

**CIFER<sup>®</sup>-MATLAB Interfaces:**  
**Development and Application**

A Thesis  
Presented to the Faculty of  
California Polytechnic State University  
San Luis Obispo

In Partial Fulfillment  
Of the Requirements for the Degree of  
Master of Science in Aerospace Engineering

By  
Brian K. Rupnik  
April 2005

**© Copyright 2005**

**Brian K. Rupnik**

**All Rights Reserved**

## Approval Page

TITLE: CIPHER<sup>®</sup>-MATLAB Interfaces: Development and Application

AUTHOR: Brian K. Rupnik

DATE SUBMITTED: April 2005

Dr. Daniel J. Biezad (AERO)

---

Advisor & Committee Chair

Dr. Mark B. Tischler (NASA/Army)

---

Committee Member

Dr. Eric Mehiel (AERO)

---

Committee Member

Dr. Lanny V. Griffin (CENG)

---

Committee Member

## **Abstract**

### **CIFER<sup>®</sup>-MATLAB Interfaces: Development and Application**

**Brian K. Rupnik**

The Army/NASA Rotorcraft Division, Flight Controls Group, Ames Research Center, has developed and is maintaining a software package called CIFER<sup>®</sup> or Comprehensive Identification from FrEQuency Responses. CIFER<sup>®</sup> allows system identification in the frequency domain and is considered to be one the top resources for frequency analysis. It provides methods to derive frequency responses, transfer functions and state-space models from a time sweep data.

The interface for CIFER<sup>®</sup> was developed long enough ago that there is a significant demand for a modernization of the software. To address the demand in the most complete manner would involve updating a very complex series of programs with modern graphical and command-line interfaces. This project is beyond the scope of an Aerospace Master's thesis. However, before the Army devotes resources to this task, they desire a 'proof of concept.'

This thesis is that proof of concept. Many users of CIFER<sup>®</sup> agree that having CIFER<sup>®</sup> programs and utilities usable from the MATLAB command-line or modernized graphical interface would be a major benefit. The Army agreed that development of a CIFER<sup>®</sup>-MATLAB interface would be both a useful tool and a stepping-stone for where they would like to take CIFER<sup>®</sup> in the future.

There are two main tasks that make up this thesis. The first task is the development of a CIFER<sup>®</sup>-MATLAB interface, both at the command line and in a graphical user interface. This interface covers some, but not all of the programs in CIFER<sup>®</sup> – enough to show that the interface works and makes use of CIFER<sup>®</sup> more efficient. The second task is to validate the new interface through a series of projects including analysis of a modern Unmanned Aerial Vehicle (UAV). Both tasks were successful in the eyes of the Army sponsors and ongoing work is being conducted to implement the work from this thesis into the whole of the CIFER<sup>®</sup> program suite.

## **Acknowledgements**

The author would like to recognize and thank Dr. Daniel J. Biezad, Department Chair at Cal Poly, San Luis Obispo, CA, and Dr. Mark B. Tischler, Flight Control Group Leader, Army/NASA Rotorcraft Division, Ames Research Center, CA, for their invaluable support, guidance and encouragement throughout this project. Dexter Hermstad was instrumental in providing knowledge and input throughout the programming development process and deserves many thanks. The efforts of Dr. Colin Theodore, Dr. Jeff Lusardi, Kenny Cheung, Chad Frost, and Rendy Cheng on behalf of this endeavor are also greatly appreciated. Everyone in the Flight Control Group was highly supportive of the effort, and they all deserve recognition for their assistance at various stages of this thesis.

# Table of Contents

<b>TABLE OF FIGURES .....</b>	<b>VIII</b>
<b>LIST OF TABLES.....</b>	<b>IX</b>
<b>NOMENCLATURE .....</b>	<b>X</b>
<b>NOMENCLATURE .....</b>	<b>X</b>
<b>CHAPTER 1: INTRODUCTION .....</b>	<b>1</b>
1.1 ABOUT CIFER® (COMPREHENSIVE IDENTIFICATION FROM FREQUENCY RESPONSES) .....	2
1.1.1 What CIFER® Encompasses .....	3
1.1.2 Ames Research Center Planning Meeting .....	5
1.2 PROJECT SCOPE .....	7
<b>CHAPTER 2: SYSTEM IDENTIFICATION USING CIFER® .....</b>	<b>10</b>
2.1 CREATING A FREQUENCY RESPONSE – FRESPID .....	12
2.2 MULTIPLE INPUT ANALYSIS – MISOSA .....	13
2.3 COMBINING WINDOWS – COMPOSITE .....	14
2.4 ANALYSIS UTILITIES .....	15
2.5 PARAMETRIC MODELING – NAVFIT, DERIVID, VERIFY .....	17
<b>CHAPTER 3: PROGRAMMING AND CODE DEVELOPMENT .....</b>	<b>19</b>
3.1 COMMAND-LINE DEVELOPMENT .....	19
3.1.1 Development Process .....	20
3.1.2 Problems Encountered and Solutions.....	22
3.1.2.1 Precision Errors.....	23
3.1.2.2 Retaining Structure of Code.....	26
3.1.3 Complexity of Use.....	27
3.2 GRAPHICAL USER INTERFACE DEVELOPMENT .....	28
3.2.1 Development Process .....	29
3.2.2 Modern Updates to the Original Interface .....	29
3.2.3 Problems Encountered and Solutions.....	31
<b>CHAPTER 4: VALIDATION AND APPLICATION .....</b>	<b>33</b>
4.1 SAMPLE CIFER CASES .....	33
4.2 MASS-SPRING-DAMPER SYSTEM .....	34
4.3 UH-60 SIMULATION .....	37

4.4	ANALYSIS ON <i>SHADOW</i> 200 TUAV .....	42
4.4.1	<i>Data Consistency checks:</i> .....	43
4.4.2	<i>RMS and crossover comparisons:</i> .....	49
4.5	SUMMARY OF VALIDATIONS .....	57
<b>CHAPTER 5: CONCLUSIONS .....</b>		<b>58</b>
5.1	CODE DEVELOPMENT .....	58
5.2	ANALYSIS .....	59
5.3	FUTURE WORK .....	60
<b>BIBLIOGRAPHY .....</b>		<b>61</b>
<b>APPENDIX A: SCREEN LAYOUT COMPARISON.....</b>		<b>63</b>
<b>APPENDIX B: HELP DOCUMENTATION FOR COMMAND-LINE INTERFACE.....</b>		<b>68</b>
<b>APPENDIX C: ONLINE HELP FOR MAIN PROGRAMS .....</b>		<b>83</b>
<b>APPENDIX D: STRUCTURE FIELD SPECIFICS FOR MAIN PROGRAMS .....</b>		<b>88</b>
<b>APPENDIX E: ONLINE HELP FOR ANALYSIS UTILITIES .....</b>		<b>90</b>
<b>APPENDIX F: STRUCTURE FIELD SPECIFICS FOR ANALYSIS UTILITIES .....</b>		<b>96</b>
<b>APPENDIX G: ONLINE HELP FOR SUPPORT FUNCTIONS .....</b>		<b>98</b>
<b>APPENDIX H: MASS-SPRING-DAMPER CASE EXAMPLE.....</b>		<b>100</b>
<b>APPENDIX I: XVLATSWP CASE EXAMPLE .....</b>		<b>102</b>

## Table of Figures

Figure 1.1: The Role of System Id .....	1
Figure 1.2: Frequency Sweep Example (UAV Flight Data).....	3
Figure 1.3: Doublet Example (UAV Flight Data) .....	5
Figure 1.4: Example CIPHER <sup>®</sup> Screen .....	6
Figure 1.5: Mass-Spring-Damper (Left), XV-15 (Right) .....	9
Figure 1.6: NASA Sikorsky UH-60 RASCAL (Left), <i>Shadow 200</i> TUAV (Right) .....	9
Figure 2.1: Example Coherence Plot.....	11
Figure 2.2: Example FRESPID Screen .....	13
Figure 2.3: Example RMS Prompts.....	16
Figure 3.1: Name-value Pairs and Structures .....	23
Figure 3.2: Percent Error Comparisons .....	25
Figure 3.3: Comparison of Navigation Menus .....	30
Figure 4.1: XVLATSWP Validation Examples .....	34
Figure 4.2: SISO Mass-Spring-Damper System.....	35
Figure 4.3: Mass-Spring-Damper Simulink Block Diagram.....	35
Figure 4.4: Mass-Spring-Damper System Input and Output .....	36
Figure 4.5: Matlab to CIPHER <sup>®</sup> Comparison .....	36
Figure 4.6: CIPHER <sup>®</sup> to ‘bode’ Comparison .....	36
Figure 4.7: Feedback Block Diagram.....	38
Figure 4.8: Error Channel Verification .....	38
Figure 4.9: Broken Loop Roll Gain/Phase Margin Results.....	39
Figure 4.10: Roll Bandwidth for Lateral Stick Input to Roll Angle Response.....	40
Figure 4.11: Stick and Actuator Input Autospectra.....	41
Figure 4.12: Cutoff Frequency Compared to Bandwidth Frequency .....	42
Figure 4.13: <i>Shadow 200</i> TUAV .....	42
Figure 4.14: Roll Angle to Rate Comparison.....	45
Figure 4.15: Pitch Angle to Rate Comparison.....	45
Figure 4.16: Longitudinal Velocity Perturbations.....	46
Figure 4.17: Lateral Velocity Perturbations .....	46
Figure 4.18: Vertical Velocity Perturbations.....	46
Figure 4.19: Lateral Velocity Perturbations, Flight Data .....	48
Figure 4.20: Vertical Velocity Perturbations, Flight Data.....	49



Figure 4.21: Comparisons with Exact 1/s Value .....	49
Figure 4.22: Aileron - Roll Rate Responses .....	51
Figure 4.23: Elevator - Pitch Rate Responses .....	51
Figure 4.24: Rudder to Yaw Rate Response.....	52
Figure 4.25: Rudder Output Autospectrum .....	52
Figure 4.26: Aileron to Roll Attitude Response .....	53
Figure 4.27: Elevator to Pitch Attitude Response .....	53
Figure 4.28: Rudder to Yaw Attitude Response.....	54
Figure 4.29: Scaled Roll Bandwidth Criteria .....	55
Figure 4.30: On Axis Roll Attitude Response.....	55
Figure 4.31: Scaled Pitch Category C Flight Criteria.....	56
Figure 4.32: On Axis Pitch Attitude Response.....	56
Figure 4.33: Scaled Pitch Category A Flight Criteria .....	57
Figure A1: Original Interface: Screen 1 .....	63
Figure A2: Original Interface: Screen 2 .....	63
Figure A3: Original Interface: Screen 3 .....	64
Figure A4: Original Interface: Screen 4 .....	64
Figure A5: Original Interface: Final Screen .....	64
Figure A6: MATLAB GUI: Screen 1 .....	65
Figure A7: MATLAB GUI: Screen 2.....	65
Figure A8: MATLAB GUI: Screen 3.....	66
Figure A9: MATLAB GUI: Screen 4.....	66
Figure A10: MATLAB GUI: Final Screen.....	67
Figure A11: MATLAB GUI: Two Data Loading Screens .....	67

## List of Tables

Table 4.1: Roll Gain/Phase Margin Results .....	38
Table 4.2: Roll Bandwidth Results.....	40
Table 4.3: Cutoff Frequency Results.....	41
Table 4.4: Full Range RMS Values.....	50
Table 4.5: Frequency RMS compared to Time RMS .....	51
Table 4.6: Cutoff Frequencies via RMS.....	52
Table 4.7: -135-Degree Bandwidth Frequencies .....	53

## Nomenclature

$a_x$	Longitudinal Acceleration	$\phi$	Roll Attitude
$a_y$	Lateral Acceleration	$\phi$	Phase Angle
$a_z$	Vertical Acceleration	$\gamma$	Coherence
$b$	Damping Coefficient	$\theta$	Pitch Attitude
$e$	Error Channel	$\tau$	Time Constant
$f$	Feedback Channel	$\tau_p$	Phase Delay
$G$	Autospectrum	$\omega_{180}$	180-Degree Frequency
$k$	Spring Coefficient	$\psi$	Yaw Attitude
$L$	Length		
$M$	Mass		
$N$	Scale Factor		
$p$	Roll Body Rate		
$q$	Pitch Body Rate		
$r$	Yaw Body Rate		
$s$	Frequency Domain Variable		
$U_0$	Longitudinal Velocity		Subscripts
$\dot{u}$	Longitudinal Body Acceleration	$a$	Actual Vehicle
$V_0$	Lateral Velocity	ail	Aileron
$\dot{v}$	Lateral Body Acceleration	dot	Time Derivative
$W_0$	Vertical Velocity	ele	Elevator
$\dot{w}$	Vertical Body Acceleration	$m$	Model Vehicle
$Y$	Output	rud	Rudder
$X$	Input	whl	Wheel
$\alpha$	Angle of Attack	xx	Input
$\beta$	Sideslip	xy	Cross
$\delta$	Command Channel	yy	Output

## Chapter 1: Introduction

The focus of this thesis is directed towards tools that aid in system identification. System identification is the process of taking measured data from a physical system and analyzing it to develop a mathematical model of that system. This is an important aspect of control system design as it allows for the validation of simulated system models, optimization of existing control systems, and handling qualities specification compliance. Figure 1.1 shows how system identification fits into a design cycle.

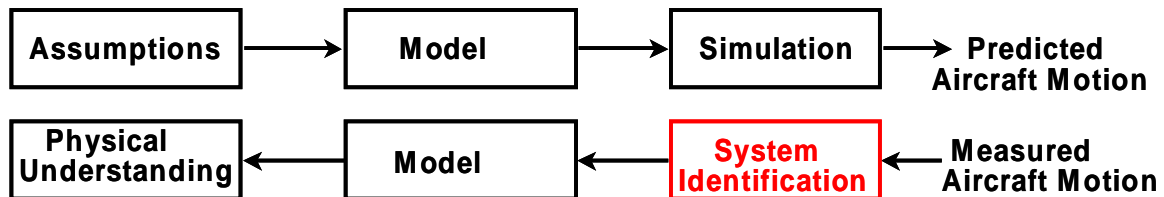


Figure 1.1: The Role of System Id

Starting at the top left of Figure 1.1, assumptions are made that result in some form of mathematical model, which describes an aircraft. The model can then be applied to a simulation that will predict the motion of the aircraft. Once a physical model of the aircraft is constructed, physical measurements can be made of its actual motion and responses to input. System identification can then be used to extract a new mathematical model of the aircraft. The new model can be compared to the old model and the assumptions used to create it for greater physical understanding of the aircraft's motion.

The mathematical model created through system identification can either be nonparametric or parametric. Nonparametric models do not assume an order or structure. They can exist either in the time-domain as an impulse response or in the frequency-domain as a frequency response.

Frequency responses are typically represented using a Bode plot format that graphs magnitude on a log scale and phase of an input-to-output ratio against frequency. Nonparametric models are useful in determining characteristics such as bandwidth, time-delay, and pilot-in-the-loop behavior. They can also be used to validate math models and determine parametric model structure and order. This project will be dealing primarily with frequency response analysis.

A parametric model assumes an order and structure with primary representations including transfer functions and state-space models. Transfer functions are pole-zero representations of individual input-output pairs. State-space models describe an entire system in terms of stability and control derivatives. Parametric models are used primarily in control system design and for wind tunnel or math model validation.

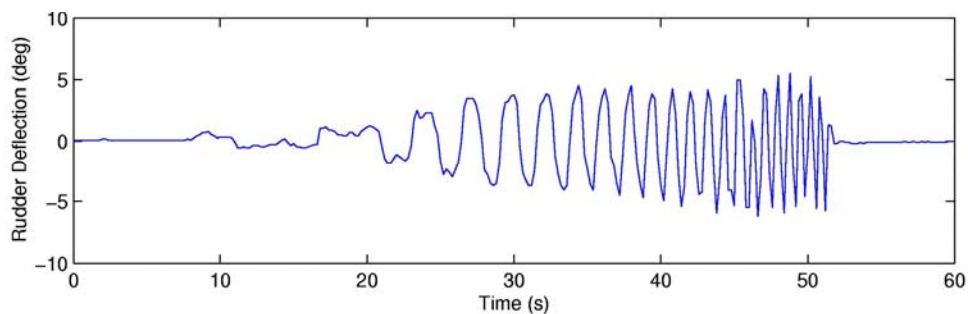
### **1.1 About CIPHER<sup>®</sup> (Comprehensive Identification from FrEQUENCY Responses)**

There are many programs that offer time domain analysis<sup>1</sup> of system response data but comparatively few that offer analysis in the frequency domain. One very successful frequency domain program, and the focus of this project, is called CIPHER<sup>®</sup>, or Comprehensive Identification from FrEQUENCY Responses. CIPHER<sup>®</sup> was developed by the Army/NASA Rotorcraft Division at the Ames Research Center during from 1988-1994 and has been constantly updated, modified, and improved since. It is used extensively by the Army/NASA Rotorcraft Division and also by many commercial aerospace companies. Some of the applications at the Ames Research Center have included development of control laws for the UH-60, identification of the XV-15 tilt-rotor demonstrator, assistance in CH-47 control development, investigation of slung load dynamics, and a wide variety of work involving unmanned aerial vehicles (UAVs). CIPHER<sup>®</sup> is considered to be one of the best programs available for frequency domain analysis.

### 1.1.1 What C<sub>I</sub>F<sub>E</sub>R<sup>®</sup> Encompasses

C<sub>I</sub>F<sub>E</sub>R<sup>®</sup> contains all the programs and tools necessary to convert time history data into frequency responses and use those responses to identify the system in question. This analysis includes the identification of single transfer functions, entire state-space systems, and various properties of responses such as bandwidth and crossover characteristics. Systems are not limited to flight vehicles, and can be as simple as a single-input-single-output mass-spring-damper setup or as complex as a multi-input-multi-output rotorcraft model.

The basic flow of the program begins with time history data. The data is generated using a frequency sweep maneuver in flight or simulation to excite the system over a wide range of frequencies. Frequency sweeps are generally characterized by a sinusoidal motion with a constant increase in frequency preceded and followed by a period of steady state flight as shown in Figure 1.2. The first step in C<sub>I</sub>F<sub>E</sub>R<sup>®</sup> is to transform this time data into frequency responses. More specifics on this step and all following steps will be discussed with more detail in Chapter 2. The data is divided into ‘time windows,’ which allow the algorithms to accurately extract both high and low frequency content from the data. The frequency responses are then calculated for each of the desired combination of inputs and outputs for each time window.

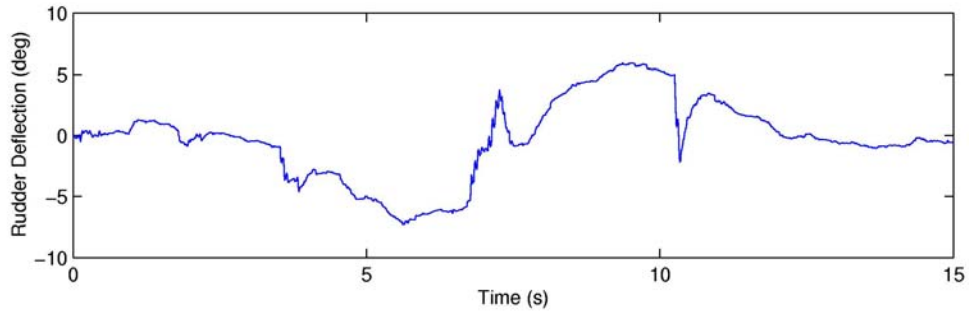


**Figure 1.2: Frequency Sweep Example (UAV Flight Data)**

In the case of a multiple input system, the frequency responses can be conditioned to remove the effects of correlation between different inputs. After this conditioning, the frequency responses from each separate window are combined to form a single frequency response based on the most accurate segments of each window. The combination is achieved by optimizing the data that has useful content in both low and high frequency regions. This is the last step in the frequency response generation process.

Less involved analysis, compared to full state-space model generation, can include calculations of RMS, cutoff frequencies, bandwidth, crossover characteristics, and data consistency checks using frequency response arithmetic. All of these tools can give useful insight to system behavior without generating more complex math models. CIFER<sup>®</sup> offers more complete identification through programs that will fit transfer functions to individual responses, and full state-space models to a series of responses. In addition to these powerful analysis tools, CIFER<sup>®</sup> offers utilities for plotting and organizing output, as well as managing the storage and organization of data.

One last important tool is the ability to validate state-space models against other time history data. These validation time histories are typically generated using a doublet maneuver as opposed to a sweep. A doublet is a short maneuver that moves through a range of motion for a control surface, beginning and ending in steady level flight as shown in Figure 1.3.

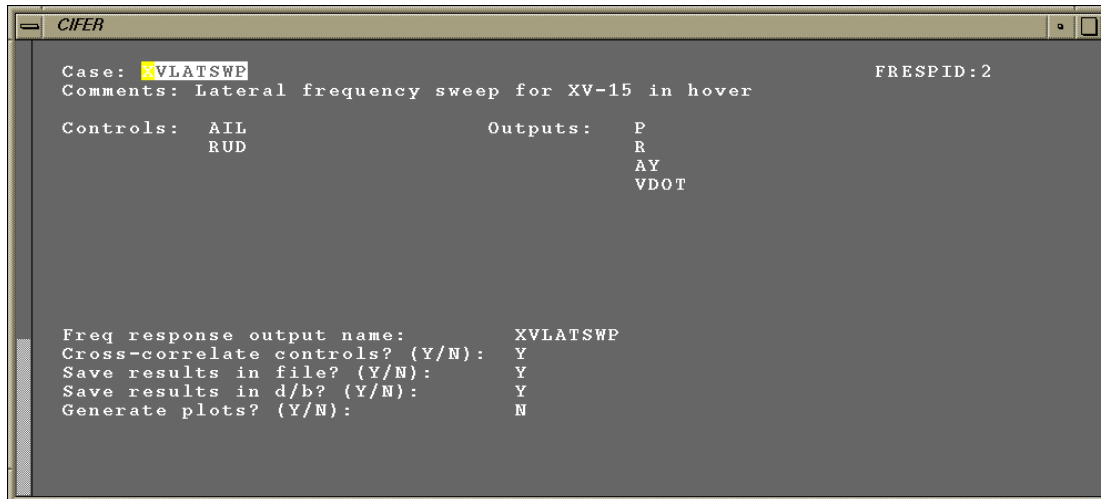


**Figure 1.3: Doublet Example (UAV Flight Data)**

### ***1.1.2 Ames Research Center Planning Meeting***

Late in the summer of 2003, a meeting within the Army/NASA Rotorcraft Division, Flight Controls Group at Ames Research Center was held to discuss the current status of CIPHER<sup>®</sup> and how it would be desirable to modify the program for future needs. This meeting was preceded by a request to industry users of CIPHER<sup>®</sup> for feedback on potential changes. The result of this activity was a summary of positive and negative aspects of CIPHER<sup>®</sup>, and a tentative plan for modernizing the program and addressing some of the negative issues. This section will detail some of the major points of the meeting that directly affected the course of this thesis.

In its current form CIPHER<sup>®</sup> is a collection of mathematically robust algorithms that have been tempered by 20 years of flight project application and experience. It can display canned results to a wide variety of formats including PostScript, X, Talaris and others. The textual interface was created in the 1980s in Curses format to run on Unix systems. It has since been ported to run on a Unix emulation environment for Windows and on Linux. An example of a CIPHER<sup>®</sup> screen is shown in Figure 1.4. The user would navigate through the screen using the arrow and function keys.



