Hybrid Simulator for Controller Synthesis

Philip Demchenko

A Thesis
Submitted to the Faculty of Graduate Studies partial fulfilment of the requirements
for the degree of Master of Science

University of Manitoba
Department of Electrical and Computer Engineering
Winnipeg Manitoba, Canada
December 2004

"Hybrid Simulator for Controller Synthesis"

BY

Philip Demchenko

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Contents

Chapter 1 - Introduction 1

Objective 4
Outline of Thesis 5

Chapter 2 - Simulation Programs 7

PSCAD/EMTDC {(POWER SYSTEMS COMPUTER AIDED DESIGN)/(ELECTROMAGNETIC TRANSIENTS including DIRECT CURRENT)} 8
PSCAD 8
EMTDC 11
MATRIX LABORATORY - "MATLAB" 12
Development Environment 13
Mathematical Function Library 14
Programming Language 15
Graphics 15
Application Program Interfacing 15
SIMULINK 16
Summary 17

Chapter 3 - H_∞ Control Synthesis 19

System Representation 20
System Performance Indicators by Signal Norms 24
H_∞ Algorithm for Control Synthesis 27
Implementation Using Weighted Inputs/Outputs 33
Conclusions 34

Chapter 4 - Interface Development 35

Interfacing PSCAD/EMTDC and MATLAB 36
SIMULINK Interface 40
Interfacing Issues 43
Features Implemented in the New Subroutine 44
Conclusion 47

Chapter5 - Solution of Complex Power Electronic Problem with the New Interface 48

Demonstration of Basic Interface Simulating PWM Control of an Induction Motor 49
Integrating Simulation Features 53

Control Synthesis for a Boost Converter 58
Conclusion73

Chapter 6 - Conclusions and Recommendations 64

Conclusion 74
Recommendations 77

References 80

Appendix A - Derivation of Boost Converter Circuit 85

Appendix B - MATLAB Code 94

List Of Figures

Fig. 2. 1	Master Library of PSCAD 9
Fig. 2. 2	Simplified Structure of EMTDC 12
Fig. 2. 3	Development Environment 14
Fig. 2. 4	SIMULINK Screen Capture 17
Fig. 3. 1	Simple System Representation 21
Fig. 3. 2	Single Input/Single Output System Including perturbations 22
Fig. 3. 3	System Representation 23
Fig. 3. 4	Portioning of D11 Matrix. 31
Fig. 3. 5	Synthesized Controller 32
Fig. 3. 6	Weighted Inputs/Outputs 33
Fig. 4.1	Matlab Interface Original Flow Structure 37
Fig. 4. 2	Flow Chart of Required Steps For Implementing Using MAT_INT 38
Fig. 4. 3	Cut and Paste Script 39
Fig. 4. 4	SIMULINK Interface Flow Structure 40
Fig. 4. 5	Modified Component Script 41
Fig. 4. 6	M-FILE Steps 42
Fig. 4. 7	Variable Input Parameters 45
Fig. 4. 8	Interfacing More Than One Data Signal in the Developed Interface 46
Fig. 5. 1	PSCAD/EMTDC Interfaced with SIMULINK 50
Fig. 5. 2	m-File Interface Code 51
Fig. 5. 3	Induction Motor Results 52
Fig. 5. 4	Simulated Circuit 54
Fig. 5. 5	Proportional Integral Controller Settings 54
Fig. 5. 6	Parameter configuration of the component block 55
Fig. 5. 7	MATLAB command window with initialization parameters and script 56

Fig. 5. 8 Simulation 57
Fig. 5. 9 Boost Converter 59
Fig. 5. 10 State Equations of the Boost Converter Circuit 60
Fig. 5. 11 Response of the Circuit Utilizing the State Equations 61
Fig 5. 12 Magnitude Bode plot of the applied Weight 63
Fig. 5. 13 System Interconnection 64
Fig. 5. 14 Results for the Synthesized Controller 65
Fig. 5. 15 Simulink Simulation 65
Fig. 5. 16 IGBT Pulse Generation 67
Fig. 5. 17 Implemented hybrid simulation to test the H_∞ controller 70
Fig. 5. 18 Hybrid Simulation Input and Output Voltage 70
Fig. 5. 19 Duty Cycle Modulation 72
Fig. A1. 1 IGBT ON 86
Fig. A1. 2 IGBT OFF 87

CHAPTER 1 Introduction

Electric power system studies are facilitated by various software packages. Some examples are listed below. The simulation package PSCAD/EMTDC is a powerful tool in performing the transient analysis of an electrical circuit where the package provides a graphical interface to construct the circuit based on developed library blocks inter-connected, an engine component to simulate, and displays to view simulation results. A second program MATLAB, structured to facilitate in the implementation of matrix computations is useful when the electrical system is developed from its mathematical equations where eigenvalue or frequency domain analysis can easily be applied. Another popular package is PSPICE which provides different tools in evaluating electrical circuits, where the analysis can be done in the frequency or time domain. The three programs differ in the functions that they provide but all have developed graphical interfaces to aid in their utilization. Depending on the application and the requirement of the analysis one program will be beneficial to use over the others, however there are cases when analyzing the electrical system requires more then one program, and the aim of this research is in using the synergy of two separate programs in one simulation study.

Such is the case in integrating the performance of power electronic equipment with control systems. For control design, analysis and simulation, the programs within MATLAB provide the most powerful tools. Within the package there are additional toolboxes for control systems, robust control design, simulink control design, simulink response optimizer, and many more associated within the "MATLAB Product Family" [21]. Within the toolboxes there are graphical aids to generate root locus or Nyquist plots and provide a display of the system connections, and view the system transfer functions to list a few. The toolboxes also contain routines to convert transfer functions to state values or state values to transfer functions. The important point from listing these features is that MATLAB delivers an exceptional control development environment. In the area of power electronics PSCAD/EMTDC is an exceptional program where a large number of circuit elements are readily available within its library. PSCAD/EMTDC also has the ability to handle large electrical circuits, where the limitation for an electrical circuit is the number of circuit nodes and not the number of elements. The program also incorporates methods to accurately simulate the switches within a circuit, the exact turning off instances of a device, and methods to remove solution inaccuracies, to list a few. Again the point of listing some of the feature within PSCAD/EMTDC is to show that the program provides an exceptional environment to handle power electronics.

The problem that exists is when the power electronics are used to stabilize a circuit. The timing of when these devices are turned on and off can remove unwanted signal characteristics, however finding the appropriate controller and controlling the timing pulses is a very difficult task. If these two programs can be made to work synergetically it would allow for the development of controllers to systems where quality is just as important as to quantity. In some electronic equipment the supply voltages have to maintain a constant level and if it is not maintained it could affect other parts of the circuit.

Before one begins to establish an interface between two individual programs one should as far as possible understand the nuances of each program to avoid interfering with established program operations. Limiting the abilities of either program can be devastating where users inadvertently select a feature within that program that now must be detected and disabled when the interface is required. In establishing the interface to allow features of each program to work unconditionally required no detection, which in effect reduces the complexity of its implementation. Users should also have the basic knowledge on how to implement the interface but do not have to be totally familiar with its detailed operation. Also with a hybrid simulation one package will generally incorporate a capability that is not present in the other. Through the interface, methods must be developed to extend the capabilities of the other software package such as not to limit any features during the interface.

When executing a dynamic simulation, basic simulation parameters need to be synchronized. These parameters are the simulation start and stop time, processing of functions that transpire during the simulation (i.e. saving of data at a specific time interval), the incremental time steps or in some cases maximum allowable incremental time step, and finally the storing of results for display or plotting purposes. These critical tasks will have to be addressed in the interface where one program must be selected to initiate all these decisions. This however only leaves the other software program isolated in its solving methods.

1.1 Objective

The goal of this thesis is to explain methods of interfacing diverse simulation and modeling packages. In particular the interfacing between PSCAD/EMTDC and MATLAB/SIMULINK. Within system operating platforms there are routines to allow shared memory among programs which allow the reading and writing of data to memory addresses outside a programs assigned memory. In Windows there are routines to transfer blocks of text from one program to another. An example is the transfer of text from Word to Excel through cut and paste. The challenge with the purposed interface is the flow of data has to be continuous in between the two programs during the entire simulation run. Where a simulation run can take a matter of a few seconds to complete or several hours depending on the complexity.

The second goal is to solve a difficult engineering problem by using the combined systems of PSCAD/EMTDC and MATLAB/SIMULINK. Although PSCAD/EMTDC is a powerful software package, it is limited by the number of models provided within the package. SIMULINK, which is incorporated into MATLAB, has different toolboxes for different fields of study and depending on user requirements additional toolboxes can be added. Such as a toolbox to synthesize a controller. The difficult problem to be solved is in the synthesizing of a controller where unknowns exist within the circuit, however the controller must still be able to stabilize the circuit. The routine to synthesize the controller, which is further discussed in Chapter 3 is iterative, ideal for the MATLAB environment.

To incorporate the same routine for PSCAD would require integrating the mathematical routines into the FORTRAN language, increasing the development time for a new function which may not be applicable to electrical systems. The use of an interface will allow users to model the electrical system in PSCAD/EMTDC and the controller in MATLAB/ SIMULINK for a simulation of the entire system. Based on the results of the simulation a decision can be made on the effectiveness of the synthesis routine. If the toolbox is advantageous for applications to electrical systems then a decision can be made to integrate this particular function into PSCAD/EMTDC.

In integrating both programs together the different disciplines of study can effectively be applied to the electrical system under consideration solving the difficult problem with relative ease. However with these two powerful programs executing at the same time it is expected that the time to complete the simulation will be longer than if one package performing the whole simulation. Within the operating system the two simulation programs are vying to run simultaneously, both requiring processing time and memory. However there is the advantage that the synergy of the combined capabilities will enable solutions to problems that were not solvable before the establishment of the interface.

1.2 Outline of Thesis

To reach the goals of this thesis in developing an interface to incorporate the power of two simulation programs into one study this thesis contains the following. Chapter 2 highlights the features of each software package, where an analysis of each software capabilities is presented. Chapters 3 to 5 present examples of how a difficult problem can be solved with this approach. The difficult problem to be solved is the "H_∞ Control Problem" where it is used to synthesis a controller for an electrical circuit. Chapter 3 describes the principles behind the H_∞ Control Problem. Within the chapter there is the description of what the problem is and how it is solved for systems with measured feedback. How this is applied to the electrical circuit is presented in Chapter 5. Chapter 4 is a discussions of the issues related to creating an interface, the problems encountered and how they were solved. Chapter 5 presents the results of three different simulations. The first two detail the efficiency of the interface and last simulation presents the control synthesis solution using both programs. Finally, Chapter 6 is a discussion on the results with future recommendations.

CHAPTER 2 Simulation Programs

There are several software packages developed to simulate an electrical system, PSCAD/EMTDC, ATP/EMTP, LABVIEW, PSPICE, and MATLAB/SIMULINK to list a few, where each one has its own advantages and disadvantages. Developed into each of these programs are a library of components where users can build their electrical system based on these building blocks. The richness of these libraries, the capabilities of the program, the Graphical User Interface and generally the versatility of the program to identify errors are some of the factors to consider when selecting one software package over another.

Each program develops its library tailored to its solution algorithm, and there are cases that a program has developed libraries that does not exist in another or unable to be easily reproduced. Thus to be able to use developed libraries users will have to switch over to the other software program, or develop their own component for that particular program. An alternative method is to allow communication between two separate simulation packages and utilize the components from both packages. However before establishing any communication between two programs the capabilities of each program must be elaborated. This thesis deals with creating an interface between PSCAD/EMTDC and MATLAB/SIMULINK and thus the following outlines the capabilities of each program.

2.1 PSCAD/EMTDC {(POWER SYSTEMS COMPUTER AIDED DESIGN)/(ELECTROMAGNETIC TRANSIENTS including DIRECT CURRENT)}

PSCAD/EMTDC is a type of Electromagnetic Transient Simulation Program (emtp) available for simulating short term transient phenomena in an electrical network [12,26]. Since its inception the simulation package has been developed into two interacting programs, EMTDC the solution engine and PSCAD the graphical interface tool to provide a virtual display of the electrical network.

2.2 PSCAD

PSCAD is a graphical interface program used to eliminate script writing by the user. Primarily the program provides the visual display of the network with all its interconnections. The electrical network is created by building up the system from individual components that are interconnected by wires in the design page or if required for the system the use of a transmission line. Some of these individual components are provided in the standard library of the package, others can be purchased as an extension to the library, and they can also be developed by the user. A screen capture of the program viewing some of its library components is shown in Figure 2.1. Within the master library there is part of the assembly of a HVDC converter with controls which demonstrates how individual component are connected through a wire to build up a circuit. Since electrical circuits are presented through a graphical display the software package provides tools to aid in the creation of the new components and editors to view the added script.

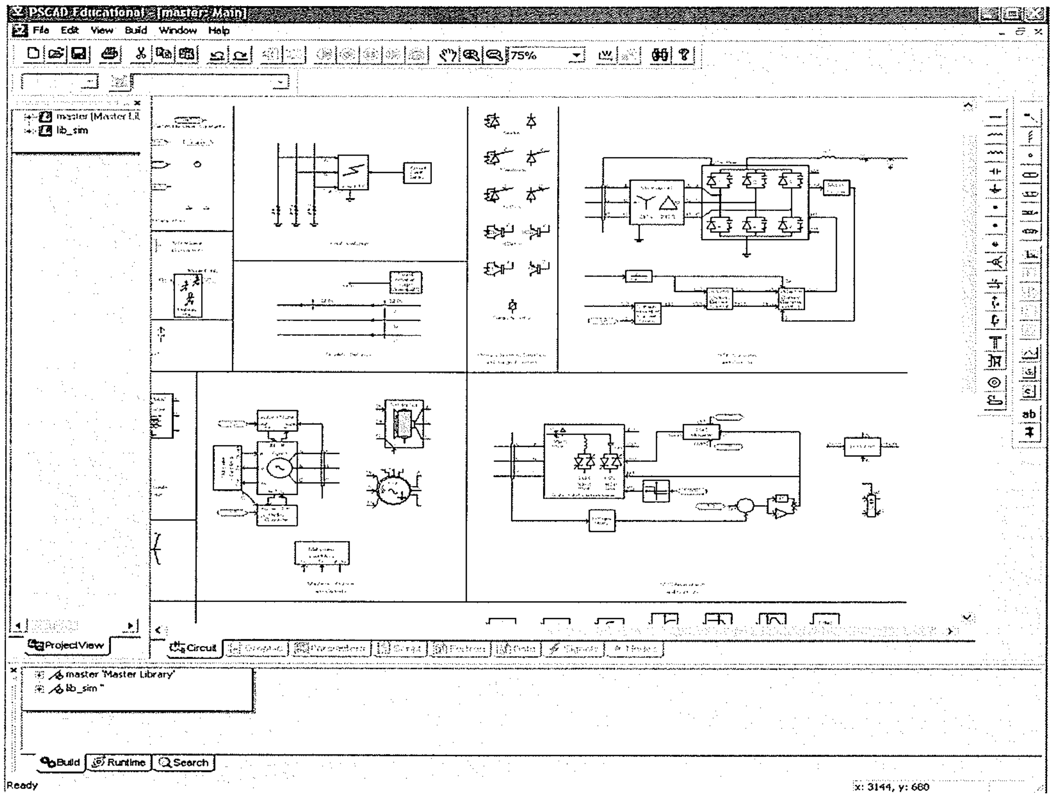


Fig. 2. 1 Master Library of PSCAD

The program also provides a visual display of the simulation results. For an electrical system these results include physical quantities of the devices utilized such as voltage measurements, current measurements, real and reactive power, or electrical/mechanical torque. Within the standard library of the package there are different displays such as graphs and meters that the user can select. PSCAD also includes library components to convert the simulation time domain solution to the phase domain allowing the user the choice of output format.

To input a change into the program there are control blocks in the library package that take the form of buttons, sliders, switches, and timers. These controls are updated at each time step to ensure that the new input value corresponds to the simulation time interval.

The main advantage of the program is that users can construct electrical networks quickly by copying components from the package library to their design page. The more complex the system, the more time required to construct and verify all the parameters in the simulation. Designs can also be incorporated into several pages where data is transferred from one page to another. For PSCAD users who need to develop their own components, there exists development tools to aid in creating the component's graphics and development tools to edit the component's script. If the new component is part of the electrical network there are subroutines that the user must be familiar with to retrieve node data and in most cases update the node data with its solution. In writing the script for the model the user must also be familiar with the solution techniques used in EMTDC.

What occurs at the start of a simulation run in PSCAD is that the graphical representation of the system has to be converted into executable program code. There are several steps involved in this processes however the general flow is such that the graphical representation generates a FORTRAN script based on how the blocks are interconnected. The FORTRAN file is then compiled into an executable program, which is processed within EMTDC. The compiler can be the freeware GNU compiler, or the commercially available compiler DIGITAL FORTRAN. In the conversion process data files containing network information are also generated. These additional files are updated only when changes are made in the system. If the system is constructed by several subsystems then for each subsystem there is a set of data files and those data files are only updated when changes are made in that subsystem. This eliminates the need to convert all the data files for the whole system at the start of a simulation run. When the graphical representation of the system is converted into executable format the simulation process begins by engaging the EMTDC engine.

2.3 EMTDC

The basic function of the EMTDC engine is solving the instantaneous electromagnetic and electromechanical transient time domain of values in an electrical system based on the trapezoidal integration rule. The fundamental loop of the EMTDC is shown in Figure 2.2 [26]. Within the figure there are two locations in the looping algorithm to handle system dynamics; "Master Dynamically Subroutine" DSDYN and "Output Definition Subroutine" DSOUT integrated with the network solution algorithm. DSDYN is intended to handle most control dynamics, while DYOUT handles output quantities after the network solution. By allowing the user to select where the dynamics of the component are incorporated into the calculations they can avoid time step delays between the controls and the electrical network.

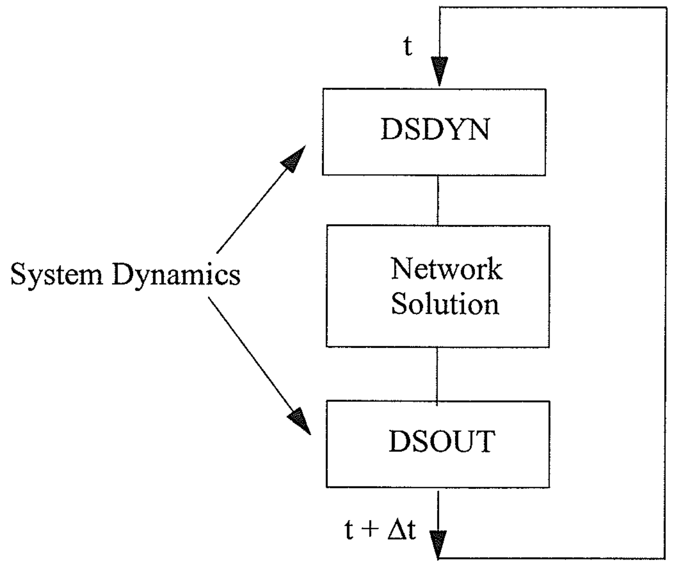


Fig. 2. 2 Simplified Structure of EMTDC

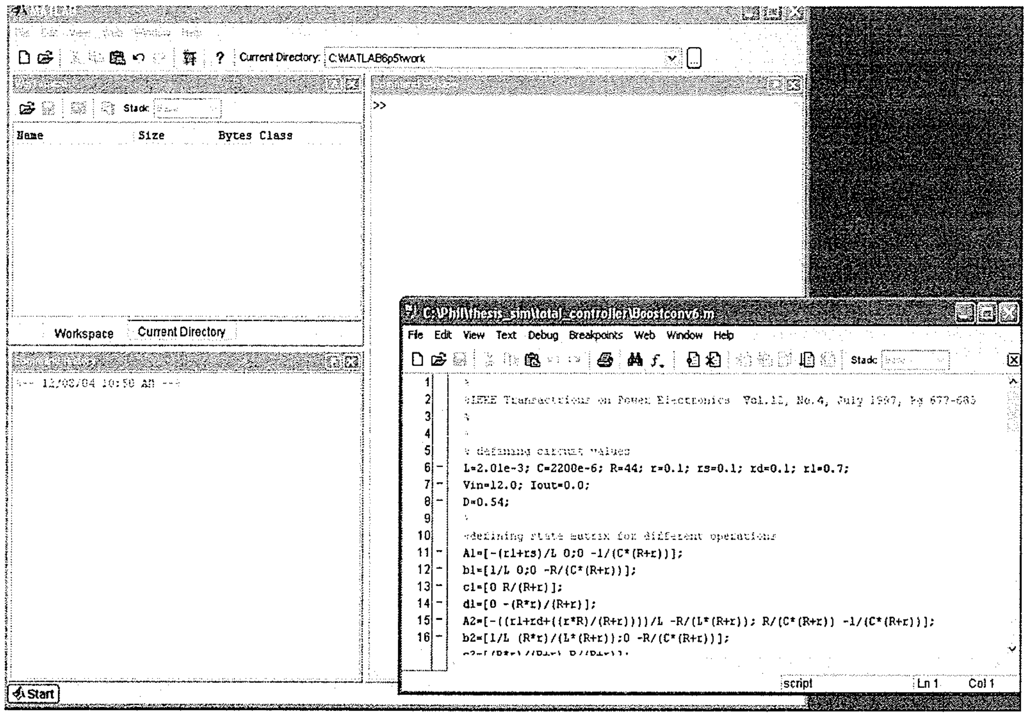
The simulation of an electrical system is based on converting the elements within the stem into their Norton Equivalent. Changes to the system are modeled using current source injectors that are updated at each time step. The solution to the network then reduces to solving; $[V]=[G]^{-1}*[I]$. Utilizing the information from the previous time step and trapezoidal integration with a fixed time step the solution to the instantaneous time values can be obtained. The choice of using trapezoidal integration is for its preservation of stability over a range of time steps, which holds for linear systems.[3,25]

2.4 MATRIX LABORATORY - "MATLAB"

MATLAB is a software program developed to implement mathematical routines in the computer environment. The software program provides an interfacing display and a library of routines that can perform any mathematical calculation available on a high performance calculator. The program exceeds the ability of the calculator by allowing users to develop new routines or to utilize several developed routines to solve mathematical problems. The major advantage of MATLAB is how it handles data arrays without indexing or first dimensioning. The program also assumes all data input to be an array and mathematical operators are operators for matrices. The MATLAB program consists of 5 basic parts: The Development Environment, Mathematical Function Library, Programming Language, Graphics, and Application Program Interface (API) [20,21,22].

