0}'' \\ e_{dS0}'' \\ e_{qS0}' \end{bmatrix}$$

$$b = B_v v + B_S s_0$$

$$x = A^{-1} b$$

Calculating Export Variables

First the currents in machine reference frame

$$i_e = \begin{bmatrix} i_d \\ i_q \\ i_0 \\ i_{fd} \end{bmatrix}$$

$$C = \begin{bmatrix} \frac{1}{x_d''} & 0 & -\frac{1}{x_d''} & 0 & 0 \\ 0 & \frac{1}{x_q''} & 0 & \frac{1}{x_q''} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{x_{kd}}{x_{ad}(x_f - x_{kd})} & 0 & \frac{x_f}{x_{ad}(x_f - x_{kd})} \end{bmatrix}$$

$$i_e = Cx$$

Now the currents in system reference frame

$$I_e = \begin{bmatrix} I_D \\ I_Q \\ I_0 \\ I_{FD} \end{bmatrix}$$

$$R_I = \begin{bmatrix} \frac{I_{MB}}{I_{SB}} M(C_\Theta) & \frac{I_{MB}}{I_{SB}} M(S_\Theta) & 0 & 0 \\ -\frac{I_{MB}}{I_{SB}} M(S_\Theta) & \frac{I_{MB}}{I_{SB}} M(C_\Theta) & 0 & 0 \\ 0 & 0 & \frac{I_{MB}}{I_{SB}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

$$I_e = R_I i_e$$

Torque Equation

$$T_{epu} = \psi_d i_q - \psi_q i_d$$

$$T_{acc} = \frac{2H p_p}{\omega_{bs}} \frac{d\omega_m}{dt}$$

$$T_{epu} = M(\psi_d) i_q - M(\psi_q) i_d$$

$$T_{acc} = \frac{2H p_p}{\omega_{bs}} S^{-1} \omega_m$$

$$T_m = \frac{T_{SB}}{T_{MB}} (T_{acc} - T_{epu})$$

Structural Jacobian

The structural jacobian for the DQ0 model is given by:

$$J_{DS} = \begin{bmatrix} N & N & 0 & N & N \\ N & N & 0 & N & N \\ 0 & 0 & 0 & 0 & 0 \\ N & N & 0 & N & N \\ N & N & 0 & N & N \\ N & N & 0 & N & N \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Jacobian Calculations

Calculating the jacobian of the export variables with respect to the import variables is straight forward with the exception of the partials with respect to the mechanical frequency. First of all, nothing depends on $\boldsymbol{V_0}$, hence all of its partials are zero. In the following derivations, the Device Jacobian is partitioned such that the voltages and currents are split from the mechanical speed and torque.

$$I_e = R_I C A^{-1} (B_v R_v V + B_S s_0)$$

$$\frac{\partial I_e}{\partial V} = R_I C A^{-1} B_v R_v$$

Calculating the partials with respect to the mechancial frequency:

$$Ax = B_v R_v V + B_S s_0$$

$$A\frac{\partial x}{\partial \omega_m} + \frac{\partial A}{\partial \omega_m} x = B_v \frac{\partial R_v}{\partial \omega_m} V$$

$$\frac{\partial x}{\partial \omega_m} = A^{-1}\left(B_v \frac{\partial R_v}{\partial \omega_m} V - \frac{\partial A}{\partial \omega_m} x \right)$$

$$\frac{\partial I_e}{\partial \omega_m} = R_I C \frac{\partial x}{\partial \omega_m} + \frac{\partial R_I}{\partial \omega_m} Cx$$

where

$$\frac{\partial A}{\partial \omega_m} = \begin{bmatrix} 0 & -Sp_p & 0 & 0 & 0 \\ Sp_p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and

$$\frac{\partial \Theta}{\partial \omega_m} = -Sp_p$$

$$\frac{\partial R_v}{\partial \omega_m} = \begin{bmatrix} \frac{V_{SB}}{V_{MB}} M(S_\Theta) Sp_p & \frac{V_{SB}}{V_{MB}} M(C_\Theta) Sp_p & 0 & 0 \\ -\frac{V_{SB}}{V_{MB}} M(C_\Theta) Sp_p & \frac{V_{SB}}{V_{MB}} M(S_\Theta) Sp_p & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial R_I}{\partial \omega_m} = \begin{bmatrix} \frac{I_{MB}}{I_{SB}} M(S_\Theta) S p_p & -\frac{I_{MB}}{I_{SB}} M(C_\Theta) S p_p & 0 & 0 \\ \frac{I_{MB}}{I_{SB}} M(C_\Theta) S p_p & \frac{I_{MB}}{I_{SB}} M(S_\Theta) S p_p & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The Torque equation Jacobians are given by:

$$\frac{\partial T_m}{\partial V} = -\frac{T_{SB}}{T_{MB}} \frac{\partial T_{epu}}{\partial V}$$

$$\frac{\partial T_{epu}}{\partial V} = M(\psi_d) \frac{\partial i_q}{\partial V} + M(i_q) \frac{\partial \psi_d}{\partial V} - M(\psi_q) \frac{\partial i_d}{\partial V} - M(i_d) \frac{\partial \psi_q}{\partial V}$$

$$\frac{\partial \psi_d}{\partial V} = [I \quad 0 \quad 0 \quad 0 \quad 0] \frac{\partial x}{\partial V}$$

$$\frac{\partial \psi_q}{\partial V} = [0 \quad I \quad 0 \quad 0 \quad 0] \frac{\partial x}{\partial V}$$

$$\frac{\partial i_d}{\partial V} = [I \quad 0 \quad 0 \quad 0] \frac{\partial i_e}{\partial V}$$

$$\frac{\partial i_q}{\partial V} = [0 \quad I \quad 0 \quad 0] \frac{\partial i_e}{\partial V}$$

Now with respect to the mechanical frequency:

$$\frac{\partial T_m}{\partial \omega_m} = \frac{T_{SB}}{T_{MB}} \left(\frac{\partial T_{acc}}{\partial \omega_m} - \frac{\partial T_{epu}}{\partial \omega_m} \right)$$

$$\frac{\partial T_{acc}}{\partial \omega_m} = \frac{2H p_p}{\omega_{bs}} S^{-1}$$

$$\frac{\partial T_{epu}}{\partial \omega_m} = M(\psi_d) \frac{\partial i_q}{\partial \omega_m} + M(i_q) \frac{\partial \psi_d}{\partial \omega_m} - M(\psi_q) \frac{\partial i_d}{\partial \omega_m} - M(i_d) \frac{\partial \psi_q}{\partial \omega_m}$$

$$\frac{\partial \psi_d}{\partial \omega_m} = [I \quad 0 \quad 0 \quad 0 \quad 0] \frac{\partial x}{\partial \omega_m}$$

$$\frac{\partial \psi_q}{\partial \omega_m} = [0 \quad I \quad 0 \quad 0 \quad 0] \frac{\partial x}{\partial \omega_m}$$

$$\frac{\partial i_d}{\partial \omega_m} = [I \quad 0 \quad 0 \quad 0] \frac{\partial i_e}{\partial \omega_m}$$

$$\frac{\partial i_q}{\partial \omega_m} = [0 \quad I \quad 0 \quad 0] \frac{\partial i_e}{\partial \omega_m}$$

Putting the Jacobian all together:

$$J_D = \begin{bmatrix} \frac{\partial I_D}{\partial V_D} & \frac{\partial I_D}{\partial V_Q} & 0 & \frac{\partial I_D}{\partial V_F} & \frac{\partial I_D}{\partial \omega_m} \\ \frac{\partial I_Q}{\partial V_D} & \frac{\partial I_Q}{\partial V_Q} & 0 & \frac{\partial I_Q}{\partial V_F} & \frac{\partial I_Q}{\partial \omega_m} \\ 0 & 0 & 0 & 0 & 0 \\ \frac{\partial T_m}{\partial V_D} & \frac{\partial T_m}{\partial V_Q} & 0 & \frac{\partial T_m}{\partial V_F} & \frac{\partial T_m}{\partial \omega_m} \\ \frac{\partial I_D}{\partial V_D} & \frac{\partial I_D}{\partial V_Q} & 0 & \frac{\partial I_D}{\partial V_F} & \frac{\partial I_D}{\partial \omega_m} \\ \frac{\partial I_Q}{\partial V_D} & \frac{\partial I_Q}{\partial V_Q} & 0 & \frac{\partial I_Q}{\partial V_F} & \frac{\partial I_Q}{\partial \omega_m} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

F-1.2 ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Stator Phase A	$\boldsymbol{V}_{\mathbf{A}}$ (import)	$\boldsymbol{I}_{\mathbf{A}}$ (export)	(1) Normal
Stator Phase B	$\boldsymbol{V}_{\mathbf{B}}$ (import)	$\boldsymbol{I}_{\mathbf{B}}$ (export)	(1) Normal
Stator Phase C	$\boldsymbol{V}_{\mathbf{C}}$ (import)	$\boldsymbol{I}_{\mathbf{C}}$ (export)	(1) Normal
Mechanical	ω_m (import)	$\boldsymbol{T}_{\mathbf{m}}$ (export)	(0) Normal
Field Voltage	$\boldsymbol{V}_{\mathbf{FD}}$ (import)		Information
Stator Phase A Current	$\boldsymbol{I}_{\mathbf{AI}}$ (export)		Information
Stator Phase B Current	$\boldsymbol{I}_{\mathbf{BI}}$ (export)		Information
Stator Phase C Current	$\boldsymbol{I}_{\mathbf{CI}}$ (export)		Information
Field Current	$\boldsymbol{I}_{\mathbf{F}}$ (export)		Information

The import $\boldsymbol{x}_{\boldsymbol{imp}}$ and export $\boldsymbol{x}_{\boldsymbol{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} V_A \\ V_B \\ V_C \\ V_{FD} \\ \omega_m \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_D \\ I_Q \\ I_0 \\ I_F \\ T_m \\ I_{DI} \\ I_{QI} \\ I_{0I} \end{bmatrix}$$

Parameters

All Parameters are identical to the DQ0 Model

States

All States are identical to the DQ0 Model

Equations

Constant Definitions

All Constants definitions are identical to the DQ0 Model

Angle Calculations

For this model, the angle is the actual rotor angle of the synchronous machine:

$$\Theta = \int \omega_m p_p dt + \Theta_{S0}$$

$$\Theta = S\omega_m p_p + \Theta_{S0}$$

Note: When calculating Θ_{S1} it would be wise to limit its range to $\pm\pi$.