**Figure 1.4: Example CIFER<sup>®</sup> Screen**

Due to its long life and the constant work that goes into improving and updating CIFER<sup>®</sup>, it has many positive aspects that make it a premier frequency response analysis tool. The algorithms have been proven to work, providing quality results without program crashes. Much effort has been given to creating robust methods of analyzing time history data and creating frequency responses. CIFER<sup>®</sup> includes many tools and utilities to assist in the analysis of frequency responses once they have been created, such as frequency response arithmetic and bandwidth calculations. The Curses interface is consistent and linearly-driven, which helps ensure that users enter data correctly.

Unfortunately, many of the features that make CIFER<sup>®</sup> accurate and robust contribute to making it difficult or tedious to use at times. There is a steep learning curve to become familiar with the myriad of functionality the program offers. While the program is robust, it is not always instructive in alerting users to the nature of a problem; error messages can often scroll too quickly and are cleared from the screen before they can be read. The status of running batch jobs can be difficult to assess. Additionally, moving data from responses and plots into modern programs such as MATLAB or Igor can be very challenging.



One almost universal concern with the program is the interface. Modern engineers are growing less familiar with the Unix-based Curses interface and much of the user feedback requested an update to the interface. Even to experienced users the same linear interface that gives CIFER<sup>®</sup> its power and robustness can be a serious hamper; data fields must be retyped for new cases with limited cut and paste functionality. Another major concern is that there is no way to script the processes of CIFER<sup>®</sup>; setting up a series of multiple cases for an involved flight test can take days of repetitive data entry.

One conclusion of this meeting was that a modern graphical user interface (GUI) and a way to use command-line calls to script CIFER<sup>®</sup> processes should be developed. A thoughtfully laid out GUI should be able to retain all the robustness of its Curses counterpart while offering new features that make modern GUIs versatile such as browsing capability and easy navigation. It was determined that the work encompassed by this thesis would be a prototype or test bed for these capabilities that will result in modernized functionality for CIFER<sup>®</sup>.

## **1.2 Project Scope**

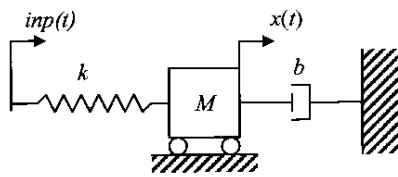
The goal of this project was primarily to develop a command-line interface and a modernized GUI layout for CIFER<sup>®</sup>. It was decided to use The Mathworks' MATLAB<sup>2</sup> as a medium for this development as many users of CIFER<sup>®</sup> also use MATLAB and a more developed communication between the two programs would be very useful. The ultimate goal of CIFER<sup>®</sup> modernization would be to make it independent of other programs so users do not have to purchase additional and potentially unnecessary software licenses to benefit from the upgrades.

Some CIPHER<sup>®</sup> users did express concern with the development of the MATLAB functionality because they were not also MATLAB users. This concern can be addressed by a compiler developed by The Mathworks that allows MATLAB files to be compiled and run independently of MATLAB itself. The MATLAB compiler that is available allows MATLAB M-files to be compiled into C code, which removes the necessity to have a MATLAB license to run the M-files. The capability for scripting and the modern GUI on which this thesis is based will be a significant benefit to CIPHER<sup>®</sup> users when it is fully developed. It was deemed acceptable to use MATLAB as a base to develop prototypes of these new interfaces as many companies already use it and the compiler addresses the concerns of those that do not.

CIPHER<sup>®</sup> is a large collection of programs and to develop command-line functions to mimic the entirety of its capability is well beyond the scope of a thesis project. It was agreed that functions to drive three of the major programs and several of the supporting analysis utilities would be a sufficient demonstration of a command-line interface. The goal is to create a function that can be called with a single-line command that will perform all of the data input and error checking of its equivalent CIPHER<sup>®</sup> counterpart. Development of the GUI was also limited in scope for the same reasons as for the command-line functions. The goal of the GUI is to show how a modern interface can enhance the functionality of CIPHER<sup>®</sup>. Thus only one program will be used as a demonstrator.

In order to further tie the project into aerospace applications, the code developed will be validated using real-world problems. In addition to checks verifying that results run in MATLAB are equivalent to those run from CIPHER<sup>®</sup>, the MATLAB functions will be used to aid in a NASA research project. Validations will include a simple mass-spring-damper system, data from XV-15 tilt rotor aircraft flight tests, UH-60 simulation data from a training course, and finally a project

involving handling qualities analysis of *Shadow 200*, a small reconnaissance UAV, all of which are shown in Figure 1.5 and Figure 1.6.



**Figure 1.5: Mass-Spring-Damper (Left), XV-15<sup>3</sup> (Right)**



**Figure 1.6: NASA Sikorsky UH-60 RASCAL<sup>3</sup> (Left), *Shadow 200* TUAV<sup>4</sup> (Right)**

A final important aspect of the project is that the code must be developed and structured such that developers at Ames Research Center can continue the work beyond the scope of this project with the final goal of distributing it to CIFER<sup>®</sup> users. This translates into creating the appropriate documentation and code structure to allow this project to be integrated into the existing CIFER<sup>®</sup> code structure efficiently.

## Chapter 2: System Identification using CIPHER<sup>®</sup>

This chapter primarily contains information found in the CIPHER<sup>®</sup> user's manual<sup>5</sup> and is intended to give readers a solid understanding of how CIPHER<sup>®</sup> generates a frequency response. This will help provide insight into the analysis of the validations described in Chapter 4.

CIPHER<sup>®</sup> produces both nonparametric and parametric models for systems in the frequency domain. The nonparametric frequency response should be considered the core of this thesis, however parametric modeling is still very relevant to CIPHER<sup>®</sup> as a whole. A frequency response is a complex-valued function that relates the Fourier Transform of system output to system input. The general form of a frequency response is shown in Equation 2.1. The frequency response is a full description of system dynamics, stable or unstable, that does not require assumptions of the system's structure.

$$H(f) = \frac{Y(f)}{X(f)} \quad [2.1]$$

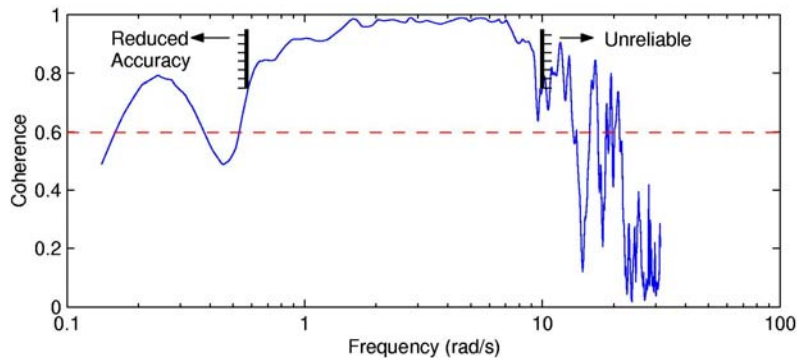
CIPHER<sup>®</sup> uses a version of the Fast Fourier Transform (FFT) known as the Chirp-Z Transform (CZT). This transform removes many of the restrictions placed on the discrete Fourier Transforms and thus is very flexible as an algorithm. Users have greater freedom to specify sample rates and resolution. The algorithm only runs on a specified frequency range, thus there are no wasted data points. The CZT algorithm generates three important values that represent the energy of the system as a function of frequency: input autospectrum ( $G_{xx}$ ), output autospectrum ( $G_{yy}$ ), and cross spectrum ( $G_{xy}$ ). The frequency response is then calculated using Equation 2.2, which is unbiased for output noise and biased for input noise.

$$H = \frac{G_{xy}}{G_{xx}} \quad [2.2]$$

One key feature of frequency response calculations is the coherence function, which is a measure of data accuracy or content. Coherence is determined by Equation 2.3, which will yield a value between 0 and 1.

$$\hat{\gamma}_{xy}^2 = \frac{|G_{xy}|^2}{|G_{xx}| |G_{yy}|} \quad [2.3]$$

Coherence represents the energy of the system or the fraction of output power that is linearly related to input power. When the system has high energy, or excitation, the coherence will be closer to 1 and the data content is considered to be more accurate. If the system does not have enough excitation or is low energy, the coherence will drop, indicating that the data content is poor. Data is also considered poor or unreliable if the coherence is rapidly changing as illustrated in Figure 2.1. Poor data can result from noise, gusts, or off-axis control activity. CIPHER<sup>®</sup> algorithms use coherence weighting to determine the frequency-ranges of a frequency response that have the most accurate data and only fit parametric models to these sections. It is generally accepted that frequency ranges with coherence equal or greater than 0.6 are considered usable if they are not rapidly changing.



**Figure 2.1: Example Coherence Plot**

CIPHER<sup>®</sup> contains six primary programs to conduct analysis: FRESPIID, MISOSA, COMPOSITE, NAVFIT, DERIVID, and VERIFY. In addition, there are three main analysis utilities that can calculate RMS values, bandwidth properties, and perform frequency response arithmetic. The

functionality of these programs and utilities will be discussed in the following sections along with a brief description of the linear screen interface that drives them. In addition to these tools there are a large number of other utilities for viewing results and organizing the CIPHER<sup>®</sup> database. The primary CIPHER<sup>®</sup> programs this project dealt with were the first three main programs, all three analysis utilities, and a small selection of other programs.

## **2.1 Creating a Frequency Response – FRESPID**

FRESPID (Frequency Response Identification) is the first step in any CIPHER<sup>®</sup> analysis. This program takes time history data and generates frequency responses based on the input and output channel measurements. A very important aspect of CIPHER<sup>®</sup> occurs in FRESPID, which is the “windowing” of the data. The data comprises a number of discrete frequency points over a time duration. FRESPID uses the CZT to average the data points of a smaller time window of data. It breaks the full time history into segments based on this window and performs computations based on averages for each given window size.

There is a distinct tradeoff in the selection of window lengths relative to the total time history length. Windows that are a smaller fraction of the total length provide a large number of averages that can more easily identify high-frequency responses by countering noise effects. Unfortunately, low-frequency identification degrades because low-frequency responses tend to occur in larger intervals than the small windows encompass. When the window size is enlarged, low-frequency identification becomes more accurate at a cost to the high-frequency end due to a decrease in the number of averages.

CIPHER<sup>®</sup> solves this trade-off issue by allowing the user to specify up to five different windows in FRESPID. A frequency response is generated based on each window. This ensures that both low

and high-frequency content is captured from the data. These preliminary responses are later combined together using COMPOSITE, which will be discussed below.

Of the three programs to be interfaced with MATLAB, FRESPIID is the most complex. There are a total of nine screens in the Curses interface that accept user input and one of the screens can lead to two sub-screens. The user must specify information about the time history data, build the controls and outputs using the measured data channels, determine which frequency responses to calculate, condition the data as desired, specify the windowing information, and set plotting and output options. All of this functionality is supported by robust error checking that must be maintained in the command-line interface. Figure 2.2 shows an example of the FRESPIID screen that controls the windowing of data.

```

CIFER
Window Parameters:                                FRESPIID:8
Final DT (seconds) 0.40000E-01

On? Window Id (20 chars)    TWIN      # In    # Out   Decim  Min Freq  Max Freq
(sec)      points    points  Ratio  (rad/sec) (rad/sec)
-----
A * 45 SECOND WINDOW      45.00000  1125   923     1      0.1396   31.4159
B * 40 SECOND WINDOW      39.96000   999   1049    2      0.1572   31.4159
C * 30 SECOND WINDOW      30.00000   750   274     1      0.2094   31.4159
D * 20 SECOND WINDOW      20.00000   500   524     1      0.3142   31.4159
E * 15 SECOND WINDOW      15.00000   375   137     1      0.4189   31.4159

Notes:
1. Window size guidelines (TWIN, secs) = # inputs * DT
   As a function of individual time history events:
     Enforced: (Window size) < or = {Smallest individual event}
     Recommended: (Window size) < or = {1/2 smallest individual event}
   As a function of total linked time history duration (all events):
     Min recommended: (Window size) < or = {1/3 linked record length}
     Optimum choice: (Window size) < or = {1/5 linked record length}
2. Min ID freq (for each window) = 2*pi/TWIN           Max ID freq = 2*pi/(5*DT)
3. The # inputs plus # outputs must equal a power of 2.

```

Figure 2.2: Example FRESPIID Screen

## 2.2 Multiple Input Analysis – MISOSA

Engineering problems involving flight vehicles tend to involve systems with multiple controls. During flight tests, frequency sweeps are performed for a single axis. However, there may be

secondary off-axis control inputs. This coupling has significant potential to distort the response identification of the system. CIPHER<sup>®</sup> addresses this issue with a program called MISOSA or Multi-Input / Single-Output Spectral Analysis.

MISOSA analysis on the user level is very straightforward. The primary control of interest is specified along with any other controls that are to be considered secondary. The program then performs spectral analysis to remove the effects of the secondary controls from the primary control response. It uses a matrix inversion of the input autospectrum at each frequency point to accomplish this. This general form is shown in Equation 2.4.  $G_{xx}$  is the matrix of auto and cross-spectra for the inputs and  $G_{xy}$  is a vector containing the cross-spectra for each control input and the single output in question.

$$T(f) = G_{xx}^{-1}(f)G_{xy}(f) \quad [2.4]$$

There are much fewer screens that drive MISOSA compared to FRESPID. The primary information needed is details about where the frequency responses are stored (whether in a file or in the database), names for the controls and outputs, and the desired combinations of responses to calculate. There is no conditioning or windowing of data involved, which results in less error checking.

### **2.3 Combining Windows – COMPOSITE**

Normally the optimization of window sizes in the FFT calculation would be a very time consuming process. For a four-input, nine-output system there would be thirty-six responses that would each have to be individually optimized. CIPHER<sup>®</sup> employs COMPOSITE (for composite windowing) to combine the individual windowed responses from FRESPID into a single combined response, thus automating the optimization. It uses a nonlinear, least-squares optimization of a cost function based on the auto and cross-spectra to combine the window data.



Thus the low-frequency content of larger windows is combined with the high-frequency content of smaller windows into a single response.

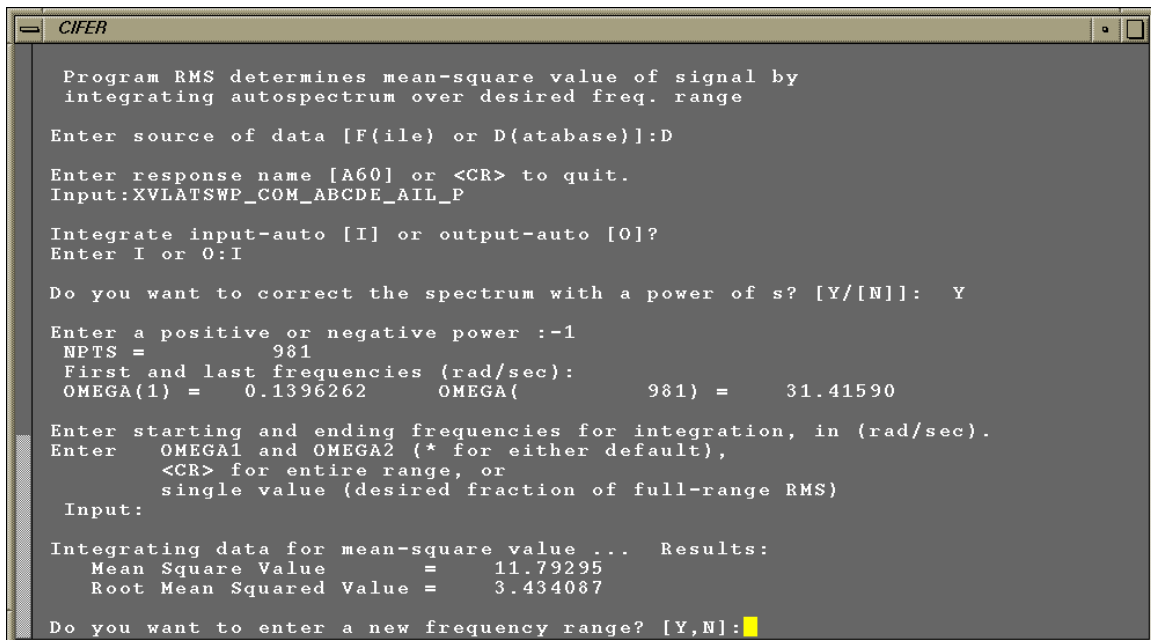
COMPOSITE is set up almost identically to MISOSA in terms of the screen interface. There are some very slight differences in specifying the sources of data, but otherwise the interfaces for the two programs gather the same information. Since the required input information for each of the three main programs is of similar nature, it was easy to standardize the general layout of the command-line interface created to drive them in MATLAB in terms of variable names, structure and error checking.

## **2.4 Analysis Utilities**

There are three primary analysis utilities included in CIPHER<sup>®</sup>. The first is the RMS program which is used to integrate the autospectrum over a desired frequency range to determine the mean-square value of a signal. In addition, it can locate a frequency where the integrated RMS is a specific fraction of the full-range value. This is useful for determining properties such as cutoff frequency.

The second utility allows both the calculation of handling qualities and crossover characteristics. The handling quality analysis looks for  $-180$ -degree,  $-135$ -degree, and 6dB bandwidth frequencies. It provides values for the gain and phase margins at these frequencies. Bandwidth is a useful indication of handling quality that can be identified from the nonparametric frequency response without first fitting a parametric model to the data. The crossover calculation examines the magnitude of the response for sign changes and is useful for determining broken-loop characteristics. The bandwidth utility also provides some useful plotting features that will allow users to include a least squares fit over a portion of the phase curve and to solve for phase delay.

The RMS and bandwidth utilities are driven by an interface different from that of the main programs described above. Rather than using screens, they use a linear prompt-based interface. Users are prompted for various data and results are then posted to the screen. These programs were more challenging to capture in a command-line format because the inputs had to be entered at one time and the outputs displayed at one time. Figure 2.3 shows an example of an RMS calculation and the prompts that drive it.



```
Program RMS determines mean-square value of signal by
integrating autospectrum over desired freq. range

Enter source of data [F(file) or D(atabase)]:D

Enter response name [A60] or <CR> to quit.
Input:XVLATSWP_COM_ABCDE_AIL_P

Integrate input-auto [I] or output-auto [O]?
Enter I or O:I

Do you want to correct the spectrum with a power of s? [Y/[N]]: Y

Enter a positive or negative power :-1
NPTS =          981
First and last frequencies (rad/sec):
OMEGA(1) =    0.1396262      OMEGA(          981) =    31.41590

Enter starting and ending frequencies for integration, in (rad/sec).
Enter OMEGA1 and OMEGA2 (* for either default),
      <CR> for entire range, or
      single value (desired fraction of full-range RMS)
Input:

Integrating data for mean-square value ... Results:
Mean Square Value      =    11.79295
Root Mean Squared Value =    3.434087

Do you want to enter a new frequency range? [Y,N]:
```

**Figure 2.3: Example RMS Prompts**

The last function performs frequency response arithmetic. Responses can be modified individually using scale factors and powers of  $s$  and then combined with basic arithmetic operations. The utility will perform these operations for magnitude, phase, coherence, and the auto and cross-spectra as desired. The option to perform this arithmetic allows users to check data

consistency by reconstructing responses for various signals based on kinematic laws of motion. Examples of this type of analysis can be found in Chapter 4.

The arithmetic function uses the screen interface from the main programs but only has two screens. Thus the conversion to command-line interface was reasonably straightforward.

## **2.5 Parametric Modeling – NAVFIT, DERIVID, VERIFY**

The three remaining main programs of CIPHER<sup>®</sup> will be covered in less detail as this project does not involve them. It is important to note their use as this constitutes a major portion of CIPHER<sup>®</sup>'s potential and the direction that future work on modernizing the interface will take.

NAVFIT allows users to fit a transfer function to a single frequency response. The interface uses a cost function to give the user an indication of how closely a particular fit matches the response data. The user specifies the order of the transfer function desired and then the program iterates to find an optimum fit.

The interface used by NAVFIT is similar to the RMS and bandwidth utilities except it has many more input options and more loops at certain segments. Additionally, it was built up on code that was originally created by McDonnell Douglas Aircraft outside of NASA. Thus, it would be particularly challenging to adapt to a command-line format. This was one major factor for not including NAVFIT within the scope of this project.

DERIVID constructs a state-space system based on a series of related frequency responses. It generates the appropriate control derivatives for the coefficient matrices of a state-space system. This is a very powerful ability as it enables the creation or validation of math models for

simulation and wind tunnel testing. DERIVID works in a fashion similar to NAVFIT, using a cost function to notify the user of the accuracy of the fit.

The final main program, VERIFY, allows users to validate a state-space model found in DERIVID with new time history data in the time domain. Typically the validation employs flight data taken from a doublet maneuver as opposed to a frequency sweep. VERIFY will run the state-space model using the time history inputs and compare the model outputs to the measured outputs. This is an important step to provide confidence in the state-space solution.

DERIVID and VERIFY use a combination of the screen interface and the prompt interface. There are 17 and 18 screens, respectively, that accept user input and a lengthy series of prompts that guide the user through the calculations of the state-space model. Due to this complexity they were excluded from the scope of the current feasibility project.

## **Chapter 3: Programming and Code Development**

The programming requirements set by the Army for this project included the development of a command-line interface for three main CIPHER<sup>®</sup> programs: FRESPID, MISOSA, and COMPOSITE, and an additional seven utilities: RMS, Bandwidth, Frequency Response Arithmetic, plotting, case listing, data storage and data retrieval. These programs constitute a “light” version of CIPHER<sup>®</sup> that takes the user through all the frequency response conditioning and allows assorted manipulation and analysis of the responses. The goal was to generate functional building blocks that had stand alone capability and upon which further development could take place.

In addition to the command-line interface, development of a GUI interface was also required. The goal of the GUI development was to create a feasibility study to show how advances in GUIs could improve on the existing Curses interface. Thus, the development was focused on one main program, COMPOSITE. The primary concern was the layout and interface, thus once one program was shown to work as a GUI, others could be adapted fairly quickly. This chapter discusses the process of development of both new interfaces for CIPHER<sup>®</sup> and the major problems encountered during the development process.

### **3.1 Command-Line Development**

A function in MATLAB accepts a list of inputs, returns a list of outputs, and typically is called from a single line, often referred to as “command-line.” The key advantage of creating such an interface for CIPHER<sup>®</sup> is the ability to script multiple calls to the function. The scripting ability greatly reduces the amount of time needed to create and run the large numbers of CIPHER<sup>®</sup> cases typical of any detailed analysis. The error checking associated with the data entry process for

CIFER<sup>®</sup> inputs from the old interface, which was split up over a long series of screens, provided significant challenge. The command line function had to emulate all of that error checking structure at one time. Appendix A contains example screen shots of all the screens for the COMPOSITE program within CIFER<sup>®</sup>. Each screen is navigated through using the keyboard, and error checking occurs when the user progresses from one screen to another.