2.4.1 Development Environment

The Development Environment as shown in Figure 2.3 is the main graphical interface to the program, and the starting and ending point to all MATLAB mathematical computations, or programs including simulations. The Development Environment encompasses several working environments for executing a MATALB program, creating MATLAB scripts, displaying results, and displaying simulation/environment parameters. The Development Environment consists of MATLAB Desktop, where users have access to the Command Window to perform all tasks, the editor window to type MATLAB script, debugger to aid in programming issues, browsers to view various data contained in MATLAB memory and previous commands executed.


Current Directory: C:\MATLAB6p5\work
Name Size Bytes Class
Workspace
Current Directory
>>
C:\Phil\thesis_sim\total_controller\Boostconv6.m
File Edit View Text Debug Breakpoints Web Window Help
Stack
script
Ln 1
Col 1
Start

```
%
%IEEE Transactions on Power Electronics  Vol.12, No.4, July 1997, Pg 677-685
%
%
% defining circuit values
L=2.01e-3; C=2200e-6; R=44; r=0.1; rs=0.1; rd=0.1; rl=0.7;
Vin=12.0; Iout=0.0;
D=0.54;
%
%defining state matrix for different operations
A1=[-(rl+rs)/L 0;0 -1/(C*(R+r))];
b1=[1/L 0;0 -R/(C*(R+r))];
c1=[0 R/(R+r)];
d1=[0 -(R*r)/(R+r)];
A2=[-((rl+rd+((r*R)/(R+r))))/L -R/(L*(R+r)); R/(C*(R+r)) -1/(C*(R+r))];
b2=[1/L (R*r)/(L*(R+r));0 -R/(C*(R+r))];
```

Fig. 2. 3 Development Environment

2.4.2 Mathematical Function Library

The Mathematical Function Library is a library of computational algorithms included with the software package. MATLAB is developed to handle matrix computations and hence incorporates into its library packages with necessary algorithms. Also included into the library are standard mathematical routines found on any scientific calculator. The library package is developed such that large and complex computational routines can be built from smaller ones existing within the standard library, and newly developed routines can easily be shared with others.

2.4.3 Programming Language

MATLAB has developed a high level programming language for users to easily implement their own routines. MATLAB incorporates standard programming features such as conditional statements, data structures, and object oriented programming. Within the language users can develop complex applications with appropriate Graphical Users Interface with simple commands and formatting issues in the text editor provided with the program.

2.4.4 Graphics

To aid in displaying the results, MATLAB incorporates graphical tools. These tools aid in displaying two-dimensional or three-dimensional graphs, along with creating presentations involving animation. For user-developed programs there are tools to aid in construction of a display window to input or output program parameters. These complex graphical interfaces can be customized through the Graphical Development Editor.

2.4.5 Application Program Interfacing

Application Program Interface (API) is a library of routines developed to allow communication with other applications. One advantage in utilizing API is that the programs or routines already developed in another computer language such as C, C++, or FORTRAN can be incorporated into a MATLAB program. This communication channel can also be reversed to allow MATLAB to integrate into C, C++ or FORTRAN programs. Previously developed routines in either language can be interchanged through the use of the API routines, decreasing the development time of the simulation. API can also be used to increase the performance of a simulation. For calculations involving multiple iterations the computational time of MATLAB is quite high. To increase its performance, functional routines may be developed in another language where the efficiency is quite high in repetitive looping and only the result is channeled back into MATLAB. The reverse is that the MATLAB environment may be utilized to execute a routine already available within its library package.

2.4.6 SIMULINK

An additional extension to MATLAB is the ability to study system dynamics with the environment dependent program SIMULINK as shown in Figure 2.4 [18,21]. This additional program with its GUI provides an environment to model and to simulate dynamical systems similar to that of PSCAD/EMTDC. A capability of SIMULINK allows for variations of how the user can define their models. Because MATLAB is a mathematical program a component block in SIMULINK can either be represented by simple transfer function or it can be represented by its states. SIMULINK also allows for the modeling of the component for discrete time studies (z-domain). Within SIMULINK users can specify simulation parameters similar to PSCAD along with a different solver: Rung-Kutta, Domand-Prince, and Euler to name a few. Unlike EMTDC where the program only utilizes trapezoidal integration as the solver, SIMULINK is equipped with different solution algorithms. Within the simulation parameters section the user can select result data to be written back into MATLAB, allowing other routines to access the data. These capabilities make SIMULINK an ideal environment to study system dynamics an it is widely used in multiple disciplines where real elements are represented by their mathematical equations.

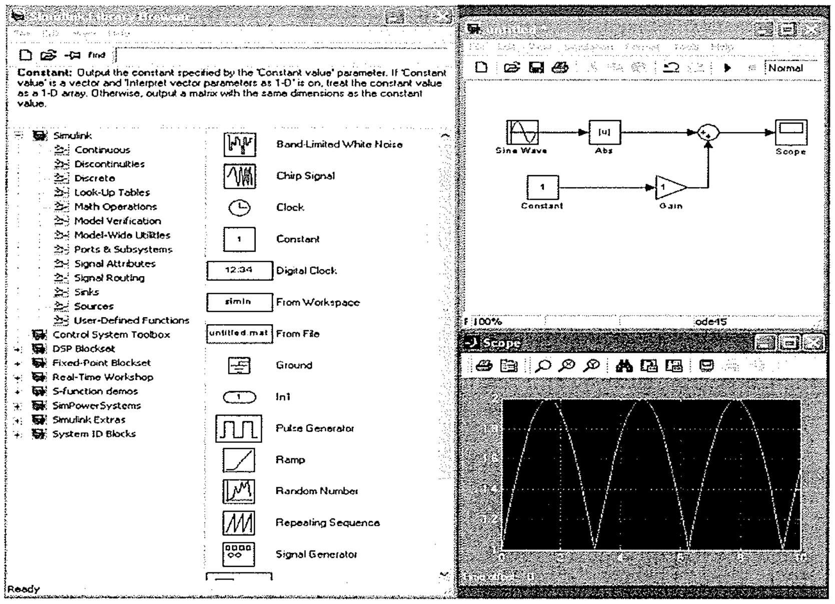

Constant: Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Fig. 2. 4 SIMULINK Screen Capture

2.5 Summary

PSCAD/EMTDC is designed to simulate the transients of an electrical system on a computer. The structure of the program allows users to develop the electrical system by a graphical interface. The program has the previously mentioned features of an added library, which contains electrical components that users can use to make up their system, and tools to develop new components. The advantage of such a program is that users can construct new systems by applying the basic building blocks within the added library, and test how these systems will react under applied conditions. Although the advantages of the program are mainly with the graphical interface, the program would not exist without the solution engine EMTDC, which preforms the calculations for the electrical network. The EMTDC program is a continuous loop that solves the network equations by using trapezoidal integration. MATLAB on the other hand is a different type of program that was primary developed to implement mathematical routines through its graphical interface. Provided with the software package is a development environment that users can use to create new routines or implement existing ones to solve complex mathematical problems. An additional development environment that is part of MATLAB is SIMULINK, used for studying system dynamics. The flexibility of how the system can be modeled either by transfer function or states, and the variations of solution algorithms are some of the features of the program. Independently the performance of these two software packages is exceptional, however interfacing them together there is the expectation of providing an even stronger simulation environment where the benefit is in the extended libraries that would be accessible without having to redevelop them for the other package. This interface can be a vital tool to solving difficult engineering problems that each program alone would not be capable of performing.

CHAPTER 3 H_∞ Control Synthesis

A complex problem in control engineering is designing controllers for systems that contain unknown variables or are heavily subjected to a disturbance. In designing a controller for this environment it must be stable. That is, when the controller is active the output of the system has to maintain a given operating point within a set tolerance. One method to find such a controller is known as the H_∞ control synthesis problem that has the ability to synthesize a controller for these environments. What differs this method from others is the way the performance of the system is measured, known by the H_∞ norm.

The ability of the synthesis procedure is that given a system in terms of its mathematical equations there exists a procedure to find a controller that maintains its performance objectives. How one accurately models the system or determines the performance of the output will determine the effectiveness of the controller. Given any inaccuracies within the model, the controller must still maintain its performance objectives. How one incorporates the inaccuracies into the design and the method of measuring the performance of the systems depend on the method implemented. Furthermore tests to verify the controllers performance can be established by having different software packages simulate the different systems environment. Such is the case for the application of the control synthesis routine applied to a power electronic circuit with switching. PSCAD accurately simulates the electrical circuit and SIMULINK the controller. The first section of this chapter details on the method to include inaccuracies into the system model. The discussion first begins with systems that do not include disturbances and then progresses to the system representing the H_∞ controller.

The core of the synthesis procedure uses an iterative-based method to determine the optimal controller. The iterative procedure used is discussed within this chapter with the overall goal of utilizing the technique in a PSCAD/EMTDC simulation. The developed routine for the synthesis procedure already exists in MATLAB and will be integrated into a PSCAD/SIMULINK simulation to ascertain control designs with applications to power electronics.

3.1 System Representation.

The designing of controllers begins with specifying what the output process must maintain given an input. With the specifications the next step is the choice of controller. A feedback controller that is located in the feedback path or a feed forward controller that is shown in Figure 3. 1 [5,6,8,29] Also designers must take into account what influences affect the system. This involves situations where the input levels are not always constant and the output levels fluctuate independent of the input. These permutations are common to real world systems, and how one accounts for these permutations can differ.

Consider a simple single-input/single-output system (SISO) shown in Figure 3. 1 which shows a standard unity feedback system for robust control design.

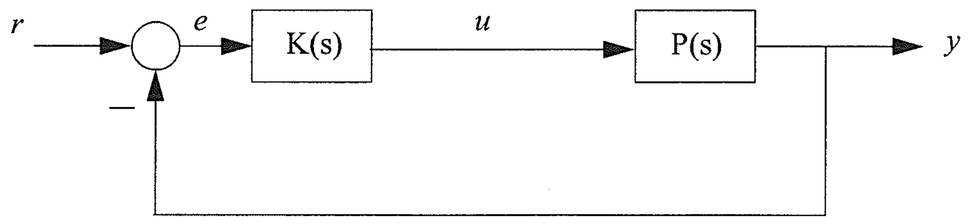


Fig. 3. 1 Simple System Representation

The variables in the figure are:

u = Control signal / (Controller output)
y = Plant output
e = Tracking error
r = Reference input

This system uses a tracking error signal (e) derived from the difference between an applied reference signal (r) and the measured system output (y). The tracking error signal is applied to the controller (K(s)) that produces a control output (u) to the system (P(s)) in turn affecting the system output (y). This representation does not include the uncertainties such as disturbances within the system that the plant is subjected to in its operating environment, and measurement noise introduced through the sensors. One method to include theses perturbations is shown in Figure 3. 2. [5,6,8,29]

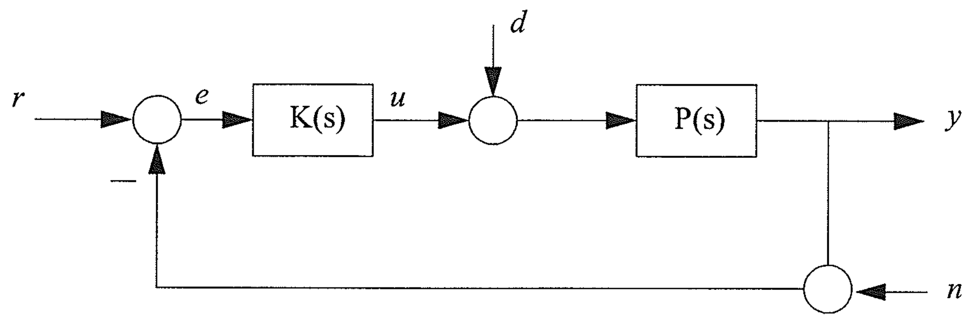


Fig. 3. 2 Single Input/Single Output System Including perturbations

Where the variables to Figure 3.2 are;

u = *Control signal / Controller output*

y = *Plant output*

e = *Tracking error*

r = *reference input*

n = *Sensor noise*

d = *Plant disturbances*

In this representation there are two extra input signals added to Figure 3.1, one titled as the Sensor Noise (n), and the other as Plant Disturbances (d). The Sensor Noise input represents the noise associated with measurement devices, and noise that affects the reference signal. The Plant Disturbance input accounts for conditions when the system output varies under a stable control output signal. It is assumed that the additional inputs must be bounded and that the perturbations to the system are additive.

When dealing with multiple-input/multiple-output systems (MIMO) system representation becomes more complicated due to the interaction between the input signals and system outputs. A simplified representation that includes both SISO and MIMO systems is shown in Figure 3. 3 [2,4,8,15,26,29].

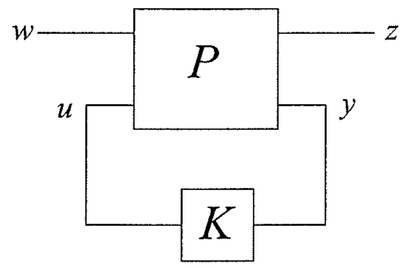


Fig. 3. 3 System Representation

Reducing the variable of Figure 3.3 to:

w = *External inputs (disturbances, sensor noise, and commands)*
z = *Performance output (error signal)*
y = *Measured variables*
u =*Control input*

Although Figure 3.3 is in a simplified form, one has to associate the external inputs to the appropriate location as shown in Figure 3.2. With the system represented in Figure 3.3 all the inputs are arranged to the left side of the figure blocks and all outputs are arranged to the right side of the blocks. Taking this configuration and reversing the orientation of the inputs and outputs one can see that this system representation follows the format of an equation and thus the use of state equations to describe the system becomes equivalently ideal for SISO or MIMO systems.

The state equations used to represent the plant in Figure 3.3 are given in Equation 3.1 [4,28,29].

$$\frac{dx}{dt} = Ax(t) + B1w(t) + B2u(t)$$
$$z = C1x(t) + D11w(t) + D12u(t) \qquad \text{(EQ 3.1)}$$
$$y = C2x(t) + D21w(t) + D22u(t)$$

In state space representation, matrices can now be utilized for systems with multiple states, multiple inputs and multiple outputs. Another advantage of the state space format is that the time domain analysis of the system can easily be performed from the state equations using a simple integration technique. However if frequency domain analysis is required the state space equations can be converted by utilizing Laplace Transform on Equation 3.2 to represent the system in terms of its transfer functions.

$$\begin{bmatrix} \frac{dx}{dt} \\ z \\ y \end{bmatrix} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \qquad \text{(EQ 3.2)}$$
$$\Updownarrow$$
$$G(s) = C(sI - A)^{-1}B + D$$

3.2 System Performance Indicators by Signal Norms

To measure and compare the performance of the system, the preferable practice is to use signal norms. Signal norms are very similar to a vector norm (a measurement of length) and thus must satisfy the basic properties of any norm:

$$\begin{aligned}
&\|u\| > 0 \\
&\|u\| = 0 \Leftrightarrow u(t) = 0 \\
&\|au\| = |a|\|u\| \qquad \forall a \in R \\
&\|u + v\| \le \|u\| + \|v\|
\end{aligned}$$

In control engineering there are several ways to define a signal norm; however, for this document it will be limited to one type.

The signal norm of interest is known as the H_∞ norm. This norm is a measure of the worst possible performance of a system. The choice of using the H_∞ norm has the advantage that the performance of the system can be analyzed in the frequency domain, while the specifications can be given in the time domain. Most systems operate in a specific frequency range, and disturbances may occur within the range of operation. How much of a problem these disturbances can cause on the output, and at what frequencies these disturbances cause the maximum damage are found by calculating the H_∞ norm. Thus with this calculation, filters can be designed on the input to block these disturbances, or a feedback controller can be designed to compensate for disturbance on the input to keep the overall system stable and within specified design parameters.

The definition of the H_∞ norm [4,31] is represented by Equation 3.3 which can be interpreted as the maximum factor by which a sinusoidal input is magnified by the system over the entire frequency range, adjusted incrementally. Note the norm does not indicate at what frequency this occurs only the level of magnification. Defining the inputs as the disturbances, the norm gives the worst possible disturbance that the system endures. To calculate Equation 3.3 the system modelling is done in the frequency domain, however the plant model is derived by its states requiring the conversion to its frequency domain equivalent shown in Equation 3.2.

$$\|G\|_{\infty} = \sup_{\omega} \ \sigma_{max}[G(j\omega)] \qquad \text{(EQ 3.3)}$$

σ_{max} is the maximum singular value

sup is the Supermum or Least Upper Bound

An alternative description of what the H_{∞} norm represents is given by the transfer function of the system, and magnitude plots of every possible input to output transfer function. The maximum peak value on the magnitude plot identifies the H_{∞} norm. Its the highest possible gain at a specific frequency, although the norm only retains the gain information. Ideally the gain from disturbances to the output should be small.

In systems with a wide frequency range the method for finding the H_{∞} norm by the definition is unpractical due to the excessive number of incremental steps, or if the incremental frequency step is too large the peak value will not be detected. An alternative method for finding the H_{∞} norm is an iterative based method derived from the systems state equations that works on decreasing a variable until conditions are no longer satisfied. Proofs on this method are in the reference paper [4]. The method begins with forming the Hamilton Matrix Equation 3.4.

$$H = \begin{bmatrix} A & \gamma^{-2}BB^T \\ -C^TC & -A^T \end{bmatrix} \qquad \text{(EQ 3.4)}$$

where the Hamilton Matrix must satisfy the following two conditions:

I. it has no eigenvalues on the imaginary axis (jω axis)

II. If (I) holds then the matrix will have n eigenvalues with Re(s) < 0 and n eigenvalues with Re(s) > 0, and there exists basis vectors within the invariant subspace corresponding to the eigenvalues with Re(s) < 0 (the eigenvectors of the matrix). Finding these basis vectors and partitioning them in the form $Im\begin{bmatrix} X_1 \\ X2 \end{bmatrix}$, where X_1 is non singular. [4]

The iterative search starts by providing an initial value of γ. The matrix H is then checked for eigenvalues on the imaginary axis. If H has no eigenvalues on the imaginary axis γ is then decreased until H has eigenvalues on the imaginary axis. When this happens γ is then increased. The H_∞ norm of the system G(s) is then defined as $\|G(s)\|_\infty < \gamma$.

With the definition of the H_∞ norm we can now proceed to designing a controller based on this performance measure.

3.3 H_∞ Algorithm for Control Synthesis

The procedure for synthesizing a controller implementing the H_∞ technique begins with the objective of "*finding a controller K, such that the H_∞ norm of the closed loop transfer function from w to z is less than a given positive number* γ, *under the constraint that K stabilizes P*" [4] based on Figure 3.3. The controller K in this definition is a sub-optimal controller which, by selecting a value for γ sets the minimal requirements of the control objective allowing the convergence to a solution which may not be the optimal one. The implementation of γ within the control synthesis procedure follows the same procedure as in finding the H_∞ norm. It begins with γ and tests whether there is a violation within the constraints; γ is then decreased until there is a violation of the constraint or a preselected $\gamma_{\min}$ is reached. In defining γ there is an initial starting point and by defining a $\gamma_{\min}$ there is an expectation of what the controller is to achieve, which may not be obtainable.

To begin there are various techniques to solve the H_∞ control synthesis problem: Coprime Factorization - polynomial approach [8,14], Frequency Domain method - J-Spectral Factorization [15,16], and Time Domain method - state space / Riccati equations [4]. The technique used in this thesis is the Time Domain - state space method and thus further discussion on solving the H_∞ control synthesis problem is limited to this technique. The state space method was introduced in 1989 and solves for a contingence of feedback problems. The procedure outlined in this section is limited to systems with measured feedback, which applies to practical applications.