Variable Rotation and Scaling

To convert from the ***V*** vector to the ***v*** vector, Parks transformation should be used:

$$V = \begin{bmatrix} V_A \\ V_B \\ V_C \\ V_{FD} \end{bmatrix}$$

$$R_v = \frac{2V_{SB}}{3V_{MB}} \begin{bmatrix} M(\cos(\Theta)) & M\left(\cos\left(\Theta - \frac{2\pi}{3}\right)\right) & M\left(\cos\left(\Theta + \frac{2\pi}{3}\right)\right) & 0 \\ -M(\sin(\Theta)) & -M\left(\sin\left(\Theta - \frac{2\pi}{3}\right)\right) & -M\left(\sin\left(\Theta + \frac{2\pi}{3}\right)\right) & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{3V_{MB}}{2V_{fB}} \end{bmatrix}$$

$$v = R_v V$$

Solving the electrical dynamical equations

The electrical dynamical equations are solved in exactly the same way as for the DQ0 model.

Calculating Explicit Variables

The only difference for calculating the explicit variables are the following matrices:

$$I_e = \begin{bmatrix} I_A \\ I_B \\ I_C \\ I_{FD} \end{bmatrix}$$

$$R_I = \frac{I_{MB}}{I_{SB}} \begin{bmatrix} M(\cos(\Theta)) & -M(\sin(\Theta)) & 1 & 0 \\ M\left(\cos\left(\Theta - \frac{2\pi}{3}\right)\right) & -M\left(\sin\left(\Theta - \frac{2\pi}{3}\right)\right) & 1 & 0 \\ M\left(\cos\left(\Theta + \frac{2\pi}{3}\right)\right) & -M\left(\sin\left(\Theta + \frac{2\pi}{3}\right)\right) & 1 & 0 \\ 0 & 0 & 0 & \frac{I_{SB}}{I_{MB}} \end{bmatrix}$$

Structural Jacobian

The structural jacobian for the ABC model is given by:

$$J_{DS} = \begin{bmatrix} N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \end{bmatrix}$$

Jacobian Calculations

The only differences for calculating the jacobian matrices are the following:

$$\frac{\partial \Theta}{\partial \omega_m} = S p_p$$

$$\frac{\partial R_v}{\partial \omega_m} = \frac{2V_{SB}}{3V_{MB}} \begin{bmatrix} -M(\sin(\Theta))Sp_p & -M\left(\sin\left(\Theta - \frac{2\pi}{3}\right)\right)Sp_p & -M\left(\sin\left(\Theta + \frac{2\pi}{3}\right)\right)Sp_p & 0 \\ -M(\cos(\Theta))Sp_p & -M\left(\cos\left(\Theta - \frac{2\pi}{3}\right)\right)Sp_p & -M\left(\cos\left(\Theta + \frac{2\pi}{3}\right)\right)Sp_p & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial R_I}{\partial \omega_m} = \frac{I_{MB}}{I_{SB}} \begin{bmatrix} -M(\sin(\Theta))Sp_p & -M(\cos(\Theta))Sp_p & 0 & 0 \\ -M\left(\sin\left(\Theta - \frac{2\pi}{3}\right)\right)Sp_p & -M\left(\cos\left(\Theta - \frac{2\pi}{3}\right)\right)Sp_p & 0 & 0 \\ -M\left(\sin\left(\Theta + \frac{2\pi}{3}\right)\right)Sp_p & -M\left(\cos\left(\Theta + \frac{2\pi}{3}\right)\right)Sp_p & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Putting the Jacobian all together:

$$J_D = \begin{bmatrix} \frac{\partial I_A}{\partial V_A} & \frac{\partial I_A}{\partial V_B} & \frac{\partial I_A}{\partial V_C} & \frac{\partial I_A}{\partial V_F} & \frac{\partial I_A}{\partial \omega_m} \\ \frac{\partial I_B}{\partial V_A} & \frac{\partial I_B}{\partial V_B} & \frac{\partial I_B}{\partial V_C} & \frac{\partial I_B}{\partial V_F} & \frac{\partial I_B}{\partial \omega_m} \\ \frac{\partial I_C}{\partial V_A} & \frac{\partial I_C}{\partial V_B} & \frac{\partial I_C}{\partial V_C} & \frac{\partial I_C}{\partial V_F} & \frac{\partial I_C}{\partial \omega_m} \\ \frac{\partial T_m}{\partial V_A} & \frac{\partial T_m}{\partial V_B} & \frac{\partial T_m}{\partial V_C} & \frac{\partial T_m}{\partial V_F} & \frac{\partial T_m}{\partial \omega_m} \\ \frac{\partial I_A}{\partial V_A} & \frac{\partial I_A}{\partial V_B} & \frac{\partial I_A}{\partial V_C} & \frac{\partial I_A}{\partial V_F} & \frac{\partial I_A}{\partial \omega_m} \\ \frac{\partial I_B}{\partial V_A} & \frac{\partial I_B}{\partial V_B} & \frac{\partial I_B}{\partial V_C} & \frac{\partial I_B}{\partial V_F} & \frac{\partial I_B}{\partial \omega_m} \\ \frac{\partial I_C}{\partial V_A} & \frac{\partial I_C}{\partial V_B} & \frac{\partial I_C}{\partial V_C} & \frac{\partial I_C}{\partial V_F} & \frac{\partial I_C}{\partial \omega_m} \end{bmatrix}$$

F-2 Voltage Regulator Model

This is a simple voltage regulator model. The voltage regulator is assumed to be of a PI type controller. This design does not have any clipping on the output waveform to ensure the field voltage is kept within a reasonable range. This model is intended for single generator operation since it has no provision for reactive power sharing with paralleled generators.

F-2.1 DQ0 Model

Interface Variables

Terminal	Potential Variable	Flow Variables	Type
Line Direct Voltage	$\boldsymbol{V_D}$ (import)		Information
Line Quadrature Voltage	$\boldsymbol{V_Q}$ (import)		Information
Reference Voltage	$\boldsymbol{V_{ref}}$ (import)		Information
Field Voltage	$\boldsymbol{V_{FD}}$ (export)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} V_D \\ V_Q \\ V_{ref} \end{bmatrix} \qquad x_{exp} = [V_{FD}]$$

Parameters

$\boldsymbol{k_I}$	Integrating factor (1 / sec)
$\boldsymbol{k_P}$	Proportional factor (PU)
$\boldsymbol{k_{DQ0}}$	Voltage Magnitude Conversion factor (PU)

States

$\boldsymbol{V_{fdS}}$	Field Voltage

Equations

Calculate the terminal voltage:

$$V_t = k_{DQ0}\sqrt{V_D \cdot V_D + V_Q \cdot V_Q}$$

Calculate the error voltage:

$$V_{err} = V_{ref} - V_t$$

Calculate the Field Voltage:

$$V_{FD} = \int k_I V_{err} dt + k_p V_{err} + V_{fdS}$$

Structural Jacobian

The structural jacobian for the DQO model is given by:

$$J_{DS} = [N \quad N \quad L]$$

Jacobian

$$\frac{\partial V_{FD}}{\partial V_{ref}} = k_I S + k_p I$$

$$\frac{\partial V_{FD}}{\partial V_t} = -k_I S - k_p I$$

$$\frac{\partial V_{FD}}{\partial V_D} = \frac{\partial V_{FD}}{\partial V_t}\frac{\partial V_t}{\partial V_D}$$

$$\frac{\partial V_{FD}}{\partial V_Q} = \frac{\partial V_{FD}}{\partial V_t}\frac{\partial V_t}{\partial V_Q}$$

Note the partials of V_t with respect to V_D and V_Q must be determined from the square root function:

$$\frac{\partial V_t}{\partial V_D} = k_{DQ0}\frac{V_D}{V_t}$$

$$\frac{\partial V_t}{\partial V_Q} = k_{DQ0}\frac{V_Q}{V_t}$$

The device jacobian is given by:

$$J_D = \begin{bmatrix} \frac{\partial V_{FD}}{\partial V_D} & \frac{\partial V_{FD}}{\partial V_Q} & \frac{\partial V_{FD}}{\partial V_{ref}} \end{bmatrix}$$

F-2.2 ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	Type
Phase A Voltage	$\boldsymbol{V_A}$ (import)		Information
Phase B Voltage	$\boldsymbol{V_B}$ (import)		Information
Phase C Voltage	$\boldsymbol{V_C}$ (import)		Information
Reference Voltage	$\boldsymbol{V_{ref}}$ (import)		Information
Field Voltage	$\boldsymbol{V_{FD}}$ (export)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_A \\ V_B \\ V_C \\ V_{ref} \end{bmatrix} \qquad x_{exp} = [V_{FD}]$$

Parameters

$\boldsymbol{k_I}$	Integrating factor (1 / sec)
$\boldsymbol{k_P}$	Proportional factor (PU)
$\boldsymbol{k_{ABC}}$	Voltage Magnitude conversion factor (PU)