### ***3.1.1 Development Process***

The first step in the development process was to determine the best method to get information from the CIFER<sup>®</sup> database into the MATLAB workspace. The solution lay in MATLAB's ability to create "mex" functions that can interface with Fortran or C code. CIFER<sup>®</sup>'s original Fortran code is structured such that it has internal functions and subroutines to access important information. By making the appropriate calls in the mex code, these functions were successful in passing data from the CIFER<sup>®</sup> Fortran into the MATLAB workspace and vice versa. The mex functions became the building blocks of all the CIFER<sup>®</sup>-MATLAB interface code.

The concept of using internal CIFER<sup>®</sup> functions with mex code was initially tested with some very basic functions designed purely to retrieve a few specific pieces of data from CIFER<sup>®</sup>, such as a frequency response name and description, and display them in MATLAB. Modifications were made to the data in MATLAB and fed back into CIFER<sup>®</sup>. Accessing the frequency response in CIFER<sup>®</sup> verified whether or not the change was successful.

When data is first processed in CIFER<sup>®</sup> it becomes known as a 'case.' which encompasses a series of measurements from a system. For example, one of the sample cases included with CIFER<sup>®</sup> is called XVLATSWP, which is for lateral responses from the XV-15 research vehicle in hover. The case structure has all the information that dictates where data is located, how to

interpret it, how to condition it, and how to window it. It is this information that must be passed into MATLAB in order to facilitate running a case from a command-line interface.

The mex building blocks were expanded upon to read and write the full gamut of information CIFER<sup>®</sup> uses to create and run its cases. The problem with these building blocks is that they have many required inputs or outputs (depending on read or write) and no inherent error checking. This made use of the initial code blocks themselves very difficult. It was necessary to create higher level MATLAB M-files that would call the mex building blocks. These functions were designed with the full range of error checking found in the CIFER<sup>®</sup> screens and could format input data to be fed into the mex code blocks.

Designing the final interface for the M-files was an iterative process in which several Ames engineers and programmers were included. Initial concepts were presented to the engineers on paper and the layout was refined based on their critiques. When those providing input were largely satisfied, the design was implemented in code. Once the M-files were finished they were distributed back to the engineers for evaluation. The evaluation process was constantly in place for the duration of the code development phase of the project.

The final result was the desired series of command-line functions that could successfully mimic their CIFER<sup>®</sup> counterparts. The various codes encompassed 49 functions and spanned 14,000 lines of code. Extensive testing on the part of Ames engineers and the developer facilitated more robust code than might otherwise have been achieved. One example of the success of the code was from one engineer running a series of CIFER<sup>®</sup> cases in a few hours that might have otherwise taken two to three days to finish. Suffice it to say that the engineers who regularly interact with CIFER<sup>®</sup> were very enthusiastic about this new capability.

### ***3.1.2 Problems Encountered and Solutions***

The first, and perhaps most fundamental, problem encountered while developing this code was the structure of the interface itself. Creating an interface is difficult because no one knows exactly what is desired until they see and interact with it. Additionally, no solution is necessarily the right or best solution. On one hand, there was a significant drive to have the interface of new code mimic the old CIFER<sup>®</sup> interface. On the other hand, it made equal sense to update older CIFER<sup>®</sup> methods to more modern implementation, and to make more drastic changes to the layout. This question was more an issue for the GUI, but still affected the command-line interface.

One method that MATLAB uses to gather large amounts of input is through a series of name-value pairs. Essentially, the name of a variable is given as input immediately followed by its value. The pros are that the lists of variables do not have to be provided in a particular order, and variables meant to retain default values need not be specified. The con is that inputs are twice as long as they might otherwise need to be. Figure 3.1 shows a simple example of name-value input to a fictional function. Another possible input method is to use a structure. Structures are a ‘parent’ data type in which various other data types can be stored. Thus one structure might contain integers, arrays, and strings organized in the ‘fields’ of the structure. Structures are reasonably well organized, and easy to deal with from a programming standpoint; however the format can be intimidating to non-programmers. Figure 3.1 also shows a simple structure layout, and how one structure used as input could replace name-value pairs.



```
>> myfun('Name1', 'Example', 'Name2', 4.2, 'Name3', 42)

>> structure_example

structure_example =

    ex_string: 'Example'
    ex_real: 4.2000
    ex_int: 42

>> myfun(structure_example)
```

**Figure 3.1: Name-value Pairs and Structures**

The final interface for the command-line code combined both name-value pairs with the structure data type. The structure was used in the background to track and store all the information from a CIFER<sup>®</sup> case. The functions accepted input as either a series of name-value pairs or a structure. Output had to be presented as a structure; any other method would have produced too much clutter in the MATLAB workspace, especially when multiple CIFER<sup>®</sup> cases were considered. The use of the structure helped streamline the internal error checking process. Essentially, for each field in the structure, there was a series of checks run that mimicked the checks run in between each CIFER<sup>®</sup> screen.

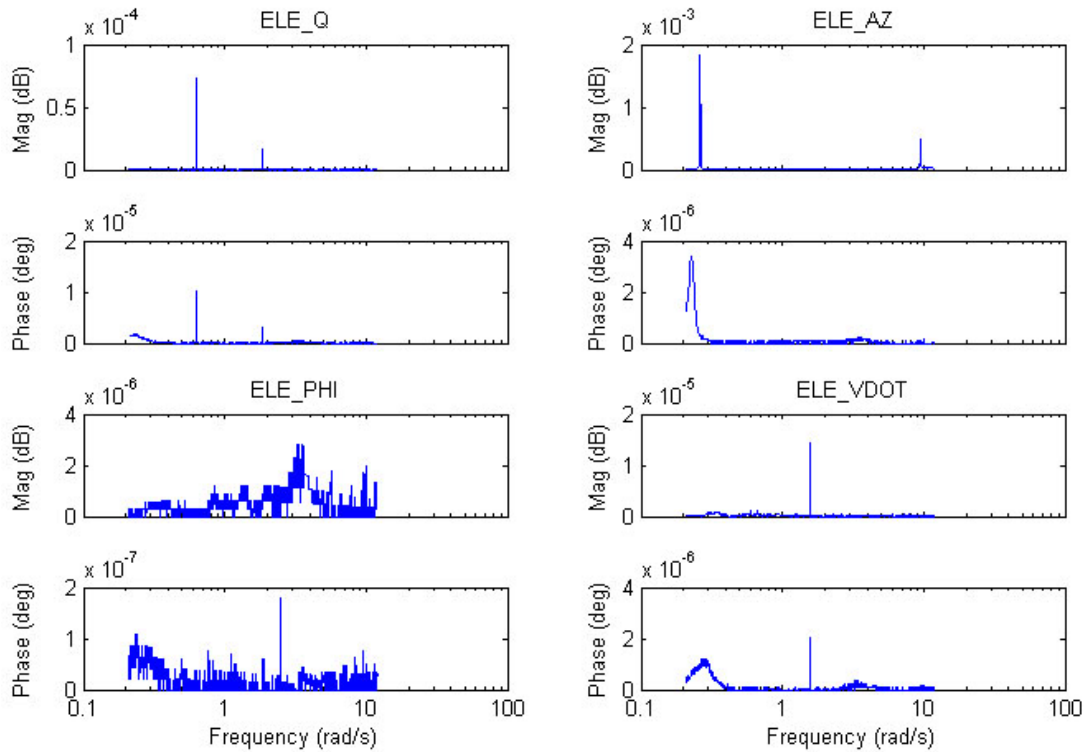
#### 3.1.2.1 Precision Errors

During the earlier phases of development a small error caused by the transfer of data between CIFER<sup>®</sup> and MATLAB was discovered. The cause of this error was likely numeric precision discrepancies, probably due to MATLAB using double precision compared to Fortran's usual single precision. The result of these errors was that the same number stored in the CIFER<sup>®</sup> database would return to MATLAB with variation in the ten or hundred thousandths decimal

place. To date no major discrepancy has been observed in the overall results generated by CIPHER<sup>®</sup>.

Several tests were conducted to locate a more exact cause of the error. The first method was to create a case in MATLAB using the interface functions and then create a second identical case (except in name) the traditional way from within CIPHER<sup>®</sup>. From here the frequency response results could be extracted and compared. This was how the error was initially discovered.

Comparisons of the errors in magnitude and phase were plotted as shown in Figure 3.2, which revealed that slightly larger errors occurred at high or low frequency, regions that tend to exhibit low coherence. The trend was not completely true in all cases, but most plots tended to exhibit that behavior. Several different cases including the XV-15 data, *Shadow 200* data (Figure 3.2), UH-60 simulation data, and an example mass-spring-damper system were examined in this manner, with the results generally being the same. Figure 3.2 is a plot of the percent error; as such, it can be seen that these errors are small in magnitude.



**Figure 3.2: Percent Error Comparisons**

In order to determine if the error was originating in MATLAB, another set of tests was conducted. First, a case was set up in CIPHER<sup>®</sup> and run (using different case names) in CIPHER<sup>®</sup> and in MATLAB. Then, an identical case was constructed in MATLAB and again run in CIPHER<sup>®</sup> and MATLAB. The results compared identically between either set of cases created using the same program. However, comparison of any combination of cases from differing setup methods resulted in the minuscule numeric error. When overlaid in a plot, there was no visual difference in the plots and thus no significant discrepancy between the two methods of creating a case. The origin of the error was not located, but suspected to result from MATLAB's use of double precision.

The conclusion was that, unless time permitted, or more significant discrepancy between results was found, the problem would be considered minor. For the most part, the errors occurred in

regions of low coherence where less weight would be assigned to the results, regardless of small numeric error. The short-term solution was to document the existence of the issue as a warning to users.

### *3.1.2.2 Retaining Structure of Code*

A major concern of the project was to create new code that would be relatively easy to maintain and modify. In order for the functions created for the MATLAB-CIFER<sup>®</sup> interface to be maintainable by CIFER<sup>®</sup> lead programmers, the code was written to mimic existing Fortran code wherever possible. In other cases, such as the MATLAB M-files, code was made uniform both in structure and in naming conventions so as to facilitate both ease of use and ease of maintenance.

Several specific methods were employed in this endeavor. First, in dealing with the Fortran mex files, the names of variables transferring data from MATLAB into the CIFER<sup>®</sup> common blocks were made identical to the common block variables but with an 'x' appended on the end. This notation would allow easy identification of variables by programmers already familiar with the old Fortran code. Second, the variables were renamed when they were passed into the MATLAB workspace because the Fortran variables have names suited to programming that may not be meaningful to an engineering user. Last, the various Makefiles which create the Fortran mex files were condensed and grouped according to the utilities and programs they created. Thus, all the functions that facilitate FRESPIID use are created using a single Makefile.

CIFER<sup>®</sup> programs and utilities vary slightly in their method of interfacing with the user, as described in Chapter 2. Some use a screen interface where the user fills in fields and advances using the function keys. When the end of the screens is reached, the screen programs may call additional programs that send information to be processed by CIFER<sup>®</sup>. Other programs use a

command-line-based interface that runs calculations and returns results as the user steps through via prompting.

The three main programs for this project, FRESPID, MISOSA, and COMPOSITE, when run in CIFER<sup>®</sup> use the screen interface and have separate functions to store, retrieve, and compute information. The MATLAB interface uses these same functions, however, to provide CIFER<sup>®</sup> the information without using the screen interface. The MATLAB structure that holds the case information is set up to collect the same information that the user would normally provide through the screens. All of this functionality was accomplished without modifying any original CIFER<sup>®</sup> code, thus no provisions will be necessary in order to maintain that original code when the new interface is added to the full CIFER<sup>®</sup> software package.

Other utilities such as the RMS, plotting, and arithmetic calculations have the screen interface embedded within the functions that run the calculations. For the MATLAB interface to work, the original CIFER<sup>®</sup> functions had to be altered to disable the screen interface. These alterations are well marked with comments as CIFER<sup>®</sup> lead programmers will ultimately need to incorporate new methods within existing code to accommodate the changes and prevent the need to maintain two separate source files. The utilities that use the command-line interface (RMS, plotting, and handling qualities) were dealt with much the same as the screen-based utilities due to the command-line prompts being embedded within the code for calculations.

### ***3.1.3 Complexity of Use***

A side effect of the command-line interface was that it required users to be reasonably familiar with CIFER<sup>®</sup>. The interface is not as intuitive as a graphical interface because everything happens at once. There are no screens with helpful notes and error checking to step through. If a case has

errors, they are all displayed at once when the program finishes. Thus, a new CIFER<sup>®</sup> user could get more lost and confused by starting off with the command-line interface. The power of the interface, as mentioned before, is the ability to script multiple cases, which is something beginning users would find less beneficial than experienced users.

A detailed help document was created for the command-line interface to help users understand how to employ it. The text of this document can be found in Appendix B, and Appendices C through I contain the appendices from the original document. Ames engineers were provided copies of the document along with the software when they employed the new interface in their projects. The document was written with the assumption that a reader would already be familiar with CIFER<sup>®</sup>.

### **3.2 Graphical User Interface Development**

COMPOSITE was selected as the test program around which to develop a MATLAB GUI. It was deemed sufficiently complex to make a useful example, and it is one of the more commonly used CIFER<sup>®</sup> programs from those selected for this project. The primary goal of the GUI development was to show that, first, it could be done, and second, the process of data entry could be enhanced using more modern GUI tools. An important secondary goal was to create the GUI as a rough template that could be efficiently applied to the rest of the CIFER programs in the future. The GUI was written to work with the command-line interface, using graphics to gather the information to be sent to the command-line functions without the user needing to know how the command-line functions work. It is important to note that the GUI programming does not stand alone as the command-line does and is to demonstrate how advances in GUI programming can make CIFER<sup>®</sup> more user friendly.

### ***3.2.1 Development Process***

The most difficult aspect of developing the GUI was adjusting the look and feel to appeal to the widest audience of potential users. The iterative method used for the command-line development was employed for the GUI as well, with NASA engineers and programmers included during the design process. Several initial concepts were sketched on paper before any code was written.

When ideas were put into code, the first programs were designed only around the layout of the GUI, lacking any real functionality. This allowed reviewers a clear idea of what the finished product might look like. Once the design for the layout was sufficiently refined, the functionality was written into the code. As the functionality was based on the command-line interface, the focus of the GUI development was the layout.

The GUI was largely built up around the command-line code that already could interface with CIPHER<sup>®</sup>. Data entry in the GUI was largely handled by toggles and text fields. The error checking from the command line was superseded by similar checks within the GUI to make its use closely resemble the old CIPHER<sup>®</sup> interface. Appendix A shows the COMPOSITE screens from both the old interface and the new GUI.

### ***3.2.2 Modern Updates to the Original Interface***

Perhaps the most visible change between the original interface and the new MATLAB GUI was the added navigational features. CIPHER<sup>®</sup> was originally designed to run on VAX/VMS systems and as a result, uses the keyboard function keys for navigation. There were menus that could be accessed for navigation within each program; however they are not always intuitive in use. Thus two primary navigation bars were added to the GUI interface. The left bar, found in Appendix A,

Figures A6 through A10, offers users a visual description of each screen in the COMPOSITE program and allows users to access any of those screens with a mouse click. The bottom menu, displayed on the bottom of Figure 3.3, is a re-creation of the original CIFER<sup>®</sup> menu, shown on top in Figure 3.3, and offers quick access to simple navigation and save or exit options.



**Figure 3.3: Comparison of Navigation Menus**

Aside from the new navigation, the largest change to the first screen, Figure A1 and Figure A6, is the addition of a browse feature. This button opens the first window shown in Figure A11 and allows the users to browse the database for CIFER<sup>®</sup> cases. When a case is selected from the list, its description is provided to aid in selecting the desired case. The case name can then be loaded into the text field on screen one.

Screen 2, Figure A2 and Figure A7, originally completely consisted of text field entries. Many of these fields corresponded to data that could only be one of two choices – yes or no, for example. The MATLAB GUI uses toggle buttons to simplify this process and reduce the amount of error checking necessary. Additional browsing capacity has been added to help data entry; inputs and outputs can be selected from a list of all inputs or outputs in that case. For example, a user might load a case into COMPOSITE and not remember which inputs and outputs were used in previous FRESPIID cases. The “input” and “output” buttons (Figure A7) call up a new screen, the second window in Figure A11, that displays all the inputs or outputs that exist in a particular case by querying previous MISOSA or FRESPIID cases. The desired inputs or outputs can be selected and then loaded back into screen 2.



The third screen is used to match which input and output pairs the program will operate on. Originally these pairs were selected through an asterisk, as shown in Figure A3. The new interface, Figure A8, is toggle button-based, and more features have been added to allow an entire row or column to be selected by clicking on the heading button. Similarly, toggle buttons were used for screen 4 as shown in Figure A4 and Figure A9.

The last screen in the original code offers three choices to exit, save and exit, or save and run the batch job. If a batch job is run, the output information is displayed to screen, shown in Figure A5, but users can press a key to continue using CIFER<sup>®</sup>, which will remove the information from the screen. Once the batch job is completed, additional information is displayed which can disrupt work if one has moved on while the batch job was running.

The final screen in the new interface, Figure A10, has the three options from the original code as well as several new features. There is now a dedicated window to display the output from the batch job, ensuring that the information will not be lost. Additionally, an option to view the output log file has been added. The log file contains a summary of the information from processing the case and is very useful for debugging a case that generated errors.

### ***3.2.3 Problems Encountered and Solutions***

The most challenging aspect of the GUI development was creating a layout that the largest number of engineers were comfortable with. It was ultimately made a requirement for the new layout to mimic the old interface as closely as possible. Before this requirement was set, several layouts were considered that would have been a significant change to the old look and feel. The driving factor was to keep the interface similar so longtime users would not have to make major

adjustments to their understanding of the program, while adding the modern GUI features that might shorten the learning curve for new users.

The second major challenge lay in generalizing the code structure to allow other programmers to easily adapt the code to work for other CIFER<sup>®</sup> programs, which was accomplished using the aforementioned added navigation functionality. These navigation tools were set up to work for a general series of windows. For this project, these were for COMPOSITE, but if windows for another program were created, they could be easily linked. This concept was illustrated by the lead programmer for CIFER<sup>®</sup> at NASA, who was able to adapt the code to the MISOSA program in a few days as opposed to the initial development, which spanned several weeks.

The development of the GUI was a significantly smaller undertaking than the development of the command line. The GUI largely added to and enhanced the already present functionality of the command line. Thus there were fewer technical problems associated with development. The practices set in place from the work on the command-line interface continued to be employed for the GUI development.

## Chapter 4: Validation and Application

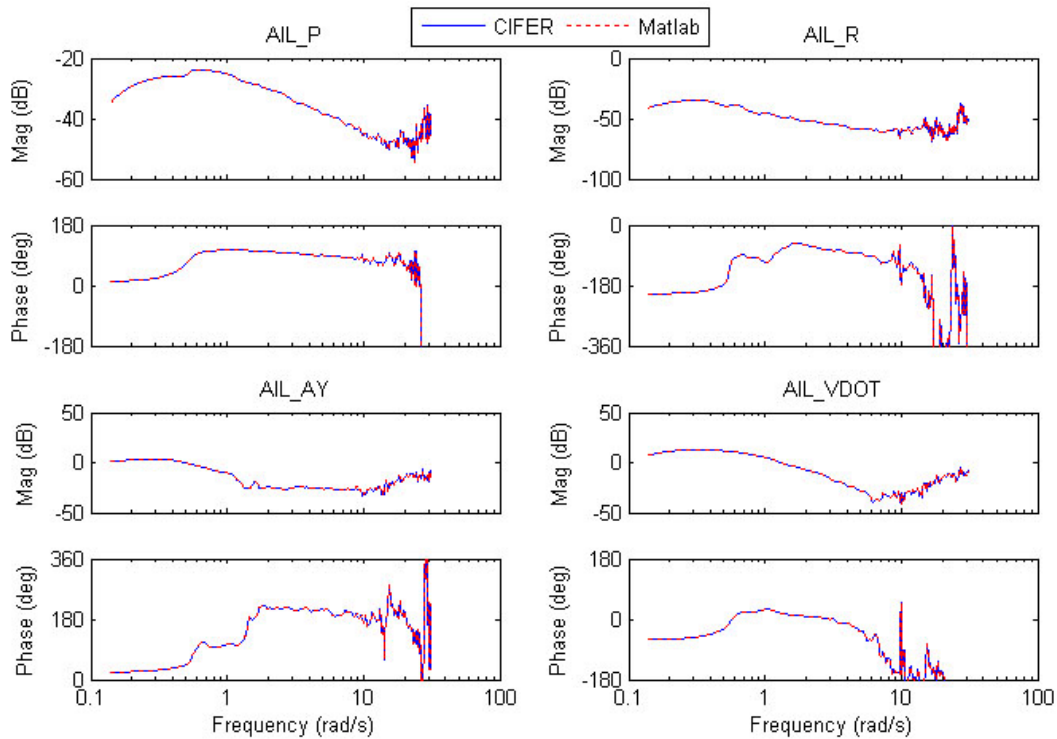
In order to confirm that the new code would accurately move information between CIPHER<sup>®</sup> and MATLAB, many tests were conducted. In the early stages of the code development, various sample cases provided with CIPHER<sup>®</sup> installations, such as the XVLATSWP case mentioned previously, were used. Also at this stage, a simple mass-spring-damper system was modeled as a potential example to new users of CIPHER<sup>®</sup>. Once the codes were more developed, more complicated validations were employed. The first was a closed-loop investigation of a UH-60 simulation, and the second was a series of data consistency checks and cutoff frequency analysis on the *Shadow 200* Tactical Unmanned Aerial Vehicle (TUAV)<sup>6</sup>.

### 4.1 Sample CIPHER Cases

CIPHER<sup>®</sup> installations come with a series of time histories for the XV-15, and its documentation<sup>5</sup> uses related sample cases as examples. New users of CIPHER<sup>®</sup> typically learn the interface by working through the set-up of these cases. Thus, the cases made excellent initial comparisons to determine if the MATLAB interface was working correctly. At this stage, the analysis was essentially creating plots of cases set up and run from CIPHER<sup>®</sup> and cases set up and run from MATLAB. The only major problem found was the numeric precision error discussed in Chapter 3. Examples of the scripts used to set up one example case can be found in Appendix I.

A series of example comparisons for the lateral sweep are shown in Figure 4.1. The aileron input is plotted against roll rate, yaw rate, lateral acceleration, and vertical velocity perturbation. The important aspect of the figure is that the plots comparing the results set up and run from MATLAB, to those set up and run from CIPHER<sup>®</sup> match. Additional plots and analysis for the XV-

15 cases have not been included as the analysis is largely the subject matter of the CIFER<sup>®</sup> documentation.