The first part of the Time Domain method begins with an iterative search in finding the minimum γ, or finding the H_∞ norm of the plant. From the plant state equations two separate Hamiltionian matrices are derived as shown (3.5) and (3.6). These two matrices hold the properties associated with the Hamilton Matrix of (3.4).

$$H = \begin{bmatrix} A & R \\ Q & -A^T \end{bmatrix} \qquad \text{(EQ 3.5)}$$

$$H_\infty = \begin{bmatrix} A & 0 \\ -C_1^T C_1 & -A^T \end{bmatrix} - \begin{bmatrix} B \\ C_1{}^T D_{1\infty} \end{bmatrix} R^{-1} \begin{bmatrix} D_{1\infty}^T C_1 & B^T \end{bmatrix}$$

(EQ 3.6)

$$R = D_{1\infty}^T D_{1\infty} - \begin{bmatrix} \gamma^2 \cdot I & 0 \\ 0 & 0 \end{bmatrix} \qquad where \quad D_{1\infty} = \begin{bmatrix} D_{11} & D_{12} \end{bmatrix}$$

$$J_\infty = \begin{bmatrix} A^T & 0 \\ -B_1 B_1^T & -A \end{bmatrix} - \begin{bmatrix} C^T \\ B_1 D_{\infty 1}^T \end{bmatrix} \tilde{R}^{-1} \begin{bmatrix} D_{\infty 1} B_1^T & BC^T \end{bmatrix}$$

(EQ 3.7)

$$\tilde{R} = D_{\infty 1} D_{\infty 1}^T - \begin{bmatrix} \gamma^2 \cdot I & 0 \\ 0 & 0 \end{bmatrix} \qquad where \quad D_{\infty 1} = \begin{bmatrix} D_{11} \\ D_{21} \end{bmatrix}$$

The basis for defining the above matrices to hold the properties of the Hamiltionian Matrix is that the Hamiltionian Matrix represent a solution to the Algebraic Riccati Equation (3.8), which is also a solution to the control synthesis problem.

$$A^T X + XA + XRX - Q = 0$$

(EQ 3.8)

where

$$X = X_2 \cdot X_1^{-1} 1$$

(EQ 3.9)

From the work done by Doyle the solution to the problem reduces to finding three conditions [4,2]:

1. Within the matrix H_∞ there is $X_\infty = X_2 * X_1^{-1}$ such that $X_\infty \geq 0$ (solution to the Algebraic Riccati Equation)

2. Within the matrix J_∞ there is $Y_\infty = Y_2 * Y_1^{-1}$, such that $Y_\infty \geq 0$

3. The spectral radius of $(X_\infty Y_\infty)$ (3.10) is less than γ^2.

$$\rho(X_\infty) = \underset{1 \le i \le n}{max|\lambda_i|} \qquad \textbf{(EQ 3.10)}$$

Included into the initial iterative procedure is not only the finding of the minimum value of γ but also the finding of γ to satisfy the three conditions above. If all conditions exist then there is a solution to the control synthesis problem and the synthesis procedure becomes a simple matter of solving equations utilizing the values of X_∞ and Y_∞:

$$F_\infty = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = -R^{-1}(D_{1\infty}^T C_1 + B^T X_\infty) \qquad \textbf{(EQ 3.11)}$$

where $F_1 = \begin{bmatrix} F_{11} \\ F_{12} \end{bmatrix}$

$$L_\infty = \begin{bmatrix} L_1 & L_2 \end{bmatrix} = -(B_1 D_{\infty 1}^T + Y_\infty C^T)\tilde{R}^{-1} \qquad \textbf{(EQ 3.12)}$$

where $L_1 = \begin{bmatrix} L_{11} & L_{12} \end{bmatrix}$

The next step in the synthesis of the controller requires subdividing the relationship between the performance variables (z) and the external inputs (w). The the D_{11} elements of matrix are further divided to form $D_{11} = \begin{bmatrix} D_{1111} & D_{1112} \\ D_{1121} & D_{1122} \end{bmatrix}$. Where this subdivision of the matrix is based on indexing D_{11} by the number of measurements (y), number of control inputs (u), total number of outputs (z + y), total number of inputs (w + u) and the offset indexing value of 1 Figure 3. 4.

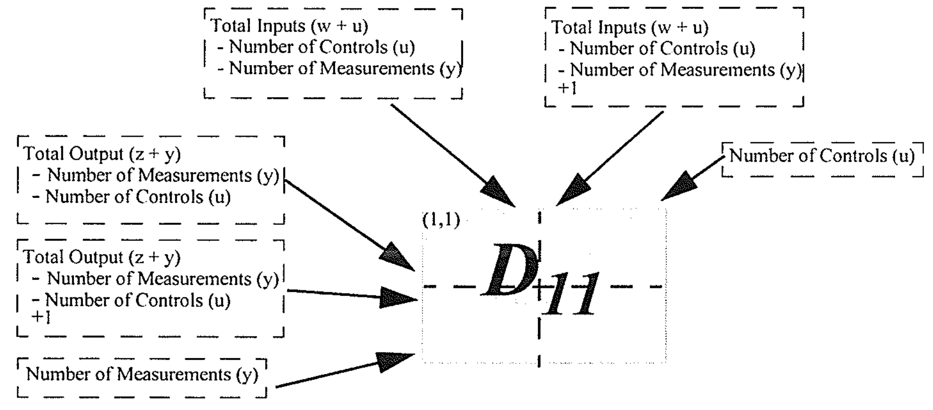


Fig. 3. 4 Portioning of D_{11} Matrix.

Note that in forming this matrix elements of D_{11xx} can be null; to emphasize this if D_{11} consists of only one row then D_{1111}, D_{1112}, $D_{1121}= 0$ and $D_{1122} = D_{11}$. Depending on the existence on elements of D_{11xx} a secondary matrix is calculated.

$$\boxed{\begin{array}{ll} \hat{D}_{11} = -D_{1122} & \textit{If } D_{1111} \textit{ or } D_{1112} \textit{ is a null matrix} \\ \hat{D}_{11} = -D_{1121}D_{1111}^T(\gamma^2 I - D_{1111}D_{1111}^T)^{-1}D_{1112} - D_{1122} & \textit{otherwise} \end{array}} \quad \text{(EQ 3.13)}$$

$$\boxed{\begin{array}{ll} \hat{D}_{21} = I & \textit{If } D_{1112} \textit{ is a null matrix} \\ \hat{D}_{21} = I - \gamma^{-2}D_{1112}^T D_{1112} & \textit{elseIf } D_{1111} \textit{ is a null matrix} \\ \hat{D}_{21} = I - D_{1112}^T(\gamma^2 I - D_{1111}D_{1111}^T)^{-1}D_{1112} & \textit{otherwise} \end{array}} \quad \text{(EQ 3.14)}$$

$$\boxed{\begin{array}{ll} \hat{D}_{12} = I & \textit{If } D_{1121} \textit{ is a null matrix} \\ \hat{D}_{12} = I - \gamma^{-2}D_{1121}D_{1121}^T & \textit{elseIf } D_{1111} \textit{ is a null matrix} \\ \hat{D}_{12} = I - D_{1121}(\gamma^2 I - D_{1111}^T D_{1111})^{-1}D_{1121} & \textit{otherwise} \end{array}} \quad \text{(EQ 3.15)}$$

With all the above values computed the controller can not be synthesized using the following equations:

$$K_a = \left[\begin{array}{c|cc} \hat{A} & \hat{B}_1 & \hat{B}_2 \\ \hline \hat{C}_1 & \hat{D}_{11} & \hat{D}_{12} \\ \hat{C}_2 & \hat{D}_{21} & 0 \end{array}\right] \qquad \text{(EQ 3.16)}$$

$$\hat{A} = A + BF_\infty + \hat{B}_1 \hat{D}_{12}^{-1} \hat{C}_2 \qquad \text{(EQ 3.17)}$$

$$\hat{B}_1 = -Z_\infty^{-1} L_2 + \hat{B}_2 \hat{D}_{12}^{-1} \hat{D}_{11} \qquad \text{(EQ 3.18)}$$

$$\hat{B}_2 = Z_\infty^{-1}(B_2 + L_{12})\hat{D}_{12} \qquad \text{(EQ 3.19)}$$

$$\hat{C}_1 = F_2 + \hat{D}_{11} \hat{D}_{21}^{-1} \hat{C}_2 \qquad \text{(EQ 3.20)}$$

$$\hat{C}_2 = -\hat{D}_{21}(C_2 + F_{12}) \qquad \text{(EQ 3.21)}$$

$$Z_\infty = (I - \gamma^{-2} X_\infty Y_\infty)^{-1} \qquad \text{(EQ 3.22)}$$

The final step of the algorithm is to set the final limits of Ka to match that of the desired controller K.

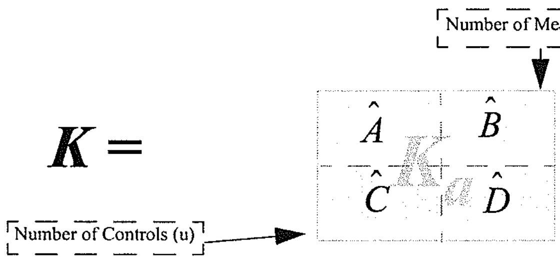


Fig. 3. 5 Synthesized Controller

With any system there still is the possibility that there is no solution for controller. One method for increasing the possibility of finding a controller is to apply weights to the system, where weights define what is important to the system.

3.4 Implementation Using Weighted Inputs/Outputs

The application of weights in controller design is to find controllers of reduced order while still maintaining design control objectives. In the above method the synthesized controller is designed for a wide bandwidth. To reduce the bandwidth of the controller one can apply weights to the inputs/outputs line to emphasize the frequency range that is highly susceptible to disturbances while decreasing the importance of the controller response to frequencies outside this range, while still maintaining control objectivity. This weighting technique is known as frequency dependent weighting and is used most commonly. An example of where weighting can be utilized is in monitoring the system outputs where, different sensors have their own characteristics and their own frequency range, where they are prone to noise. In utilizing the weighting method one applies weights to where the specific frequency range of noise may occur and then synthesizes the controller to compensate for this flaw.

The implementation of weights in the synthesis procedure affects the finding of the H_∞ norm of the transfer function from w to z as seen in Figure 3. 6. [1,2,16,19,17]

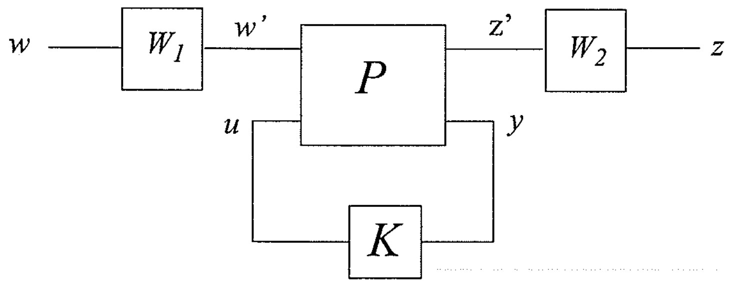


Fig. 3. 6 Weighted Inputs/Outputs

By applying the weights the transfer function from w to z meets the design criteria, while the transfer function w' to z' is designed for a practical (cost effective, reduced complexity) controller. When implementing the weights into the design it will be assumed that the system is decoupled, thus allowing for a diagonal weight matrix.

3.5 Conclusions

The H_∞ controller has the advantage of including into the design uncertainties about the environment the system will operate in and uncertainties about the plant. These uncertainties are included into the model as perturbations to the system. The effect of these perturbations and the control inputs on the output of the system are measured by the use of signal norms and for this case the norm of interest is the H_∞ norm. This norm classifies the maximum gain of the system and in the synthesis procedure of the controller the performance measure is the maximum gain of the system output that is to be monitored to the perturbation signals affecting the system. There is also the application of weights to reduce the order of the controller and to ensure a convergence to a solution of the problem definition.

CHAPTER 4 Interface Development

This chapter discusses the development of the interface between different simulation tools, and is hence a key contribution of this thesis. In developing a method to interface the simulation packages PSCAD and SIMULINK the interface between PSCAD and MATAB was first utilized to understand the complexity of interfacing the two simulation programs. The PSCAD/MATLAB interface, developed by The Manitoba HVDC Research Center consisted of a subroutine that interface PSCAD with MATLAB's engine. The make-up of this interface and how it was utilized is further discussed within this chapter. Using this subroutine and adding extra code in MATLAB a method to interface SIMULINK was developed. Once establishing the interface using this developed component several changes became necessary to handle conditions arising when interfaced to SIMULINK. These conditions are also discussed within this chapter. Other changes were to reduce the comprehensiveness of the interface by developing it into an implemented component through the GUI's of each program with no scripting requirements, unlike the original interface between PSCAD and MATLAB. These changes were accomplished by designing a new subroutine SIM_MAT along with an appropriate user interface available in PSCAD. This chapter begins with a brief discussion of the interface between PSCAD and MATALB and how it was used to interface to SIMULINK, followed by a discussion of the problems encountered and foreseen. The following sections then progress to the new method used to interface PSCAD/EMTDC to SIMULINK with discussions on the new routine's functionality and how it alleviates the issues from the original method.

4.1 Interfacing PSCAD/EMTDC and MATLAB

The interface between PSCAD/EMTDC and MATALB operates by connecting PSCAD/EMTDC to the MATLAB engine and then calling the listed m-file, see Figure 4.1 [11]. The interface requires several steps to implement through PSCAD/EMTDC, utilizing the two component development tools: New Component Workshop and Edit Definition/Component Workshop. The steps required to establish the interface were to first design a component block in PSCAD, identify the signals that are transferred in between the programs, write script to write and read variables from the stack STORF (a variable within EMTDC), evoke the subroutine MAT_INT, and update used pointers NSTORF (a variable within EMTDC). Within EMTDC there are assigned variables users can access for their component. For example if the user needs to know the present time of the simulation they can access the EMTDC variable TIME. The Stack and the Stack Pointer are helpful for retaining information throughout the entire run, where the Stack Pointer is updated as to not overwrite the data stored on the stack. For the case in the interface between PSCAD/EMTDC and MATLAB the stack is the actual memory location that is shared between the two programs. What data is placed on the stack from one program is then read by the other.

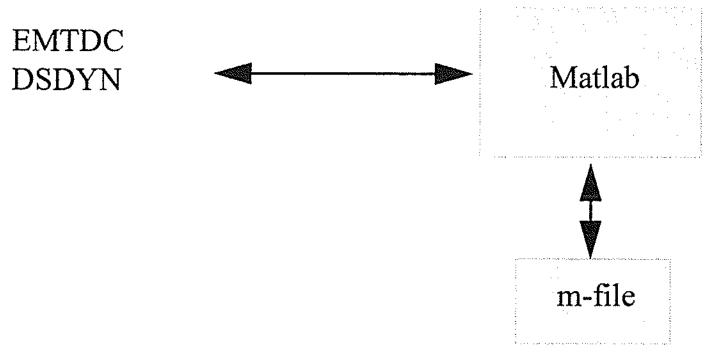
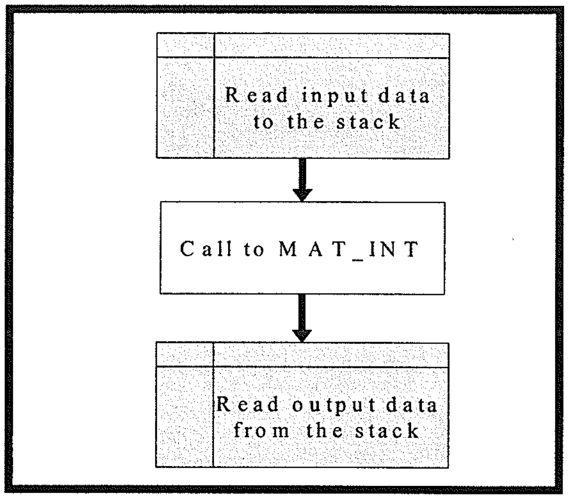


Fig. 4.1 Matlab Interface Original Flow Structure

The actual procedure to implement the interface can be found in the manual of the program PSCAD/EMTDC [26], however the core to interface is the subroutine MAT_INT which handles the communication from PSCAD/EMTDC to MATLAB. The subroutine handles all the functions to establish the connection with the user only supplying specific parameter information identified in the users manual. Incorporated into the subroutine are the identification of potential connection issues, MATLAB installation requirements, format errors, and errors for detecting invalid data width.

The MAT_INT subroutine utilizes the ENGroutines, MXroutines, and MATroutines from the Application Program Interface (API) of MATLAB to handle the actual data transfer. These routines are part of MATLAB's library files located within the installed files of MATLAB and are dependent on the version of MATLAB, the programming language (C,C++ or FORTRAN), and the compiler (Visual, Borland, Microsoft, Whatcom and LLC C). In linking PSCAD and MATALB the programming language would be FORTRAN, and the required compiler would be Visual. No other options exist for this interface. Before the simulation run is started in PSCAD (or compiling the program to execute) the location of the routine library files needs to be specified. To do this in PSCAD the data is entered through the "Project Settings - FORTRAN" pull-down menus.

The script section of the interface block that calls the subroutine MAT_INT has a standard flow as shown in Figure 4.2.



Input data (Data lines into he new component) must be stored on the STORF stack (index of the stack utilizes the variable NSTORF)

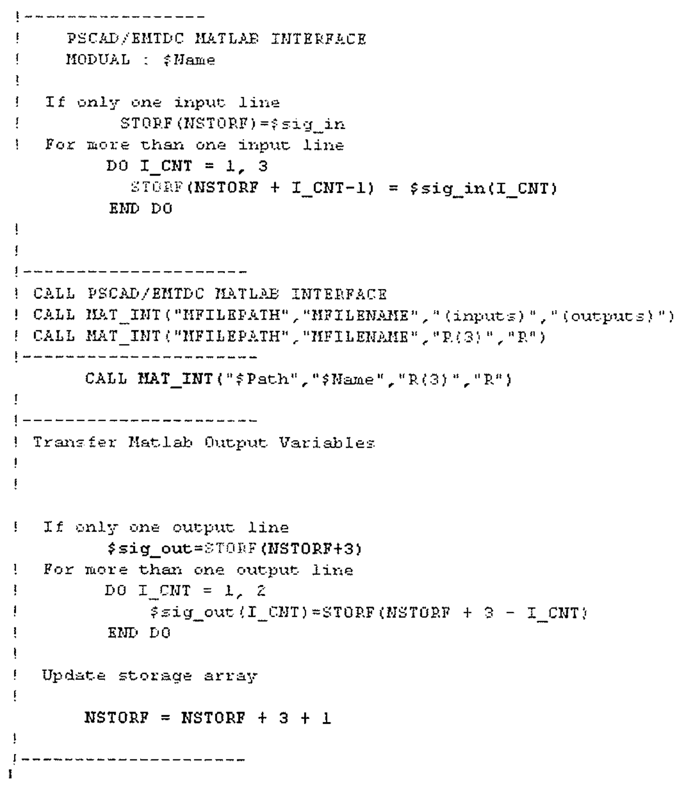
Call to MAT_INT(******) with the appropriate parameters filled in

Output data is retrieved form the stack STORF and allocated to the appropriate output lines (Note at least one data value must be read back from Matlab). The return data from Matlab is place just below the input data read onto the stack

Update stack pointer NSTORF

Fig. 4. 2 Flow Chart of Required Steps For Implementing Using MAT_INT

This script can be considered as a standard block of code where variations occur in the number of inputs and outputs utilized, their dimensions and the name and location of the m-file used. These changes can be made very easily and thus a cut and paste method can be used (see Figure 4.3). Note that in order to aid in the reuse of the component block a "Configuration" section could be added into the "Parameters" of the block in the Edit Definition/Component Workshop. In the parameters section text boxes can be added to input the name of the m-file and the location stored as a variable name, where the variable name would be used in the cut and paste script.



```
!-----------------
!    PSCAD/EMTDC MATLAB INTERFACE
!    MODUAL : $Name
!
!  If only one input line
!         STORF(NSTORF)=$sig_in
!  For more than one input line
        DO I_CNT = 1, 3
          STORF(NSTORF + I_CNT-1) = $sig_in(I_CNT)
        END DO
!
!
!---------------------
! CALL PSCAD/EMTDC MATLAB INTERFACE
! CALL MAT_INT("MFILEPATH","MFILENAME","(inputs)","(outputs)")
! CALL MAT_INT("MFILEPATH","MFILENAME","R(3)","R")
!----------------------
      CALL MAT_INT("$Path","$Name","R(3)","R")
!
!----------------------
! Transfer Matlab Output Variables
!
!


!  If only one output line
        $sig_out=STORF(NSTORF+3)
!  For more than one output line
!       DO I_CNT = 1, 2
!           $sig_out(I_CNT)=STORF(NSTORF + 3 - I_CNT)
!       END DO
!
!  Update storage array
!
      NSTORF = NSTORF + 3 + 1
!
!---------------------
!
```

Fig. 4. 3 Cut and Paste Script

Finally the m-file that the original interface communicates with is a MATLAB Function File. Within the Function File there is the ability to perform calculations, make logical decisions (using logical statements), generate plots (graphics) and issue command line statements. An example of a command line statement is to clear the command window "clc" or to close a specific plot window "Close Figure NO. *X*". Having the ability to issue the command line call allows for the extension into SIMULINK. However when using Function File there is an expectation of having a return value.

4.2 SIMULINK Interfacing

SIMULINK is a GUI based wrapper for MATALB rather then program code, icons can be place together to model various control systems. An important contribution of this thesis is the creation of an interface between a PSCAD and SIMULINK. This interface was first established by the original MATLAB interface structure and by issuing Command Line Statements to execute a SIMULINK simulation in the m-file. The sequence of events is similar to that of the original routine with the extension from the m-file to SIMULINK as shown in Figure 4.4. The commands required to execute the SIMULINK simulation are: "*simset*" and "*sim*". The "*simset*" command is a Command Line Function to set the simulation properties and the "*sim*" command sets Simulation Parameters and executes the simulation.