States

$\boldsymbol{V_{fdS}}$	Field Voltage

Equations

Calculate and subtract out the DC offset of the common potential:

$$V_0 = \frac{V_A + V_B + V_C}{3}$$

$$V_{A0} = V_A - V_0$$

$$V_{B0} = V_B - V_0$$

$$V_{C0} = V_C - V_0$$

Calculate the Terminal voltage[1]:

$$V_t = k_{ABC}\sqrt{\frac{2}{3}(V_{A0}\cdot V_{A0} + V_{B0}\cdot V_{B0} + V_{C0}\cdot V_{C0})}$$

Calculate the Error voltage:

$$V_{err} = V_{ref} - V_t$$

Calculate the Field Voltage:

$$V_{FD} = \int k_I V_{err} dt + k_p V_{err} + V_{fdS}$$

Note 1: Derivation of Terminal Voltage:

Assume phase voltages are balanced three phase:

$$V_{A0} = V_T \cos(\Theta)$$

$$V_{B0} = V_T \cos\left(\Theta + \frac{2\pi}{3}\right)$$

$$V_{C0} = V_T \cos\left(\Theta - \frac{2\pi}{3}\right)$$

$$V_{A0}^2 + V_{B0}^2 + V_{C0}^2 = \frac{V_T^2}{2}(1 + \cos(2\Theta) +$$

$$1 + \cos(2\Theta)\cos\left(\frac{4\pi}{3}\right) - \sin(2\Theta)\sin\left(\frac{4\pi}{3}\right) +$$

$$1 + \cos(2\Theta)\cos\left(\frac{4\pi}{3}\right) + \sin(2\Theta)\sin\left(\frac{4\pi}{3}\right))$$

$$V_{A0}^2 + V_{B0}^2 + V_{C0}^2 = \frac{3}{2}V_T^2$$

Structural Jacobian

The structural jacobian for the ABC model is given by:

$$J_{DS} = [N \quad N \quad N \quad L]$$

Jacobian

$$\frac{\partial V_{FD}}{\partial V_{ref}} = k_I S + k_p I$$

$$\frac{\partial V_{FD}}{\partial V_t} = -k_I S - k_p I$$

$$\frac{\partial V_{FD}}{\partial V_A} = \frac{\partial V_{FD}}{\partial V_t}\frac{\partial V_t}{\partial V_A}$$

$$\frac{\partial V_{FD}}{\partial V_B} = \frac{\partial V_{FD}}{\partial V_t}\frac{\partial V_t}{\partial V_B}$$

$$\frac{\partial V_{FD}}{\partial V_C} = \frac{\partial V_{FD}}{\partial V_t}\frac{\partial V_t}{\partial V_C}$$

Note the partials of $\boldsymbol{V_t}$ with respect to $\boldsymbol{V_A}$, $\boldsymbol{V_B}$, and $\boldsymbol{V_C}$ must be determined from the square root function:

$$\frac{\partial V_t}{\partial V_{A0}} = k_{ABC}\frac{V_{A0}}{V_t}\sqrt{\frac{2}{3}}$$

$$\frac{\partial V_t}{\partial V_{B0}} = k_{ABC}\frac{V_{B0}}{V_t}\sqrt{\frac{2}{3}}$$

$$\frac{\partial V_t}{\partial V_{C0}} = k_{ABC}\frac{V_{C0}}{V_t}\sqrt{\frac{2}{3}}$$

$$\frac{\partial V_t}{\partial V_A} = \frac{2}{3}\frac{\partial V_t}{\partial V_{A0}} - \frac{1}{3}\frac{\partial V_t}{\partial V_{B0}} - \frac{1}{3}\frac{\partial V_t}{\partial V_{C0}}$$

$$\frac{\partial V_t}{\partial V_B} = -\frac{1}{3}\frac{\partial V_t}{\partial V_{A0}} + \frac{2}{3}\frac{\partial V_t}{\partial V_{B0}} - \frac{1}{3}\frac{\partial V_t}{\partial V_{C0}}$$

$$\frac{\partial V_t}{\partial V_C} = -\frac{1}{3}\frac{\partial V_t}{\partial V_{A0}} - \frac{1}{3}\frac{\partial V_t}{\partial V_{B0}} + \frac{2}{3}\frac{\partial V_t}{\partial V_{C0}}$$

The device jacobian is given by:

$$J_D = \left[\frac{\partial V_{FD}}{\partial V_A} \quad \frac{\partial V_{FD}}{\partial V_B} \quad \frac{\partial V_{FD}}{\partial V_C} \quad \frac{\partial V_{FD}}{\partial V_{ref}} \right]$$

F-3 Prime Mover

This is a rather crude model of a PI controller on a prime mover. The dynamics of the controller are assumed to dominate the response of the prime mover.

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Mechanical	ω_m (import)	T_m (export)	(0) Normal
Information	ω_{ref} (import)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are defined by:

$$x_{imp} = \begin{bmatrix} \omega_m \\ \omega_{ref} \end{bmatrix} \qquad x_{exp} = [T_m]$$

Parameters

k_I	Integrating Torque factor (1 / sec)
k_P	Proportional Torque factor (PU)
ω_{bs}	base frequency (rad/sec)
P_{SB}	base System Power (watts)
P_{MB}	base Machine Power (watts)

States

T_{mS}	mechanical torque

Equations

$$T_m = \frac{P_{MB}}{P_{SB}}\left(\int \frac{k_I}{\omega_{bs}}(\omega_{ref} - \omega_m)dt + \frac{k_P}{\omega_{bs}}(\omega_{ref} - \omega_m) \right) + T_{mS}$$

Structural Jacobian

The structural jacobian for the Prime Mover model is given by:

$$J_{DS} = [L \quad L]$$

Jacobian

$$\frac{\partial T_m}{\partial \omega_{ref}} = \frac{P_{MB}}{P_{SB}} \frac{k}{\omega_{bs}} S$$

$$\frac{\partial T_m}{\partial \omega_m} = -\frac{\partial T_m}{\partial \omega_{ref}}$$

$$J_D = \begin{bmatrix} \frac{\partial T_m}{\partial \omega_m} & \frac{\partial T_m}{\partial \omega_m} \end{bmatrix}$$

F-4 Three Phase Switch

F-4.1 DQO Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
TD1	$\boldsymbol{V_{D1}}$ (Import)	$\boldsymbol{I_{D1}}$ (Export)	(1) Normal
TQ1	$\boldsymbol{V_{Q1}}$ (Import)	$\boldsymbol{I_{Q1}}$ (Export)	(2) Normal
T01	$\boldsymbol{V_{01}}$ (Import)	$\boldsymbol{I_{01}}$ (Export)	(3) Normal
TD2	$\boldsymbol{V_{D2}}$ (Import)	$\boldsymbol{I_{D2}}$ (Export)	(1) Normal
TQ2	$\boldsymbol{V_{Q2}}$ (Import)	$\boldsymbol{I_{Q2}}$ (Export)	(2) Normal
T02	$\boldsymbol{V_{02}}$ (Import)	$\boldsymbol{I_{02}}$ (Export)	(3) Normal
SW	$\boldsymbol{S_W}$ (Import)		Information

All Interface variables are on a Per Unit (PU) Basis

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_{D1} \\ V_{Q1} \\ V_{01} \\ V_{D2} \\ V_{Q2} \\ V_{02} \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_{D1} \\ I_{Q1} \\ I_{01} \\ I_{D2} \\ I_{Q2} \\ I_{02} \end{bmatrix}$$

Parameters

$\boldsymbol{G_{on}}$	On Conductance (PU)
$\boldsymbol{G_{off}}$	Off Conductance (PU)

States

There are no states for this model.

Equations

The equations for the switch are very simple. First, we define the conductance $\boldsymbol{G}$ of the waveform:

$$\begin{aligned} \text{If} \quad & S_W > 0 \\ \text{Then} \quad & G = G_{on} \\ \text{Else} \quad & G = G_{off} \end{aligned}$$

Now Generate the Conductance Matrix $\boldsymbol{G_D}$:

$$G_D = \begin{bmatrix} M(G) & 0 & 0 & M(-G) & 0 & 0 & 0 \\ 0 & M(G) & 0 & 0 & M(-G) & 0 & 0 \\ 0 & 0 & M(G) & 0 & 0 & M(-G) & 0 \\ M(-G) & 0 & 0 & M(G) & 0 & 0 & 0 \\ 0 & M(-G) & 0 & 0 & M(G) & 0 & 0 \\ 0 & 0 & M(-G) & 0 & 0 & M(G) & 0 \end{bmatrix}$$

The export variables are simply:

$$x_{exp} = G_D x_{imp}$$

Structural Jacobian

The structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} N & 0 & 0 & N & 0 & 0 & N \\ 0 & N & 0 & 0 & N & 0 & N \\ 0 & 0 & N & 0 & 0 & N & N \\ N & 0 & 0 & N & 0 & 0 & N \\ 0 & N & 0 & 0 & N & 0 & N \\ 0 & 0 & N & 0 & 0 & N & N \end{bmatrix}$$

Jacobian Calculations

The jacobian matrix is very similar to $\boldsymbol{G_D}$:

$$J_D = \begin{bmatrix} M(G) & 0 & 0 & M(-G) & 0 & 0 & M(V_{D1} - V_{D2})\dfrac{\partial G}{\partial S_W} \\ 0 & M(G) & 0 & 0 & M(-G) & 0 & M(V_{Q1} - V_{Q2})\dfrac{\partial G}{\partial S_W} \\ 0 & 0 & M(G) & 0 & 0 & M(-G) & M(V_{01} - V_{02})\dfrac{\partial G}{\partial S_W} \\ M(-G) & 0 & 0 & M(G) & 0 & 0 & M(V_{D2} - V_{D1})\dfrac{\partial G}{\partial S_W} \\ 0 & M(-G) & 0 & 0 & M(G) & 0 & M(V_{Q2} - V_{Q1})\dfrac{\partial G}{\partial S_W} \\ 0 & 0 & M(-G) & 0 & 0 & M(G) & M(V_{02} - V_{01})\dfrac{\partial G}{\partial S_W} \end{bmatrix}$$

Where $\frac{\partial G}{\partial S_W}$ is determined by the **If-Then-Else** function.