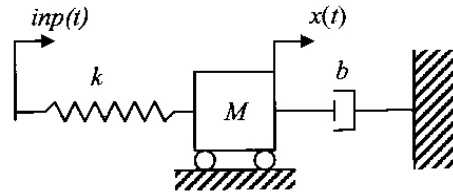


**Figure 4.1: XVLATSWP Validation Examples**

## 4.2 Mass-Spring-Damper System

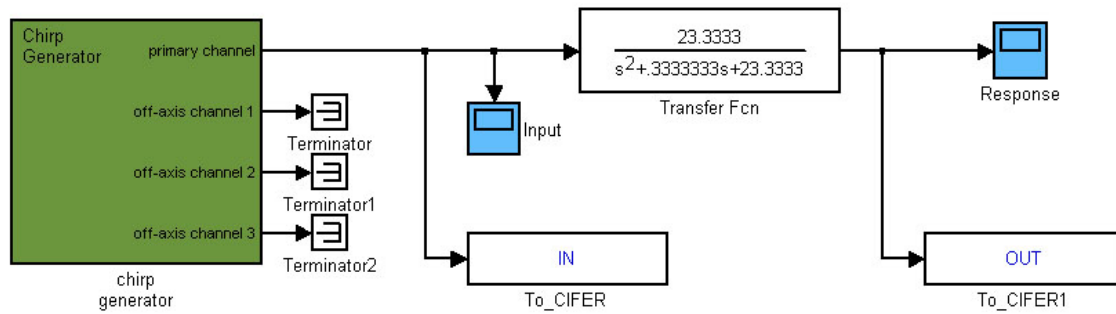
A second elementary check on the MATLAB interface was conducted using a single-input-single-output (SISO) mass-spring-damper system as an example. The primary goal for the example was to provide students with a simple system to analyze using CIFER<sup>®</sup>. The system used is shown in Figure 4.2, and its transfer function was solved from the governing differential equations in the form of Equation 4.1.

$$\frac{OUT(s)}{IN(s)} = \frac{k}{Ms^2 + bs + k} \quad [4.1]$$



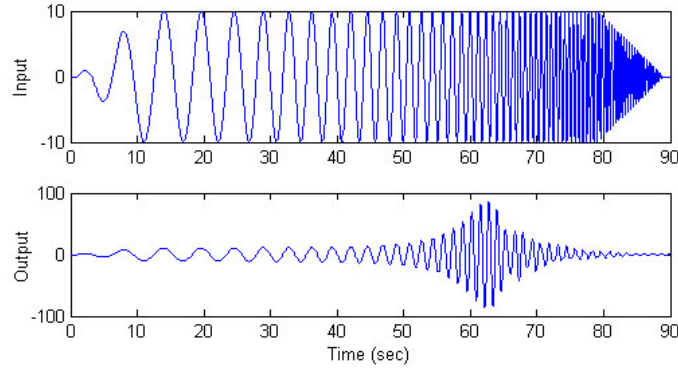
**Figure 4.2: SISO Mass-Spring-Damper System**

The system was modeled in Simulink by attaching a custom chirp signal generator to the transfer function as shown in Figure 4.3. The chirp generator was written to provide a frequency sweep as input. The spring constant  $K$  was set to 7, and variables  $M$ , and  $b$  set to 0.3 and 0.1, respectively. These values correspond to a natural frequency of 4.83 rad/s and a damping of 0.035.



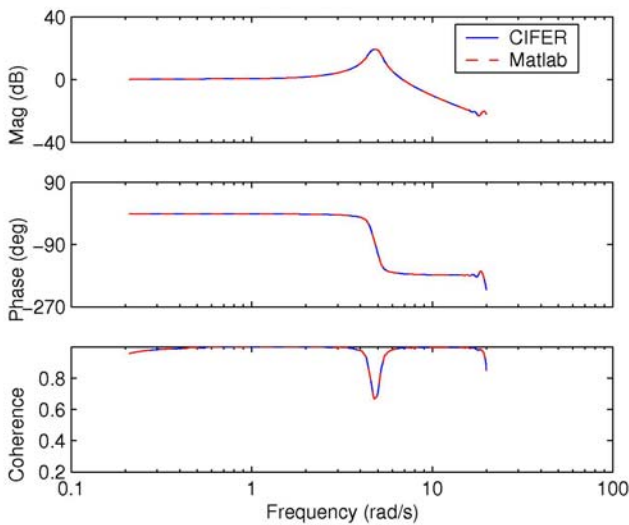
**Figure 4.3: Mass-Spring-Damper Simulink Block Diagram**

The results of the simulation, Figure 4.4 below, were collected and formatted into a file readable by CIFER<sup>®</sup>. The time history then made the basis for new CIFER<sup>®</sup> case that was created entirely in MATLAB. The script that set up the case can be found in Appendix H.

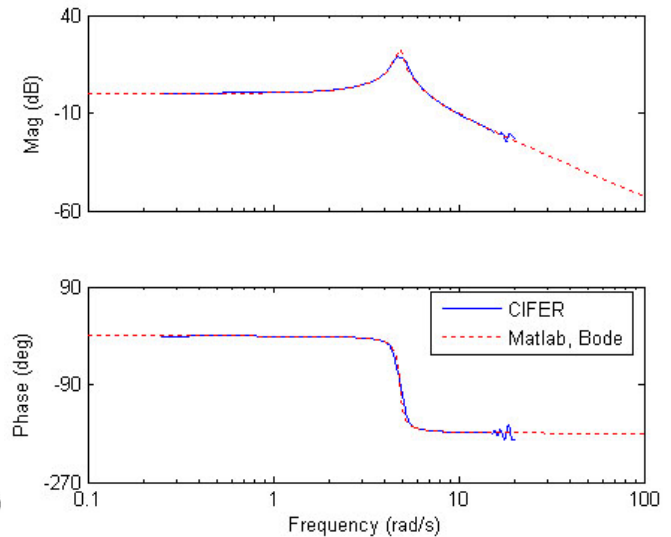


**Figure 4.4: Mass-Spring-Damper System Input and Output**

The frequency response of the simulation is shown in Figure 4.5 below. The natural frequency of the mode occurs is almost 5 rad/s, which corresponds to the calculated value of 4.83 rad/s. The case was also run from within CIPHER<sup>®</sup> for comparison to the MATLAB case, the results of which are also shown in Figure 4.5, where the results overlay precisely. In addition, the CIPHER<sup>®</sup> results were compared to the results from using the MATLAB ‘bode’ command on the transfer function as shown in Figure 4.6. There is a slight difference found at the mode and at high frequency, which correlates to the drop in coherence at those regions.



**Figure 4.5: Matlab to CIPHER<sup>®</sup> Comparison**



**Figure 4.6: CIPHER<sup>®</sup> to ‘bode’ Comparison**

### 4.3 UH-60 Simulation

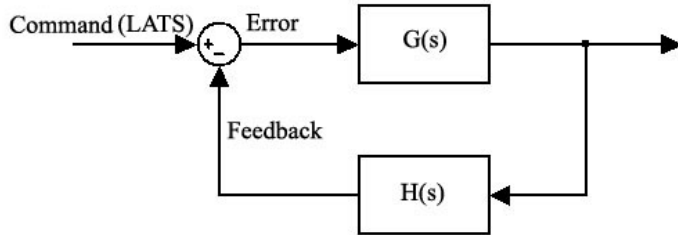
The first in-depth validation of the CIPHER<sup>®</sup>-MATLAB interface was based on UH-60 simulation data provided in the Combined-CIPHER-CONDUIT-RIPTIDE Training course<sup>6</sup>. This course was designed to give engineers a brief introduction to the three programs developed and distributed by the Flight Controls Group, of which CIPHER<sup>®</sup> is one. The second program is CONDUIT<sup>®7</sup>, which is designed to optimize a control system around a parametric model for a system. The third program is RIPTIDE<sup>®8</sup>, which is a simulation program that will allow users to fly systems modeled in CONDUIT<sup>®</sup>. Together, the programs constitute a very powerful control systems design suite.

The course data was examined for crossover and bandwidth characteristics using CIPHER<sup>®</sup> utilities. The reference values for these properties were already provided from CONDUIT<sup>®</sup> analysis and were used as a check to ensure the correctness of the CIPHER<sup>®</sup> results. The course manual provided closed-loop data necessary for bandwidth analysis. In order to investigate crossover characteristics it was necessary to generate additional simulated flight recordings, in RIPTIDE<sup>®</sup>, of the feedback and error channels. Only the roll channel was examined for this analysis.

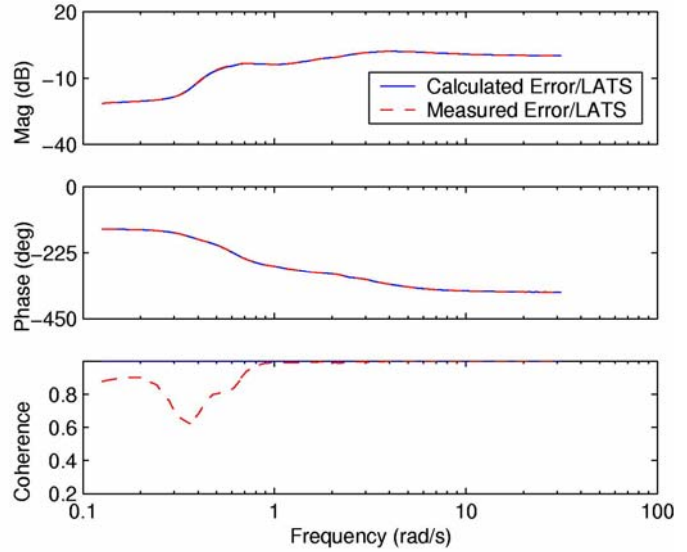
As a first step in the analysis, frequency response arithmetic (CIPHER<sup>®</sup> utility 9) was used to confirm the consistency of data channels used for error and feedback. Equations 4.2 and 4.3 show the relation of the error signal to the input and feedback signals for a conventional feedback setup as depicted in Figure 4.7. Figure 4.8 shows the plot of the error response to stick input compared to the response solved with the MATLAB version of CIPHER<sup>®</sup> utility 9 using Equation 4.3. The results overlay precisely, which was expected given that the data comes from a simulation.

$$e = \delta - f \quad [4.2]$$

$$\frac{e}{\delta} = \frac{\delta}{\delta} - \frac{f}{\delta} = 1 - \frac{f}{\delta} \quad [4.3]$$



**Figure 4.7: Feedback Block Diagram**



**Figure 4.8: Error Channel Verification**

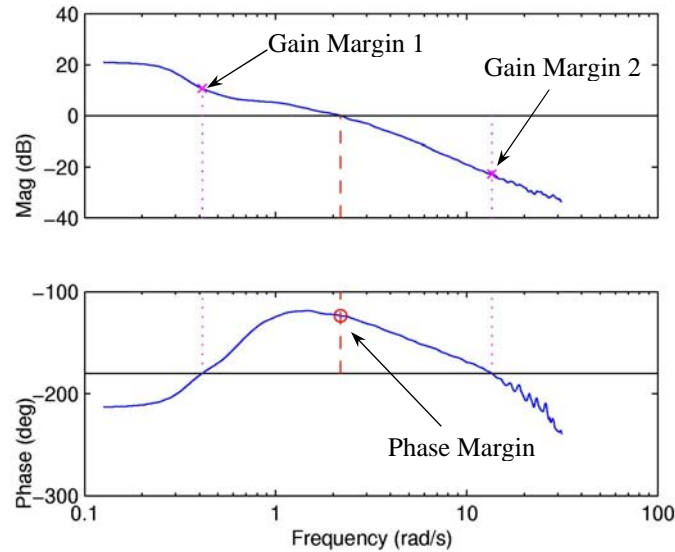
Using the MATLAB version of CIPHER<sup>®</sup>'s utility 8 for crossover characteristics, the feedback response to error in the lateral axis was analyzed. A time delay of 0.0402 was incorporated into the CIPHER crossover analysis because the time delay was not present in the simulated flight recordings. This modification is not entirely correct because it puts the time delay over the entire system as opposed to just within the forward loop where it actually occurs. It is, however, a reasonable approximation. Table 4.1 shows the results from CIPHER<sup>®</sup> as compared in Unix and from Matlab to those printed in the training course manual. There was no discrepancy between the two difference CIPHER<sup>®</sup> calculations.

**Table 4.1: Roll Gain/Phase Margin Results**

	CIPHER, From Unix		CIPHER, From MATLAB		CONDUIT		% Difference	
	Margin	Frequency	Margin	Frequency	Margin	Frequency	Margin	Frequency
Gain (1)	-10.67	0.41	-10.67	0.41	-10.3	0.48	3.57	13.18
Gain (2)	22.84	13.54	22.84	13.54	24.42	14.85	6.47	8.82
Phase	56.35	2.19	56.35	2.19	57.2	2.13	1.49	2.64



The differences between the two calculations are very small with the exception of the first gain margin frequency. The difference in frequency could be a result of the time delay mentioned above, however this would have more effect at higher frequencies. It is reasonable that the results do not precisely match as they were obtained by different analytical methods. Figure 4.9 contains a plot of the response with the gain and phase margins called out.



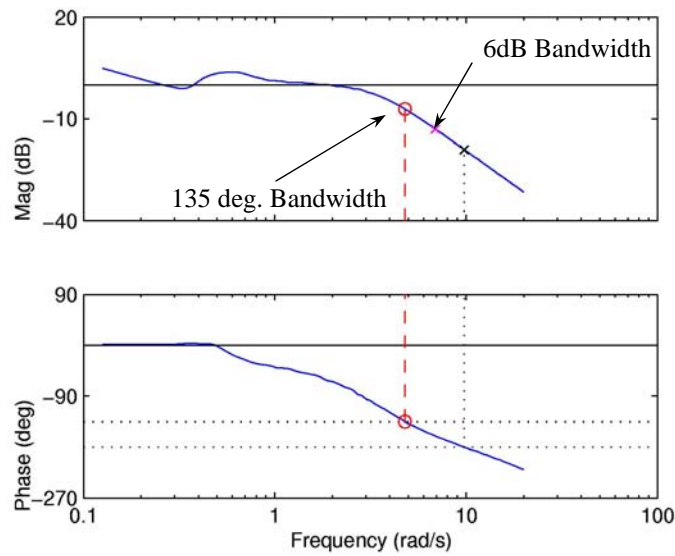
**Figure 4.9: Broken Loop Roll Gain/Phase Margin Results**

In addition to the broken loop gain and phase margins, CIPHER<sup>®</sup> can also determine bandwidth values from frequency response data. Using both the MATLAB and Unix version of CIPHER<sup>®</sup> utility 8 for bandwidth calculations, the roll attitude response to lateral stick was examined and compared to results obtained from CONDUIT<sup>®</sup>. These comparisons are shown in Table 4.2, and the difference between the two programs is very small. There was no difference between the MATLAB and Unix runs of CIPHER<sup>®</sup>. The ability to analyze nonparametric frequency responses is a useful feature of CIPHER<sup>®</sup> as it allows handling qualities of a system to be determined without identification of the full parametric math model.

**Table 4.2: Roll Bandwidth Results**

	CIFER From Unix	CIFER From MATLAB	CONDUIT	% Diff
45 deg Phase Margin	4.79	4.79	4.77	0.42
6 db Gain Margin	6.89	6.89	7.05	2.27

The response for lateral input to roll rate output was used to generate the data from Table 4.2. First it was integrated to yield the attitude response and then the time delay of 0.0402 was applied. The graph of this response with the bandwidths marked is shown in Figure 4.10.

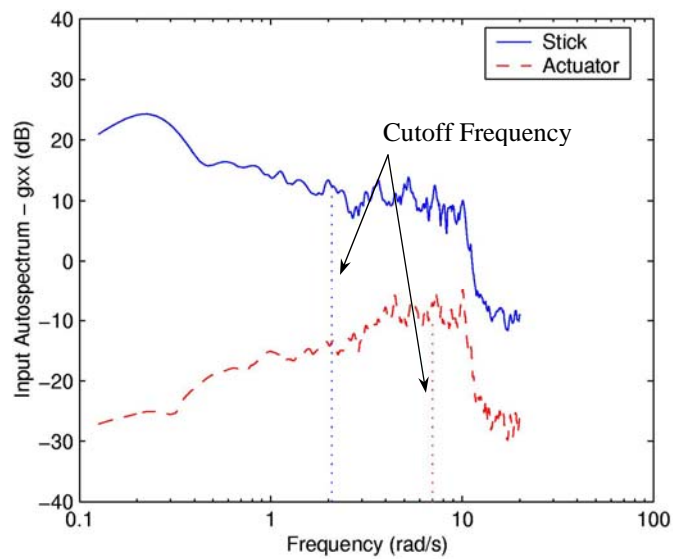


**Figure 4.10: Roll Bandwidth for Lateral Stick Input to Roll Angle Response**

CIFER<sup>®</sup> has an RMS utility that is capable of solving for the cutoff frequency of a response based on the energy content of the data. Table 4.3 shows the cutoff frequencies solved for using both lateral actuator (LATA) and stick (LATS) inputs with roll attitude and rate responses by the Unix and MATLAB versions of CIFER<sup>®</sup>. The results show that the piloted input loses energy at a lower frequency, and the actuator picks up content and operates at a higher frequency. These results are confirmed by the input autospectra, shown in Figure 4.11 with cutoff frequencies marked. There is no difference in the results between Unix and MATLAB.

**Table 4.3: Cutoff Frequency Results**

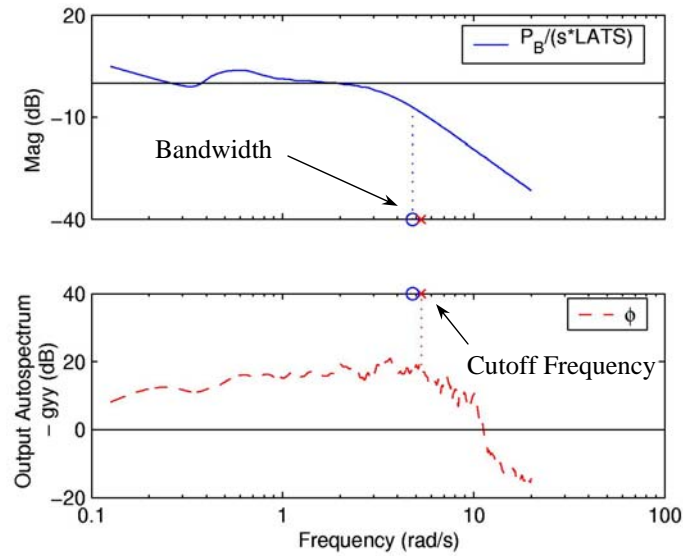
Channel	CIFER from Unix	CIFER from MATLAB
	Cutoff Freq.	Cutoff Freq.
LATA	7.00	7.00
- PB	5.32	5.32
- PHI	2.09	2.09
LATS	1.93	1.93
- PB	3.75	3.75
- PHI	0.57	0.57



**Figure 4.11: Stick and Actuator Input Autospectra**

The cutoff frequency for actuators can be indicative of the bandwidth frequency for some systems because the 135-degree crossover tends to occur during the phase shift that occurs at cutoff. The roll rate tends to have more frequency content than the attitude, as indicated by Table 4.3, which was why it was used for the previous bandwidth calculations and integrated to get the attitude response. The cutoff frequency for roll attitude compares reasonably closely to the 45-degree phase margin bandwidth frequency for the lateral stick to roll attitude response, 5.32 to 4.79,

respectively. Figure 4.12 shows the two frequencies marked on their respective magnitude and output autospectrum plots.



**Figure 4.12: Cutoff Frequency Compared to Bandwidth Frequency**

#### 4.4 Analysis on *Shadow 200* TUAV

The following analysis was conducted on AAI Corporation’s *Shadow 200* TUAV<sup>4</sup> for an Ames Research Center research project. *Shadow 200* has a wingspan of 12.75 feet, length of 11.17 feet, takeoff gross weight of 328 pounds, and carries 60 pounds of payload. It cruises between 65 and 85 knots, up to a 15,000-foot ceiling, with endurance better than five hours. Figure 4.13 shows *Shadow* at launch.



**Figure 4.13: *Shadow 200* TUAV<sup>4</sup>**

Two sets of data were used: one was a simulation of the UAV, and the second from a series of flight tests on the vehicle. The goals of the analysis were to investigate data consistency using the frequency response arithmetic utility, and to explore the RMS and cross-over characteristics of the vehicle. The results of the analysis were compared to scaled bandwidth criteria for manned aircraft. This analysis was the final, most involved, validation of the CIFER<sup>®</sup>-MATLAB interface.

#### 4.4.1 Data Consistency checks:

Checking data for kinematic consistency assures that measured data obeys kinematic laws and does not contain hidden scale factors or delays. The frequency arithmetic feature of CIFER<sup>®</sup> allows reconstruction of parameters not measured during flight tests from the responses of those that were measured. Additionally, it can be used to show whether or not data is consistent with kinematic laws. There are several relations among commonly measured rates and attitudes for aircraft as shown in Equations 4.4 through 4.8<sup>5</sup>. Herein, it was assumed that  $V_0$  and  $W_0$  were small.

$$p = \dot{\phi} \quad [4.4]$$

$$q = \dot{\theta} \quad [4.5]$$

$$\dot{u} = a_x - g\theta - W_0q + V_0r \quad [4.6]$$

$$\dot{v} = -U_0\dot{\beta} = a_y - U_0r + W_0p + g\phi \quad [4.7]$$

$$\dot{w} = U_0\dot{\alpha} = a_z + U_0q - V_0p \quad [4.8]$$

The simulation data includes measurements of phi and theta as well as the rates and accelerations in all axes, however, alpha and beta channels were not provided. Velocity perturbations were reconstructed from the time domain data in the FRESPID program using Equations 4.6 through 4.8. The frequency responses could then be calculated for the velocity perturbations. Using

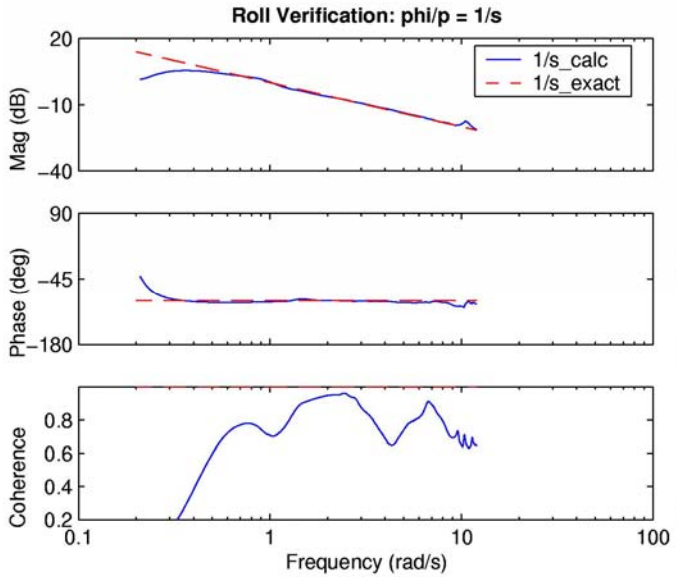
frequency response arithmetic, data consistency can be examined for all five of the above relations. The arithmetic feature in CIPHER<sup>®</sup> allows a single, basic operation (addition, subtraction, multiplication, and division) to be performed on two frequency responses. Each individual response can be modified by a scale factor and a power of  $s$  if desired.

In the frequency domain Equations 4.4 and 4.5 can be rearranged into Equations 4.9 and 4.10 respectively, thus allowing the comparison of the measured responses to the exact value of  $1/s$  in the frequency domain. Frequency response arithmetic was used to calculate the  $\phi/p$  and  $\theta/q$  responses from the on-axis responses for rates and attitudes. Figure 4.14 and Figure 4.15 show the arithmetic results for Equations 4.9 and 4.10 and compare these responses to the theoretical value of  $1/s$ .