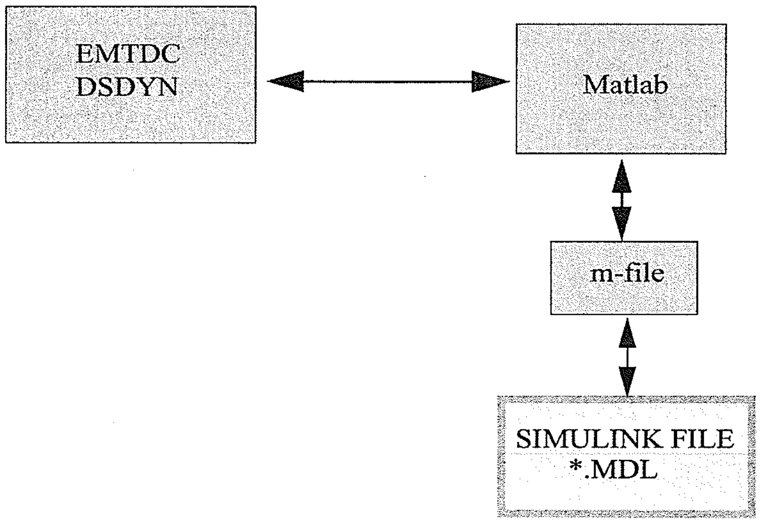
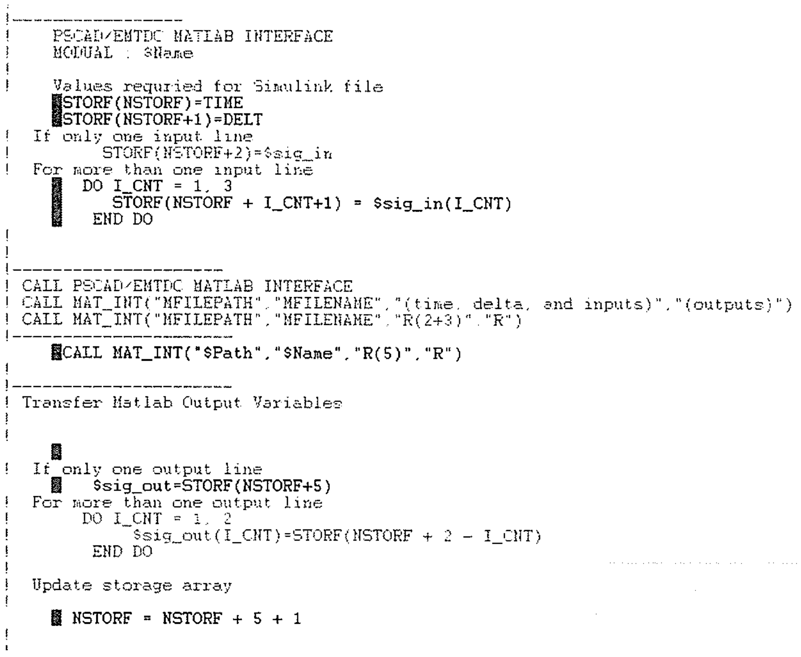


Fig. 4. 4 SIMULINK Interface Flow Structure

To eliminate one simulation program advancing faster then the other both programs execute the same time step before advancing to the next. Having the programs synchronized in this matter simplifies the implementation. By having the PSCAD/EMTDC incorporate the time and the time step variables into the data stream, we are then able to compute the time frame that SIMULINK must execute. An alternative method is to pass the full simulation parameters from EMTDC to MATLAB and then exchange data at the appropriate time intervals (time step). However the solvers in SIMULINK can vary and having to synchronize the time intervals becomes cumbersome. The additional information of the simulation time and the time step is taken care of in the component script by loading those values onto the stack STORF before the input data is stored as seen in Figure 4.5.

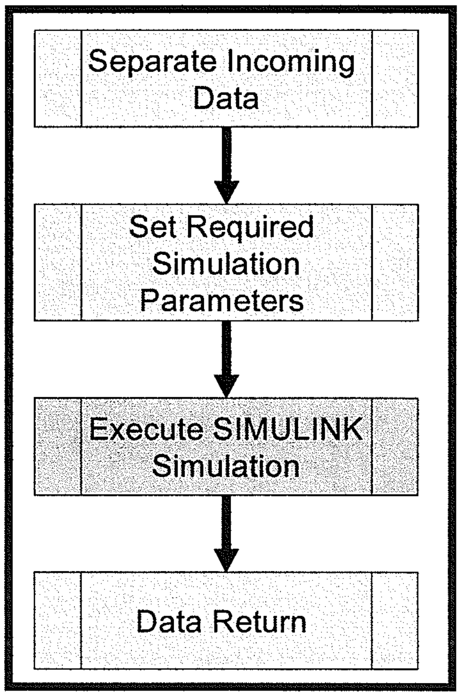


```
!------------------
!     PSCAD/EMTDC MATLAB INTERFACE
!     MODUAL : $Name
!
!     Values requried for Simulink file
      STORF(NSTORF)=TIME
      STORF(NSTORF+1)=DELT
!   If only one input line
!         STORF(NSTORF+2)=$sig_in
!   For more than one input line
           DO I_CNT = 1, 3
              STORF(NSTORF + I_CNT+1) = $sig_in(I_CNT)
            END DO
!
!
!----------------------
! CALL PSCAD/EMTDC MATLAB INTERFACE
! CALL MAT_INT("MFILEPATH","MFILENAME","(time, delta, and inputs)","(outputs)")
! CALL MAT_INT("MFILEPATH","MFILENAME","R(2+3)","R")
!----------------------
      CALL MAT_INT("$Path","$Name","R(5)","R")
!
!----------------------
! Transfer Matlab Output Variables
!
!

!   If only one output line
          $sig_out=STORF(NSTORF+5)
!   For more than one output line
!          DO I_CNT = 1, 2
!              $sig_out(I_CNT)=STORF(NSTORF + 2 - I_CNT)
!           END DO
!
!   Update storage array
!
       NSTORF = NSTORF + 5 + 1
!
```

Fig. 4. 5 Modified Component Script

Within the Simulink Interface Flow Structure the m-file to communicate with SIMULINK. The script of the m-file to establish the communication was developed such that it can be reusable in other simulations with very little modification required. The modification required to reuse the code is in identifying the name of the SIMULINK file and the location of the file. The algorithm for the script begins by partitioning the incoming data into simulation parameters and simulation data and performing the necessary calculations required for defining all simulation parameters of SIMULINK. The flow continues by executing the SIMULINK simulation, save the output data in the MATLAB engine and then export the return data back to PSCAD as shown in Figure 4.6.


Separate Incoming Data
Set Required Simulation Parameters
Execute SIMULINK Simulation
Data Return

The input data is subdivide into it's appropriate components. (time, time step, signal-data, state data).

Simulink simulation parameters are set simset (Simulink file name, solver, maximum number of data points, workspace identification, start time, stop time, signal data and state data)

Call to Simulink file - SIM, and retrieve output data along with any state variable data.

Pass all output data back to EMTDC

Fig. 4. 6 M-FILE Steps

Within the EMTDC structure all the data is read back from MATLAB by the subroutine MAT_INT before the EMTDC loop continues.

4.3 Interfacing Issues

Experimenting with the interface to SIMULINK using the original subroutine helped with the discovery of several issues that needed to be addressed.

I. The design objective was to have the interface as a standard library component block in PSCAD's master library and to eliminate the need to utilize the New Component Wizard. This component block should be adaptive to changes in the number of inputs/outputs. The problem that exists is when a component block is created in the component workshop environment it must be assigned a fixed parameter for the input dimension and a fixed parameter for the output dimension. This dictates how many fixed input and output signals are used by the component block a set structure of PSCAD/EMTCD. To develop a dynamic input/output requires changes within the program structure of PSCAD/EMTDC

II. In implementing the call routine MAT_INT within the EMTDC loop it was found by the way MATLAB is structured, it does not retain its memory from call to call. MAT_INT is designed to use MATLAB functions, which have defined characteristics. One characteristic of a MATLAB function is to keep values within the function autonomous from the originating program. The purpose of this characteristic is that the initiating program and the function routine can bestow the same variable name to hold two different values. Variables used within a function routine only exist when the function is being executed and only the return value is retained upon exiting the function. However due to the time step synchronization implemented SIMULINK requires the retention of memory when certain library components are implemented. To overcome the memory problem data of the present time step of SIMULINK is exported back to PSCAD then reimported back into SIMULINK on the next time step. It is important to note that this is not a very effective method to utilize when simulation speed is a factor.

III. The subroutine MAT_INT does not allow assigning of variables or pre-simulation computations within the MATLAB engine. The problem is that the state variables used in a SIMULINK simulation can be defined by a variable name which holds the state information. To utilize the full potential of SIMULINK there has to be the ability to allow the assignment of variables that hold state information. The problem is not only of loading the values into MATLAB memory but also retaining the values for the whole simulation run.

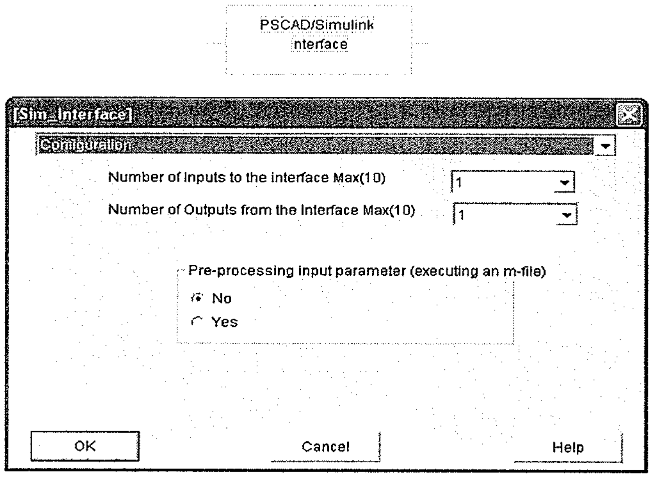
IV. The feature of taking of a snapshot and starting from a snapshot in PSCAD is non-functional when interfaced to SIMULINK. In its present form the interface does not have the ability to retain the memory of MATLAB or to issue a command to save data at a specified time interval. Since there is no data for MATLAB the process of starting from a snapshot must be disabled.

4.4 Features Implemented in the New Subroutine

The solution to the above problems was to create a new subroutine to replace the existing one. The new subroutine addressed the issues mentioned in the previous section along with restructuring the subroutine for efficiency. The modifications are as follows:

I. From the flow chart in Figure 4.4 the subroutine is restructured to replace the m-file. The communication path will exclude the linking of the m-file and will directly communicate with the SIMULINK file through the MATLAB engine. The command line code that was utilized in the m-file is transferred to the subroutine script using the ENGroutines and MXroutines in MATLAB API.

II. Requested for this new component, the developers of PSCAD arranged a method to allow dynamic input/output dimensioning. This is handled by allowing variable names to be assigned to input/output dimensions and use the Components Properties dialog to assign a value to the input/output dimensions Figure 4.7. Within the developed interface dialog users specify the number of inputs and outputs through a pull down menu.



Number of Inputs to the Interface Max(10): 1

Number of Outputs from the Interface Max(10): 1

Pre-processing input parameter (executing an m-file): [✓ No] Yes

Fig. 4. 7 Variable Input Parameters

III. Restructuring the input/output port to one, and requiring the use of data merge and data taps in PSCAD for components that require multiple signals as shown in Figure 4.8. The choice of this format was to simplify the input parameters of the subroutine defining the number of inputs and outputs. Since we are specifying the input dimension of the component block thought a variable name, the same variable name can be utilized to specify input and output parameters to MATLAB. Changing the last two parameter fields of the subroutine from a string format to a numerical value allows for the elimination of the code and decreasing the number of branches that are occurring within the original subroutine to increase the efficiency of subroutine.

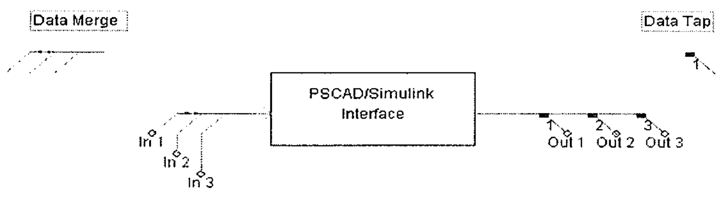


Fig. 4. 8 Interfacing More Than One Data Signal in the Developed Interface

IV. Allowing the initialization of MATLAB memory through the execution of an additional m-files when the simulation is initialized. Within the newly designed library component block there is an additional input parameter called "PreProcessing" where specified MATLAB files can be executed to initialize variables within MATLAB at the start of a simulation.

V. Included in the subroutine are extra routines that will save MATLAB engine data as "*mlab_data.mat*" and if required will load data from the same filename. This ability of saving and retrieving data is a significant factor in the implementation of snapshots in

PSCAD. What happens during the snapshot event is that a command is sent to MATLAB to save the data, and when starting the PSCAD simulation from a snapshot, a command is sent to load the saved file. Storage of these file is kept to the same folder that PSCAD stores its data when taking a snapshot, i.e. in the *.emt folder.

VI. Utilization of the workspace to retain simulation data from time step to time step. In the new subroutine, state data is retained in MATLAB's workspace under the variable name "*xFinal*". The choice of the name is based on its identification name within the simulation parameters list of SIMULINK, where xFinal allocates a memory storage name of the state data.

4.5 Conclusion

Implementing an interface in PACAD/EMTDC with SIMULINK required restructuring of the original subroutine to allow the full utilization of both programs. These changes were implemented in the new subroutine SIM-INT to be used when linking PSCAD/ EMTDC to SIMULINK. The inherent problems with the old routine were discussed along with the modifications that were implemented.

CHAPTER 5 Solution of Complex Power Electronic Problem With the New Interface

This chapter discusses the demonstration of the developed interface as it is applied to three separate problems. The results observed on the interface are to evaluate its effectiveness with later recommendations as to when such a tool should be utilized. These performance evaluations are conducted within the first two simulation results of this chapter. The first result details the performance of the interface with a Pulse Width Modulation (PWM) Controller modeled in Simulink and an Induction Motor in PSCAD. The simulation also uses the original method to interface between PSCAD and SIMULINK and the newly developed library component. The second simulation uses only the newly developed library component to test the integration of simulation features through the interface. One of the features tested is the use of state blocks within SIMULINK which is an important test as it requires the interface to remember past states.

The final simulation uses the interface to implement the complex H_∞ control synthesis solution. The difficult problem of finding a solution to the control problem is handled through MATLAB with the controller simulated in SIMULINK. To find the control synthesis solution required the circuit to be modeled by its state space equivalents and one of the results presents a total SIMULINK simulation, with and without the controller active. The interface simulation ideally should be a verification of the above simulation however the results present a different case.

5.1 Demonstration of Basic Interface Simulating PWM control of an Induction Motor

This simple example demonstrates the initial method used to interface PSACD/ EMTDC and SIMULINK. The initial results use the linking process in Figure 4.4, the PSCAD/MATLAB interface with the added m-file of Figure 5.2 to establish the connection with SIMULINK. Shown in the m-file is the SIM command with its required parameter fields and the segmenting of data for required simulation parameters and input/output data. The required parameter fields of SIM commands are simulation start and stop times which coexist with the present time step of PSCAD and the limiting the output data to its final solution. The application of the interface is used to control an induction motor by applying a PWM controller Figure 5.1 [11].

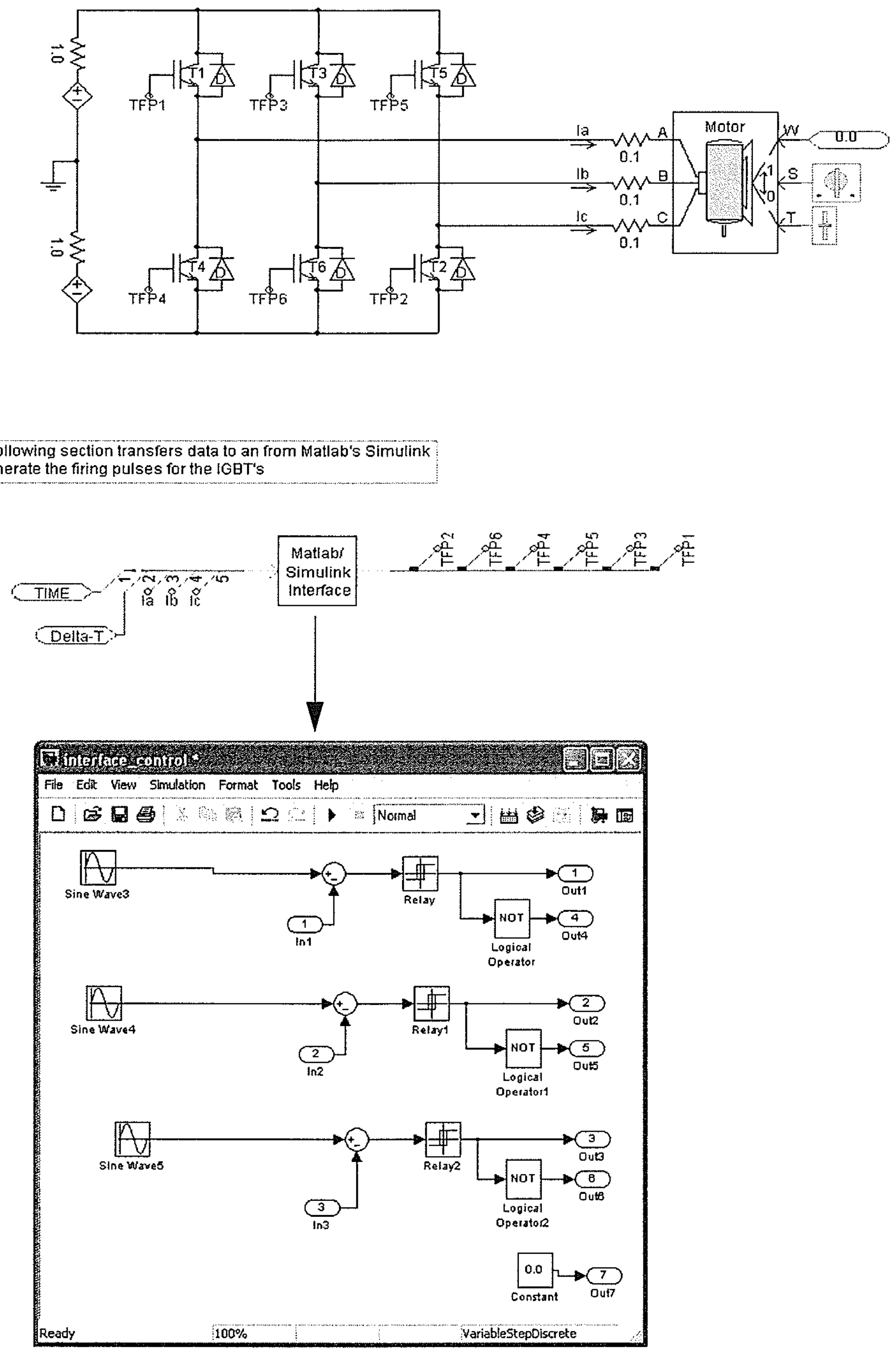
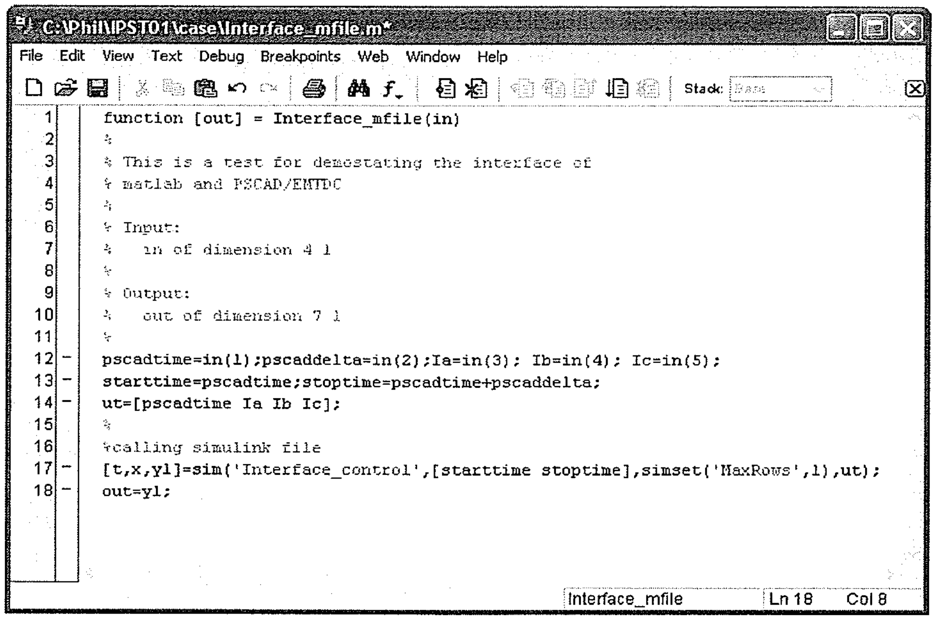

The following section transfers data to an from Matlab's Simulink to generate the firing pulses for the IGBT's

Fig. 5. 1 PSCAD/EMTDC Interfaced with SIMULINK

I


C:\Phil\IPST01\case\Interface_mfile.m*
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
Interface_mfile
Ln 18 Col 8

```
function [out] = Interface_mfile(in)
%
% This is a test for demostating the interface of
% matlab and PSCAD/EMTDC
%
% Input:
%   in of dimension 4 1
%
% Output:
%   out of dimension 7 1
%
pscadtime=in(1);pscaddelta=in(2);Ia=in(3); Ib=in(4); Ic=in(5);
starttime=pscadtime;stoptime=pscadtime+pscaddelta;
ut=[pscadtime Ia Ib Ic];
%
%calling simulink file
[t,x,y1]=sim('Interface_control',[starttime stoptime],simset('MaxRows',1),ut);
out=y1;
```

Fig. 5. 2 m-File Interface Code

To describe the structure of the simulation, Simulink was used as a current reference PWM controller for an induction motor starting scheme. Current measurements to the motor are transferred to Simulink and are compared to a reference signal. Based on this comparison the choice of turning the thyristors on or off is then transmitted back to PSCAD/EMTDC. The technical aspects of the circuit are not the focal point of the results. However the interface block and the overall simulation time hold more interest. Within the PSCAD/EMTDC simulation some characteristics of the motor are plotted: input currents, mechanical and electrical torque and speed. With the current reference frequency set to 10 Hz, 10 Amps peak applied to a load torque of 0.1 p.u. the motor reaches a maximum speed of 0.16 p.u as shown in Figure 5.3.

$$\omega_S = \frac{\omega_r}{\omega_{base}} = \frac{2\pi 10}{2\pi 60} = 0.167 \qquad \textbf{(EQ 5.1)}$$

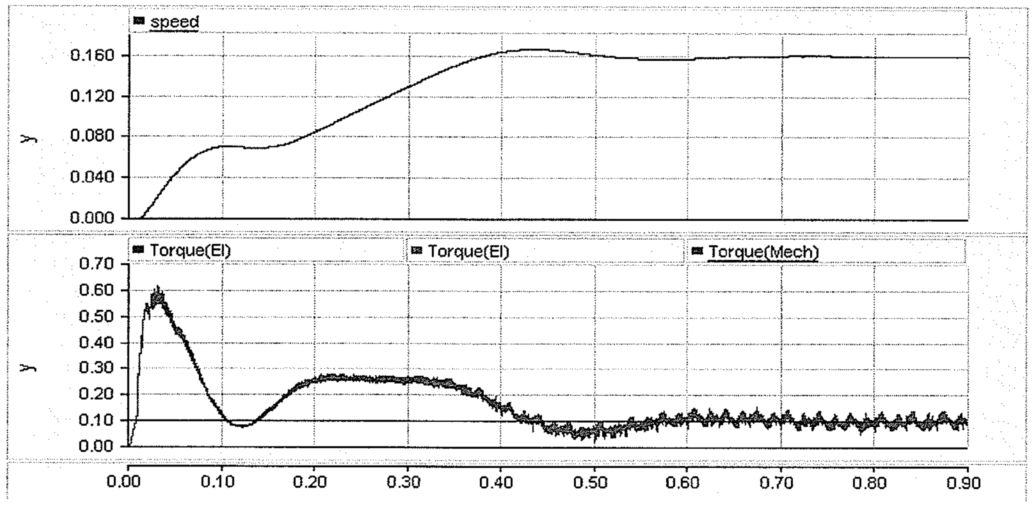


Fig. 5. 3 Induction Motor Results

The first result from this simulation shows the objective of establishing an interface between the two software programs has clearly been established through the data output displayed in the PSCAD/EMTDC simulation program. Although it is a cumbersome approach to establish the interface, it represents the vital fact that such a task can be accomplished. In evaluating the performance of the hybrid simulation a typical standard to incorporate is the performance in overall simulation time for an application. The comparison in performance is not only dependent on the machine but also on the operating system installed onto the machine. For this evaluation several different computers are chosen with different operating systems installed, and the time comparison is made with machine running the hybrid simulation and the same machine running the simulation within one chosen package PSCAD. The first comparison is on 500 MH Pentium III, with 128 MB RAM running Windows 98, required 4 minutes of CPU time to execute 1 minute of simulation time with a time step of 50 μs in the hybrid case. With the simulation confined to PSCAD, the time to complete the simulation was only 15 seconds. The second computer, a 2.5 GHz Pentium III with 128 MB RAM running Windows XP Professional, required 5.2 minutes of CPU time to execute the hybrid simulation, while in PSCAD it only requires 2.58 seconds to complete. Clearly, as far as speed is concerned for this simulation modeling the circuit using one simulation package would be more effective. The difference in time between the two operating platforms Windows 98 and Windows XP is due to the number of processes that the system requires to keep active, viewed within the Task Manager Windows. In Windows XP the simulation program is competing for cpu time against standard function carried out by the operating platform, hence requiring longer simulation time than the Windows 98 simulation. However, the interface approach though slower has the advantage that developed models in both programs become immediately available.