F-4.2 ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
TA1	$\boldsymbol{V_{A1}}$ (Import)	$\boldsymbol{I_{A1}}$ (Export)	(1) Normal
TB1	$\boldsymbol{V_{B1}}$ (Import)	$\boldsymbol{I_{B1}}$ (Export)	(2) Normal
TC1	$\boldsymbol{V_{C1}}$ (Import)	$\boldsymbol{I_{C1}}$ (Export)	(3) Normal
TA2	$\boldsymbol{V_{A2}}$ (Import)	$\boldsymbol{I_{A2}}$ (Export)	(1) Normal
TB2	$\boldsymbol{V_{B2}}$ (Import)	$\boldsymbol{I_{B2}}$ (Export)	(2) Normal
TC2	$\boldsymbol{V_{C2}}$ (Import)	$\boldsymbol{I_{C2}}$ (Export)	(3) Normal
SW	$\boldsymbol{S_W}$ (Import)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_{A1} \\ V_{B1} \\ V_{C1} \\ V_{A2} \\ V_{B2} \\ V_{C2} \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_{A1} \\ I_{B1} \\ I_{C1} \\ I_{A2} \\ I_{B2} \\ I_{C2} \end{bmatrix}$$

Parameters

$\boldsymbol{G_{on}}$	On Conductance (PU)
$\boldsymbol{G_{off}}$	Off Conductance (PU)

States

There are no states for this model.

Equations

The equations for the switch are very simple. First, we define the conductance ***G*** of the waveform:

$$\begin{array}{rl} \text{If} & S_W > 0 \\ \text{Then} & G = G_{on} \\ \text{Else} & G = G_{off} \end{array}$$

Now Generate the Conductance Matrix $\boldsymbol{G_D}$:

$$G_D = \begin{bmatrix} M(G) & 0 & 0 & M(-G) & 0 & 0 & 0 \\ 0 & M(G) & 0 & 0 & M(-G) & 0 & 0 \\ 0 & 0 & M(G) & 0 & 0 & M(-G) & 0 \\ M(-G) & 0 & 0 & M(G) & 0 & 0 & 0 \\ 0 & M(-G) & 0 & 0 & M(G) & 0 & 0 \\ 0 & 0 & M(-G) & 0 & 0 & M(G) & 0 \end{bmatrix}$$

The export variables are simply:

$$x_{exp} = G_D x_{imp}$$

Structural Jacobian

$$J_{DS} = \begin{bmatrix} N & 0 & 0 & N & 0 & 0 & N \\ 0 & N & 0 & 0 & N & 0 & N \\ 0 & 0 & N & 0 & 0 & N & N \\ N & 0 & 0 & N & 0 & 0 & N \\ 0 & N & 0 & 0 & N & 0 & N \\ 0 & 0 & N & 0 & 0 & N & N \end{bmatrix}$$

Jacobian Calculations

The jacobian matrix is very similar to $\boldsymbol{G_D}$:

$$J_D = \begin{bmatrix} M(G) & 0 & 0 & M(-G) & 0 & 0 & M(V_{A1} - V_{A2})\dfrac{\partial G}{\partial S_W} \\ 0 & M(G) & 0 & 0 & M(-G) & 0 & M(V_{B1} - V_{B2})\dfrac{\partial G}{\partial S_W} \\ 0 & 0 & M(G) & 0 & 0 & M(-G) & M(V_{C1} - V_{C2})\dfrac{\partial G}{\partial S_W} \\ M(-G) & 0 & 0 & M(G) & 0 & 0 & M(V_{A2} - V_{A1})\dfrac{\partial G}{\partial S_W} \\ 0 & M(-G) & 0 & 0 & M(G) & 0 & M(V_{B2} - V_{B1})\dfrac{\partial G}{\partial S_W} \\ 0 & 0 & M(-G) & 0 & 0 & M(G) & M(V_{C2} - V_{C1})\dfrac{\partial G}{\partial S_W} \end{bmatrix}$$

Where $\frac{\partial G}{\partial S_W}$ is determined by the **If-Then-Else** function.

F-5 Transmission Line

DQ0 Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
T_{D1}	$\boldsymbol{V_{D1}}$ (Import)	$\boldsymbol{I_{D1}}$ (Export)	(1) Normal
T_{Q1}	$\boldsymbol{V_{Q1}}$ (Import)	$\boldsymbol{I_{Q1}}$ (Export)	(2) Normal
T_{01}	$\boldsymbol{V_{01}}$ (Import)	$\boldsymbol{I_{01}}$ (Export)	(3) Normal
T_{D2}	$\boldsymbol{V_{D2}}$ (Import)	$\boldsymbol{I_{D2}}$ (Export)	(1) Normal
T_{Q2}	$\boldsymbol{V_{Q2}}$ (Import)	$\boldsymbol{I_{Q2}}$ (Export)	(2) Normal
T_{02}	$\boldsymbol{V_{02}}$ (Import)	$\boldsymbol{I_{02}}$ (Export)	(3) Normal

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_{D1} \\ V_{Q1} \\ V_{01} \\ V_{D2} \\ V_{Q2} \\ V_{02} \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_{D1} \\ I_{Q1} \\ I_{01} \\ I_{D2} \\ I_{Q2} \\ I_{02} \end{bmatrix}$$

Parameters

$\boldsymbol{R}$	Resistance (ohms)
$\boldsymbol{X}$	Reactance (ohms)

States

(There are no states for this model)

Equations

Constant Definitions

$$G = \frac{R}{X^2 + R^2} I$$

$$Y = -\frac{X}{X^2 + R^2} I$$

Calculate the Export Variables

$$J_D = \begin{bmatrix} G & -Y & 0 & -G & Y & 0 \\ Y & G & 0 & -Y & -G & 0 \\ 0 & 0 & \frac{I}{r} & 0 & 0 & -\frac{I}{r} \\ -G & Y & 0 & G & -Y & 0 \\ -Y & -G & 0 & Y & G & 0 \\ 0 & 0 & -\frac{I}{r} & 0 & 0 & \frac{I}{r} \end{bmatrix}$$

$$x_{exp} = J_D x_{imp}$$

Structural Jacobian

The Structural Jacobian is given by:

$$J_{DS} = \begin{bmatrix} D & D & 0 & D & D & 0 \\ D & D & 0 & D & D & 0 \\ 0 & 0 & D & 0 & 0 & D \\ D & D & 0 & D & D & 0 \\ D & D & 0 & D & D & 0 \\ 0 & 0 & D & 0 & 0 & D \end{bmatrix}$$

Jacobian Calculations

The matrix $\boldsymbol{J_D}$ is the Jacobian matrix.

ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
T_{A1}	$\boldsymbol{V_{A1}}$ (Import)	$\boldsymbol{I_{A1}}$ (Export)	(1) Normal
T_{B1}	$\boldsymbol{V_{B1}}$ (Import)	$\boldsymbol{I_{B1}}$ (Export)	(2) Normal
T_{C1}	$\boldsymbol{V_{C1}}$ (Import)	$\boldsymbol{I_{C1}}$ (Export)	(3) Normal
T_{A2}	$\boldsymbol{V_{A2}}$ (Import)	$\boldsymbol{I_{A2}}$ (Export)	(1) Normal
T_{B2}	$\boldsymbol{V_{B2}}$ (Import)	$\boldsymbol{I_{B2}}$ (Export)	(2) Normal
T_{C2}	$\boldsymbol{V_{C2}}$ (Import)	$\boldsymbol{I_{C2}}$ (Export)	(3) Normal

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_{A1} \\ V_{B1} \\ V_{C1} \\ V_{A2} \\ V_{B2} \\ V_{C2} \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_{A1} \\ I_{B1} \\ I_{C1} \\ I_{A2} \\ I_{B2} \\ I_{C2} \end{bmatrix}$$

Parameters

$\boldsymbol{R}$	Series Resistance (ohms)
$\boldsymbol{G}$	Parallel Conductance (mhos)
$\boldsymbol{L}$	Inductance (henries)

States

$\boldsymbol{I_{AL}}$	Phase A Inductor Current
$\boldsymbol{I_{BL}}$	Phase B Inductor Current
$\boldsymbol{I_{CL}}$	Phase C Inductor Current

Equations

Each of the three phases can be treated independently of one another. In the equations which follow replace a subscripted $\boldsymbol{X}$ with the appropriate phase letter:

First write the equations describing the phase:

$$\begin{bmatrix} R & I \\ I & -\frac{S}{L} - G \end{bmatrix} \begin{bmatrix} I_{X1} \\ V_{XN} \end{bmatrix} + \begin{bmatrix} -I & 0 & 0 \\ 0 & G + \frac{S}{L} & -I \end{bmatrix} \begin{bmatrix} V_{X1} \\ V_{X2} \\ I_{XL0} \end{bmatrix} = 0$$

$$\begin{bmatrix} I_{X2} \\ I_{XL} \end{bmatrix} = \begin{bmatrix} -I & 0 \\ 0 & -\frac{I}{R} - G \end{bmatrix} \begin{bmatrix} I_{X1} \\ V_{XN} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \frac{I}{R} & G & 0 \end{bmatrix} \begin{bmatrix} V_{X1} \\ V_{X2} \\ I_{XL0} \end{bmatrix}$$