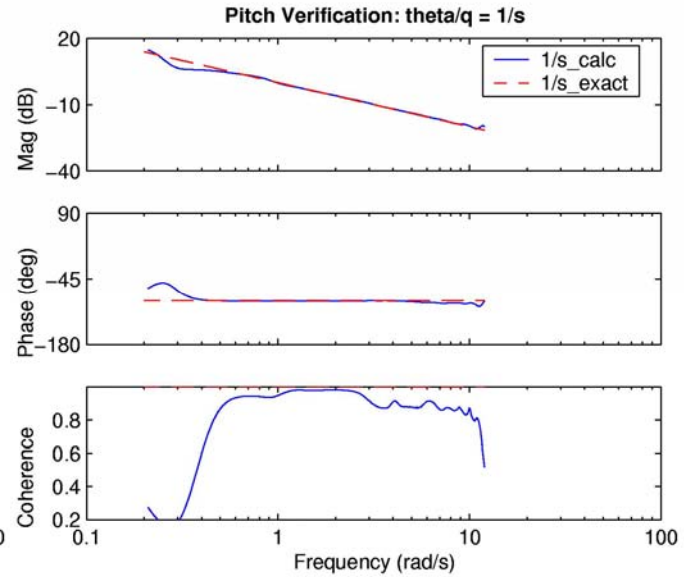
$$\frac{\phi / \delta_{ail}}{p / \delta_{ail}} = \frac{\phi}{p} = \frac{1}{s} \quad [4.9]$$

$$\frac{\theta / \delta_{ele}}{q / \delta_{ele}} = \frac{\theta}{q} = \frac{1}{s} \quad [4.10]$$

Both plots compare very well, which is to be expected as the measured values from the simulation should be kinematically correct to start with. At low frequency there is a more significant difference, primarily due to the low coherence. The difference suggests that the values from the CIPHER<sup>®</sup> response are unreliable due to little or no input at those frequencies. This is true of the higher frequencies where coherence also degrades. Another cause of these discrepancies could be a result of the hardware in the loop, which could introduce sensor and actuator noise and biases.



**Figure 4.14: Roll Angle to Rate Comparison**



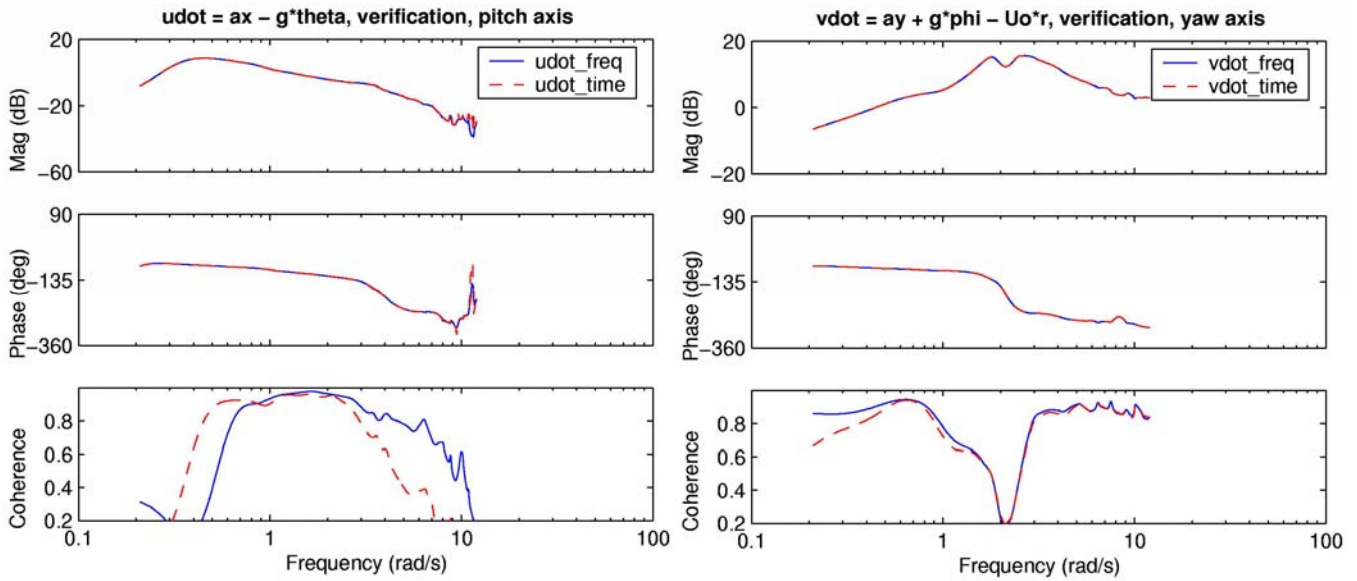
**Figure 4.15: Pitch Angle to Rate Comparison**

The next check using the simulation data was performed for Equations 4.6 through 4.8. Frequency response arithmetic was used to combine the on-axis responses of rates and attitudes to calculate the velocity perturbations in the frequency domain from the appropriate frequency responses. These were compared to the corresponding velocity perturbation responses that were initially reconstructed in the FRESPID program using the same math on the time history data. Equation 4.11 shows an example of how the frequency responses were combined using arithmetic.

$$\frac{\dot{u}}{\delta_{ele}} = \frac{a_x}{\delta_{ele}} - g \frac{1}{s} \frac{q}{\delta_{ele}} \quad [4.11]$$

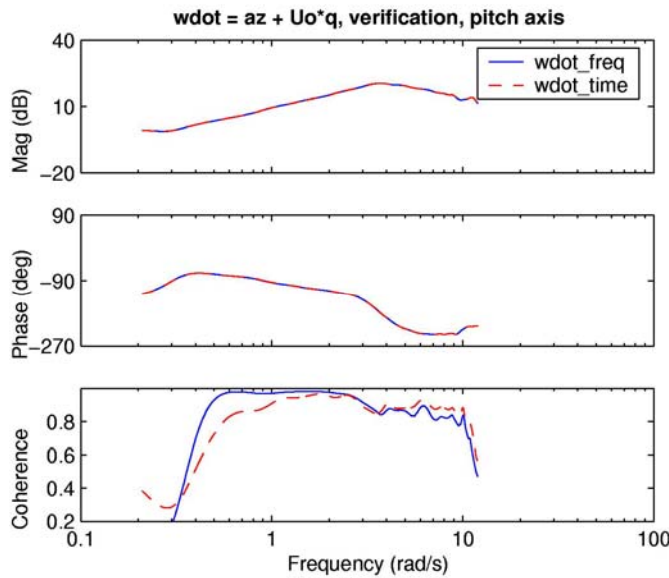
These comparisons are shown in Figure 4.16 through Figure 4.18 using a forward velocity of 65 knots. Both methods of calculation overlay nearly precisely, which makes sense as both plots are based on the same data, one created straight from the time domain and one built through frequency response arithmetic. Even in regions of low coherence, both calculations include the same errors and thus arrive at the same answers. The coherence plots vary due to a convention in

the arithmetic utility that applies the coherence from one or the other of the constituent responses to the resulting response.



**Figure 4.16: Longitudinal Velocity Perturbations**

**Figure 4.17: Lateral Velocity Perturbations**



**Figure 4.18: Vertical Velocity Perturbations**

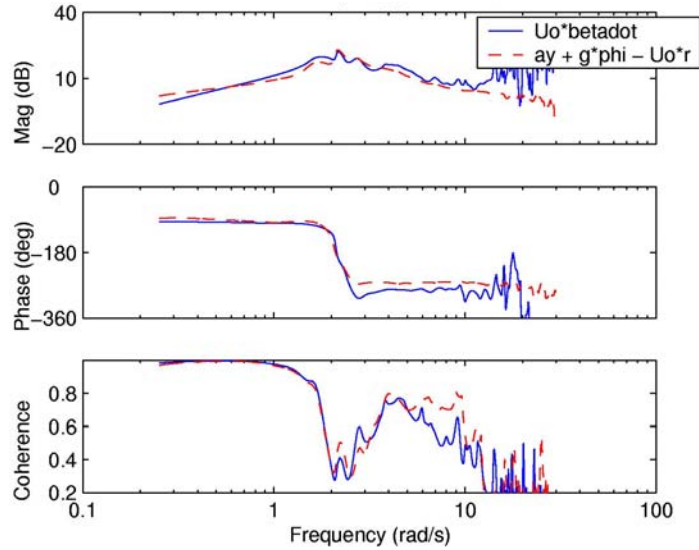
Performing consistency checks on the simulation was useful to ascertain that the models were correctly set up and identify any discrepancies that may have resulted from hardware in the loop.



The same checks were run on the flight data as well. Measurement instruments in flight are not inherently bound by the math that makes a simulation correct, and it is worthwhile to confirm that kinematic laws still hold true for them. Analyzing spurious results allows engineers to accurately correct flight data or fix the instruments in order to acquire, hopefully, more accurate data. The flight data includes measurements for the rates and accelerations, with the addition of alpha and beta measurements taken from a nose boom. These allow consistency checks for Equations 4.7 and 4.8.

Measurements of phi and theta were not included in the flight time history files and thus there is no benchmark with which to compare Equations 4.4 and 4.5. However, as phi and theta can be reconstructed from p and q, Equations 4.6 through 4.8 could be used to generate responses for the velocity perturbations. Equation 4.6 was not used with the flight data as there would only be one source of  $\dot{u}$  calculations for comparison. However, as alpha and beta were included in the time histories,  $\dot{v}$  and  $\dot{w}$  could be calculated by two methods and those results compared to check the data consistency of the time histories.

Figure 4.19 shows  $\dot{v}$  comparisons, where the solid line is the calculation of  $\dot{v}$  using yaw rate, roll angle, and lateral acceleration, and the dotted line is  $\dot{v}$  solved using the beta response for a forward velocity of 85 knots. The coherence drops rapidly at a very low frequency, just over 2 rad/s, however the trends of the two calculations still match up closely. There appears to be a small offset in phase between the two after the 180-degree phase shift. The offset could be due to the low coherence or possibly due to a hysteresis effect.

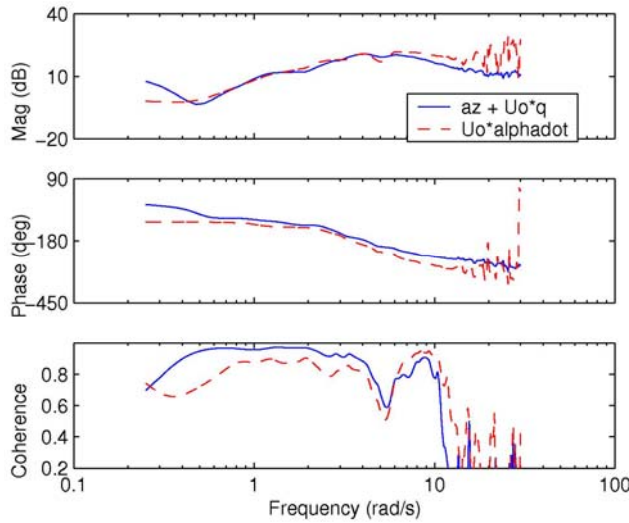


**Figure 4.19: Lateral Velocity Perturbations, Flight Data**

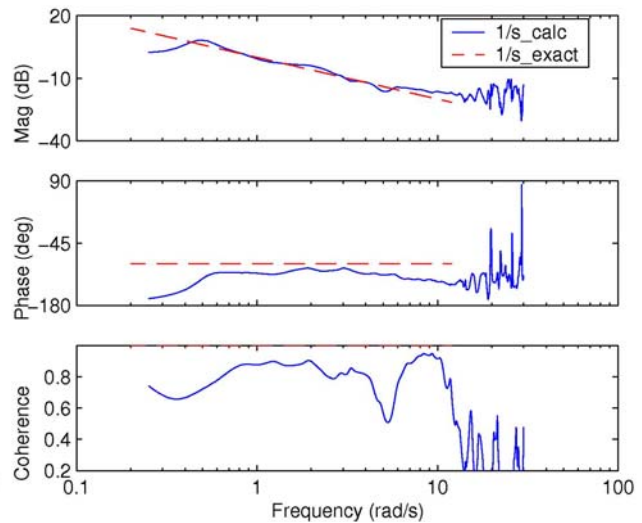
The second and final consistency check on the flight data was for  $\dot{w}$  calculations for pitching motion. Direct comparison of the two methods of calculation can be seen in Figure 4.20. Both calculated responses appear to have the same trend. For a more exact comparison of the measured data to the theoretical solution, Equation 4.8 can be rearranged to relate  $\alpha$ ,  $a_z$  and  $q$  to  $1/s$  as shown in Equation 4.12.

$$\frac{U_o \alpha}{a_z + U_o q} = \frac{1}{s} \quad [4.12]$$

A series of frequency response arithmetic operations were used to generate the left side of Equation 4.12, which was then compared to the plot of  $1/s$  as shown in Figure 4.21. Overall the data matches the  $1/s$  value quite well. Discrepancies in the magnitude plots can be directly correlated to spikes in the coherence. The most noticeable difference occurs in the phase plot; there is a small phase offset from  $-90^\circ$ . The offset is fairly constant with frequency, which suggests a hysteresis effect. The result is typical of airboom measurements such as were used in the flight test, which tend to exhibit some amount of hysteresis.



**Figure 4.20: Vertical Velocity Perturbations,  
Flight Data**



**Figure 4.21: Comparisons with Exact 1/s  
Value**

#### **4.4.2 RMS and crossover comparisons:**

CIFER<sup>®</sup> has utilities which calculate the root mean squared (RMS) value and some handling qualities metrics for a given frequency response. The results of these calculations can be compared to handling qualities specifications such as those laid out in MIL-STD 1797A<sup>9</sup> or the Neal-Smith Criteria<sup>10</sup>. The values between both simulation and flight data can be compared to check consistency of the simulation. Additionally, RMS calculations can be made on the time history data and compared to the equivalent frequency domain RMS values.

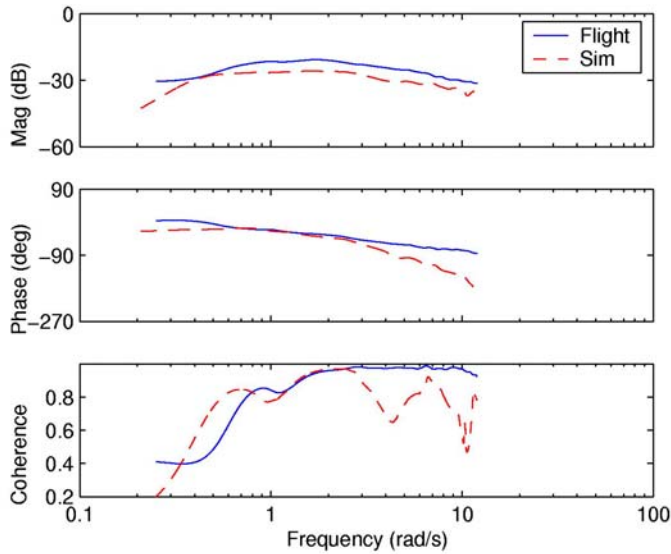
The first check was a comparison between flight and simulation full-range RMS values for control inputs and output responses from the main on-axis channels. These results are tabulated in Table 4.4. It should be noted that RMS values are based on the input or output autospectrum for a given response and are a measure of the energy or excitation of the system.

**Table 4.4: Full Range RMS Values**

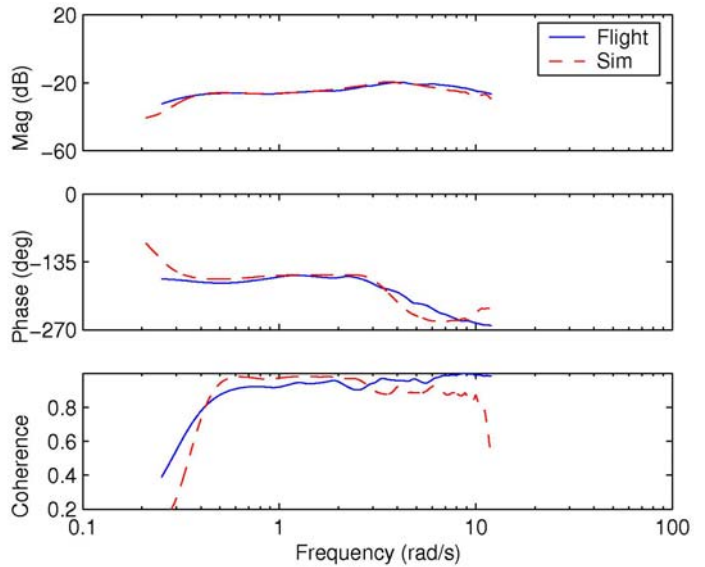
	Flight:	Sim:
AIL	6.05	9.07
P	0.350	0.330
ELE	2.66	3.98
Q	0.193	0.281
RUD	2.36	2.94
R	0.064	0.075
WHL	1.10	1.77
R	0.064	0.075
Beta	0.047	none

The RMS values for the aileron, elevator, and wheel inputs from the simulation are higher in magnitude than the flight RMS. The likely explanation is that the simulation operators gave larger inputs to the control system than the flight operators to the flight test. The rudder RMS values compare more closely between flight and simulation, however simulation is still larger in magnitude. The RMS comparisons of the pitch and yaw rates have the same trend as their respective inputs. The values for the roll rate are almost the same, which is not consistent to the difference in the magnitudes of the aileron inputs.

Figure 4.22 shows the comparison between the flight and simulation roll rate responses. For additional comparison, Figure 4.23 shows the same comparison in the pitch axis. The results for the roll axis show a discrepancy in magnitude which could be due to the x-axis moment of inertia estimated too large or the aileron control power derivative being estimated too small in the simulation. Additionally, the phase roll-off is steeper at high frequency for the simulation which suggests there might be a time delay error. The pitch response also shows an anomaly in the phase slope at high frequency, which could be a time delay error as well.



**Figure 4.22: Aileron - Roll Rate Responses**



**Figure 4.23: Elevator - Pitch Rate Responses**

RMS values for the time history data were manually calculated in MATLAB and compared to the equivalent values from CIFER<sup>®</sup> calculations as based on the frequency responses. These comparisons were conducted for both flight and simulation data. Table 4.5 shows the results of the study. The time RMS values were obtained from the pure control input data. The two calculations compare well between the two domains.

**Table 4.5: Frequency RMS compared to Time RMS**

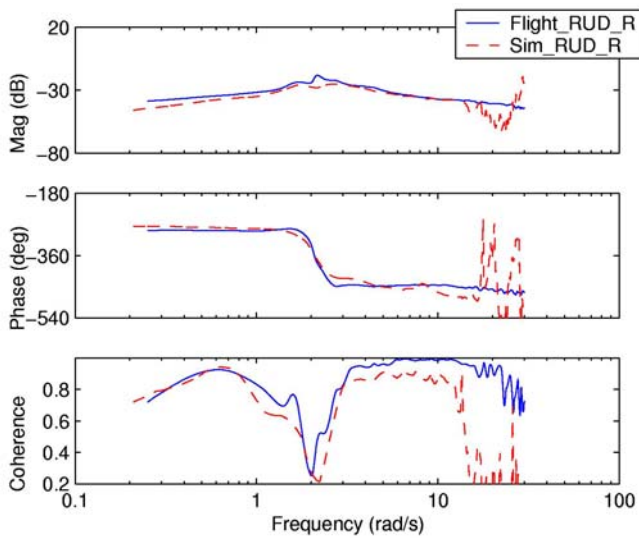
	Flight		Simulation	
	Freq.	Time	Freq.	Time
Ail:	6.05	5.83	9.07	9.01
Ele:	2.66	2.79	3.98	3.77
Rud:	2.36	1.96	2.95	2.85
Whl:	1.10	1.05	1.77	1.74

The crossover frequencies can also be examined by using the CIFER<sup>®</sup> RMS calculations. Flight and simulation results of the same on-axis responses are shown in Table 4.6. All of the cutoff frequencies based on inputs agree well, which means the flight control performance is consistent between the simulation and flight.

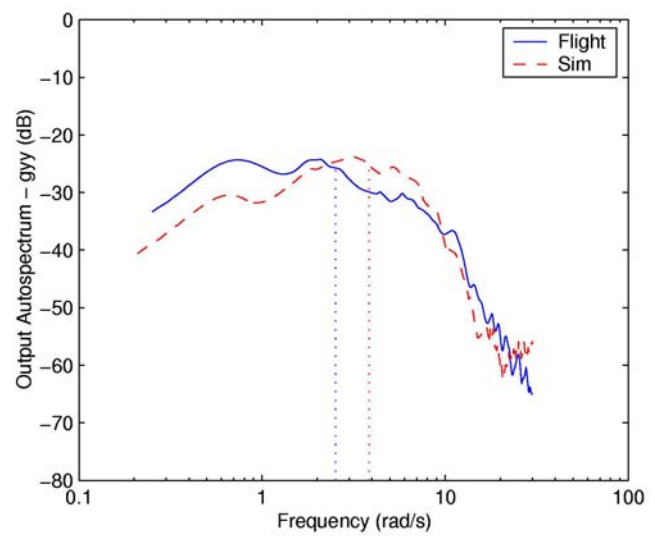
**Table 4.6: Cutoff Frequencies via RMS**

	Flight:	Sim:
AIL	4.82	4.43
P	3.53	3.70
ELE	7.03	6.82
Q	5.62	4.87
RUD	6.21	6.03
R	2.53	3.89
WHL	6.04	5.59
R	2.53	3.83
Beta	.85	none

The differences in the rudder and wheel output calculations between simulation and flight are greater. The phase slopes are different at the mode, which suggests the simulation is more highly damped. The outcome is surprising because the responses are very similar as shown in Figure 4.24. The primary difference is in the spike in the flight data around 2 rad/s, which is due to the poor coherence. When the output autospectrum is plotted, Figure 4.25, the difference between the flight and the simulation data is more pronounced. This is the data that is integrated in the RMS calculation. The plot of simulation data begins to roll off, losing energy, around 4 rad/s, which corresponds to the frequency from Table 4.6. The flight data begins to roll off just above 2 rad/s, also corresponding to Table 4.6; these autospectrum plots are the source of the discrepancy in the table.



**Figure 4.24: Rudder to Yaw Rate Response**



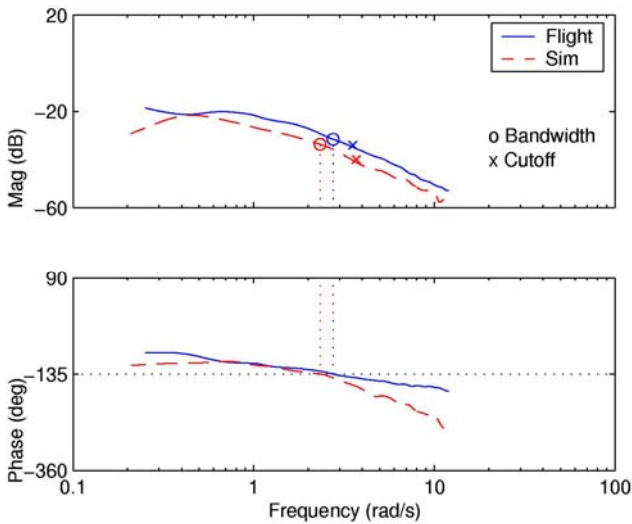
**Figure 4.25: Rudder Output Autospectrum**

CIFER<sup>®</sup> offers a utility for analyzing bandwidth and crossover properties of frequency responses. Use of the handling qualities portion of the Bandwidth utility allows identification of  $-180$ -degree and  $-135$ -degree bandwidth frequencies and gains. Table 4.7 shows the  $-135$ -degree bandwidth frequencies for the main on-axis responses in flight and simulation. The attitude bandwidths were determined by integrating the corresponding rate response.