5.2 *Integrating Simulation Features*

In this example we verify that the newly designed features do not interfere with the capabilities of either program and address the issues discussed in Chapter 4. A comparative study between the original interface used in the above section and the newly implemented subroutine is used to examine the effectiveness of the procedure implemented.

The simulation models the electrical circuit in PSCAD and the controller in SIMULINK (Figure 5.4). The circuit is a simple Voltage Source externally controlled with a connection to a passive load the basic building block of any circuit. The controller for the circuit is a simple Proportional Integral (PI) controller represented in state blocks. Because MATLAB is a mathematical program, its modeling within SIMULINK can be represented by differential equations within a state block or by a simple transfer function. By representing the controller in states space, MATLAB must retain within its memory the state information as the simulation executes.

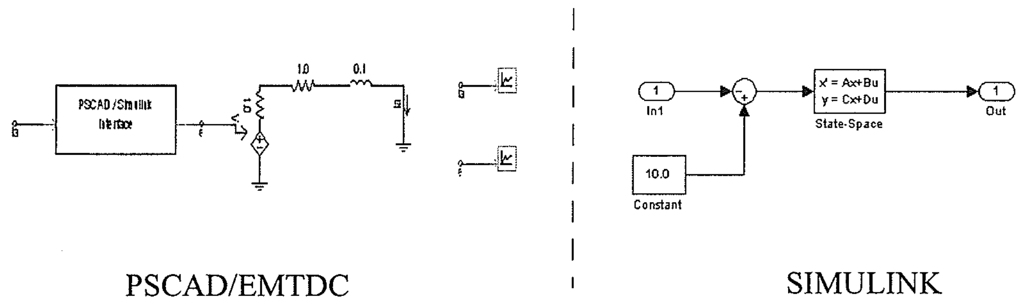


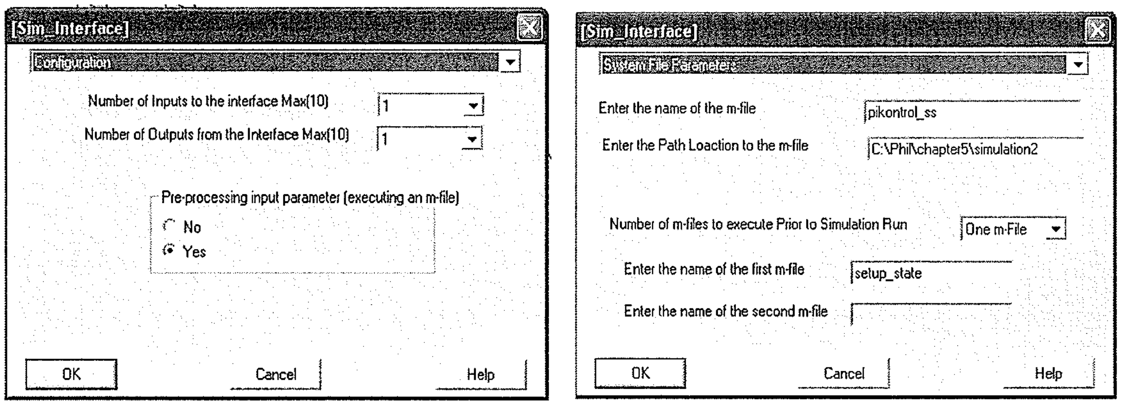
Fig. 5. 4 Simulated Circuit

The objective of the controller is to reduce the rise time to steady state value (reference 10 Amps) of the load current while keeping the voltage source magnitude within reasonable limits. Selecting values for the gain and the time constant were done arbitrarily, Figure 5.5, however adjustments could be made to meet a designed performance specifications. Tuning of the controller is done by adjusting the Gain for percent overshoot and the Time Constant for the rise and settling time. Finally if the control parameter were to be defined as a transfer function, one can use toolboxes available within MATLAB to convert the state values to a transfer function, highlighting the multi-functions available within the package.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{1}{T} & K \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 40 & 3 \end{bmatrix} \quad \begin{array}{l} \text{Gain (K)} = 3.0 \\ \text{Time Constant} = 0.025 \end{array}$$

Fig. 5. 5 Proportional Integral Controller Settings

Within this case study the various operations developed within the interface are highlighted beginning with the method of selecting the number of inputs/outputs data signals communicated between the two programs through the configurations dialog of the component block Figure 5.6. A second feature demonstrated in the case study is presented by the technique used to initialize the state information within SIMULINK. The initial set up of the SIMULINK state block defines the block parameters as variable names (A,B,C,D). In the initialization of the simulation the variable names are then predefined in MATLAB's memory with a value (Figure 5.5). The process was achieved by running an additional MATLAB file in preprocessing. The process also defines the variables devoted to retaining information in MATLAB (current and xFinal), and a variable containing simulation parameter delta the time step. The verification that the proper commands were executed is by inspecting MATLAB Command Window which displays the contents of its memory during the simulation Figure 5.7.


[Sim_Interface]

Configuration

Number of Inputs to the interface Max(10): 1

Number of Outputs from the Interface Max(10): 1

Pre-processing input parameter (executing an m-file): No [✓ Yes]

[Sim_Interface]

System File Parameters

Enter the name of the m-file: pikontrol_ss

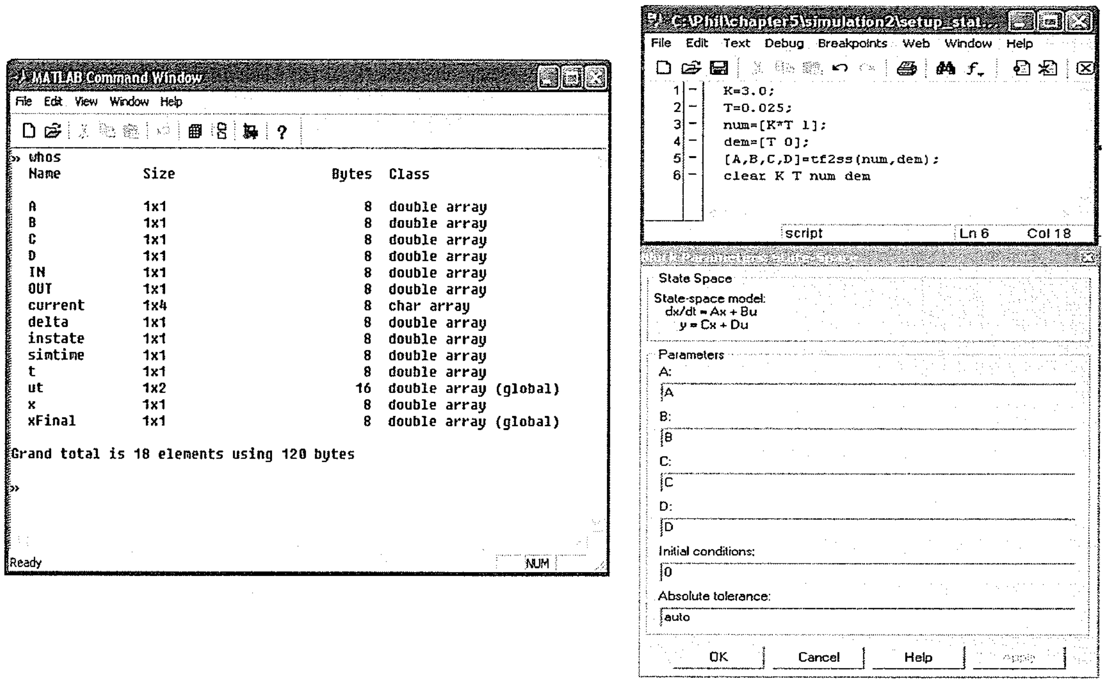
Enter the Path Loaction to the m-file: C:\Phil\chapter5\simulation2

Number of m-files to execute Prior to Simulation Run: One m-File

Enter the name of the first m-file: setup_state

Enter the name of the second m-file:

Fig. 5. 6 Parameter configuration of the component block


MATLAB Command Window
File Edit View Window Help
C:\Phil\chapter5\simulation2\setup_stat...
File Edit Text Debug Breakpoints Web Window Help
State Space
Parameters
OK
Cancel
Help
Apply

```
>> whos
  Name          Size                    Bytes  Class

  A             1x1                         8  double array
  B             1x1                         8  double array
  C             1x1                         8  double array
  D             1x1                         8  double array
  IN            1x1                         8  double array
  OUT           1x1                         8  double array
  current       1x4                         8  char array
  delta         1x1                         8  double array
  instate       1x1                         8  double array
  simtime       1x1                         8  double array
  t             1x1                         8  double array
  ut            1x2                        16  double array (global)
  x             1x1                         8  double array
  xFinal        1x1                         8  double array (global)

Grand total is 18 elements using 120 bytes

>>
```

Ready NUM

```
K=3.0;
T=0.025;
num=[K*T 1];
dem=[T 0];
[A,B,C,D]=tf2ss(num,dem);
clear K T num dem
```

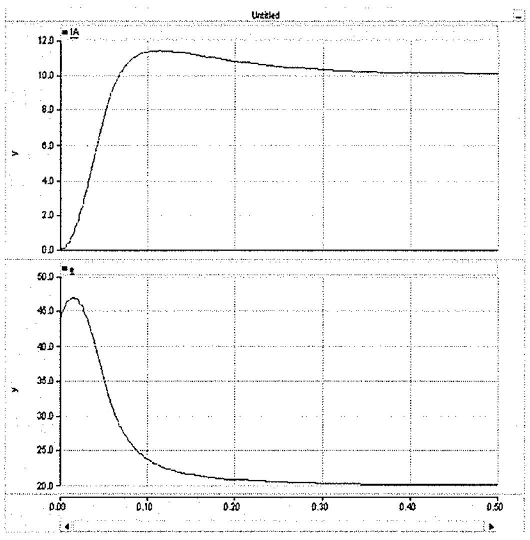
script Ln 6 Col 18

State-space model:
dx/dt = Ax + Bu
y = Cx + Du

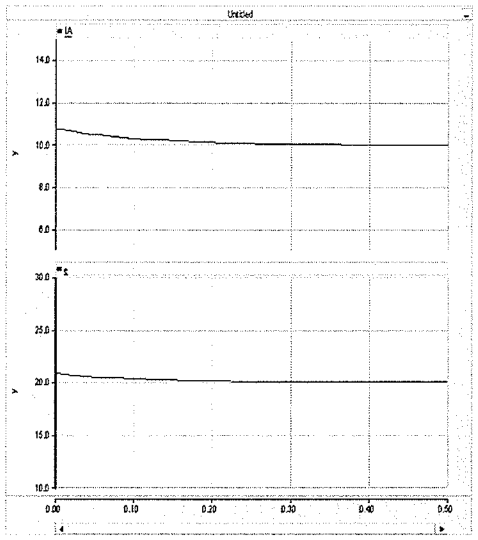
A: A
B: B
C: C
D: D
Initial conditions: 0
Absolute tolerance: auto

Fig. 5. 7 MATLAB command window with initialization parameters and script

Integrated into the simulation is also the method to store data when a snapshot is taken. A snap shot is a point in time within the simulation that all the node data of the simulation is stored. The benefit of taking a snap shot is that following simulation can begin from the point the snapshot was taken. This is an ideal tool for the analysis of events that occur after the system is in steady state operation. The snapshot command integrated into the interface stores the data within MATLAB as a *.mat file, at that same time interval when the snapshot occurs in PSCAD. The process of starting from a snapshot is the reverse where data is loaded in the initialization of the simulation. The results generated by the simulation are presented in Figure 5.8 where a snapshot is taken at 0.2 seconds into the simulation and the simulation is restarted from the snapshot.



Initialization (taking a snap shot)



Starting from a Snap Shot

Fig. 5. 8 Simulation

Testing of the interface shows that the routines utilized to retain memory in MATLAB work effectively, decreasing the amount of data transfer required. In structuring the interface for the user to specify the number of inputs/outputs simplifies its implementation. The original interface is based on a subroutine that users have to develop their component block around; in the new routine the component block is now a standard block that can be incorporated into the library of PSCAD. However testing the performance of the new subroutine shows an increase in CPU time when running the above simulation by initiating the call to execute the simulink file directly from the MATLAB engine as compared to issuing the call to execute the simulink simulation from the m-file. By eliminating the process of the m-file from the loop Figure 4.4 and executing the commands for the engine we found an increase in simulation time. In the original routine the amount of CPU time to complete the simulation was 76.25 seconds. Implementing the updated routine with the same simulation circuit required 114.16 seconds, an increase of 37.91 seconds. This increase in time is an unexpected result due to the fact that then new subroutine streamlines the code by reducing the number of branches and reduces the number of executable statements, however the outcome effect is an increase in simulation time. The same simulation using only transfer function blocks to model the controller produces similar results, with the original interface the cpu time was 84.14 seconds and the CPU time with the updated interface was 115.34 seconds.

The comparison that can be made from the results is that there is no difference in the simulation process time if state blocks or a transfer function blocks are used, however to implement the use of the state blocks required extra features to be added into the subroutine for storage of state values. A second comparison is that updating the subroutine to incorporate the latest API library had no effect on the simulation process time.

The above results show that there is considerable advantage in simulation performance by maintaining the status of the old routine. However in remaining with the old routine there is a loss in the new features incorporated into the new routine. Although there is a loss in simulation performance there are other gains in reusability of the block as its applied to other simulations.

5.3 Control Synthesis for a Boost Converter

This example uses the control toolbox in MATLAB to synthesis a controller based on the H_∞ criterion. Analysis is performed on how robust the controller is to damping out induced disturbances on the system [7,23,27]. Since the synthesis procedure requires the derivation of the system states, the initial analysis on the synthesized controller is performed within the MATLAB environment. To accurately model the switching events of the circuit the next step was to integrate the controller with a PSCAD simulation to verify its effectiveness. Along with the analysis of the controller is also the analysis of the hybrid simulation, with its capabilities and its limitations when incorporating it into a complex simulation.

The test system is a boost converter to step up voltage from 12 to approximately 24 kV DC as shown in Figure 5.9. Disturbances modeled into the system are input voltage fluctuations and load variations. The input voltage fluctuations are introduced into the simulation as a voltage source where load fluctuations are modeled as a variable current source. To implement the H_∞ algorithm discussed in the chapter 3 the circuit needs to be expressed in terms of its state variables. The derived state expression for the circuit is presented in Figure 5.10, with the derivation in Appendix A.

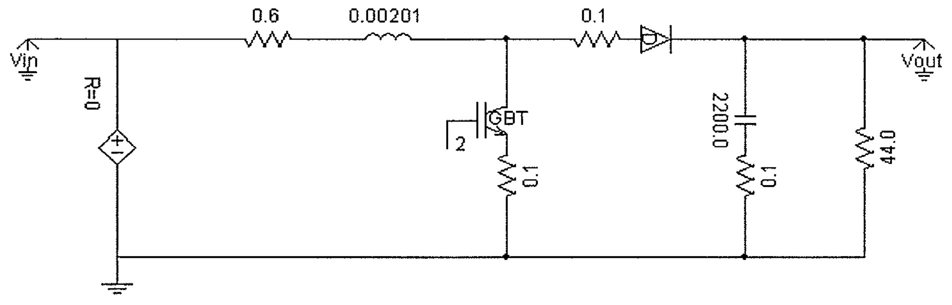


Fig. 5. 9 Boost Converter

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax(t) + B1w(t) + B2u(t) \qquad x = \begin{bmatrix} \hat{i}_l \\ \hat{v}_c \end{bmatrix} \quad w = \begin{bmatrix} \hat{Vin} \\ \hat{Iout} \end{bmatrix} \quad u = \begin{bmatrix} \hat{d} \end{bmatrix}$$

$$z = C1x(t) + D11w(t) + D12u(t)$$

$$y = C2x(t) + D21w(t) + D22u(t) \qquad z = \begin{bmatrix} \hat{Vout} \end{bmatrix} \quad y = \begin{bmatrix} \hat{Vout} \\ \hat{Vin} \end{bmatrix}$$

$$A = \begin{bmatrix} \frac{(-D\cdot((R+r_c)\cdot(r_s-r_d)-r_cR))}{L\cdot(R+r_c)} + \frac{-(r_l+r_d)\cdot(R+r_c)-r_cR}{L\cdot(R+r_c)} & \frac{(-R)\cdot(1-D)}{L\cdot(R+r_c)} \\ \frac{R\cdot(1-D)}{C\cdot(R+r_c)} & \frac{-1}{C\cdot(R+r_c)} \end{bmatrix}$$

$$B1 = \begin{bmatrix} \frac{1}{L} & \frac{(1-D)\cdot R\cdot r_c}{L\cdot(R+r_c)} \\ 0 & \frac{-R}{C\cdot(R+r_c)} \end{bmatrix}$$

$$B2 = \begin{bmatrix} \frac{12\cdot[-(r_l+r_s)\cdot(R+r_c)+(r_l+r_d)\cdot(R+r_c)+r_c\cdot R+R^2\cdot(1-D)]}{L\cdot\{D\cdot[(R+r_c)\cdot(r_s-r_d)-r_cR]+(r_l+r_d)\cdot(R+r_c)+r_c\cdot R+R^2\cdot(1-D)\}} \\ \frac{-12\cdot R}{C\cdot\{D\cdot[(R+r_c)\cdot(r_s-r_d)-r_cR]+(r_l+r_d)\cdot(R+r_c)+r_c\cdot R+R^2\cdot(1-D)\}} \end{bmatrix}$$

$$C1 = \begin{bmatrix} \frac{(1-D)\cdot R\cdot r_c}{(R+r_c)} & \frac{R}{(R+r_c)} \end{bmatrix} \qquad C2 = \begin{bmatrix} \frac{(1-D)\cdot R\cdot r_c}{(R+r_c)} & \frac{R}{(R+r_c)} \\ 0 & 0 \end{bmatrix}$$

$$D11 = \begin{bmatrix} 0 & \frac{-(R\cdot r_c)}{(R+r_c)} \end{bmatrix}$$

$$D12 = \begin{bmatrix} \frac{-12\cdot R\cdot r_c}{\{D\cdot[(R+r_c)\cdot(r_s-r_d)-r_cR]+(r_l+r_d)\cdot(R+r_c)+r_c\cdot R+R^2\cdot(1-D)\}} \end{bmatrix}$$

$$D21 = \begin{bmatrix} 0 & \frac{-(R\cdot r_c)}{(R+r_c)} \\ 1 & 0 \end{bmatrix} \qquad D22 = \begin{bmatrix} D12 \\ 0 \end{bmatrix}$$

Fig. 5. 10 State Equations of the Boost Converter Circuit

From the derived state expressions the frequency response of the circuit can be obtained as shown in Figure 5.11. As seen from the figure, the circuit behaves as a low pass filter to the induced noise, whereas frequencies below 400 Hz are amplified and frequencies above the 400 Hz limit are beginning to be damped out. Frequencies above 10 kHz have marginal effect on the output. High frequency signals have little effect on the output voltage due to the presence of the capacitor in the circuit, which maintains the output voltage constant to the high frequency disturbances the circuit is subjected to.