Manipulating the first matrix equation, we can get an expression for the terminal 1 current:

$$G_{iv} = \left(\frac{R}{L} S + RG + I \right)^{-1} \begin{bmatrix} \left(\frac{S}{L} + G \right) & -\left(\frac{S}{L} + G \right) & I \end{bmatrix}$$

$$I_{X1} = G_{iv} \begin{bmatrix} V_{X1} \\ V_{X2} \\ I_{XL0} \end{bmatrix}$$

Once this is known, the other variables are easy to calculate:

$$V_{XN} = V_{X1} - RI_{X1}$$

$$I_{XL} = -\left(\frac{I}{R} + G \right) V_{XN} + \frac{V_{X1}}{R} + GV_{X2}$$

$$I_{X2} = -I_{X1}$$

Structural Jacobian

The Structural Jacobian is given by:

$$J_{DS} = \begin{bmatrix} L & 0 & 0 & L & 0 & 0 \\ 0 & L & 0 & 0 & L & 0 \\ 0 & 0 & L & 0 & 0 & L \\ L & 0 & 0 & L & 0 & 0 \\ 0 & L & 0 & 0 & L & 0 \\ 0 & 0 & L & 0 & 0 & L \end{bmatrix}$$

Jacobian Calculations

The Jacobian Matrix can be directly constructed from the first element of $\boldsymbol{G_{iv}}$:

$$G_{11} = \left(\frac{R}{L} S + RG + I \right)^{-1} \left(\frac{S}{L} + G \right)$$

$$J_D = \begin{bmatrix} G_{11} & 0 & 0 & -G_{11} & 0 & 0 \\ 0 & G_{11} & 0 & 0 & -G_{11} & 0 \\ 0 & 0 & G_{11} & 0 & 0 & -G_{11} \\ -G_{11} & 0 & 0 & G_{11} & 0 & 0 \\ 0 & -G_{11} & 0 & 0 & G_{11} & 0 \\ 0 & 0 & -G_{11} & 0 & 0 & G_{11} \end{bmatrix}$$

F-6 Constant Impedance Loads

DQ0 Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
T_D	$\boldsymbol{V_D}$ (import)	$\boldsymbol{I_D}$ (export)	(0) Normal
T_Q	$\boldsymbol{V_Q}$ (import)	$\boldsymbol{I_Q}$ (export)	(0) Normal
T_0	$\boldsymbol{V_0}$ (import)	$\boldsymbol{I_0}$ (export)	(0) Normal

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_D \\ V_Q \\ V_0 \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_D \\ I_Q \\ I_0 \end{bmatrix}$$

Parameters

$\boldsymbol{R}$	Load resistance (ohms)
$\boldsymbol{X}$	Load reactance (ohms)
$\boldsymbol{G_{gnd}}$	Zero Sequence conductance to ground.

States

(There are no states for this model)

Equations

First calculate the admitance

$$G = \frac{R}{R^2 + X^2}$$

$$Y = -\frac{X}{R^2 + X^2}$$

Now calculate the Admitance Matrix:

$$G_{vi} = \begin{bmatrix} G & -Y & 0 \\ Y & G & 0 \\ 0 & 0 & G_{gnd} \end{bmatrix}$$

$$x_{exp} = G_{vi} x_{imp}$$

Structural Jacobian

The Structural Jacobian is given by:

$$J_{DS} = \begin{bmatrix} D & D & 0 \\ D & D & 0 \\ 0 & 0 & D \end{bmatrix}$$

Jacobian Calculations

The jacobian matrix is the $\boldsymbol{G_{vi}}$ matrix.

ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
T_A	$\boldsymbol{V_A}$ (import)	$\boldsymbol{I_A}$ (export)	(0) Normal
T_B	$\boldsymbol{V_B}$ (import)	$\boldsymbol{I_B}$ (export)	(0) Normal
T_C	$\boldsymbol{V_C}$ (import)	$\boldsymbol{I_C}$ (export)	(0) Normal

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix}$$

Parameters

$\boldsymbol{R}$	Series Resistance (ohms)
$\boldsymbol{G}$	Parallel Conductance (mhos)
$\boldsymbol{L}$	Inductance (henries)
$\boldsymbol{R_{GND}}$	Resistance of center to Ground (ohms)

States

$\boldsymbol{I_{AL}}$	Phase A Inductor Current
$\boldsymbol{I_{BL}}$	Phase B Inductor Current
$\boldsymbol{I_{CL}}$	Phase C Inductor Current

Equations

This load model can be considered to be a transmission line where all the terminals of one side are connected together to a resistor going to ground. As such, we can use some of the derivations from the transmission line model:

$$G_{31} = \left(\frac{R}{L} S + RG + I \right)^{-1}$$

$$G_{11} = G_{31} \left(\frac{S}{L} + G \right)$$

$$I_X = G_{11} V_X - G_{11} V_{X2} + G_{13} I_{XL0}$$

$$V_{X2} = (I_A + I_B + I_C) R_{GND}$$

This can be rewritten by defining the following matrix:

$$G_{ABC} = \begin{bmatrix} (I + R_{GND}G_{11}) & R_{GND}G_{11} & R_{GND}G_{11} \\ R_{GND}G_{11} & (I + R_{GND}G_{11}) & R_{GND}G_{11} \\ R_{GND}G_{11} & R_{GND}G_{11} & (I + R_{GND}G_{11}) \end{bmatrix}$$

Which allows the following equation to be written:

$$G_{ABC}\begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} = \begin{bmatrix} G_{11} & 0 & 0 \\ 0 & G_{11} & 0 \\ 0 & 0 & G_{11} \end{bmatrix}\begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix} + \begin{bmatrix} G_{31} & 0 & 0 \\ 0 & G_{31} & 0 \\ 0 & 0 & G_{31} \end{bmatrix}\begin{bmatrix} I_{AL0} \\ I_{BL0} \\ I_{CL0} \end{bmatrix}$$

Or if we rewrite the equation:

$$\begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} = G_{ABC}^{-1}\begin{bmatrix} G_{11} & 0 & 0 \\ 0 & G_{11} & 0 \\ 0 & 0 & G_{11} \end{bmatrix}\begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix} + G_{ABC}^{-1}\begin{bmatrix} G_{31} & 0 & 0 \\ 0 & G_{31} & 0 \\ 0 & 0 & G_{31} \end{bmatrix}\begin{bmatrix} I_{AL0} \\ I_{BL0} \\ I_{CL0} \end{bmatrix}$$

The inductor current for phase ***X*** can be determined from:

$$V_{XN} = V_X - RI_X$$

$$I_{XL} = -\left(\frac{I}{R} + G\right)V_{XN} + \frac{V_X}{R} + GR_{GND}(I_A + I_B + I_C)$$

Structural Jacobian

The structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} L & L & L \\ L & L & L \\ L & L & L \end{bmatrix}$$

Jacobian Calculations

The Jacobian Matrix is given by:

$$J_D = G_{ABC}^{-1} \begin{bmatrix} G_{11} & 0 & 0 \\ 0 & G_{11} & 0 \\ 0 & 0 & G_{11} \end{bmatrix}$$

F-7 Reduction Gear

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
S1	ω_1 (export)	$\boldsymbol{T_1}$ (export)	(0) Normal
S2	ω_2 (import)	$\boldsymbol{T_2}$ (import)	(0) Normal

The potential variables are measured in radians/second while the flow variables are measured in Newton-meters.

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} \omega_2 \\ T_2 \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} \omega_1 \\ T_1 \end{bmatrix}$$

Parameters

$\boldsymbol{n_1}$	Number of *teeth* on shaft 1.
$\boldsymbol{n_2}$	Number of *teeth* on shaft 2.
η	Efficiency of Reduction Gears.

States

There are no states for this model.

Equations

The rotational speed of the shafts are proportional to the gear ratio:

$$\omega_1 = \frac{n_2}{n_1}\omega_2$$

The transmitted torque however, must be scaled by the efficiency. Which side of the equation the efficiency applies depends on the direction of the power flow:

if $T_2\omega_2 > 0$

then $T_1 = -\frac{n_1}{n_2}\eta T_2$

else $T_1 = -\frac{n_1}{n_2\eta} T_2$

The first zero crossing of the power should be passed back to the system as a suggested recalculation time.

Structural Jacobian

The structural jacobian is given by:

$$J_{DS} = \begin{bmatrix} D & 0 \\ N & N \end{bmatrix}$$

Jacobian Calculations

The jacobian is given by:

$$J_D = \begin{bmatrix} \frac{n_2}{n_1} & 0 \\ \frac{\partial T_1}{\partial \omega_2} & \frac{\partial T_1}{\partial T_2} \end{bmatrix}$$

The partials of T_1 depend on the partial derivatives of the **if-then-else** function. If the direction of the power flow remains constant over the interval, then the partials are given by:

$$\frac{\partial T_1}{\partial \omega_2} = 0$$

$$\frac{\partial T_1}{\partial T_2} = \frac{T_1}{T_2}$$

F-8 Propeller

The relationship between the torque, angular speed, forward velocity and forces on a propeller are highly complex and nonlinear. While much information is known about the steady-state operation of propellers traversing in the forward direction, little information is available for nonstandard operating conditions. The classical approach is to generate $\boldsymbol{K_t}$ vs $\boldsymbol{J}$ and $\boldsymbol{K_q}$ vs $\boldsymbol{J}$ curves where:

$$J = \frac{V_p}{nD}$$

$$F = K_T(J)\rho D^4 n^2$$

$$T_m = K_Q(J)\rho D^5 n^2$$

where the variables are described in the following sections.