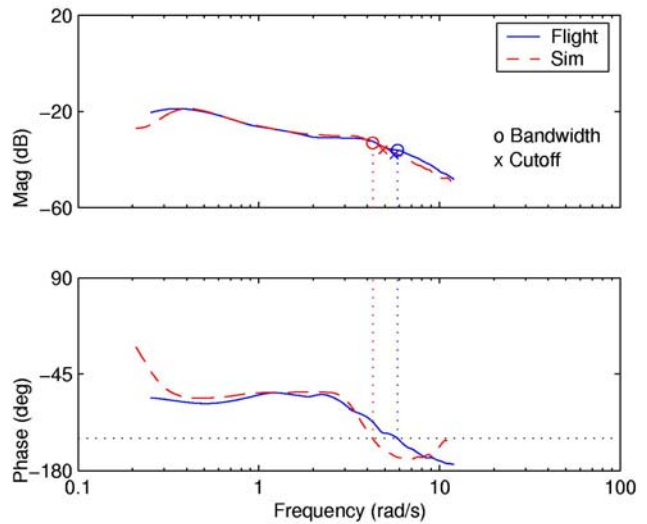
**Table 4.7: -135-Degree Bandwidth Frequencies**

	Flight:	Sim:
PHI/AIL	2.75	2.33
THETA/ELE	5.86	4.27
PSI/RUD	2.32	2.39
PSI/WHL	2.62	3.78
BETA/WHL	1.53	none

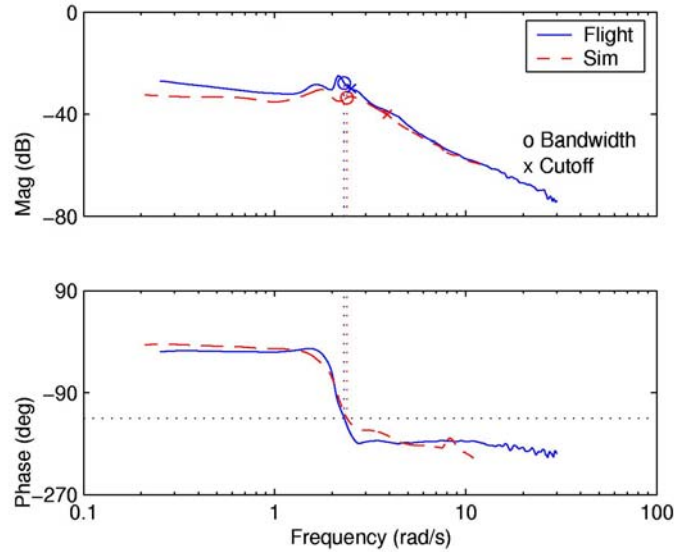
These results compare reasonably well between flight and simulation. As the cutoff frequency can be indicative of the bandwidth frequency it is useful to show comparisons between the two. Figures 4.26 through 4.28 show these comparisons for each axis on the attitude response for the aircraft. The results for the roll and pitch axes as well as the yaw flight data compare reasonably well. The cutoff and bandwidth frequencies for the yaw simulation data do not compare as well, which likely is a result of the differing output autospectra.



**Figure 4.26: Aileron to Roll Attitude Response**



**Figure 4.27: Elevator to Pitch Attitude Response**



**Figure 4.28: Rudder to Yaw Attitude Response**

Bandwidth can be a useful indication of aircraft handling qualities and thus can be used to assist in control system optimization. Classically, bandwidth criteria for fixed-wing aircraft are discussed in MIL-STD 1797A and several subsequent studies. These criteria use both bandwidth and phase delay to determine handling qualities. Phase delay is based on the twice 180-degree frequency as shown in Equation 4.13.

$$\tau_p = -\frac{(\phi_{2\omega_{180}} + 180^\circ)}{57.3 * 2\omega_{180}} \quad [4.13]$$

One concern with comparing the *Shadow 200* data to the specifications as defined in these reports is that the studies were conducted for much larger, piloted aircraft. MIL-STD 1797A uses AFFDL-TR-70-74<sup>10</sup> as a basis for its bandwidth criteria. This study was conducted to examine control system design criteria for fighter aircraft. Thus the criteria from these documents will be scaled to *Shadow* as though it were a scaled down fighter. Compared to the average fighter wingspan, *Shadow* has a scale factor of about 3. Equations 4.14 through 4.16 show the form of the dynamic similarity laws<sup>11</sup> that govern scaling where subscripts *a* and *m* denote actual and model, respectively, and N is scale factor.



Length:  $L_m = \frac{L_a}{N}$  [4.14]

Time Constant:  $\tau_m = \frac{\tau_a}{\sqrt{N}}$  [4.15]

Frequency:  $\omega_m = \omega_a \sqrt{N}$  [4.16]

Figure 4.29 shows results for the roll axis plotted on the scaled WL-TR-84-3162<sup>12</sup> handling specification for roll attitude. The flight bandwidth is within the level 1 handling qualities boundary while the simulation result is level 2. Though the bandwidths of the two tests are similar, the phase delay is significantly different: 0.093 for simulation and 0.001 for flight. This is a result of the differences in the phase curve of the frequency response, shown in Figure 4.30 with the 180-degree and twice 180-degree frequencies marked. The simulation data rolls off much more quickly than the flight data. The discrepancy between the two tests would need to be cleared up before a solid conclusion on the handling qualities could be reached.

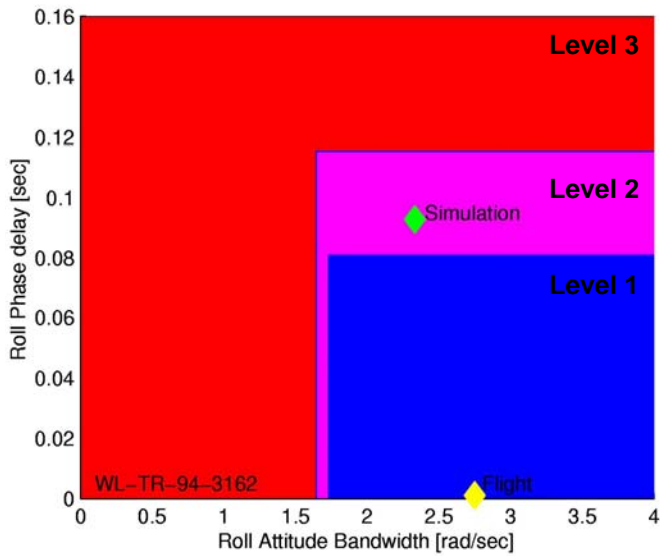


Figure 4.29: Scaled Roll Bandwidth Criteria

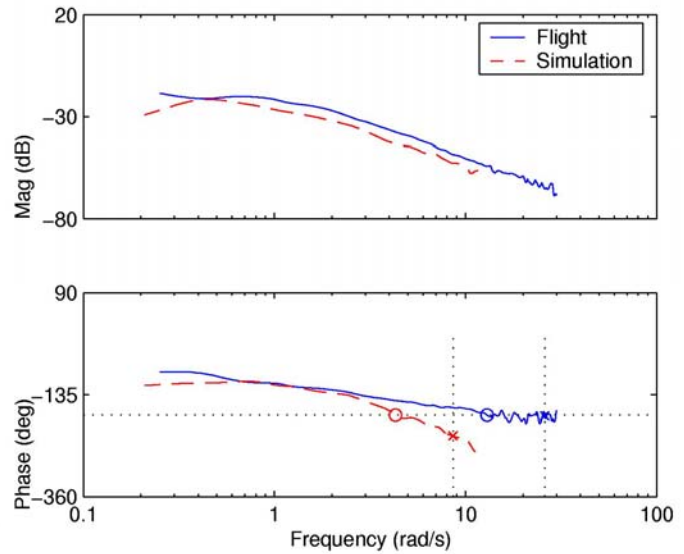


Figure 4.30: On Axis Roll Attitude Response

The analogous scaled specification for pitch attitude bandwidth is shown in Figure 4.31. Only flight data was calculated as the simulation data did not exhibit a -180-degree phase crossing at

high frequency, as shown on Figure 4.32. Flight data with marks for the 180-degree and twice 180-degree frequencies is also shown. This handling specification is for Category C flight which covers terminal flight phases that typically involve non-aggressive maneuvers and accurate flight-path control such as takeoff, approach, and landing. The results predict level 1 handling qualities. The phase delay, 0.009, is fairly small. As the calculations had to be conducted on a low coherence portion of the phase plot, it is probable that the phase delay is higher, though cleaner data would be required for verification.

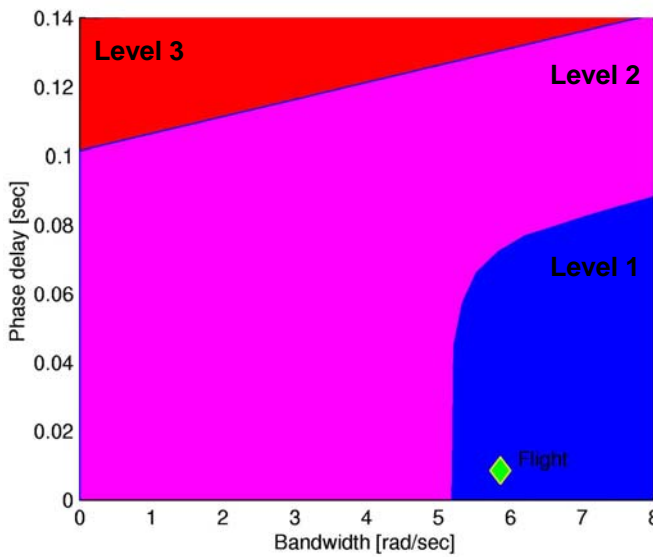


Figure 4.31: Scaled Pitch Category C Flight Criteria

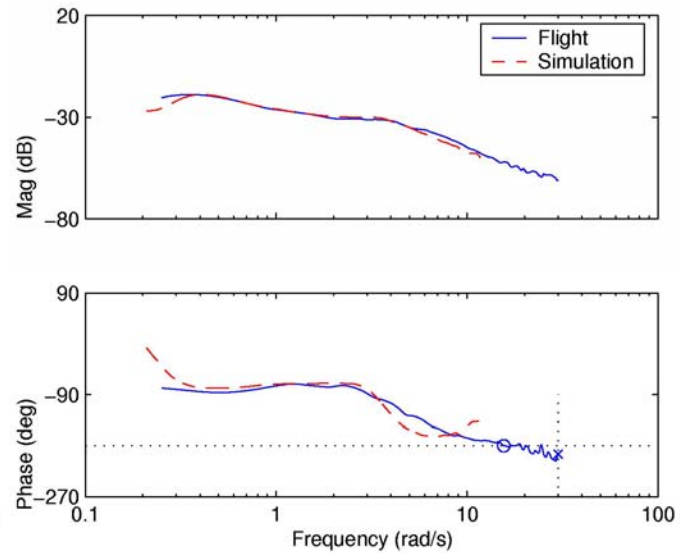


Figure 4.32: On Axis Pitch Attitude Response

A third bandwidth criterion for Category A flight encompasses non-terminal phases that require rapid maneuvering, precision tracking or precise flight-path control. Examples include air-to-air combat, weapon delivery/launch, reconnaissance or terrain-following. While *Shadow 200* is not a combat aircraft, it might be expected to perform aggressive maneuvers during its transit to and from observation targets. Figure 4.33 shows *Shadow* to be well into the level 2 region for Category A flight. The level 1 region scales up to a very high bandwidth, which may be unreasonable. Given that the spec is derived from fighter aircraft data, a scaled down fighter style

UAV might conceivably achieve bandwidths necessary to yield level 1 handling qualities by these scaled criteria.

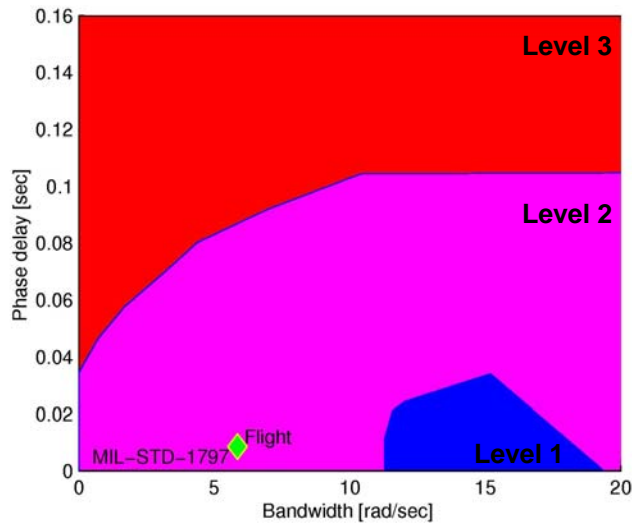


Figure 4.33: Scaled Pitch Category A Flight Criteria

#### 4.5 Summary of Validations

The extensive series of validation examples suggest that the CIFER<sup>®</sup>-MATLAB interface from the command line works very well. No major fundamental errors are still present in the code. The validations were central to the programming development as they allowed problems with interface structure to be fixed by the programmer as seen from the viewpoint of a user. In addition to the validations conducted by the author discussed in this chapter, engineers at NASA were supplied preliminary and completed versions of the interface to test in their own projects. This allowed the engineers more input to the layout of the interface and provided additional testing for the new code.

## Chapter 5: Conclusions

### 5.1 Code Development

The primary goal of this thesis was to create a modernized interface for CIPHER<sup>®</sup> that utilized both command-line functions and a GUI interface within MATLAB. Command-line functionality was introduced for CIPHER<sup>®</sup> programs FRESPID, MISOSA, COMPOSITE, the utilities for RMS, Bandwidth, and Frequency Response Arithmetic as well as additional plotting and data storage/retrieval utilities. This library of functions will provide a base from which further modernizations of CIPHER<sup>®</sup> can take place. The command-line functions proved to be complex enough that only an experienced user should employ them, however, the time savings to that user are significant. In-house users of CIPHER<sup>®</sup> at Ames Research Center were pleased with the new capability to script the set-up and running of cases.

The development of the GUI as a feasibility study also proved successful; the general look and feel of the CIPHER<sup>®</sup> screen interface was preserved while adding various elements that enhanced and accelerated the case set-up process over the previous Curses interface. The introduction of modern navigation tools such as browsers and menus will afford users more awareness of where they are in the set-up process and improve the learning curve of new users.

The last important aspect of the programming was the attention to the fact that the code will be built upon by other programmers at NASA. Thus the code was well commented and documented. The general structure of the code was made to be as uniform as possible. While the author's development of this code has ceased, programmers at Ames are continuing development and report that the efforts made to create an easily modifiable code were successful. GUIs for both

FRESPID and MISOSA have been created with a minimum hassle. Much of this success is due to the reviews of in-house users and programmers.

## 5.2 Analysis

The most important conclusion from the analysis was that the code developed for this thesis worked. Numerous comparisons between cases run in CIPHER<sup>®</sup> and cases run in MATLAB were made, and no appreciable difference was found. Slight discrepancies were discovered, but careful examination of these suggested that they were caused by machine precision issues, probably because MATLAB automatically makes all numeric variables double precision. The differences due to precision error were both negligible and tended to occur in portions of the data that would not be used in analysis, due to low coherence values.

The analysis provided a good illustration of the capability introduced with the ability to script the functionality of CIPHER<sup>®</sup>. The scripts that ran most of the analysis and generated the plots would have taken far longer to enter by hand into the CIPHER<sup>®</sup> interface. Many bugs and errors with the MATLAB code that might not have been found till much later, or at all, were uncovered by the extensive use of the functions during the analysis.

In addition to validating the programming efforts, the analysis tasks provided valuable experience to the author in real-world application of controls and handling analysis. The UH-60 example provided important insight into open and closed-loop handling qualities. The work on *Shadow* illustrated the need to verify that measurement devices are correctly installed and provided good experience in determining the accuracy of a simulation as compared to flight data.

### 5.3 Future Work

The functions created for this thesis are a significant step forward for CIFER<sup>®</sup>, but there is significant work that must still be completed in order to ready the command-line and modern GUI capability for commercial release, which is the ultimate intended goal. Command-line primitives for the remaining programs and utilities need to be created as well as a full graphical interface. While DERIVID and VERIFY are too interactive and complex to be encompassed by a single command-line call, combined functions could still allow experienced users to efficiently script setting up cases. Both of these programs would benefit greatly from updated GUI interfaces, which could easily run multiple command-line functions behind the screens.

In addition to finishing development on the remaining CIFER<sup>®</sup> functionality, the code will need to be production tested to assure that it is robust. Engineers at Ames Research Center are making the first steps towards this process by applying the current code to larger, more involved tasks. Once the code is fully developed and its capacity verified to the satisfaction of industry users the new code can be released as a standard feature of the CIFER<sup>®</sup> suite. The work accomplished in this thesis goes far towards realizing the possibilities of the new interfaces as a reality that will greatly benefit all users of CIFER<sup>®</sup>.

## Bibliography

### Works Cited:

- 1.) Hamel, P.G. (Editor), "Rotorcraft System Identification," AGARD-AR-280, 1991.
- 2.) The Mathworks, Inc., [www.mathworks.com](http://www.mathworks.com), 2004.
- 3.) Thompson, K.D., Ames Imaging Library Server, [ails.arc.nasa.gov/](http://ails.arc.nasa.gov/)
- 4.) AAI Corporation "Shadow TUAV" [www.shadowtuav.com](http://www.shadowtuav.com), Accessed 2004.
- 5.) Tischler, M.B., "CIFER version 2.1 - Comprehensive Identification from Frequency Responses, Vol 1 - Class Notes; Vol 2 - User's Manual," Army TR-94-A-017/8, NASA CP 10149/10150, September 1993.
- 6.) Tischler, M.B., Mansur, M.H., "Combined CIFER/CONDUIT/RIPTIDE Training," Moffett Field, Ames Research Center, 2004.
- 7.) "CONDUIT Version 4.1 User's Guide," University of California, Santa Cruz 41-071403, July 2003.
- 8.) Manur, M.H., Dai, W.L., "Real-time Interactive Prototype Technology Integration/Development Environment (RIPTIDE): Installation and User's Guide", UARC, UC Santa Cruz, Ames Research Center.
- 9.) United States Department of Defense, "Flying Qualities of Piloted Aircraft," MIL-STD-1797A, 1990.
- 10.) Neal, T.P., Smith, R.E., "An In-Flight Investigation to Develop Control System Design Criteria for Fighter Airplanes, Volumes 1 & 2," AFFDL-TR-70-74, Cornell Aeronautical Laboratory, Inc., Dec. 1970.
- 11.) Mettler, B., Tischler, M.B., Kanade, T., "System Identification of Small-Size Unmanned Helicopter Dynamics," American Helicopter Society, 1999.
- 12.) Mitchell, D.G., Hoh, R.H., "Proposed Incorporation of Mission-oriented Flying Qualities into MIL-STD-1797A", WL-TR-94-3162, 1994.

### References:

1. Marchand, P., *Graphics and GUIs with Matlab*, Boca Raton, CRC Press, 1996.
2. McRuer, D., Irving, A., Dunstan, G., *Aircraft Dynamics and Automatic Control*, New Jersey: Princeton University Press, 1973.

3. Nelson, R.C. *Flight Stability and Automatic Control*, 2<sup>nd</sup> ed., New York: McGraw-Hill, Inc., 1998.
4. Nise, N.S., *Control Systems Engineering*, New York: John Wiley & Sons, Inc., 2000.
5. Nyhoff, L.R., Leestma, S.C., *Fortran 90 for Engineers and Scientists*, New Jersey: Prentice Hall, 1997.
6. Thurling, A.J., "Improving UAV Handling Qualities Using Time Delay Compensation," M.S. Thesis, Air Force Institute of Technology, 2000.
7. Tischler, M.B., Cauffman, M.G., "Frequency-Response Method for Rotorcraft System Identification: Flight Applications to BO-105 Coupled Rotor/Fuselage Dynamics," *Journal of the American Helicopter Society*, Vol 37, No 3, pgs 3-17, July 1992.



## Appendix A: Screen Layout Comparison

This Appendix contains a series of screen shots that show each screen in COMPOSITE for both the original CIFER<sup>®</sup> screen interface and the new GUI interface. A more detailed description of the differences can be found in Chapter 3.