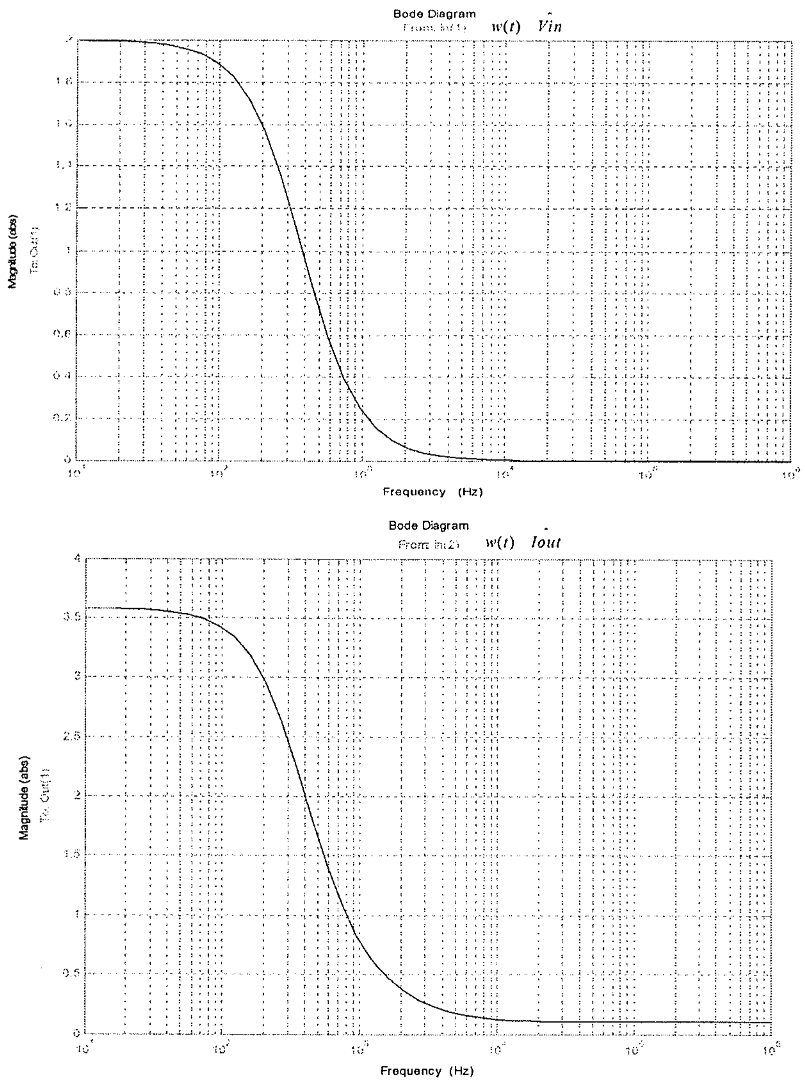


Fig. 5. 11 Response of the Circuit Utilizing the State Equations

Thus for analysis the noise induced into the system will need to be contained in the low frequency range from 1Hz to 4 kHz range, above this range the circuit adequately suppress the disturbances.

With the state equations derived, the next process is considering the weights that will be applied. One could choose the weights to be constant values or select an appropriate transfer function. The weights applied for this example utilize a transfer function that follows the typical frequency-weighting function that has a small gain in the low frequency range and attenuation in the upper frequency range. Given the transition from gain to unity gain there is the defined bandwidth ω_b of the weighted function, and it occurs when the function reaches unity gain (or more precisely 3dB from unity gain). The gain value identifies the maximum allowable tolerance in performance that is acceptable to the systems small signal disturbances at low frequencies, represented by the variable M, ideally small. The bandwidth ω_b of the system is its ability to reject noise over a wide frequency range, ideally high. In considering the values of M and ω_b there is a trade-off as both effect γ., where small values of M and large values of ω_b reflect high values of γ [1].

The standard weighting curve is implemented is using Equation 5.2 to produce Figure 5.12.

$$W(s) = \frac{s+\omega_b}{s+a}$$

Where: $$a = \frac{\omega_b}{10^{\left(\frac{M}{20}\right)}} \qquad \text{(EQ 5.2)}$$

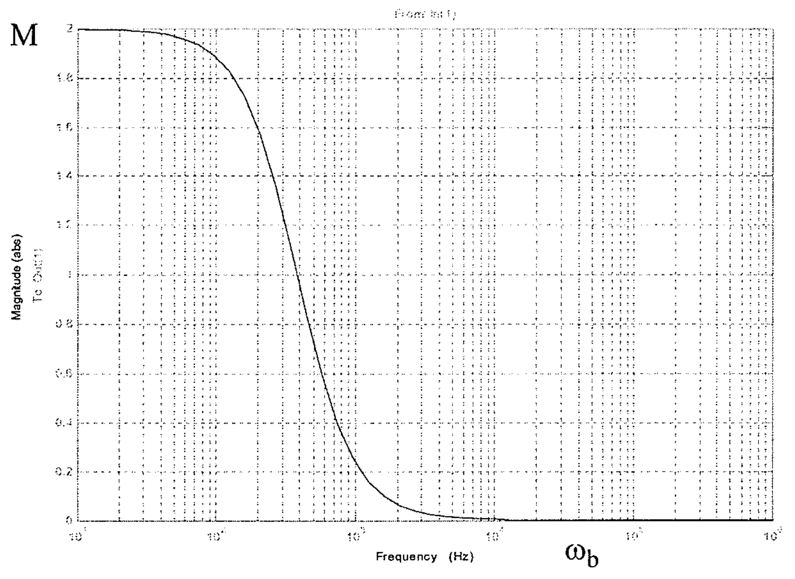


Fig 5. 12 Magnitude Bode plot of the applied Weight

The next step is to specify the interconnections between all defined systems (Plant, Weights, Actuators, ect.). This is carried out by defining variable names to identify the inputs and them assigning them to the appropriate blocks. Blocks that are interconnected are specified as the output of one block becomes the input of the other, and systems with multiple inputs can be assigned individually. One final note is that before the blocks can be interconnected each block should be represented by its states the method used to synthesize the controller. This is easily achievable within MATLAB using the Linear System Transformation routines, which provide the ability to convert from one system description to another, e.g., from state space to transfer function and reverse.

The Boost Converter (Plant) with the weight applied is shown below in Figure 5.13. In the analysis the only weight that needs to be applied to converge to a solution was one on the output z. Without the applied weight it was found that there was no viable solution in the synthesis routine.

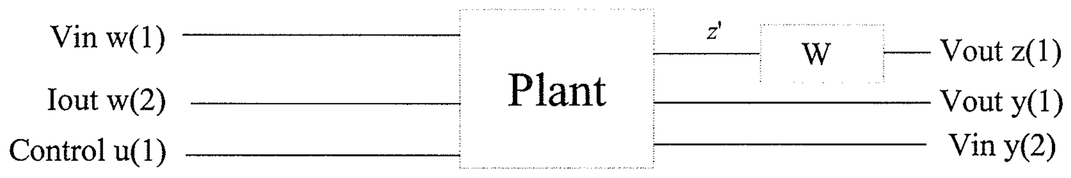


Fig. 5. 13 System Interconnection

With the entire system configured, the final step is implementing the algorithm to find the optimal controller as discussed in Chapter 3 and implemented within MATLAB's mu-toolbox. The synthesis procedure is implemented through the routine "*hinfsyn*". The required parameters utilized to implement the routine are the total system configuration, number of inputs/outputs of the controller, maximum and minimum γ value, and finally the relative tolerance between final γ values. There are additional parameters for the routine such as the selection of a solution method, error tolerances in the solutions, and display results; for this simulation the default settings were used. In this simulation the control was synthesized with two inputs and one output, the starting γ selected was 70 a value chosen slightly about the norm of the system, a minimum γ of 0.1 a value below 1, and the tolerance was chosen to be 0.001 the default value. The return quantities from the routine are the state space values for the controller and the state space values for the overall system. With the return of both quantities, one can produce multiple graphs describing the systems with bode plots of the magnitude and phase. Appendix B gives the Matlab code to model the system in terms of the state matrix by inputting the circuit values, along with the synthesis of the controller for the system Figure 5.14. Since the circuit is already developed, a simple test of the controllers performance was implemented using only simulink through the use of state blocks Figure 5.15.

```
>> seesys(k)

-9.3e+004 -1.0e+006 -9.9e+006  |  2.3e+003  2.3e+001
 7.9e+003  4.6e+004  8.9e+005  |  3.6e-014  4.6e-015
 3.5e+000  4.0e+001 -2.1e+002  |  8.7e-018  2.0e-018
-------------------------------|--------------------
-1.6e+002 -1.9e+003 -1.8e+004  |  4.2e+000 -9.4e-017

>>
```

Fig. 5. 14 Results for the Synthesized Controller

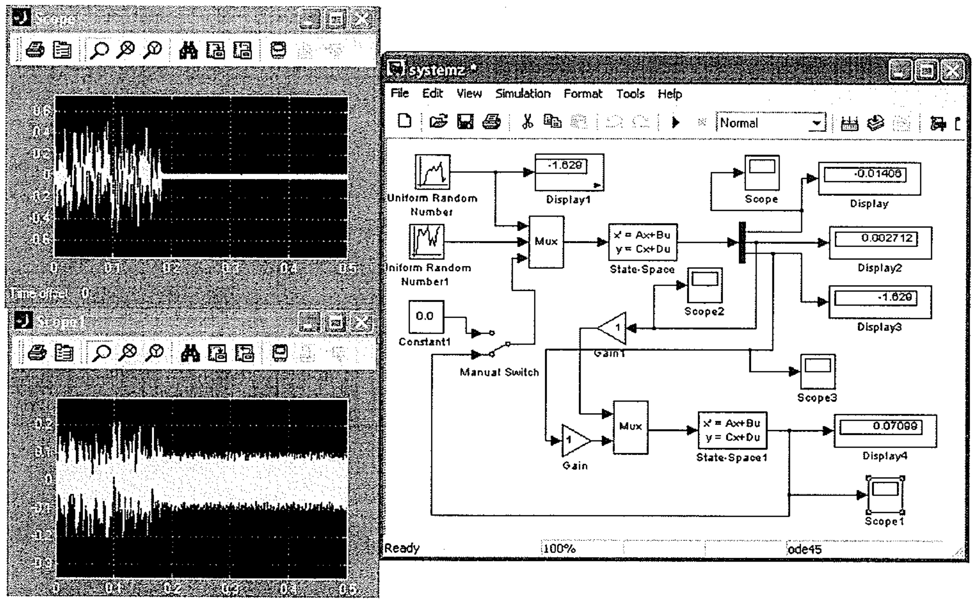


Fig. 5. 15 Simulink Simulation

The output of the SIMULINK shows the noise suppression through the figure window Scope 1 and the controller output is shown in Scope 1. With the synthesized controller in state space and the given plant in the same format SIMULINK was used to test the effectiveness of the controller. Using a random number generator to simulate the perturbations, the results of the simulation show the noise reduction of 95% of the peak value from the small signal output voltage level. Initial analysis shows that the synthesized controller is stable and that the controller is robust to control the output voltage level when subjected to a noisy environment.

The next step is integrating the simulation to include PSCAD/EMTDC with the circuit portion modeled in PSCAD to handle the dynamics of the switching and the control section remaining in Simulink. Since the perturbations are part of the electrical network the generation of those perturbations are also modeled in PSCAD/EMTDC. To introduce the noise into the electrical network an ideal current and voltage source are used, where the magnitude of the source is controlled by an external signal. The external signal is a generated random number whose magnitude is limited to the peak value of the small signal noise. PSCAD's master library includes a random number generator, however in the initial circuit analysis that noise above a frequency of 10 000 kHz is effectively damped out by the circuit. To control the frequency of the noise introduced a simple circuit is utilized to produce a pulse to trigger the event of when a new random number is generated.

The simulink file for the simulation requires some changes to implement the hybrid simulation. The method to derive state equations to represent the electrical circuit is based on the method of averaging, which includes the switching of the IGBT by modeling the states, input and output quantities as a constant value pulse a small signal value. For example the input duty cycle to the electrical circuit is $d = D + \hat{d}$, where D is a constant value the input duty cycle and $\hat{d}$ the small signal. The result of making D a constant value is that it can be multiplied throughout the state equations as shown in Appendix A. The result produces an input that is only composed of $\hat{d}$ the small signal variable. However to integrate to the hybrid simulation the circuit must be composed of $d = D + \hat{d}$. To generate D within SIMULINK the components used are: a Voltage Controller Oscillator (VCO) and a Comparator along with two input reference signal. The VCO outputs a ramp function between the limits of 0 and 1, which is oscillatory at the frequency of the first input (reference signal). The output of the VCO is then compared to a second reference signal the duty cycle through the comparator. The comparator outputs the value of 1 when the output of the VCO is less than or equal to the reference signal and 0 when the output of the VCO is greater than the reference signal. This produces the constant switching pulses for the IGBT as shown in Figure 5.16. The final portion of the controller is adding the output of the synthesized controller $\hat{d}$ to the signal generated by the comparator.

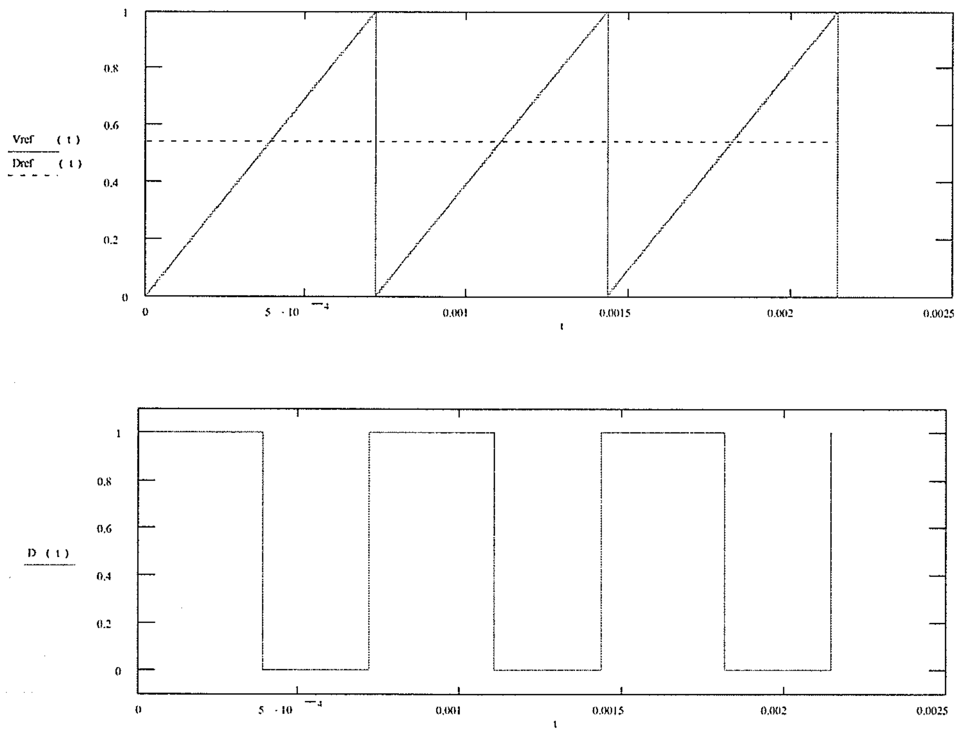


Fig. 5. 16 IGBT Pulse Generation

Finally as mentioned above with the duty cycle all the input values to the synthesized controller, the measured values of the circuit must also be adjusted accordingly. For example the measurement of the input voltage (V_{in}) is composed of $V_{in} = V_{in} + \hat{V}_{in}$, however the input to the controller synthesized uses only $\hat{V}_{in}$. To remove the reference value from the measured value a simple subtraction method is implemented based on the knowledge of dc reference value. Finally in the simulation it was discovered that the controller was sensitive to the switching noise through the IGBT. To reduce the controller response to the switching noise a low pass filter was inserted at the inputs to the synthesized controller, with a cut off frequency of 10 kHz above the frequency response of the circuit and the weight applied. The transfer function of the low pass filter is given in Eq. 5.3 .

$$T(s) = \frac{10000}{s + 1000}$$

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} -10000 & 1 \\ 10000 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ u \end{bmatrix} \qquad \textbf{(EQ 5.3)}$$

The Figures below shows the implementation of the circuit within PSCAD and SIMULINK.

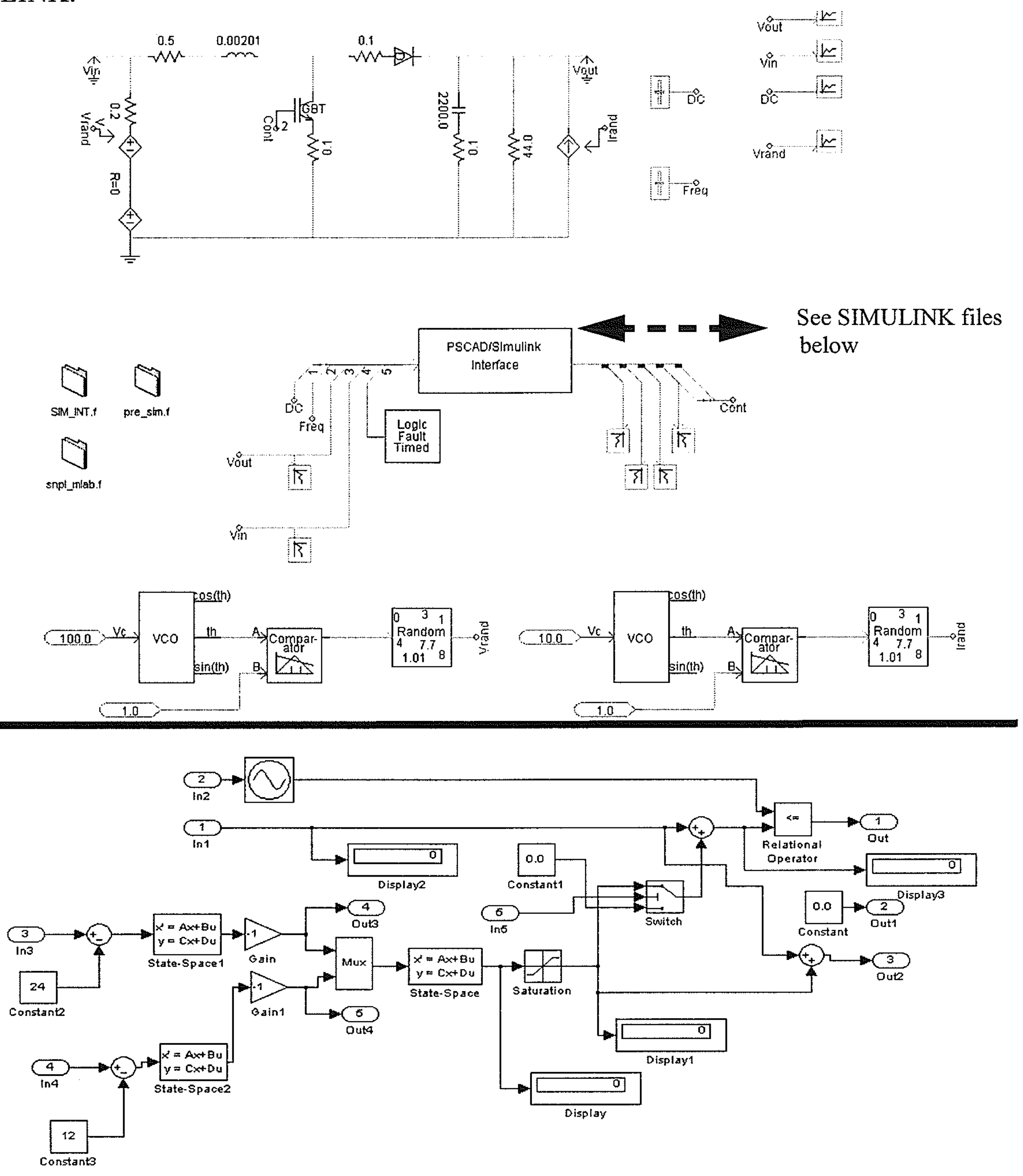


Fig. 5. 17 Implemented hybrid simulation to test the H_{∞} controller

The hybrid simulation shows a different result on the amount of damping by the controller to what was presented in the simulation results with SIMULINK. The process to incorporate the controller into a practical setting shows that the effectiveness of the controller decreases, and the amount of noise reduction is a little over 67% seen in Figure 5.18, as compared to 98% above.

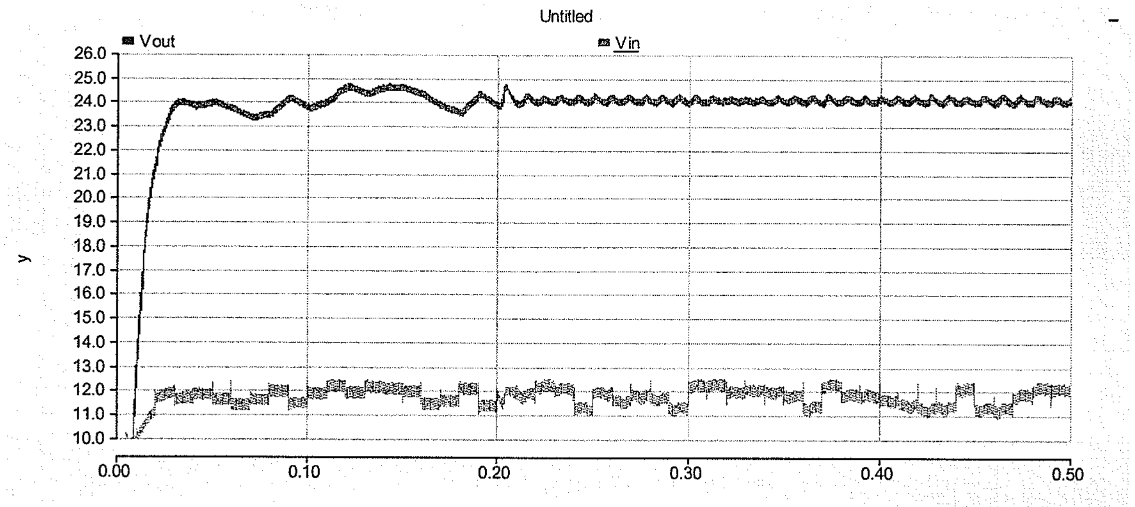


Fig. 5. 18 Hybrid Simulation Input and Output Voltage

The discrepancies in the simulation results are due to the modeling techniques used for the electrical circuit. To produce the state space representation of the circuit used to synthesis the controller required certain assumptions based on the method of averaging technique [19] and small signal analysis. These assumptions are that the duty cycle, inputs and state values remains constant with the disturbances applied to the circuit and the state values that they effect vary around these constants. By applying these assumptions the duty cycle of the IGBT can be taken into account and the state equations can be derived mathematically. In implementing a total MATLAB simulation using SIMULINK, where the modeling of the circuit includes the assumptions, the results achieve expected damping. However, because PSCAD uses a different solution technique to solve the electrical circuit, the assumptions for the circuit are not valid and produce a solution that shows the controller to damp out the injected disturbances; however the effectiveness of the controller is less than expected.