The classical approach works well when $\boldsymbol{V_p}$ (speed of propeller with respect to the fluid) and $\boldsymbol{n}$ (RPM of shaft) are both positive and $\boldsymbol{n}$ is large enough to bring $\boldsymbol{J}$ (advance coefficient) below about 1.5. Outside of this range, little data is provided for most propellers. The classical approach breaks down completely when the shaft speed is zero and $\boldsymbol{J}$ is infinite. Furthermore, there is no way to differentiate between backing down ($\boldsymbol{n}$ and $\boldsymbol{V_p}$ both negative) and having forward way on ($\boldsymbol{n}$ and $\boldsymbol{V_p}$ both positive). The method used for this model is better suited for simulation studies because it essentially uses the angle of attack on the propeller blade as the argument for the thrust and torque coefficients. This model is based on work conducted at the Naval Ship Research and Development Center, Annapolis, MD by D. W. Baker and C. L. Patterson and reflects data and theory developed by I. Ya. Miniovich.

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Mechanical	ω_m (import)	$\boldsymbol{T_m}$ (export)	(0) Normal
Hydrodynamic	$\boldsymbol{u}$ (import)	$\boldsymbol{F}$ (export)	(0) Normal

Note: units are *radians/second*, *Newton-meters*, *meters/second*, and *Newtons*

The Import $\boldsymbol{X_{imp}}$ and Export $\boldsymbol{X_{exp}}$ vectors are defined by:

$$X_{imp} = \begin{bmatrix} \omega_m \\ u \end{bmatrix} \qquad X_{exp} = \begin{bmatrix} T_m \\ F \end{bmatrix}$$

Parameters

D	Diameter of Prop (*meters*)
w	Wake Fraction (PU)
ρ	Density of water (kg/m^3)
$C_T()$	Thrust Coefficient matrix (unlimited rows by 2 columns) first column is Θ in *radians* **[-π π]** second column is Thrust Coefficient in PU.
$C_Q()$	Torque Coefficient matrix (unlimitd rows by 2 columns) first column is Θ in *radians* **[-π π]** second column is Torque Coefficient in PU.

States

(There are no states for this model)

Equations

$$V_p = (1 - w)u$$

$$n = \frac{\omega_m}{2\pi}$$

$$\Theta = \text{atan2}(nD, V_p)$$

$$F = -C_T(\Theta)\rho D^2(V_p^2 + n^2D^2)$$

$$T_m = C_Q(\Theta)\rho D^3(V_p^2 + n^2D^2)$$

Normally, $C_T()$ and $C_Q()$ are specified as data points. Hence some type of interpolations scheme is needed to determine the value of theses functions at intermediate points, as well as the value of the first derivative.

Device Structural Jacobian

The Device Structural Jacobian for all waveforms is given by:

$$J_{DS} = \begin{bmatrix} N & N \\ N & N \end{bmatrix}$$

Jacobian Calculations

Calculate partials with respect to intermediate variables

$$\frac{\partial F}{\partial u} = (1-w)\frac{\partial F}{\partial V_p}$$

$$\frac{\partial T_m}{\partial u} = (1-w)\frac{\partial F}{\partial V_p}$$

$$\frac{\partial F}{\partial \omega_m} = \frac{1}{2\pi}\frac{\partial F}{\partial n}$$

$$\frac{\partial T_m}{\partial \omega_m} = \frac{1}{2\pi}\frac{\partial T_m}{\partial n}$$

Calculate the partials

$$\frac{\partial F}{\partial V_p} = -\rho D^2\left(2C_T(\Theta)V_p + (V_p^2 + n^2D^2)\frac{\partial C_T(\Theta)}{\partial V_p}\right)$$

$$\frac{\partial F}{\partial n} = -\rho D^2\left(2C_T(\Theta)D^2 n + (V_p^2 + n^2D^2)\frac{\partial C_T(\Theta)}{\partial n}\right)$$

$$\frac{\partial T_m}{\partial V_p} = \rho D^3\left(2C_Q(\Theta)V_p + (V_p^2 + n^2D^2)\frac{\partial C_Q(\Theta)}{\partial V_p}\right)$$

$$\frac{\partial T_m}{\partial n} = \rho D^3\left(2C_Q(\Theta)D^2 n + (V_p^2 + n^2D^2)\frac{\partial C_Q(\Theta)}{\partial n}\right)$$

Calculate the partial of Θ with respect to $\boldsymbol{V_p}$ and $\boldsymbol{n}$.

$$\frac{\partial \Theta}{\partial V_p} = \frac{\frac{1}{nD}}{1 + \left(\frac{V_p}{nD}\right)^2}$$

$$\frac{\partial \Theta}{\partial n} = \frac{-\frac{V_p}{n^2 D}}{1 + \left(\frac{V_p}{nD}\right)^2}$$

Of course, the partials of $\boldsymbol{C_T(\Theta)}$ and $\boldsymbol{C_Q(\Theta)}$ with respect to Θ must be determined from the interpolation scheme used.

Putting all of this together:

$$J_D = \begin{bmatrix} \frac{\partial T_m}{\partial \omega_m} & \frac{\partial T_m}{\partial u} \\ \frac{\partial F}{\partial \omega_m} & \frac{\partial F}{\partial u} \end{bmatrix}$$

F-9 Ship Dynamics Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Ship Hydrodynamics	***u*** (import)	***F*** (export)	(0) Normal

Velocity ***u*** is measured in meters/second while force ***F*** is measured in Newtons.

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = [u] \qquad x_{exp} = [F]$$

Parameters

ρ	Density of Salt Water (kg/m^3)	1025.9 kg/m^3 @ 15˚ C.
ν	Kinematic Viscosity of Water (m^2/sec)	1.19×10^{-6} m^2/sec @15˚ C.
$\boldsymbol{G}$	Acceleration of Gravity (m/sec^2)	9.80665 m/sec^2

$\boldsymbol{L}$ Length of Ship (m)

$\boldsymbol{A_S}$ Surface Area of Ship (m^2)

$\boldsymbol{m}$ Mass of Ship (kg)

$\boldsymbol{m_{add}}$ Added Mass Multiplier (PU) (normally between 1.0 and 1.10)

$\boldsymbol{C_a}$ Correlation Allowance

$\boldsymbol{C_f(R_e)}$ Matrix of Frictional Drag Coefficients (2×*n*) or (3×*n*)

- column 1 are Reynolds Number Values
- column 2 are Frictional Drag Coefficient Values
- column 3 are optional first derivative values of the curve
- Note: Values should be provided for negative Reynolds Numbers

$\boldsymbol{C_r(F_r)}$ Matrix of Residual Drag Coefficients (2×*n*) or (3×*n*)

- column 1 are Froude Number Values
- column 2 are Residual Drag Coefficient Values
- column 3 are optional first derivative values of the curve
- Note: Values should be provided for negative Froude Numbers

States

There are no states associated with this device

Equations

The basic equations are given by:

$$R_e = \frac{uL}{\nu}$$

$$F_r = \frac{u}{\sqrt{GL}}$$

$$C_T = C_f(R_e) + C_r(F_r) + C_a$$

$$F = \frac{\rho}{2} u^2 A_S C_T + m_{add} m \frac{du}{dt}$$

The only potential difficulty is performing the evaluation of the drag coefficients $\boldsymbol{C_f(R_e)}$ and $\boldsymbol{C_r(F_r)}$.

Structural Jacobian

The structural jacobian is given by:

$$J_{DS} = [N]$$

Jacobian Calculations

$$J_D = \frac{\rho}{2} A_s \left(M(u)M(u) \left(\frac{L}{\nu} M\left(\frac{dC_f(R_e)}{dR_e} \right) + \frac{1}{\sqrt{LG}} M\left(\frac{dC_r(F_r)}{dF_r} \right) \right) + 2M(u)M(C_t(R_e)) \right) +$$

$$m_{add} m S^{-1}$$

where $\boldsymbol{S}$ is the integration matrix and $\frac{dC_f(R_e)}{dR_e}$ ($\frac{dC_r(F_r)}{dF_r}$) must be determined by either differentiating the $\boldsymbol{C_f(R_e)}$ ($\boldsymbol{C_r(F_r)}$) curve or by interpolating the third colum if so provided.

F-10 Pulse Generator

Interface Variables

Terminal	Potential Variable	Flow Variables	Type
V0	$\boldsymbol{V}$ (export)		Information

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = [] \qquad x_{exp} = [V]$$

Parameters

$\boldsymbol{V_{off}}$	Value of $\boldsymbol{V}$ when off
$\boldsymbol{V_{on}}$	Value of $\boldsymbol{V}$ when on
$\boldsymbol{t_P}$	Matrix of pulse times ($2\times n_t$): column 1 are pulse on times. column 2 are pulse off times. unlimited number of n_t rows.

States

None

Equations

If the time $\boldsymbol{t}$ falls between an on and an off time then $\boldsymbol{V = V_{on}}$, otherwise $\boldsymbol{V = V_{off}}$.

If a discontinuity falls within the time interval, the earliest discontinuity time should be passed back as a recommended recalculation time. The time of the next discontiuity after the time interval should also be made available to the system solver.

Structural Jacobian

There is no structural jacobian matrix for this device.

Jacobian Calculations

There is no jacobian matrix for this device.

F-11 Induction Motor

F-11.1 ABC Model

Interface Variables

Terminal	Potential Variable	Flow Variables	(KCL Group) Type
Phase A	$\boldsymbol{V_A}$ (Import)	$\boldsymbol{I_A}$ (Export)	(1) Normal
Phase B	$\boldsymbol{V_B}$ (Import)	$\boldsymbol{I_B}$ (Export)	(1) Normal
Phase C	$\boldsymbol{V_C}$ (Import)	$\boldsymbol{I_C}$ (Export)	(1) Normal
Neutral	$\boldsymbol{V_0}$ (Import)	$\boldsymbol{I_0}$ (Export)	(1) Normal
Mechanical	ω_m (Import)	$\boldsymbol{T_m}$ (Export)	(0) Normal

Voltages are in volts, currents in amps, angle speed in radians per second and Torque in Newton-meters.