### Old Interface:

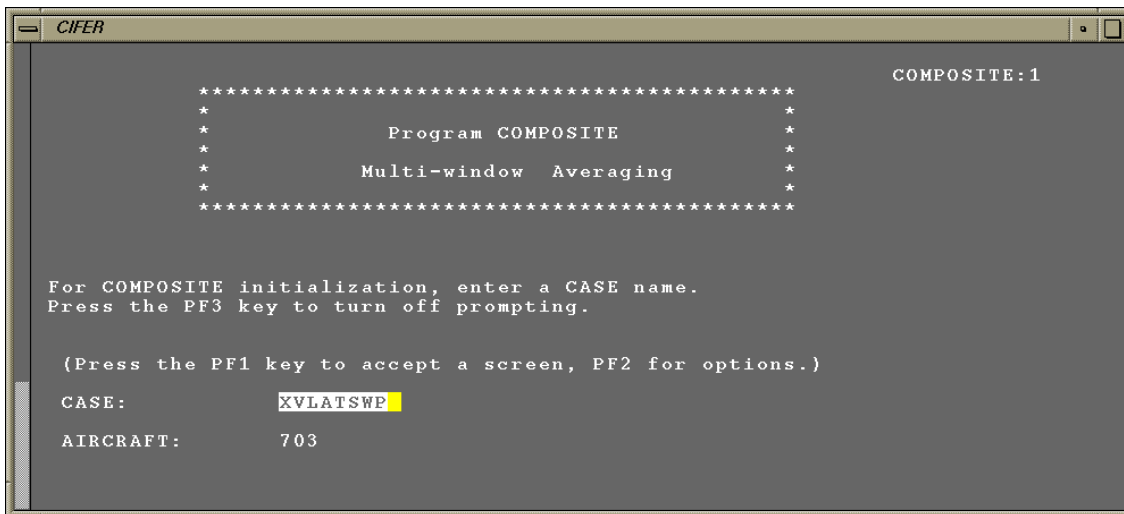


Figure A1: Original Interface: Screen 1

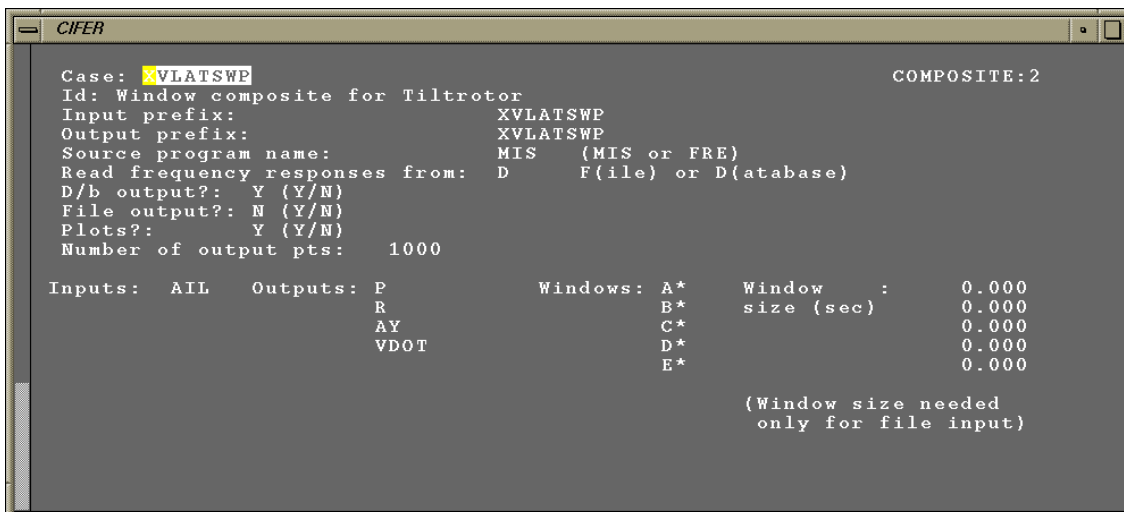


Figure A2: Original Interface: Screen 2

```
CIFER
All? (Y/N/Clear): N COMPOSITE:3
      AIL
P      *
R      *
AY     *
VDOT  *
```

Figure A3: Original Interface: Screen 3

```
CIFER
Plotting Options (1 to plot, 0 not to plot): COMPOSITE:4
Transfer fn magnitude: 1
Transfer fn phase: 1
Coherence: 0
Input auto spectrum: 0
Output auto spectrum: 0
Cross spectrum: 0
Error: 0

Heavy grid?: Y
Large plot?: Y

Plot output device: P
(V(er); C(omprs); Q(MS); T(alaris); P(ostScript))
```

Figure A4: Original Interface: Screen 4

```
CIFER

You may
(B) Submit a batch job and save case
(S) Save case only
(E) Exit without saving case

Enter your choice [E]: B
Writing case info ... Preparing batch job ...

The plot file will be: /u7/brupnik/cifer/jobs/plots/COM_XVLATSWP.PSC.02
Job file: /u7/brupnik/cifer/jobs/COM_XVLATSWP.COM.05 submitted.
Strike anykey to continue...
Message from brupnik on ronin (ttyq7) [ Wed Sep  8 13:39:52 ] ...
Job 5238571 from batch file: /u7/brupnik/cifer/jobs/COM_XVLATSWP.COM.05 has completed.
The log file is: /u7/brupnik/cifer/jobs/COM_XVLATSWP.OUT.05
No error detected.

<EOT>
```

Figure A5: Original Interface: Final Screen

New GUI Interface:

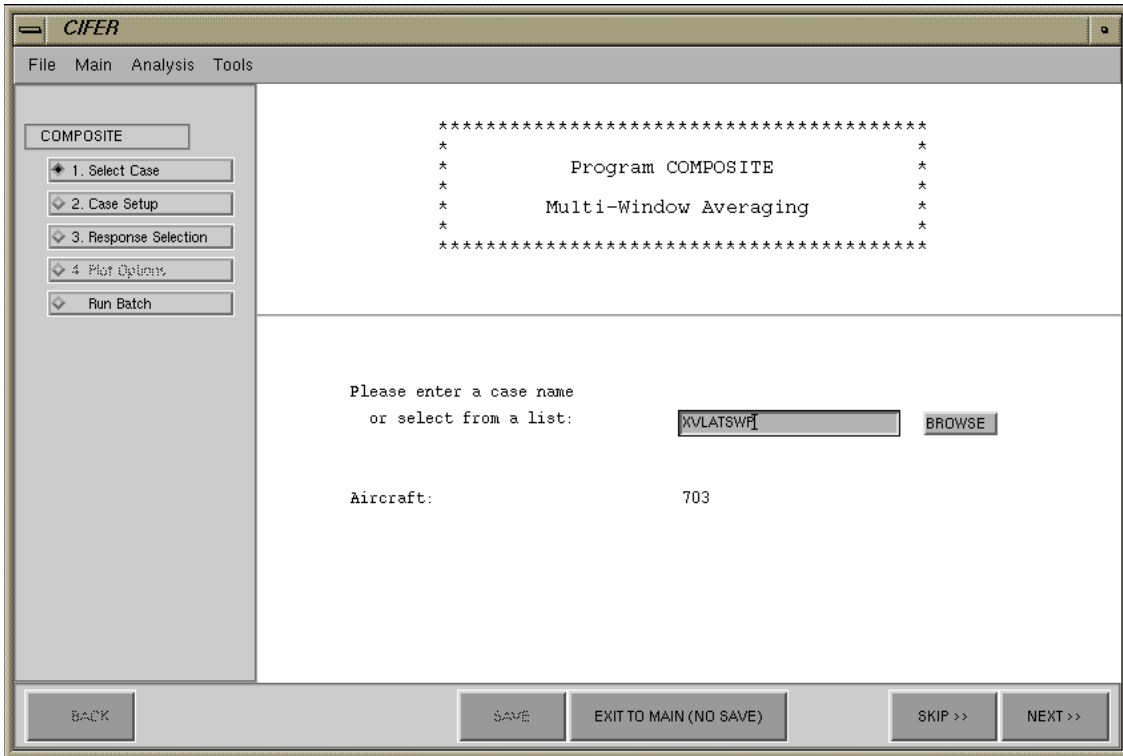


Figure A6: MATLAB GUI: Screen 1

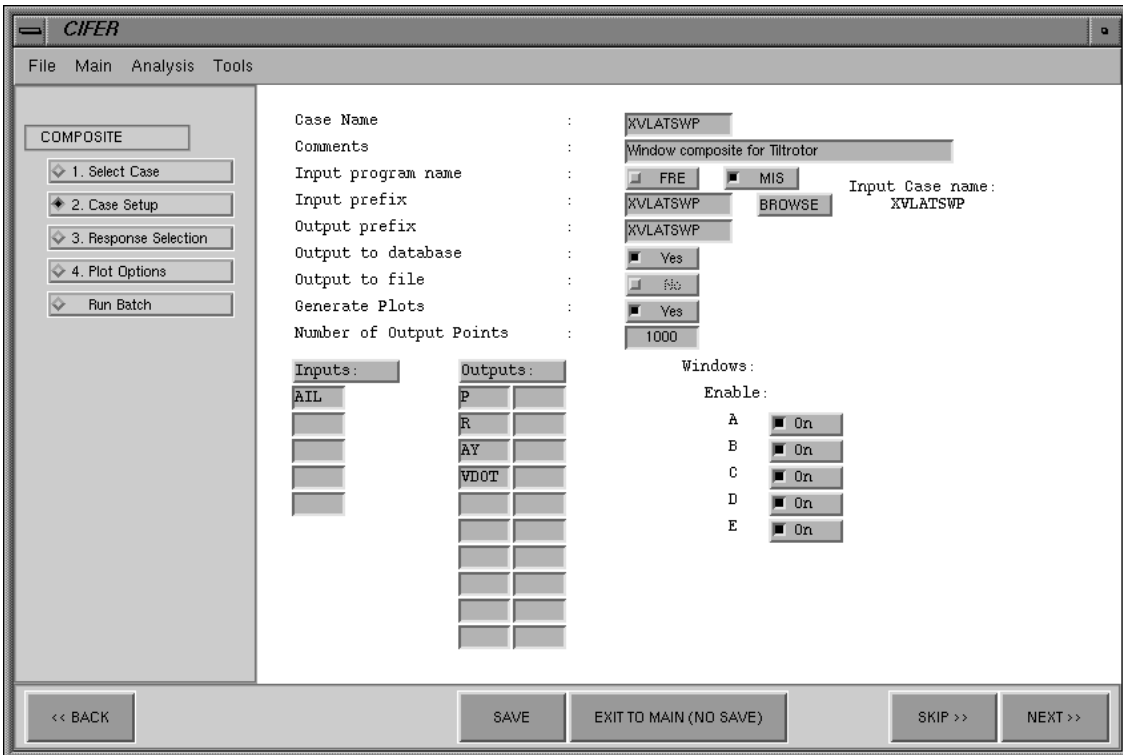


Figure A7: MATLAB GUI: Screen 2



Figure A8: MATLAB GUI: Screen 3

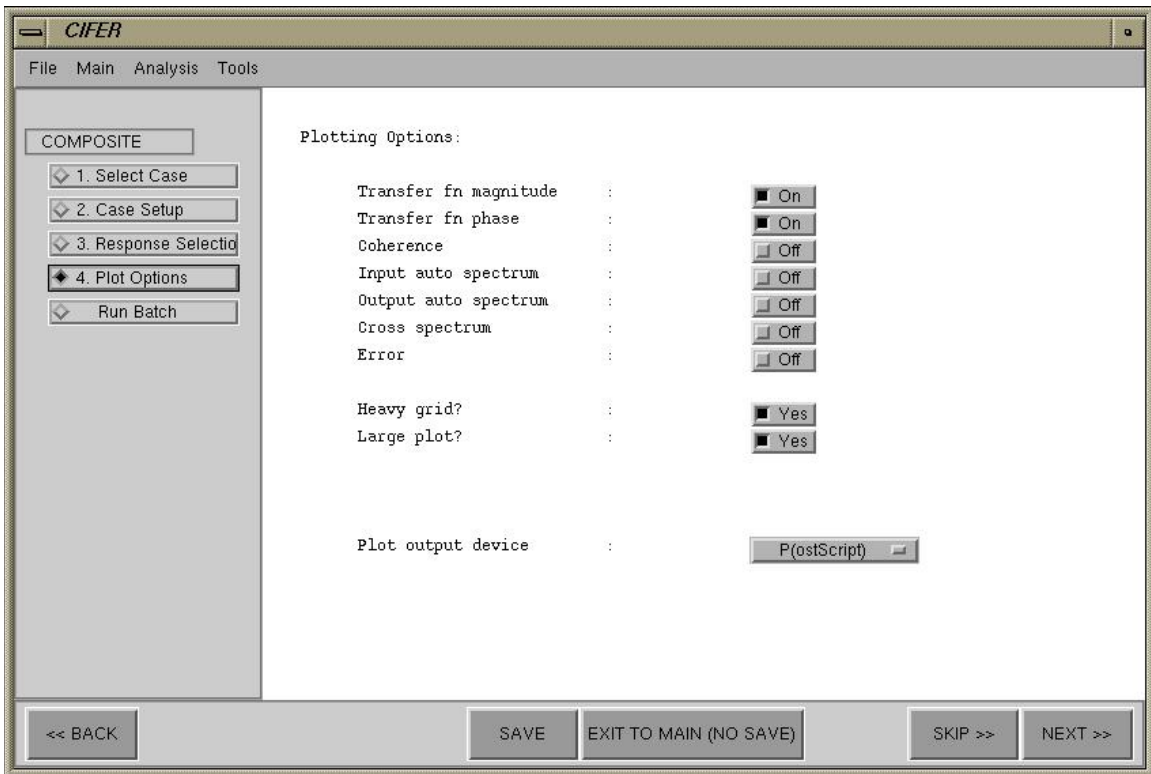


Figure A9: MATLAB GUI: Screen 4

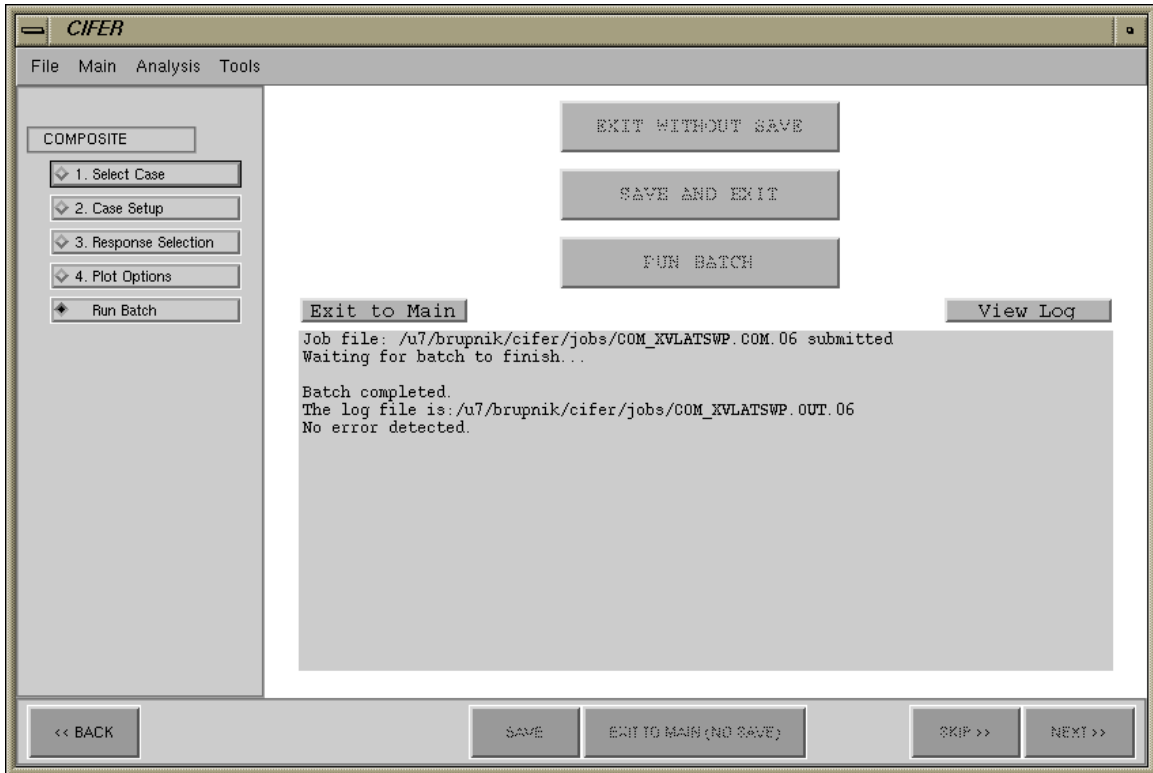


Figure A10: MATLAB GUI: Final Screen

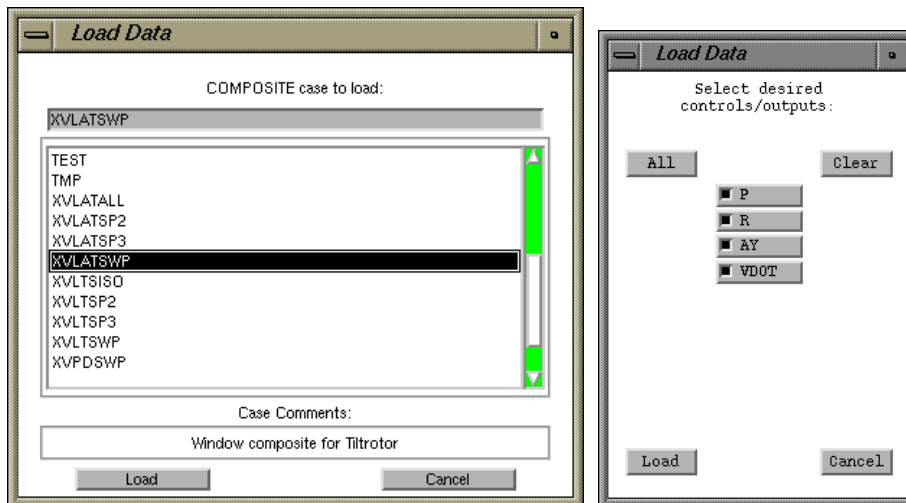


Figure A11: MATLAB GUI: Two Data Loading Screens

## **Appendix B: Help Documentation for Command-Line Interface**

### **Introduction:**

This document details the use of the command-line functions included in the CIPHER<sup>®</sup>-MATLAB interface developed by NASA Ames. CIPHER<sup>®</sup> is a tool, also developed by NASA Ames, for system identification using frequency responses. This document has been written assuming the user has background in using CIPHER<sup>®</sup>. Any questions concerning the operation of CIPHER<sup>®</sup> should be directed to the appropriate CIPHER<sup>®</sup> user manual.

This interface is designed as a tool to allow users to run CIPHER<sup>®</sup> programs and utilities from the command line of MATLAB. This enables easy retrieval of information in CIPHER<sup>®</sup> to the MATLAB workspace for analysis and allows the scripting of multiple batch jobs. The command line functions will perform all the operations and checks contained in each screen of a CIPHER<sup>®</sup> program at one time. While this allows the user great flexibility, the linear setup of the CIPHER<sup>®</sup> screens is bypassed. Thus new users are encouraged to verify the data entered in the command line by running the corresponding program within CIPHER<sup>®</sup> and checking each screen.

Contained within this document are a series of short examples detailing the various features and layout of the command-line functions. These examples are supplemented by the online help information (accessed using the command 'help functionname' in MATLAB) and two extended examples, all of which are contained in this document's appendices.

## CIFER<sup>®</sup> main programs

The three main CIFER<sup>®</sup>-MATLAB interface functions are designed to give users the functionality of the CIFER<sup>®</sup> programs FRESPID, MISOSA and COMPOSITE by making calls from the MATLAB workspace. All the functions are structured in the same manner with two required inputs. The first input is *either* the name of the desired CIFER<sup>®</sup> case *or* a MATLAB structure that contains information representing a CIFER<sup>®</sup> case. The second input is the flag to execute a particular command on that case. The commands that can be performed are to open a case, save data to a case or run a case as a batch job.

CIFER <sup>®</sup> #	CIFER <sup>®</sup> Program	MATLAB Function
1	FRESPID	<i>frespid</i>
2	MISOSA	<i>misosa</i>
3	COMPOSITE	<i>composite</i>

To facilitate the organization of information that goes into a CIFER<sup>®</sup> case, MATLAB structures are used to store case data. The fields in these structures correlate to the field entries of the CIFER<sup>®</sup> screen interface. The MATLAB interface has optional inputs that allow users to specify values for a particular field using name-value pairs. This allows the user to modify parts of the structure without dealing with the entire structure. Invoking the ‘help functionname’ command from MATLAB will display examples of using each function as well as a detailed list of the CIFER<sup>®</sup> information each field stores. The contents of these help commands can also be found in Appendix C.

CIFER<sup>®</sup> should be set to the database to be used (by selecting the appropriate SIFDEF file) prior to using these MATLAB functions. If the database is changed by changing the SIFDEF file, MATLAB must be restarted.

### Features common to all three main programs:

The structures for each of the programs contain many fields, thus the functions have been set up to do as much of the work of filling the fields in as possible for the user. Each function can be called with no inputs or outputs to return a list of what type of data is contained in each field; integers, cell arrays, scalar arrays, etc. The default value for each field is also specified in this call. This information can also be found in Appendix D. Users can initialize a mostly empty

structure by calling the functions with no inputs and a single output. This template contains all the correct fields with arrays specified to the correct size and many fields set to a default value. For instance, the option in FRESPID to cross-correlate controls is set to ‘yes.’ Fields that require case-specific information, such as the case name, are not given default values. These functions make extensive use of cell arrays to store string information and users unfamiliar with cell arrays should review the topic.

The following examples detail the basics of using the functions: opening, saving, and running cases. Details specific to individual functions will be covered in following sections. In addition to the short examples presented inline with the text, detailed examples of setting up and running a simple second order mass-spring-Damper system and the XVLATSWP case from the XV-15 sample database “703” are presented in Appendices H and I, respectively. (This database should be provided with the installation of CIFER<sup>®</sup>.)

It is very important to keep in mind that the MATLAB calls can only save frequency responses into the CIFER<sup>®</sup> database at this time. Due to complications with suppressing prompts from CIFER<sup>®</sup> for information, the option to save frequency responses as files has not been implemented in the MATLAB command line.

#### ***Creating a template structure:***

The call to create a template structure is very simple as shown below. After the calls, each of the ‘out\_x’ variables contains the basic skeleton used to create CIFER<sup>®</sup> cases. This feature is useful to ensure that the field names and data types are correctly specified and the various arrays are the correct length. The fields are all initialized to the same defaults as found in a new case of the appropriate CIFER program with the exception of plotting which defaults off for all three.

```
>> out_f = frespid;  
>> out_m = misosa;  
>> out_c = composite;
```

#### ***Opening Cases (1 as second input):***

Retrieving case information from the CIFER<sup>®</sup> database is accomplished by calling the MATLAB functions with the case name specified as the first input and ‘1’ for the second input as shown below. All of these calls would write the information from the CIFER<sup>®</sup> case XVLATSWP to the appropriate MATLAB structure ‘out\_x’. ‘out\_x’ can then be viewed and modified as needed.



```
>> out_f = frespid('XVLATSWP',1);
>> out_m = misosa('XVLATSWP',1);
>> out_c = composite('XVLATSWP',1);
```

***Saving Cases (2 as second input):***

Saving a case works much the same as opening cases. A simple call to save a case might involve specifying an existing case to be renamed as a new case. This call is shown below; the existing XVLATSWP case is opened, renamed to 'TEST', and then saved back into the database as a new case. The 'out\_x' variable retains the information about the new case.

```
>> out_f = frespid('XVLATSWP',2,'casename', 'TEST', 'caseout', 'TEST');
>> out_m = misosa('XVLATSWP',2,'casename', 'TEST', 'casein', 'TEST',
    'caseout', 'TEST');
>> out_c = composite('XVLATSWP',2,'casename', 'TEST', 'casein', 'TEST',
    'caseout', 'TEST');
```

Cases can also be specified using the appropriate structure as the first input. Shown below are calls using *frespid* that mimic the above example. In both examples, the output name (and input name for *misosa* and *composite*) is also specified; if it had not, then the output name would still be 'XVLATSWP'. The MATLAB functions do not assume the user wants to change the output name. If the output field is blank, then the functions assume the output name to be the same as the case name.

```
>> out_f = frespid('XVLATSWP',1);
>> out_f.casename = 'TEST';
>> out_f.caseout = 'TEST';
>> frespid(out_f,2);
```

Users should be careful when using old cases as a template. If the number of controls, outputs, data files, or other such parameters changes, old values can remain in the arrays of both the MATLAB structure and in the CIFER<sup>®</sup> database. Therefore it is good practice to carefully zero out unused fields in arrays when making changes. This will prevent unwanted information from being retained as changes are made.

***Running a batch job (3 as second input):***

Specifying a case to run can be as straightforward as opening a case. The three calls below will execute the batch job for the appropriate program (FRESPID, MISOSA or COMPOSITE) for the 'XVLATSWP' case. The batch call will pause MATLAB until the batch job has finished, display information about the job and log files, and report if there are potential errors with the batch job. Only the screen output for the *frespid* call is shown to conserve space.

```

>> frespid('XVLATSWP',3);
Job file: /usr/cifer/jobs/FRE_XVLATSWP.COM.01 submitted
Waiting for batch to finish...

Batch completed.
The log file is:/usr/cifer/jobs/FRE_XVLATSWP.OUT.01
No error detected.
>> misosa('XVLATSWP',3);
>> composite('XVLATSWP',3);

```

The example below will take the XVLATSWP case, copy it to TEST and then run it as a batch job all with a single command. The batch command always saves the case before it sends it to the batch job. To verify the name change was correct, the structure, which will contain information for TEST, is returned.

```

>> out_f = frespid('XVLATSWP',3,'casename','TEST','caseout','TEST');

```

### **frespid:**

This section contains information specific to the *frespid* function and will primarily cover the mechanics that mimic the functionality of FRESPID screens 7 and 8: conditioning and windowing of data.