Another discrepancy between simulation results due to the modeling technique in MATLAB is the high frequency switching noise produced by the IGBT. In taking into account the actual switching of the IGBT there exists high frequency noise within the output which is not present in the SIMULINK simulation and in the frequency analysis of the circuit state equations.

An observation that applies to both simulations is the output of the synthesized controller. Within either simulation when the output of the controller is applied (0.2 seconds into the simulation) the output order of the controller decreases. Prior to the controller engaging (before 0.2 seconds of simulation time) there is a high level of output modulation. As the controller is switched into the circuit there is a decrease in the output order as the circuit works to damp out the disturbance on the output using both input and output voltage measurements, and as the level of output disturbance deceases so does the controllers response as seen in Figure 5.19.

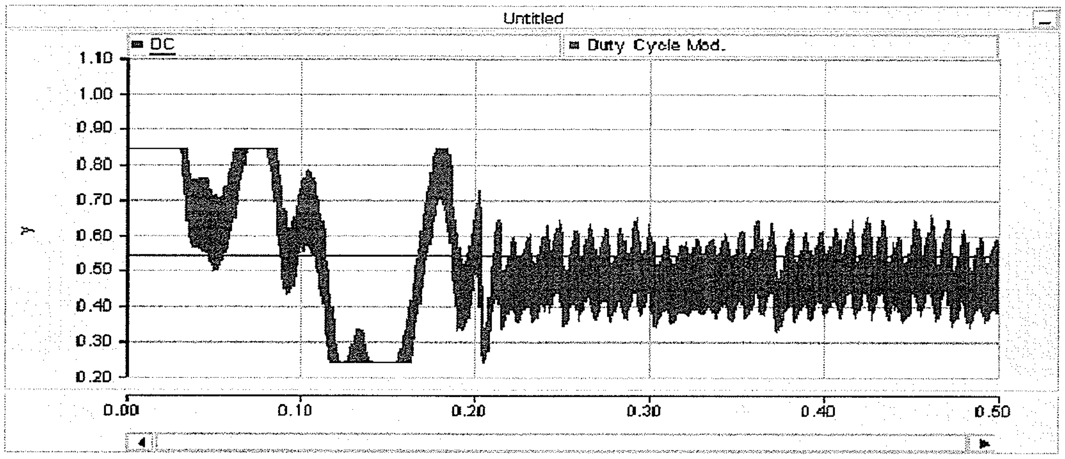


Fig. 5. 19 Duty Cycle Modulation

The application of the controller to removal of noise is an alternative method. Where other techniques can be used to remove noise such as filters tuned to the frequency range where most of the disturbances occur, to energy storage devices such as a capacitor that can minimize the noise. However in high voltage applications the cost to implement the procedure may be the deciding factor and designing efficient low cost controller my be the alternative solution to the problem.

In increasing the accuracy in the state space model (removing any assumptions in the derivation) of the electrical circuit will allow the synthesis technique to produce the controller that will be effective in the hybrid simulation. Simulations are an alternative to the construction of the electrical circuit and implementation of the synthesized controller in hardware should produce the results of the simulations. Discrepancies in the abilities or design parameters in a hardware setting can be fatal. This experience shows the need for being able to construct simulation in various platforms.

5.4 Conclusion

The design of the interface was very successful as seen through the various examples. Presented was the capabilities to handle and process state information, store data within MATLAB during a snapshot in PSCAD, and initialize a simulation from a snapshot file. Although not presented with the capabilities the developed interface as a library tool was a success by the easy integration from one simulation to the next. The integration from one simulation to the other only required editing the components parameter dialog, with no scripting. The synthesized controller using the H_∞ algorithm was implemented in a total MATLAB simulation and with the interface for a PSCAD/SIMULINK simulation. In both simulation the results show that the induced noise was reasonably damped. However because PSCAD/EMTDC incorporated all the nonlinearity due to the switching the damping is less than expected value. However the simulation indicates that the linearized assumptions on the design are still reasonable as the produced and acceptable waveforms.

CHAPTER 6 Conclusions and Recommendations

6.1 Conclusion

Presented in this document is a new approach to simulations where an interface was developed to allow two independent software program to communicate. Difficult problems that can not be solved by one software package may now be solvable through the interface of two independent software packages together. Such as the case presented in the control synthesis problem applied to the boost converter. PSCAD/EMTDC accurately simulates the electrical elements of the circuit, and MATLAB with SIMULINK to solve the control synthesis problem and provide an environment to test the results through simulations. Interfacing of the two environments creates a mutual benefit to both programs.

The process of developing the interface first began with establishing a method that would connect both software packages and then developing objectives for the interface. These objectives were based on examining what each software developer incorporates into their package and examining how the interface affects each program. Establishing the objectives also led to its implementation and structure. In having two programs that are designed to execute independently, requires the selection of one program to dominate the other when using the interface. This domination is a necessity to maintain simulation synchronization. Other issues in the development were the retention of memory and reduction in the number of executable steps within the interface. This reduction was handled by restructuring the subroutine and grouping all executable statements into the one routine

The advantages seen by establishing the interface were:

i) Choices in the type of modeling techniques available. PSCAD/EMTDC is limited in its modeling capabilities to solving the instantaneous time values using trapezoidal integration. However SIMULINK has developed several solvers and is capable of modeling elements in various formats, such as by their states or by a simple polynomial. In interfacing the two programs together the user now has the capability of selecting the most efficient method to develop their model and then later integrate it into one software program.

i1) Decreases the development time. The decrease in simulation development is accountable to the increase in available library components. The extra library components translates into less programming and design of new models for a particular software program.

These advantages are seen in the simulations presented in Chapter 5, where the control synthesis routine is only available in MATLAB and requires extensive development to be included into PSCAD. Implementing the controller as a state blocks in SIMULINK is a straightforward procedure.

If the interface was not utilized there would have been extensive programming to implement the synthesis routine of Chapter 3 into PSCAD/EMTDC. Also there would be the converting the output of the synthesis routine to a model format PSCAD/EMTDC can process.

The disadvantages seen with the interface are:

i) Simulation time. In all examples shown the amount of simulation time required with interface is several times higher than that of when the simulation is executed in one package.

ii) Ongoing support required in maintaining the interface. Since there are two independent simulation programs there has to exist support for when either program undergoes a version update.

In examining the response of the controller synthesized using the H_∞ technique, it is dependent on the model of the plant. In the SIMULINK simulation the results showed the controller to be very effective in damping out disturbances that were induced into the simulation. However when the controller was applied to the electrical circuit simulated in PSCAD/EMTDC the effectiveness of the controller decreased. In establishing the hybrid simulation modifications to the control circuit were necessary to produce the switching pulses of the IGBT, the duty cycle. Also the measurement feedback elements required the d.c. components to be removed before the measured values can be applied to the controllers inputs. Overall through this application there was the ability to include unknowns into the design of the controller which make it an ideal tool, however the synthesizing technique is limited in the ability of how the circuit can be represented by its state values.

6.2 Recommendations

With demonstrating the ability of an interface between two simulation program there exists future work for the interface block. Part of work is in integrating the component block into a reusable component. Although the block was designed with the intention of becoming a library component, there is still the issue of selecting the number of inputs and outputs. In its present state of development the component block has a minimum number of two inputs/outputs. If the user requires one input or one output there would be an error generated due to the method of defining the input/output variable data. The input/output variable names for the component block are defined as an array. FORTRAN's compiler however does not allow for declaring an array variable with the index value of one. Thus if there was only one inputs/outputs, the script within the component block needs to be adjusted.

Other work is the amendment to applications for the interface block. In most cases the interface is ideal in applications involving new techniques and theories. Or testing new theories against applications already developed within one software package. It is also the recommendation that after completing the study that validates the new technique or theory it should be incorporated into the one package the user is accustomed to.

The last case example also shows the benefit of constructing a simulation under different platforms to test the effectiveness of the new method. In the case of the hybrid simulation where the electrical circuit was modeled using one package and the controller in another, the results of the hybrid simulation showed that the controllers ability to remove unwanted noise was different when the circuit is modeled using only one software package. The method of modeling and simulating the circuit are important as to mimic real world devices where the performance in the simulation environment or the physical environment produce similar results. Taking advantage of different models in different software packages can lead to validation of the proposed method.

The final extension to the interface is the development of cases that involve electrical components modeled in both programs. This condition may not be practical to implement because there are several variables to account for such as the node voltages and currents and how the component affects the nodal values during the simulated time step. The process of how the interface will update the nodal value need to be explored and how PSCAD/EMTDC will account for the electrical component modeled outside its program. The inherent problems will be the format in parameters the model will require and the values that are returned.

The final recommendation deals with the H_∞ algorithm and analyzing a condition applied before synthesizing. In setting up the system one of the elements in the system matrix was nulled. The element in the system matrix is the D22 elements corresponds the control inputs to the feedback outputs. By nulling the matrix prior to the synthesis routine the values of the controller was synthesized with the equations presented in Chapter 3. However for the actual boost converter the system matrix D22 contains a value. In processing the values with D22 the routine synthesized a controller however it contained high gain values by the factor 10^{23}, and when simulated in the MATLAB environment the controller was unstable. However in experimenting with the routine it was found that inverting the D22 matrix reduced the overall gain factor by a factor of 10^{16}, and when this controller was simulated in the SIMULINK environment the results showed the controller stabilized the circuit. The future work in this area is understanding if the inversion of the matrix is a special case, a class or problems or a general condition. This leads to applications where the output of the synthesis routine can be applied mediately. In the case example presented if the duty cycle order were to be adjusted a new controller can be synthesized and applied immediately.

REFERENCE

[1] Balakrishanan V. Boyd, S. "Tradeoffs in Frequency-Weighted H_∞ Control", IEEE Proceedings on Computer-Aided Control System Design, March 1994, pp 7-9

[2] Balas, G.J., Doyle, J.C., Glover, K., Packard, A., Smith, R. "μ-Analysis and Synthesis Toolbox User's Guide" Version 3, MUSYN Inc. and The MathWorks, Inc. 1993-2001

[3] Dommel, H.W., "Digital Computer Solution of Electromagnetic Transients in Single- and Multi-Phase Networks", IEEE Trans. PAS, Vol. PAS-88, No. 4, April 1969, pp 388-399

[4] Doyle, J.C., Khargonekar P.P. "State-Space Solutions to Standard H_2 and H_∞ Control Problems", IEEE Transactions on Automatic Control., Vol. 34, No. 8, August 1989, pp 831-847.

[5] Doyle, J. Stein, G. “Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis” IEEE Transactions on Automatic Control, Vol AC-26, NO. 1, February 1981, pp 4-16

[6] Doyle, J. Francis,B. Tahhembaum, A. “Feedback Control Theory” Macmillan Publishing co. 1990.

[7] Erickson, R.W. “DC-DC Power Converters”, Article in Wiley Encyclopedia of Electrical and Electronics Engineering

[8] Francis, B.A. “A Course in H Control Theory” New York: Springer-Verlag 1987.

[9] Francis, B. Doyle, J.C. “Linear Control Theory with an H Optimality Criterion” Siam j. Control and Optimization, Vol. 25. No4. July 1987.

[10] A.M. Gole, A. Daneshpooy, “Towards Open Systems: A PSCAD/EMTDC to MATLAB Interface”, International Conference on Power System Transients, IPST’97, Seattle, June 22-26, 1997, Proceedings: pp 145-149.

[11] Gole, A.M., Demchenko, P., Kell, D., Irwin, G.D.,”Integreating Electromagnetic Transient Simulation with Other Design Tools”, International Conference on Power Systems Transients, (IPST’2001), Rio de Janeiro, Brazil June, 2001

[12] A.M. Gole, O.B. Nayak. T.S. Sidhu, M.S. Sachdev, “A Graphical Electromagnetic Simulation Laboratory for Power Systems Engineering Programs”, IEEE PES Summer Meeting, Portland, OR, July 1995.

[13] Kezunovic, M., Chen, Q.,” A Novel Approach for Interactive Protection System Simulation”, IEEE T&D Conference, Los Angeles, September 1996

[14] Kwakernaak, H. “The Polynomial Approach to Optimal Control”, Polynomial Methods in Optimal Control and Filtering, IEE Colloquium on, 10 May 1991 pp 4/1 - 4/3

[15] Kwakernaak, H. Gjerrit, M. “Design Methods for Control Systems: Dutch Institute of Systems and Control, 2000-2001.

[16] Kwakernaak H., “Robust Control and H_∞-Optimization- Tutorial Paper” Automatica, Vol. 29, No.2 1993 pp 255-273

[17] Lundstrom, P., Skogestad, S., Wans, Z. “Uncertainty Weight Selection for H_∞ and μ-Control Methods”, Proceedings of the 30th Conference on Decision and Control, Brighton, England, December 1991, pp 1537-1540

[18] Mahseredjian, J. and Alvarado, F.; “The Design of Time Domain Simulation Tools: the Computational Engine Approach” Proceedings of the International Conference on Power System Transients, Lisbon, Portugal, Sept. 3-7, 1995, pp 493-498

[19] Middlebrook, R.D., Slobodan C. “A general unified approach to modelling Switching-converter power stages” INT. J. Electronics, 1977, Vol. 42. No. 6, pp 521-550

[20] McFarlane, D. Glover, K. “A Loop Shaping Design Procedure using H_∞ Synthesis” IEEE Transactions on Automatic Control, VOl. 37., No. 6, June 1992, pp 759-769.

[21] The Mathworks Inc., “MATLAB, The Language for Technical Computing”, The Mathworks Inc., September 1998.

[22] The Mathworks Inc., “SIMULINK Dynamic Simulation for MATLAB”, The Mathworks Inc., December 1996

[23] “MATLAB The Language of Technical Computing Application Program Interface Reference Version 6”

[24] Nami R. Weiss G. Den-Yaakov S. “H_∞ Control Applied to Boost Power Converters” IEEE Transactions on Power Electronics Vol. 12 No. 4 July 1997 pp 677-683

[25] Onillon, E. Upton, BNL. “Feedback Design Methods Review and Comparison” Proceedings of the 1999 Particle Accelerator Conference, New York 1999 pp1109-1111.

[26] PSCAD Version 4.0 On-Line Help V4.1.0 Copyright 2003 Manitoba HVDC Research Center

[27] Rami, NAIM, Weiss, G. Ben-Yaakov S.”H_∞ Control of Boost Converter” Applied Power Electronics Conference and Exposition, 1995. APEC '95. Conference March 1995, pp719-722

[28] Stoorvogel, A.A. “The H_∞ Control Problem: A State Space Approach” Department of Electrical Engineering and Computer Science, University of Michigan, February5, 2000

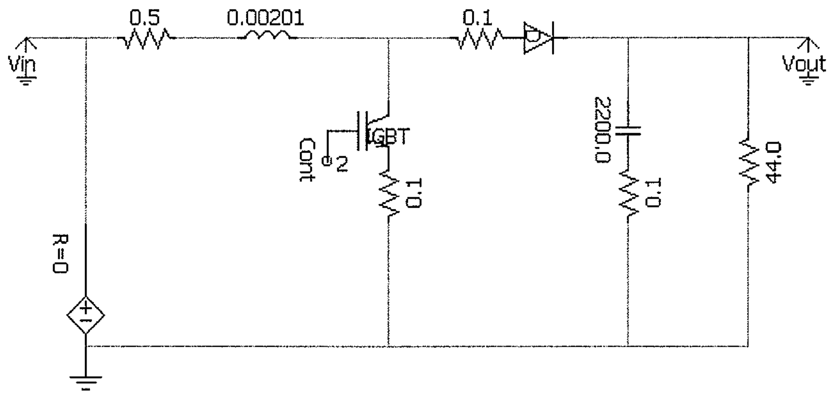
[29] Shinners, S.M. “Modern Control Systems Theory and Design, 2nd Edition”, John Wiley & Sons Inc. 1998.

[30] Willems J.C., “Least Squares Optimal Control and the Algebraic Riccati Equation” IEEE Transactions on Automatic Control, Vol. AC-16, No.6, December 1971 pp 621-634.

[31] Zames, G. “Feedback and Optimal Sensitivity: Model Reference Transformations, Multiplicative Seminorms, and Approximate Inverses” IEEE Transactions on Automatic Control, Vol. AC-26, No. 2, April 1981, pp 301-320.

APPENDIX A Derivation of Boost Converter Circuit

From the given circuit in below.



The method of averaging is based on the circuit is divided into the two conditions. When the IGBT is on and when the IGBT is off Figures 3.1 and 3.2 respectively. From each of the figures the appropriate state equations can be derived.

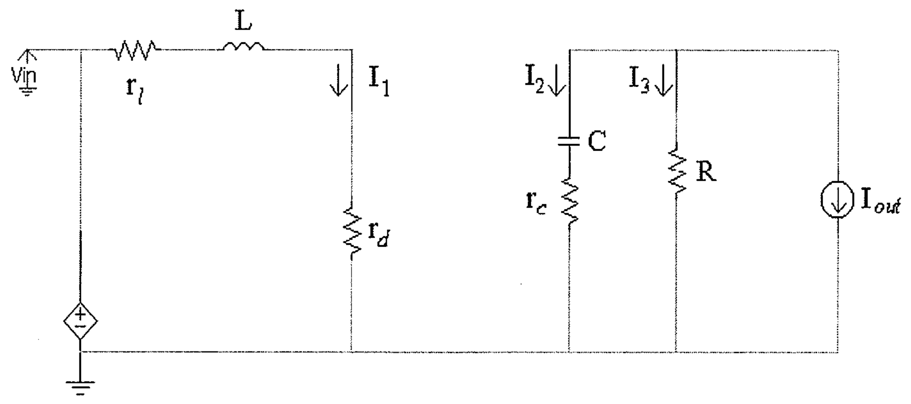


Fig. A1. 1 IGBT ON

When the IGBT is on the following equations are valid.

$$Vin = v_{rl} + v_l + v_{rd}$$

$$v_{rl} = -v_{rl} - v_{rd} + Vin$$

$$v_{rl} = -(r_l + r_d)\cdot i_l + Vin$$

$$L\cdot\frac{\mathrm{d}i}{\mathrm{d}t} = -(r_l + rd)\cdot i_l + Vin$$

$$\frac{\mathrm{d}i}{\mathrm{d}t} = -\frac{(r_l + rd)}{L}\cdot i_l + \frac{1}{L}\cdot Vin$$

$$I_2 + I_3 + Iout = 0$$

$$I_2 = -I_3 - Iout$$

$$I_2 = -\frac{V_{r3}}{R} - Iout$$

$$I_2 = -\frac{(v_c + v_{rc})}{R} - Iout$$

$$I_2 = -\frac{1}{R}\cdot v_c - \frac{r_c}{R}\cdot I_2 - Iout$$

$$\left(1 + \frac{r_c}{R}\right)\cdot I_2 = -\frac{1}{R}\cdot v_c - Iout$$

$$\left(1 + \frac{r_c}{R}\right)\cdot C\frac{\mathrm{d}v_c}{\mathrm{d}t} = -\frac{1}{R}\cdot v_c - Iout$$

$$\frac{\mathrm{d}v_c}{\mathrm{d}t} = -\frac{1}{C\cdot(R + r_c)}\cdot v_c - \frac{R}{C\cdot(R + r_c)}\cdot Iout$$

$$Vout = v_c + v_{rc}$$

$$Vout = v_c + r_c\cdot i_c$$

$$Vout = v_c + r_c\cdot(-I_2 - Iout)$$

$$Vout = v_c - \frac{r_c}{R}\cdot Vout - r_c\cdot Iout$$

$$\left(1 + \frac{r_c}{R}\right)Vout = v_c - r_c\cdot Iout$$

$$Vout = \frac{R}{R + r_c}\cdot v_c - \frac{R\cdot r_c}{R + r_c}\cdot Iout$$

$$A_{on} = \begin{bmatrix} \dfrac{-(r_l + r_s)}{L} & 0 \\ 0 & \dfrac{-1}{C \cdot (R + r_c)} \end{bmatrix} \qquad B_{on} = \begin{bmatrix} \dfrac{1}{L} & 0 \\ 0 & \dfrac{-R}{C \cdot (R + r_c)} \end{bmatrix}$$

$$C_{on} = \begin{bmatrix} 0 & \dfrac{R}{(R + r_c)} \end{bmatrix} \qquad D_{on} = \begin{bmatrix} 0 & \dfrac{-R \cdot r_c}{(R + r_c)} \end{bmatrix}$$