The import $\boldsymbol{x_{imp}}$ and export $\boldsymbol{x_{exp}}$ vectors are given by:

$$x_{imp} = \begin{bmatrix} V_A \\ V_B \\ V_C \\ V_0 \\ \omega_m \end{bmatrix} \qquad x_{exp} = \begin{bmatrix} I_A \\ I_B \\ I_C \\ I_0 \\ T_m \end{bmatrix}$$

Parameters

$\boldsymbol{R_S}$	Stator Resistance (ohms)
$\boldsymbol{R_R}$	Rotor Resistance (reflected to Stator) (ohms)
$\boldsymbol{X_{LS}}$	Stator Reactance (ohms)
$\boldsymbol{X_{LR}}$	Rotor Reactance (reflected to Stator) (ohms)
$\boldsymbol{X_M}$	Mutual Reactance (ohms)
$\boldsymbol{J}$	Moment of Inertia (Kg-m^2)
ω_{bs}	Base Frequency (radians per second)
$\boldsymbol{p_p}$	Pole Pairs
$\boldsymbol{B}$	Windage Torque Factor (Newton-Meters-second)

States

θ Electrical Rotor Angle (radians)

Equations

First calculate the electrical angle:

$$\theta = \theta_0 + \int p_p \omega_m dt$$

Now specify the following stator voltage and current vectors:

$$v_S = \begin{bmatrix} V_A - V_0 \\ V_B - V_0 \\ V_C - V_0 \end{bmatrix} \qquad i_S = \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix}$$

The Rotor voltages and currents as reflected on the stator are:

$$v_R' = \begin{bmatrix} V_{AR}' \\ V_{BR}' \\ V_{CR}' \end{bmatrix} \qquad i_R' = \begin{bmatrix} I_{AR}' \\ I_{BR}' \\ I_{CR}' \end{bmatrix}$$

Calculate the inductances:

$$L_{ls} = \frac{X_{LS}}{\omega_{bs}} I \qquad L_{lr}' = \frac{X_{LR}}{\omega_{bs}} I$$

$$M = \frac{X_M}{\omega_{bs}} I \qquad L_{ms} = \frac{2}{3} M$$

The Stator induction matrix is given by:

$$L_S = \begin{bmatrix} (L_{ls} + L_{ms}) & -\frac{1}{2} L_{ms} & -\frac{1}{2} L_{ms} \\ -\frac{1}{2} L_{ms} & (L_{ls} + L_{ms}) & -\frac{1}{2} L_{ms} \\ -\frac{1}{2} L_{ms} & -\frac{1}{2} L_{ms} & (L_{ls} + L_{ms}) \end{bmatrix}$$

The Rotor induction matrix is given by:

$$L_R' = \begin{bmatrix} (L_{lr}' + L_{ms}) & -\frac{1}{2}L_{ms} & -\frac{1}{2}L_{ms} \\ -\frac{1}{2}L_{ms} & (L_{lr}' + L_{ms}) & -\frac{1}{2}L_{ms} \\ -\frac{1}{2}L_{ms} & -\frac{1}{2}L_{ms} & (L_{lr}' + L_{ms}) \end{bmatrix}$$

The Mutual induction matrix is given by:

$$L_{SR}' = \begin{bmatrix} M(\cos(\theta))L_{ms} & M\left(\cos\left(\theta + \frac{2\pi}{3}\right)\right)L_{ms} & M\left(\cos\left(\theta - \frac{2\pi}{3}\right)\right)L_{ms} \\ M\left(\cos\left(\theta - \frac{2\pi}{3}\right)\right)L_{ms} & M(\cos(\theta))L_{ms} & M\left(\cos\left(\theta + \frac{2\pi}{3}\right)\right)L_{ms} \\ M\left(\cos\left(\theta + \frac{2\pi}{3}\right)\right)L_{ms} & M\left(\cos\left(\theta - \frac{2\pi}{3}\right)\right)L_{ms} & M(\cos(\theta))L_{ms} \end{bmatrix}$$

The Rotor and Stator resistance matrices are given by:

$$R_r' = R_R \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \qquad R_s = R_S \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}$$

The system of equations which need to be solved can be expressed as:

$$\begin{bmatrix} v_S \\ v_R' \end{bmatrix} = \begin{bmatrix} R_s + S^{-1}L_S & S^{-1}L_{SR}' \\ S^{-1}(L_{SR}')^T & R_r' + S^{-1}L_R' \end{bmatrix} \begin{bmatrix} i_S \\ i_R' \end{bmatrix}$$

For a squirrel cage induction motor $v_R' = 0$. Hence the rotor curents can be solved for:

$$i_R' = A_{RS} i_S$$

$$A_{RS} = -(R_r' + S^{-1}L_R')^{-1} S^{-1} (L_{SR}')^T$$

$$A_{RS} = -(SR_r' + L_R')^{-1} (L_{SR}')^T$$

Now the stator currents can be solved for:

$$A_{SS} = (R_s + S^{-1}L_S + S^{-1}L_{SR}'A_{RS})^{-1}$$

$$i_S = A_{SS}v_S$$

The neutral current is solved using KCL;

$$I_0 = -I_A - I_B - I_C$$

The electrical torque is given by:

$$T_e = -\frac{2X_M p_p}{3\omega_{bs}}\left\{\left(M(I_A)M\left(I_{AR}' - \frac{I_{BR}'}{2} - \frac{I_{CR}'}{2}\right) + \right.\right.$$

$$M(I_B)M\left(I_{BR}' - \frac{I_{CR}'}{2} - \frac{I_{AR}'}{2}\right) +$$

$$\left.\left.M(I_C)M\left(I_{CR}' - \frac{I_{AR}'}{2} - \frac{I_{BR}'}{2}\right)\right)\sin(\theta) + \right.$$

$$\left.\sqrt{\frac{3}{2}}(M(I_A)M(I_{BR}' - I_{CR}') + M(I_B)M(I_{CR}' - I_{AR}') + M(I_C)M(I_{AR}' - I_{BR}'))\cos(\theta)\right\}$$

The full torque equation is given by:

$$T_m = -T_e + (JS^{-1} + BI)\omega_m$$

Structural Jacobian

$$J_{DS} = \begin{bmatrix} N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \\ N & N & N & N & N \end{bmatrix}$$

Jacobian Calculations

The jacobian elements corresponding to the electrical variables are easy to calculate. If we partition the jacobian matrix as follows:

$$J_D = \begin{bmatrix} A_{SS} & A_{SS}U_1 & J_{EW} \\ U_1^T A_{SS} & U_1^T A_{SS} U_1 & U_1^T J_{EW} \\ J_{TE} & J_{TE}U_1 & J_{TW} \end{bmatrix}$$

where

$$U_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

The remaining matrices ($\boldsymbol{J_{EW}}$, $\boldsymbol{J_{TE}}$, and $\boldsymbol{J_{TW}}$) are not as easy to calculate:

$$v_S = A_{SS}^{-1} i_S$$

Take the partial wrt ω_m:

$$0 = \frac{\partial (A_{SS}^{-1})}{\partial \omega_m} i_S + A_{SS}^{-1} \frac{\partial i_S}{\partial \omega_m}$$

$$\frac{\partial i_S}{\partial \omega_m} = J_{EW} = -A_{SS} S^{-1} \frac{\partial (L_{SR}' A_{RS})}{\partial \omega_m} i_S$$

$$L_{SR}' A_{RS} = -L_{SR}' (SR_r' + L_R')^{-1} (L_{SR}')^T$$

$$\frac{\partial (L_{SR}' A_{RS})}{\partial \omega_m} = -L_{SR}' (SR_r' + L_R')^{-1} \frac{\partial (L_{SR}')^T}{\partial \omega_m} - \frac{\partial L_{SR}'}{\partial \omega_m} (SR_r' + L_R')^{-1} (L_{SR}')^T$$

$$\frac{\partial L_{SR}'}{\partial \omega_m} = -\begin{bmatrix} M(\sin(\theta)) L_{ms} S p_p & M\left(\sin\left(\theta + \frac{2\pi}{3}\right)\right) L_{ms} S p_p & M\left(\sin\left(\theta - \frac{2\pi}{3}\right)\right) L_{ms} S p_p \\ M\left(\sin\left(\theta - \frac{2\pi}{3}\right)\right) L_{ms} S p_p & M(\sin(\theta)) L_{ms} S p_p & M\left(\sin\left(\theta + \frac{2\pi}{3}\right)\right) L_{ms} S p_p \\ M\left(\sin\left(\theta + \frac{2\pi}{3}\right)\right) L_{ms} S p_p & M\left(\sin\left(\theta - \frac{2\pi}{3}\right)\right) L_{ms} S p_p & M(\sin(\theta)) L_{ms} S p_p \end{bmatrix}$$

Appendix G: WAVESIM Program Files

WAVESIM consists of ten program source code (**.c**) files plus three header (**.h**) files and must be linked to the standard math library. Six of the program source code files and two of the header files are specific to WAVESIM while the remaining files contain application independent code.

WAVESIM SPECIFIC FILES

wavesim.c	Main Executive Routine
wavesim.h	Definition of WAVESIM Structures
waveinit.c	Initialization of Structures Read **device.def**
waveinit.h	Define Initial values of all system parameters
waveread.c	Read and interpret Input File
wavebld.c	Build System
wavewrit.c	Write MATLAB .M file
wavewrta.c	Write MATLAB .M file (continued)

APPLICATION INDEPENDENT FILES

ioliba.c	String Manipulation Routines
iolbia.h	declarations of **ioliba.c** routines
get_file.c	Prompt for and open files
getline.c	Obtain string from an input stream
filebase.c	File name manipulation routines

G-1 Main Program File: `wavesim.c`

`wavesim.c` contains the **`main`** routine which performs the executive functions for WAVESIM:

1.	Initialize device definitions	**`init_devices`**
2.	Print the Header	
3.	Open Files	
4.	Read Input File	**`read_file`**
5.	Build the System	**`build_system`**
6.	Write The Output File	**`write_file`**
7.	Close the Files	

G-2 System Initialization: `waveinit.c`

`waveinit.c` contains the following routines for initializing the system:

`init_devices`	Sets Default Values Calls **`read_device_def`** to read in **`device.def`** file. Debug handler.
`read_device_def`	Reads in **`device.def`** file
`print_system_base`	Prints system base parameters
`print_device_def`	Prints a Device Definition
`print_matrix`	Prints a Matrix
`print_structural_jacobian`	Prints a Structural Jacobian
`read_terminal`	Interprets **`TERMINAL`** subordinate command
`read_parameter`	Interprets **`PARAMETER`** subordinate command
`read_state`	Interprets **`STATE`** subordinate command
`read_function`	Interprets **`FUNCTION`** subordinate command
`read_structural_jacobian`	Interprets **`STRUCTURAL JACOBIAN`** subordinate command
`strip_white`	Strips all blanks, tabs, returns from a string.
`read_matrix`	Reads a matrix.

The hierarchy for the routines in **waveinit.c** is given by:

```
init_devices
  read_device_def
    read_terminal
    read_parameter
      read_matrix
    read_state
    read_function
    read_structural_jacobian
      strip_white
  print_system_base
  print_device_def
    print_matrix
    print_structural_jacobian
```

G-3 Reading Input File: `waveread.c`

`waveread.c` contains the following routines for reading in an input file:

`read_file`	Controls other routines for reading input files Debug handler.
`read_file_device`	Reads and Interprets **`device`** command from input file.
`read_file_default`	Reads and Interprets **`default`** command from input file.
`read_file_node`	Reads and Interprets **`node`** command from input file.
`read_file_time`	Reads and Interprets **`time`** command from input file.
`read_file_debug`	Reads and Interprets **`debug`** command from input file.
`print_debug`	Print debug flag status.
`print_time`	Print simulation time parameters.
`print_device`	Print device characteristics.
`print_system`	Print system characteristics.
`print_node`	Print node characteristics.
`finish_reading_file`	Generate cross references and generally finish the process of developing the initial system.

The hierarchy for the routines in **waveread.c** is given by:

```
read_file
  read_file_device
    print_device
      print_matrix
    read_matrix
  read_file_default
    print_system_base
  read_file_node
    print_node
  read_file_time
    print_time
  read_file_debug
    print_debug
  finish_reading_file
  print_system
    print_device
    print_node
```

G-4 Building the System: `wavebld.c`

`wavebld.c` contains the following routines for building the system:

`build_system`	Identifies System variables and equations Builds cross references Builds System Structural Jacobian Reduces system
`build_system_identify`	Identifies system variables and equations.
`build_system_xref`	Builds cross references within the system.
`build_system_structural_jacobian`	Builds the system structural jacobian matrix.
`build_system_blocks`	Identifies the sequence of blocks for solving system.
`find_block`	Attempts to find a block of a given size.
`print_system_identify`	Prints system information.
`sj_add`	Adds two Structural Jacobian codes.
`sj_sub`	Subtracts two Structural Jacobian codes.
`print_system_block_sjac`	Prints the block owner of each element in the system sructural jacobian matrix.
`print_block`	Prints information about a block.

The hierarchy for the routines in **wavebld.c** is given by:

```
build_system
  build_system_identify
  print_system_identify
  build_system_xref
  build_system_structural_jacobian
    sj_add
    sj_sub
  print_structural_jacobian
  build_system_blocks
    find_block
  print_block
    print_structural_jacobian
```

G-5 Writing MATLAB M-File: `wavewrit.c` and `wavewrta.c`

`wavewrit.c` and **`wavewrta.c`** contain the following routines for writing the output MATLAB M-File:

`write_file`	Calls other routines to generate MATLAB M-File.
`write_file_header`	Prints header information to MATLAB M-File.
`write_file_initialize`	Prints System Initialization parameters to MATLAB M-File.
`write_file_time_loop`	Prints Time Loop algorithm to MATLAB M-File.
`write_file_plot_variables`	Prints algorithm to plot system variables to MATLAB M-File
`write_file_footer`	Prints Footer information to MATLAB M-File.
`write_file_solve_block`	Prints algorithm for solving block to MATLAB M-File.

The hierarchy for the routines in **`wavewrit.c`** and **`wavewrta.c`** is given by:

```
write_file
  write_file_header
  write_file_intialize
  write_file_time_loop
    write_file_solve_block
  write_file_footer
```

G-6 Application Independent Files

Several Support files containing special C functions are required by WAVESIM. These files were written by the author independently of WAVESIM and may contain routines unused by WAVESIM.

G-6.1 `ioliba.c`

`ioliba.c` contains a number of functions for manipulating strings. The functions used by WAVESIM are:

`Stoda`	Converts a string to an array of double precision floating numbers.
`strextract`	Extracts the *n*th word of a string
`strsplit`	Returns the remainder of a string after the *n*th word.
`strstrip`	Strips a string of leading and trailing spaces, tabs, and carriage returns
`strncmpa`	Case insensitive version of **`strncmp`** for comparing the first *n* characters of two strings.
`strcmpa`	Case insensitive version of **`strcmp`** for comparing two strings.

G-6.2 `getline.c`

`getline.c` contains functions for reading in lines from a file and automatically implementing the following features:

1. Comment Lines beginning with **`#`**, **`!`**, or **`%`** are ignored.
2. Blank Lines are ignored.
3. Lines can be continued with **`...`** or **`\`**.
4. If the first word of the line is **`INCLUDE`** followed by a filename, lines are read from that file. (NOTE: A check for recursive **`INCLUDE`** statements is performed to prevent infinite loops)
5. Carriage Returns are truncated.

The functions used by WAVESIM are:

`init_strm`	Initialization function
`get_line`	Function used to read the lines in.

G-6.3 `get_file.c`

`get_file.c` contains functions for opening input and output files. The following features are implemented:

1. Two strings are passed: a **default** filename string and an **argument** filename string. Either or both strings may be empty.
2. If the **argument** filename is specified, the functions attempt to open that file. If opening that file is unsuccessful, the user is prompted to enter a new filename.
3. If the **argument** filename is empty, the user is prompted to enter a filename. If the **default** filename is not empty, it is offered to the user as the default name of the file. If opening the file entered by the user is unsuccessful, the user is prompted to enter a new filename.
4. Users can exit the routine without opening a file by entering only a **q** when prompted for a filename
5. Users can obtain a directory listing by entering a **?** followed by whatever file specification the user desires.
6. Leading and trailing spaces in filenames are truncated.
7. Whatever filename is successfully opened is passed back as a new **default** filename.

The function used by WAVESIM is:

`get_input_file` Function for opening an Input File

G-6.4 `filebase.c`

`filebase.c` contains functions for stripping an extension off of a filename and for returning the extension of a filename. The following features are implemented:

1. An extension is defined as all the characters after the last period (**.**) found after the last directory delimiter (\ for IBM-DOS and / for UNIX). If no such period is found, the extension does not exist.

The function used by WAVESIM is:

`extract_base` Extract the base filename (without extension)

G-7 Makefile

The UNIX **`make`** utility greatly eases the task of developing programs by only compiling those files which have changed since the last compilation. Here is the **`Makefile`** used to generate WAVESIM on a VAXstation 3100:

```
# Makefile for wavesim
#
# For Revision 2.0
#
FILES = wavesim.c ioliba.c getline.c get_file.c filebase.c \
     waveinit.c waveread.c wavewrit.c wavewrta.c \
     wavebld.c
OBJ   = wavesim.o ioliba.o getline.o get_file.o filebase.o  \
     waveinit.o waveread.o wavewrit.o wavewrta.o \
     wavebld.o
HEADER= wavesim.h ioliba.h
CFLAG = -g
COMPILE  = cc
#
#
#
wavesim: $(HEADER) $(OBJ)
     $(COMPILE) -o wavesim  $(CFLAG) $(OBJ) -lm
wavesim.o: $(HEADER) wavesim.c
     $(COMPILE) -c $(CFLAG) wavesim.c
ioliba.o: ioliba.h ioliba.c
     $(COMPILE) -c $(CFLAG) ioliba.c
getline.o: getline.c
     $(COMPILE) -c $(CFLAG) getline.c
get_file.o: get_file.c
     $(COMPILE) -c $(CFLAG) get_file.c
filebase.o: filebase.c
     $(COMPILE) -c $(CFLAG) filebase.c
waveinit.o: $(HEADER) waveinit.h waveinit.c
     $(COMPILE) -c $(CFLAG) waveinit.c
waveread.o: $(HEADER) waveread.c
     $(COMPILE) -c $(CFLAG) waveread.c
wavewrit.o: $(HEADER) wavewrit.c
     $(COMPILE) -c $(CFLAG) wavewrit.c
wavewrta.o: $(HEADER) wavewrta.c
     $(COMPILE) -c $(CFLAG) wavewrta.c
wavebld.o: $(HEADER) wavebld.c
     $(COMPILE) -c $(CFLAG) wavebld.c

#
#
#
lint:
     lint $(FILES)
```