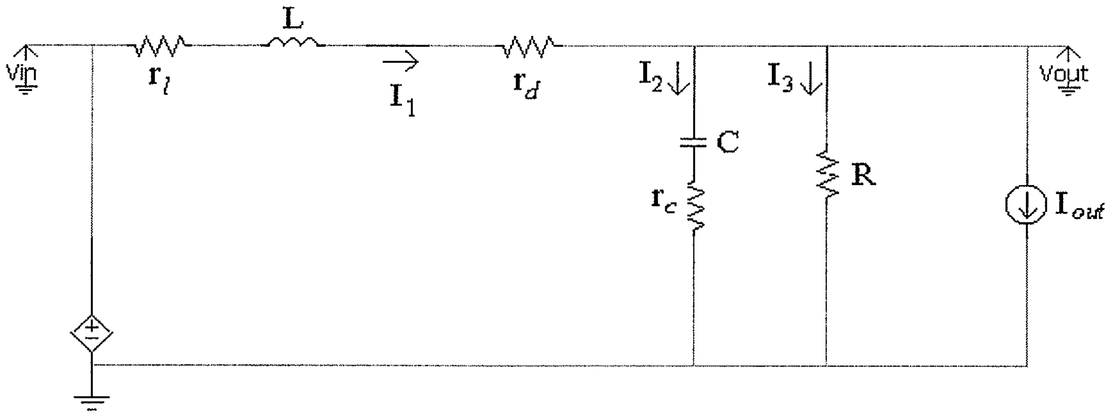


Fig. A1. 2 IGBT OFF

When the IGBT is on the following equations are valid

$$I_1 = I_2 + I_3 + Iout$$

$$i_l = i_c + \frac{Vout}{R} + Iout$$

$$i_l = i_c + \frac{(v_c + v_{rc})}{R} + Iout$$

$$i_l = i_c + \frac{1}{R} \cdot v_c + \frac{r_c}{R} \cdot ic + Iout$$

$$i_l = \left(1 + \frac{r_c}{R}\right) i_c + \frac{1}{R} \cdot v_c + Iout$$

$$\left(\frac{R + r_c}{R}\right) \cdot C \frac{dv_c}{dt} = i_l - \frac{1}{R} \cdot v_c - Iout$$

$$\frac{dv_c}{dt} = \frac{R}{C \cdot (R + r_c)} \cdot i_l - \frac{1}{C \cdot (R + r_c)} \cdot v_c - \frac{R}{C \cdot (R + r_c)} \cdot Iout$$

$$Vin = v_{rl} + v_l + v_{rd} + v_e + v_{rc}$$

$$v_l = -(r_l + r_d) \cdot i_l - v_c - r_c \cdot i_c + Vin$$

$$v_l = -(r_l + r_d) \cdot i_l - v_c - \left(r_c \cdot C \frac{dv_c}{dt}\right) + Vin$$

$$v_l = -(r_l + r_d) \cdot i_l - v_c - \left(r_c \cdot C \left(\frac{R}{C \cdot (R + r_c)} \cdot i_l - \frac{1}{C \cdot (R + r_c)} \cdot v_c - \frac{R}{C \cdot (R + r_c)} Iout\right)\right) + Vin$$

$$v_l = -\left((r_l + r_d) + \frac{R \cdot r_c}{R + r_c}\right) \cdot i_l - \left(1 - \frac{r_c}{(R + r_c)}\right) v_c + Vin + \frac{R \cdot r_c}{(R + r_c)} Iout$$

$$L \frac{di_l}{dt} = -\frac{(r_l + r_d) \cdot (R + r_c) + R \cdot r_c}{R + r_c} \cdot i_l - \frac{R}{R + r_c} \cdot v_c + Vin + \frac{R \cdot r_c}{(R + r_c)} Iout$$

$$\frac{di_l}{dt} = -\frac{(r_l + r_d) \cdot (R + r_c) + R \cdot r_c}{L \cdot (R + r_c)} \cdot i_l - \frac{R}{L \cdot (R + r_c)} \cdot v_c + \frac{1}{L} \cdot Vin + \frac{R \cdot r_c}{L \cdot (R + r_c)} Iout$$

$$I_1 = I_2 + I_3 + Iout$$

$$I_3 = i_l - i_c - Iout$$

$$\frac{Vout}{R} = i_l - C\frac{dv_c}{dt} - Iout$$

$$\frac{Vout}{R} = i_l - C\left(\frac{R}{C\cdot(R+r_c)}\cdot i_l - \frac{1}{C\cdot(R+r_c)}\cdot v_c - \frac{R}{C\cdot(R+r_c)}\cdot Iout\right) - Iout$$

$$\frac{Vout}{R} = \left(1 - \frac{R}{(R+r_c)}\right)\cdot i_l + \frac{1}{(R+r_c)}\cdot v + \left(\frac{R}{(R+r_c)} - 1\right)\cdot Iout$$

$$\frac{Vout}{R} = \frac{r_c}{(R+r_c)}\cdot i_l + \frac{1}{(R+r_c)}\cdot v - \frac{r_c}{(R+r_c)}\cdot Iout$$

$$\boxed{Vout = \frac{R\cdot r_c}{(R+r_c)}\cdot i_l + \frac{R}{(R+r_c)}\cdot v - \frac{R\cdot r_c}{(R+r_c)}\cdot Iout}$$

$$A_{off} = \begin{bmatrix} \frac{-[(r_l + r_s)\cdot(R+r_c) + r_c\cdot R]}{L\cdot(R+r_c)} & \frac{-R}{L\cdot(R+r_c)} \\ \frac{R}{C\cdot(R+r_c)} & \frac{-1}{C\cdot(R+r_c)} \end{bmatrix} \qquad B_{off} = \begin{bmatrix} \frac{1}{L} & \frac{R\cdot r_c}{L\cdot(R+r_c)} \\ 0 & \frac{-R}{C\cdot(R+r_c)} \end{bmatrix}$$

$$C_{off} = \begin{bmatrix} \frac{R\cdot r_c}{(R+r_c)} & \frac{R}{(R+r_c)} \end{bmatrix} \qquad D_{off} = \begin{bmatrix} 0 & \frac{-R\cdot r_c}{(R+r_c)} \end{bmatrix}$$

To inculde the switching of the IGBT an average of the states are applied based on the duty cycle [18].

Basic State space average model

$$\dot{x} = Ax + bu \qquad x = \begin{bmatrix} i_l \\ v_c \end{bmatrix} \qquad u = \begin{bmatrix} Vin \\ Iout \end{bmatrix}$$

$$y = cx + du \qquad y = Vout$$

D=Duty Cycle

$$A = D \cdot A_{on} + (1-D) \cdot A_{off}$$

$$b = D \cdot B_{on} + (1-D) \cdot B_{off}$$

$$c = D \cdot C_{on} + (1-D) \cdot C_{off}$$

$$d = D \cdot D_{on} + (1-D) \cdot D_{off}$$

$$A = D \cdot \begin{bmatrix} \frac{-(r_l + r_s)}{L} & 0 \\ 0 & \frac{-1}{C \cdot (R + r_c)} \end{bmatrix} + (1-D) \cdot \begin{bmatrix} \frac{-[(r_l + r_s) \cdot (R + r_c) + r_c \cdot R]}{L \cdot (R + r_c)} & \frac{-R}{L \cdot (R + r_c)} \\ \frac{R}{C \cdot (R + r_c)} & \frac{-1}{C \cdot (R + r_c)} \end{bmatrix}$$

$$A = \begin{bmatrix} D \cdot \frac{-(r_l + r_s)}{L} + (1-D) \cdot \frac{-[(r_l + r_s) \cdot (R + r_c) + r_c \cdot R]}{L \cdot (R + r_c)} & (1-D) \cdot \frac{-R}{L \cdot (R + r_c)} \\ (1-D) \cdot \frac{R}{C \cdot (R + r_c)} & D \cdot \frac{-1}{C \cdot (R + r_c)} + (1-D) \cdot \frac{-1}{C \cdot (R + r_c)} \end{bmatrix}$$

$$A = \begin{bmatrix} \frac{(-D \cdot ((R + r_c) \cdot (r_s - r_d) - r_c R))}{L \cdot (R + r_c)} + \frac{-(r_l + r_d) \cdot (R + r_c) - r_c R}{L \cdot (R + r_c)} & \frac{(-R) \cdot (1 - D)}{L \cdot (R + r_c)} \\ \frac{R \cdot (1 - D)}{C \cdot (R + r_c)} & \frac{-1}{C \cdot (R + r_c)} \end{bmatrix}$$

$$b = D \cdot \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{-R}{C \cdot (R + r_c)} \end{bmatrix} + (1 - D) \cdot \begin{bmatrix} \frac{1}{L} & \frac{R \cdot r_c}{L \cdot (R + r_c)} \\ 0 & \frac{-R}{C \cdot (R + r_c)} \end{bmatrix}$$

$$b = \begin{bmatrix} D \cdot \frac{1}{L} + (1 - D) \cdot \frac{1}{L} & (1 - D) \cdot \frac{R \cdot r_c}{L \cdot (R + r_c)} \\ 0 & D \cdot \frac{-R}{C \cdot (R + r_c)} + (1 - D) \cdot \frac{-R}{C \cdot (R + r_c)} \end{bmatrix}$$

$$\boxed{b = \begin{bmatrix} \frac{1}{L} & \frac{(1 - D) \cdot R \cdot r_c}{L \cdot (R + r_c)} \\ 0 & \frac{-R}{C \cdot (R + r_c)} \end{bmatrix}}$$

$$c = D \cdot \begin{bmatrix} 0 & \frac{R}{(R + r_c)} \end{bmatrix} + (1 - D) \cdot \begin{bmatrix} \frac{R \cdot r_c}{(R + r_c)} & \frac{R}{(R + r_c)} \end{bmatrix}$$

$$c = \begin{bmatrix} (1 - D) \cdot \frac{R \cdot r_c}{(R + r_c)} & D \cdot \frac{R}{(R + r_c)} + (1 - D) \cdot \frac{R}{(R + r_c)} \end{bmatrix}$$

$$\boxed{c = \begin{bmatrix} \frac{(1 - D) \cdot R \cdot r_c}{(R + r_c)} & \frac{R}{(R + r_c)} \end{bmatrix}}$$

$$d = D \cdot \begin{bmatrix} 0 & \frac{-R \cdot r_c}{(R + r_c)} \end{bmatrix} + (1 - D) \cdot \begin{bmatrix} 0 & \frac{-R \cdot r_c}{(R + r_c)} \end{bmatrix}$$

$$d = D \cdot \begin{bmatrix} 0 & D \cdot \frac{-R \cdot r_c}{(R + r_c)} + (1 - D) \cdot \frac{-R \cdot r_c}{(R + r_c)} \end{bmatrix}$$

$$\boxed{d = \begin{bmatrix} 0 & \frac{-(R \cdot r_c)}{(R + r_c)} \end{bmatrix}}$$

State Space equations with Perturbed

$$d(t) = D + \hat{d} \qquad x = X + \hat{x} \qquad u(t) = U + \hat{u} \qquad y(t) = Y + \hat{y}$$

$$\dot{x} = Ax + bu$$

$$\dot{x} = (D + \hat{d}) \cdot (A_{on} \cdot (X + \hat{x}) + B_{on} \cdot (U + \hat{u})) + (1 - (D + \hat{d})) \cdot (A_{off} \cdot (X + \hat{x}) + B_{off} \cdot (U + \hat{u}))$$

$$\dot{x} = (D \cdot A_{on} + (1 - D) \cdot A_{off}) \cdot X + (D \cdot B_{on} + (1 - D) \cdot B_{off}) \cdot U + (D \cdot A_{on} + (1 - D) \cdot A_{off}) \cdot \hat{x}$$

$$'''' + (D \cdot B_{on} + (1 - D) \cdot B_{off}) \cdot \hat{u} + ((A_{on} - A_{off}) \cdot X + (B_{on} - B_{off}) \cdot U) \cdot \hat{d}$$

$$'''' + \hat{d} \cdot ((A_{on} - A_{off}) \cdot \hat{x} + (B_{on} - B_{off}) \cdot \hat{u})$$

$$\dot{\hat{x}} = A \cdot X + b \cdot U + A \cdot \hat{x} + b \cdot \hat{u} + ((A_{on} - A_{off}) \cdot X + (B_{on} - B_{off}) \cdot U) \cdot \hat{d}$$

$$'''' + \hat{d} \cdot ((A_{on} - A_{off}) \cdot \hat{x} + (B_{on} - B_{off}) \cdot \hat{u})$$

$$A_{on} - A_{off} = \begin{bmatrix} \frac{-(r_l + r_s) \cdot (R + r_c)}{L \cdot (R + r_c)} & 0 \\ 0 & \frac{-1}{C \cdot (R + r_c)} \end{bmatrix} - \begin{bmatrix} \frac{-[(r_l + r_s) \cdot (R + r_c) + r_c \cdot R]}{L \cdot (R + r_c)} & \frac{-R}{L \cdot (R + r_c)} \\ \frac{R}{C \cdot (R + r_c)} & \frac{-1}{C \cdot (R + r_c)} \end{bmatrix}$$

$$A_{on} - A_{off} = \begin{bmatrix} \frac{r_c \cdot R}{L \cdot (R + r_c)} & \frac{R}{L \cdot (R + r_c)} \\ \frac{-R}{C \cdot (R + r_c)} & \frac{1}{C \cdot (R + r_c)} \end{bmatrix}$$

$$B_{on} - B_{off} = \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{-R}{C \cdot (R + r_c)} \end{bmatrix} - \begin{bmatrix} \frac{1}{L} & \frac{R \cdot r_c}{L \cdot (R + r_c)} \\ 0 & \frac{-R}{C \cdot (R + r_c)} \end{bmatrix}$$

$$\boxed{B_{on} - B_{off} = \begin{bmatrix} 0 & \frac{-(R \cdot r_c)}{L \cdot (R + r_c)} \\ 0 & 0 \end{bmatrix}}$$

$$0 = X \cdot A + B \cdot U$$

$$\boxed{X = -A^{-1} \cdot B \cdot U}$$

$$\boxed{\hat{x} = A \cdot \hat{x} + b \cdot \hat{u} + ((A_{on} - A_{off}) \cdot X + (B_{on} - B_{off}) \cdot U) \cdot \hat{d}}$$

$$y = cx + du$$

$$Y + \hat{y} = (D + \hat{d}) \cdot (C_{on} \cdot (X + \hat{x}) + D_{on} \cdot (U + \hat{u})) + (1 - (D + \hat{d})) \cdot (C_{off} \cdot (X + \hat{x}) + D_{off} \cdot (U + \hat{u}))$$

$$Y + \hat{y} = (D \cdot C_{on} + (1 - D) \cdot C_{off}) \cdot X + (D \cdot D_{on} + (1 - D) \cdot D_{off}) \cdot U + (D \cdot C_{on} + (1 - D) \cdot C_{off}) \cdot \hat{x}$$

$$"" + (D \cdot D_{on} + (1 - D) \cdot D_{off}) \cdot \hat{u} + ((C_{on} - C_{off}) \cdot X + (D_{on} - D_{off}) \cdot U) \cdot \hat{d}$$

$$"" + ((C_{on} - C_{off}) \cdot \hat{x} + (D_{on} - D_{off}) \cdot \hat{u}) \cdot \hat{d}$$

$$\boxed{Y + \hat{y} = c \cdot X + d \cdot U + c \cdot \hat{x} + d \cdot \hat{u} + ((C_{on} - C_{off}) \cdot X + (D_{on} - D_{off}) \cdot U) \cdot \hat{d}}$$

$$\boxed{Y = c \cdot X + d \cdot U}$$

$$\boxed{\hat{y} = c \cdot \hat{x} + d \cdot \hat{u} + ((C_{on} - C_{off}) \cdot X + (D_{on} - D_{off}) \cdot U) \cdot \hat{d}}$$

APPENDIX B MATLAB Code

```
%
%IEEE Transactions on Power Electronics Vol.12, No.4, July 1997, Pg 677-683
%
%
% defining circuit values
L=2.01e-3; C=2200e-6; R=44; r=0.1; rs=0.1; rd=0.1; rl=0.7;
Vin=12.0; Iout=0.0;
D=0.54;
%
%defining state matrix for different operations
A1=[-(rl+rs)/L 0;0 -1/(C*(R+r))];
b1=[1/L 0;0 -R/(C*(R+r))];
c1=[0 R/(R+r)];
d1=[0 -(R*r)/(R+r)];
A2=[-((rl+rd+((r*R)/(R+r))))/L -R/(L*(R+r)); R/(C*(R+r)) -1/(C*(R+r))];
b2=[1/L (R*r)/(L*(R+r));0 -R/(C*(R+r))];
c2=[(R*r)/(R+r) R/(R+r)];
d2=[0 -(R*r)/(R+r)];
%
clear L C R r rs rd rl
%
%Derived values for the State Matrix\
A=D*A1+(1-D)*A2;
B1=D*b1+(1-D)*b2;
C1=D*c1+(1-D)*c2;
D11=D*d1+(1-D)*d2;
%
% Defining steady state values and small signal analysis
w=[Vin;Iout];
```

```
X=(-inv(A)*B1)*w;
B2=(A1-A2)*X+(b1-b2)*w;
D12=(c1-c2)*X+(d1-d2)*w;
%
clear Vin Iout w X
%
% included variables defined by choice of feedback
C2=[C1(1) C1(2);0 0];
D21=[D11(1) D11(2);1 0];
D22=[-D12(1);0];
%
sys=pck(A,[B1 B2],[C1;C2],[D11 D12;D21 D22]);
seesys(sys);
minfo(sys);
hinfnorm(sys);
%
%
%clear A1 b1 c1 d1 A2 b2 c2 d2 D
%clear A B1 B2 C1 C2 D11 D12 D21 D22
%
%Weight function
w1=nd2sys([1 10],[1 700],10);
%wp=nd2sys([700],[1 700],1);
w2=nd2sys([1 10],[1 700],1);
w3=nd2sys([1 700],[1 10],1);
%
systemnames='sys w1 w2 w3';
inputvar='[Vin; Iout; control]';
outputvar='[w3; sys(2); sys(3)]';
input_to_sys='[w1; w2; control]';
input_to_w1='[Vin]';
input_to_w2='[Iout]';
input_to_w3='[sys(1)]';
sysoutname='sys_ic';
cleanupsysic='yes';
sysic;
minfo(sys_ic);
[k,clp]=hinfsyn(sys_ic,2,1,0.1,70,0.01)
omega=logspace(-1,6,100);
[ak,bk,ck,dk]=unpck(k);
clp_g=frsp(clp,omega);
vplot('liv,m',clp_g);
title('Frequency Response of closed loop system');
figure
sys_g=frsp(sys_ic,omega);
vplot('liv,m',sys_g)
```

```
title('Frequency Response of open loop system');
%
% Calculating the band pass filter for the controller input
%
[af,bf,cf,df]=butter(2,90,'high','s');
[afl,bfl,cfl,dfl]=butter(2,800,'low','s');
%[afl,bfl,cfl,dfl]=butter(2,800,'low','s');
%
%clear wp wp2 wp3 ans clp clp_g omega sys sys_g sys_ic k
%
%IEEE Transactions on Power Electronics Vol.12, No.4, July 1997, Pg 677-683
%
% defining circuit values
L=2.01e-3; C=2200e-6; R=44; r=0.1; rs=0.1; rd=0.1; rl=0.7;
Vin=12.0; Iout=0.0;
D=0.54;
%
%defining state matrix for different operations
A1=[-(rl+rs)/L 0;0 -1/(C*(R+r))];
b1=[1/L 0;0 -R/(C*(R+r))];
c1=[0 R/(R+r)];
d1=[0 -(R*r)/(R+r)];
A2=[-((rl+rd+((r*R)/(R+r))))/L -R/(L*(R+r)); R/(C*(R+r)) -1/(C*(R+r))];
b2=[1/L (R*r)/(L*(R+r));0 -R/(C*(R+r))];
c2=[(R*r)/(R+r) R/(R+r)];
d2=[0 -(R*r)/(R+r)];
%
clear L C R r rs rd rl
%
%Derived values for the State Matrix\
A=D*A1+(1-D)*A2;
B1=D*b1+(1-D)*b2;
C1=D*c1+(1-D)*c2;
D11=D*d1+(1-D)*d2;
%
% Defining steady state values and small signal analysis
w=[Vin;Iout];
X=(-inv(A)*B1)*w;
B2=(A1-A2)*X+(b1-b2)*w;
D12=(c1-c2)*X+(d1-d2)*w;
%
clear Vin Iout w X
%
% included variables definded by choice of feedback
C2=[C1(1) C1(2);0 0];
D21=[D11(1) D11(2);1 0];
```

```
D22=[-D12(1);0];
%
sys=pck(A,[B1 B2],[C1;C2],[D11 D12;D21 D22]);
seesys(sys);
minfo(sys);
hinfnorm(sys);
%
%clear A1 b1 c1 d1 A2 b2 c2 d2 D
%clear A B1 B2 C1 C2 D11 D12 D21 D22
%
%Weight function
w1=nd2sys([1 10],[1 700],10);
%wp=nd2sys([700],[1 700],1);
w2=nd2sys([1 10],[1 700],1);
w3=nd2sys([1 700],[1 10],1);
%
systemnames='sys w1 w2 w3';
inputvar='[Vin; Iout; control]';
outputvar='[w3; sys(2); sys(3)]';
input_to_sys='[w1; w2; control]';
input_to_w1='[Vin]';
input_to_w2='[Iout]';
input_to_w3='[sys(1)]';
sysoutname='sys_ic';
cleanupsysic='yes';
sysic;
minfo(sys_ic);
[k,clp]=hinfsyn(sys_ic,2,1,0.1,70,0.01)
omega=logspace(-1,6,100);
[ak,bk,ck,dk]=unpck(k);
clp_g=frsp(clp,omega);
vplot('liv,m',clp_g);
title('Frequency Response of closed loop system');
figure
sys_g=frsp(sys_ic,omega);
vplot('liv,m',sys_g)
title('Frequency Response of open loop system');
%
% Calculating the band pass filter for the controller input
%
[af,bf,cf,df]=butter(2,90,'high','s');
[afl,bfl,cfl,dfl]=butter(2,800,'low','s');
%[afl,bfl,cfl,dfl]=butter(2,800,'low','s');
%
%clear wp wp2 wp3 ans clp clp_g omega sys sys_g sys_ic k
```