Data conditioning in screen 7 is handled with two fields in the *frespid* structure. The first field, 'conditioning,' contains two rows; the first row stores a flag for the type of conditioning desired and the second row stores the value. The convention is that '1' denotes expansion of data, '2' denotes decimation, and '3' denotes filtering. The second field, 'condunit,' contains the units to use in the case of filtering. The example below illustrates these variables.

```

>> in = frespid;
>> in.conditioning(1:2,1:2) = [3, 2; 4, 25];
>> in.condunit(1) = 'Hz';

```

The data in the example will be filtered at 4 Hertz and decimated to 25 Hertz. These operations are lumped into one somewhat ungainly variable because of how CIPHER<sup>®</sup> internally processes the data. The default value for 'condunit' is Hertz so it need not be specified unless Radians or non-dimensional values are used. When returning conditioning output from CIPHER<sup>®</sup> to MATLAB, the function also defaults the units to Hertz. If cases created using conditioning are opened in CIPHER<sup>®</sup>, there may appear to be a small error in the ten or hundred thousandth decimal place. This is due to MATLAB using double precision while Fortran uses a combination of precisions. Numerous checks have verified that the end results are the same.

The variables that support screen 8 are analogous to the fields in that FRESPIID screen. The general convention is that if a window length is specified as zero, then automatic calculations on it will not be performed. However, if any of the other related variables, such as numbers of input and output points, are set to zero while a window is turned on, they will be automatically calculated using the same logic that CIFER<sup>®</sup> employs. If only window lengths are specified and all other related fields set to zero, then the function will generate the same values that CIFER<sup>®</sup> would if the screen were stepped through manually. Thus the following call would achieve the same result as entering the window lengths in CIFER<sup>®</sup> screen 8 and pressing the PF1 key until CIFER<sup>®</sup> finished filling in the fields.

```
>> in = frespid('NOWINS',1);  
>> frespid(in, 2, 'name', 'NEWWINS', 'winlen', [45,40,30,20,15])
```

The example retrieves a theoretical template case that has no window data entered and copies it to a case that now has fully specified window data. Users can specify their own values to fields such as maximum frequency and the automatic calculations will not occur unless the specified values violate the rules governing the variables. For instance, if the numbers of input and output points do not add to a power of two, the function will adjust them and issue a warning. Care should be used when using one case as a template for another and modifying the window lengths to make sure that old values are not written into the new case.

### **misosa and composite:**

The CIFER<sup>®</sup> programs MISOSA and COMPOSITE are less complicated than FRESPIID. Accordingly, the corresponding MATLAB functions have far fewer field entries and automatic corrections. The main issue to be alert for is in specifying the input case name. The warning is much the same as for the output case name previously stated. If the field is empty, it will be filled in to match the case name. Otherwise it retains the old entry. Thus to fully change names from one case to another, all three variables 'casename,' 'caseout,' and 'casein' must be modified.

## CIFER<sup>®</sup> analysis utilities

The CIFER<sup>®</sup>-MATLAB interface includes three frequency response analysis utilities: RMS, Handling Qualities/Stability Margin, and response arithmetic. Access to CIFER<sup>®</sup>'s handling qualities utility has been split into two parts: handling qualities and crossover characteristics. The current implementation allows full use of the various CIFER<sup>®</sup> options for each utility from the MATLAB command line. In addition to these analysis utilities, the CIFER<sup>®</sup> frequency response plotting utility may also be used from the MATLAB command line. Below is a list of the utilities and their corresponding MATLAB calling function. Appendix E contains the help files for each of these functions.

CIFER <sup>®</sup> #	CIFER <sup>®</sup> Utility	MATLAB Function
7	RMS	<i>cifrms</i>
8	Handling Qualities	<i>cifhq</i>
8	Handling Qualities	<i>cifxover</i>
9	Frequency Response Arithmetic	<i>cifarith</i>
19	Plot Frequency Response	<i>cifplot</i>

All five utilities use MATLAB structures to organize the input variables and work almost the same as the three main programs. They all use the same basic interface to convey information and run their respective utilities. Input can be specified either using a pre-initialized structure, or by passing in a flag, which generates a template structure, and a list of name-value pairs to fill in the fields of the template. The sections describing the individual functions have examples of both styles. Each function can be called with no inputs or outputs, causing a list of the information that each field requires along with its default setting to be displayed. This information is provided in Appendix F. Finally, if the functions are called with a single output and no inputs, a template structure will be returned initialized to the default settings. Examples of all these commands follow in the next sections and a detailed overview of the online help information for each function can be found in Appendix E.

### **cifrms:**

The CIFER<sup>®</sup> RMS utility is a straightforward calculation for the mean square and root mean squared values of a frequency response. The MATLAB *cifrms* function is called using a structure

as input and up to five outputs to collect the desired information. The least complex call and the corresponding output are shown below:

```
>> in = cifrms;
>> in.name = 'XVLATSWP_FRE_A0000_AIL_P';
>> cifrms(in);

**** Frequency Response Information ****
First Freq:      0.13963
Last Freq:    31.41590
Number of values in frequency response:      923

**** Mean-square value results ****
      Mean Square Value      =      9.244088
      Root Mean Squared Value =      3.040409
```

In this example the name of a frequency response is specified; when the structure is created it assumes defaults for inputs such as source and input/output integration selection. These defaults are defined in Appendix F and can be accessed by an empty call to the function as mentioned above. Single-character inputs such as those for source are not case-sensitive. No outputs need be specified; the function will print all the results to the screen. A more complex call is shown below without the printed output, and using name-value pairs for input. The name-value pairs do not have to be specified in a particular order as long as they occur in pairs.

```
>> in = cifrms;
>> name = 'XVLATSWP_FRE_A0000_AIL_P';
>> scorrect = 2;
>> minfreq = 10;
>> maxfreq = 30;
>>
[msv,rms,pts,min,max]=cifrms(in,'name',name,'spower',scorrect,'minfreq',
                             minfreq,'maxfreq',maxfreq,'toscrn','off');
```

This example specifies limits to the frequency range examined and sets the ‘power of s’ correction factor to 2. The default values can be used for these variables by assigning the inputs a value of 0. The output variables store the information that results from the function call; ‘msv’ and ‘rms’ store the mean square value and root mean squared value, respectively. The variable ‘pts’ stores the number of points in the response, and ‘min’ and ‘max’ store the minimum and maximum frequencies in the response. Fewer outputs can be specified if desired. Finally, the printed output can be turned off by the ‘toscrn’ field in the structure.

One other feature of the *cifrms* function is the ability to calculate a fraction of a full range rms value. The ‘minfreq’ and ‘maxfreq’ fields must be set equal to use this option. The printed output is slightly different and the variable outputs change slightly. ‘msv’ stores the full range rms value,

'rms' stores the rms value for the partial range, and 'freq' stores the frequency where that partial value occurs. An example call is shown below:

```
>>
[msv,rms,pts,min,max,freq]=cifrms(0,'name','XVLATSWP_FRE_A0000_AIL_P',
    'minfreq',0.707,'maxfreq',0.707);

**** Frequency Response Information ****
First Freq:      0.13963
Last Freq:      31.41590
Number of values in frequency response:      923

**** Mean-square value results ****
Full range root mean square value is:      3.040409
    Located frequency where RMS value is
    0.707000 * 3.040409 = 2.149569
    At frequency 3.837147 rad/sec the RMS is 2.158710
```

### **cifhq:**

The *cifhq* function is comprises all of the handling qualities calculations from CIPHER<sup>®</sup>'s utility 8 as well as the utility's quick plotting, linearized plotting, and least squares fits. There are several output variables available to store the information resulting from the function. The most basic call to *cifhq* is shown below where the function is called to set up a template structure which is then modified for the specific response. The full list of field names can be found in Appendix C.

```
>> in = cifhq;
>> in.name = 'XVLATSWP_FRE_A0000_AIL_P';
>> cifhq(in);

    Start Freq      Start Mag      Start Phase
    End Freq      End Mag      End Phase
    0.1396      -34.5106      2.6104
    31.4159      -39.9538      96.0240

Number of values in frequency response:      923

***** Handling Qualities Characteristics *****

Start freq:      0.1396
Start phase:      2.6104

-180 deg frequency =      29.478342 (Rad/sec)
    DB-Gain =      -63.286926 (dB)
    Linear gain =      0.000685 (Hz)
-135 deg BW freq =      29.463261 (Rad/sec)
    DB-Gain =      -61.210114 (dB)
    Linear gain =      0.000870 (Hz)
6 dB Bandwidth frequency =      29.437544 (Rad/Sec)
Another 6 dB Bandwidth frequency =      27.796064 (Rad/sec)
Another 6 dB Bandwidth frequency =      27.782501 (Rad/sec)
Another 6 dB Bandwidth frequency =      24.950647 (Rad/sec)
Another 6 dB Bandwidth frequency =      24.890440 (Rad/sec)
Another 6 dB Bandwidth frequency =      17.924093 (Rad/sec)
Another 6 dB Bandwidth frequency =      17.860493 (Rad/sec)
TWICE 180 FREQUENCY NOT FOUND
```

All the information is displayed to the screen, however, output variables can be used to preserve these numbers. There are two variables for storing *cifhq* output, one for purely numeric values, and a second that stores the words associated with each number. Information about the frequency response can be saved to two additional variables. Shown below is the function call with all outputs specified.

```
>> [words,num,pts,frinfo]=cifhq(in,'toscrn','off');
>> disp(words)
''
' -180 deg frequency = 29.478342 (Rad/sec) '
' DB-Gain = -63.286926 (dB) '
' Linear gain = 0.000685 (Hz) '
''
' -135 deg BW freq = 29.463261 (Rad/sec) '
' DB-Gain = -61.210114 (dB) '
' Linear gain = 0.000870 (Hz) '
''
' 6 dB Bandwidth frequency = 29.437544 (Rad/Sec) '
' Another 6 dB Bandwidth frequency = 27.796064 (Rad/sec) '
' Another 6 dB Bandwidth frequency = 27.782501 (Rad/sec) '
' Another 6 dB Bandwidth frequency = 24.950647 (Rad/sec) '
' Another 6 dB Bandwidth frequency = 24.890440 (Rad/sec) '
' Another 6 dB Bandwidth frequency = 17.924093 (Rad/sec) '
' Another 6 dB Bandwidth frequency = 17.860493 (Rad/sec) '
' TWICE 180 FREQUENCY NOT FOUND'
```

The above call has turned off the screen output using the 'toscrn' field. The variable 'num' contains only the resultant frequencies and gains that are shown in the 'words' variable. (Those values on the right side of the equals sign.) 'pts' saves the number of points in the response and 'frinfo' is an array containing the starting and ending frequency, magnitude, and phase of the response. It also contains starting frequency and phase for reference if modified using correction factors for 'power of s,' gain, phase shift, or time delay. The correction factors can be used with a slightly more complex call, shown below.

```
>> name = 'XVLATSWP_FRE_A0000_AIL_P';
>> corrections = [2,5,90,0.5];
>> cifhq(0,'name',name,'cor_list',corrections,'toscrn','off');
```

This call utilized the name-value pair style of input. The name-value pairs do not have to be specified in a particular order as long as they occur in pairs. Note that the flag of 0 for the first variable makes the function automatically use the default structure set up in *cifhq*. If the 0 flag is used, any response-specific inputs must be specified or the call will result in errors. The template does not assume default input or output response names.

The *cifhq* function supports the utility 8 ability to create plots and perform a least squares fit on the data. There are a number of options for turning on the plots and setting their ranges. The call below implements the full series of plotting available from utility 8.

```
>> name = 'XVLATSWP_FRE_A0000_AIL_P';
>> plot = 'Y';
>> linplot = 'Y';
>> leastsquare = 'Y';
>>
cifhq(0, 'name', name, 'mpcplt', plot, 'lpcplt', linplot, 'lsfit', leastsquare,
      'toscrn', 'off');
```

This call would display a total of three CIFER<sup>®</sup> plots to the computer screen. The values for the ranges on all the plots are initialized to a default value, which can be modified by changes to the appropriate fields (see Appendix E) in the structure.

### **cifxover:**

The second half of the handling qualities utility in CIFER<sup>®</sup> allows the user to find crossover properties for a frequency response. This ability has been split out into a separate function call for the MATLAB interface. It works nearly the same as the *cifhq* function. The most basic call and the resulting output are shown below.

```
>> in = cifxover;
>> in.name = 'XVLATSWP_FRE_A0000_AIL_P';
>> cifxover(in);
```

```
Search Range:
Min freq:    0.139626    Max freq:   31.415899
```

```
No 0dB crossing found
```

```
-180*n deg crossings for gain margin determination
```

```
First -180*n crossover in selected range is:
Freq. = 27.818336 for 180.0000 deg
Gain Margin = 54.728527 dB
```

```
First -180*n crossover in selected range is:
Freq. = 29.478342 for -180.0000 deg
Gain Margin = 63.286926 dB
```

```
First -180*n crossover in selected range is:
Freq. = 30.105280 for -180.0000 deg
Gain Margin = 52.274651 dB
```

The name-value pair method of input works the same as described above for *cifhq*. The call below shows an example of this used to run the utility with correction factors, generate a plot of the output, and save the new response. The crossover characteristics portion of utility 8 is not tied



to either the linearized phase and coherence plot or the least squares fit in CIFER<sup>®</sup> so those options are not available to the *cifxover* function.

```
>> name = 'XVLATSWP_FRE_A0000_AIL_P';
>> corrections = [2,5,90,0.5];
>> plot = 'Y';
>> save = 'Y';
>> savename = 'XVLATSWP_BAN_A0000_AIL_P';
>> cifxover(0,'name',name,'cor_list',corrections,'mpcplt',plot,'save',
    save,'savename',savename,'toscrn','off');
```

Again it should be noted that care should be used when employing the 0 flag for inputs as response names are not assumed by the code. Output information is stored in the same fashion as *cifhq*. The first two variables contain the numeric values for the crossover characteristics, and the last variable has the frequency range of the response.

```
>> [X0db,Xn180db,frng]=cifxover(0,'name','XVLATSWP_FRE_A0000_AIL_P',
    'toscrn','off');

>> disp(X0db)
    0      0

>> disp(Xn180db)
27.8183 180.0000 54.7285
29.4783 -180.0000 63.2869
30.1053 -180.0000 52.2747

>> disp(frng)
0.1396 31.4159
```

### **cifarith:**

This utility allows basic arithmetic (+,-,\*,/) to be performed on two frequency responses. The results are saved to a new response file. A very simple example of running the arithmetic function is shown below:

```
>> in=cifarith;
>> in.names = {'XVLATSWP_FRE_A0000_AIL_P','XVLATSWP_FRE_A0000_AIL_R'};
>> in.outname = 'test';
>> in.outid = 'arith resp from Matlab';
>> cifarith(in)
```

```
Response test written to the database.
*** Arithmetic operation successful ***
```

```
Minimum Frequency is: 0.1396
Maximum Frequency is: 31.4159
1000 Values in Output Response
Units are in RAD
```

The template sets all values to the defaults found when the arithmetic utility in CIFER<sup>®</sup> is first run. For example, all scale factors are set to 1, the operation is multiplication, and the source and destination point to the database. Any of these values can be modified either by changes to the appropriate field in the structure or by using the field name and value as a pair of inputs. The above example could be compressed into one call as follows. There are no output variables for this function, and as before the 'toscrn' field allows the user to turn off the printed return information.

```
>> cifarith(0,'names',{ 'XVLATSWP_FRE_A0000_AIL_P',
    'XVLATSWP_FRE_A0000_AIL_R'}, 'outname', 'test', 'outid', 'arith resp
    from Matlab', 'toscrn', 'off')
```

### **cifplot:**

The *cifplot* function is the MATLAB command line call for CIFER's utility 19, the frequency response plotting. A basic set of calls to *cifplot* are shown below:

```
>> in = cifplot;
>> in.array(1:3) = [1,2,3];
>> in.names(1) = {'XVLATSWP_FRE_A0000_AIL_P'};
>> cifplot(in);
    Reading data .....
    Plotting successful
```

After this call, the standardized CIFER<sup>®</sup> plot would be displayed to the screen. As with the previous functions, the call can be simplified using name-value pairs as input. The previous example could be reduced as follows. There are no output variables for this function.

```
>> cifplot(0,'array',[1,2,3,0,0], 'names',
    {'XVLATSWP_FRE_A0000_AIL_P', '', '', '', ''})
    Reading data .....
    Plotting successful
```

## **CIFER<sup>®</sup> support utilities:**

Three functions are available to facilitate the retrieval of data from the CIFER<sup>®</sup> database. The *getfr* function, when given a frequency response name, will return arrays for frequency, magnitude, phase and so on. The other function, *writefr*, allows the user to change values in these arrays and pass them back in to be saved to the CIFER<sup>®</sup> database. The third function is called *caselist* and allows the user to query the database for which cases are stored for a particular program.

Due to the simplicity of these functions, structures are not used to input data as is done for the other CIFER<sup>®</sup> programs and utilities. Thus, these functions do not have the empty call feature that returns a list of structure field data types. They do still include help documentation which can also be found in Appendix G.

### **getfr and writefr:**

The *getfr* function is designed to give users access to all the arrays that make up a full frequency response. The most complete call, shown below, will bring back arrays of all 10 frequency response fields. The order in which these are returned is fixed; therefore if only coherence were desired, variables to hold information for frequency, magnitude and phase would still be required as shown in the second function call, which only returns four of the frequency response data fields.

```
>> name = 'XVLATSWP_COM_ABCDE_AIL_P';  
>> [frq,mag,pha,coh,gxx,gyy,gxy,rel,img,err] = getfr(name);  
>> [frq,mag,pha,coh] = getfr(name);
```

Frequency responses can also be written back into the CIFER<sup>®</sup> database through the *writefr* function. The example below shows the creation of a new frequency response with all 10 data arrays. Again, the order is fixed and, as above in the *getfr* example, not all the inputs need be provided. If all inputs are not provided the user should keep in mind that CIFER<sup>®</sup> will not have access to any unspecified information in the event it must make calculations with the new responses. The response names do not have to conform to CIFER<sup>®</sup> naming conventions, though doing so will help track new responses.

```
>> name = 'NEW_FREQUENCY_RESPONSE';  
>> writefr(name, frq, mag, pha, coh, gxx, gyy, gxy, rel, img, err);
```

**caselist:**

Lists of cases present for any of the CIFER<sup>®</sup> programs can be obtained using the *caselist* function. The three calls below return a cell array list of FRESPID, MISOSA and COMPOSITE cases that are present in the database. The input code follows the same numbering convention found in the delete utility of CIFER<sup>®</sup> (Utility 13). This information can also be found by making a 'help' call for the function and is printed in Appendix G.

```
>> names_f = caselist(1);    % Returns FRESPID cases  
>> names_m = caselist(2);    % Returns MISOSA cases  
>> names_c = caselist(3);    % Returns COMPOSITE cases
```

## Appendix C: Online Help for Main Programs

This Appendix contains the information that is displayed when the 'help functionname' command is used in MATLAB.

### FUNCTION: frespid

DESCRIPTION: Function to open, save and run a FRESPID case.  
Function may be called with no arguments to return a template, mostly blank structure.

```
[out] = frespid(id,cmd,options)
```

#### INPUTS:

id - either frespid structure or string with case name  
Information on the details of the structure can be found with an empty call:  
>> frespid

cmd - command to have function perform a process  
1 - open FRESPID case  
2 - save a FRESPID case  
3 - save case and run batch job

options - name-value pairs to set individual data fields (optional)  
e.g.: ..., 'casename', 'XVLATSWP', ...

#### OUTPUTS:

out - return structure for frespid data structure. A template structure can be returned using an empty call:  
>> out = frespid

#### EXAMPLE CALLS:

```
[out] = frespid
returns a mostly blank template frespid structure

[out] = frespid('TEST',1)
opens case named 'TEST', returns frespid structure

[out] = frespid('TEST',2,'casename','TEST2','winlen',
[45,40,30,20,15])
saves TEST as TEST2 and changes window lengths; the
structure is returned

frespid(new_struct,3)
sends new_struct, a new FRESPID case, to batch; nothing is
returned
```

#### NOTES:

- A totally new case must be specified via a structure, not by the desired new name. (i.e., frespid('NEWNAME',2) will NOT save a blank case entitled 'NEWNAME' into the database)

---

#### DESCRIPTION OF FIELDS IN INPUT STRUCTURE

##### SCREEN 2

id.casename	FRESPID case name
id.comments	FRESPID case identifier string

id.controls	Control names (user-specified)
id.outputs	Output names
id.caseout	Case name for freq responses (may be different from the case name)
id.crosscor	Cross-correlate controls? ('Y' or 'N')
id.savdb	Save frequency response in the database ('Y' or 'N')
id.plot	Generate plots? ('Y' or 'N')
SCREEN 3	
id.evntnum	Event list (i=1,10)
id.flghtnum	Flight list (corresponds to eventnum)
id.strttim	Start time for each time history
id.stoptim	Stop times
id.source	Code for where time histories come from: 1 for stand-alone ASCII files 2 for TRENDS 3 for FLYTE 4 for READMIS >4 other sources as defined by source code
id.thdt	Time interval between samples
id.biasflag	Bias and drift removal? ('Y' or 'N')
id.thfile	Array to store time history file names
SCREEN 4	
id.conchnl	Primary channel names for each control
id.conunit	Engineering units for control channels
id.conscfac	Scale factors for each channel
SCREEN 5	
id.outchnl	Primary channel names for each output
id.outunit	Engineering units for output channels
id.outscfac	Scale factors for each channel
SCREEN 6	
id.frall	Compute all frequency responses? ('Y' or 'N')
id.frscal	Table indicating which responses to compute. (Blank if not to be computed; '*' to compute.)
SCREEN 7	
id.conditioning	Array to store conditioning information. The first row contains the code for the type of conditioning: 1 for expansion 2 for decimation 3 for filter The second row stores the actual value
id.conduunit	If the filtering option is used, this field stores the unit that the value is provided in. If left blank, Hertz is assumed.
id.savconth	Save files of conditioned time histories? 'N' no files, 'A' ASCII format only, 'U' unformatted only, 'Y' both types
SCREEN 8	
id.winon	Entry for each window: blank not to compute, '*' to compute
id.winid	Window descriptor
id.winlen	Window Lengths
id.wininpt	Number of t.h. points input to the FFT
id.winoutpt	Number of points returned by the FFT (wininpt + winoutpt = power of 2) (winoutpt must be .le. wininpt)
id.windec	Output decimation factor for frequency responses
id.minfft	Min FFT freq for this window



```

id.caseid      Descriptive text for this case
id.casein     Frequency response prefix (input)
id.caseout    'Case' part of freq resp name for output
id.source     Source for input frequency responses:
              'F' for ASCII files; 'D' for database
id.savdb     Write results to the database? ('Y' or 'N')
id.plot      Generate plots? ('Y' or 'N')
id.controls  Control names
id.outputs   Output names
id.winon     Code for windows to combine
              ('*' indicates a window to include,
              '' indicates not to include a window)

```

SCREEN 3

```

id.frall     Code indicating whether to clear/set frcalc matrix
id.frcalc   Table indicating which responses to compute
              ('*' requests a new response computation,
              '' indicates not to calculate a response)

```

SCREEN 4

```

id.plotopt  Integer array indicating which plots to generate.
id.grid     Draw a grid on the plots? ('Y' or 'N')
id.lrgplot  Large plot? ('Y' or 'N')
id.plotdev  Selected plot device: Q(MS),C(omprs),V(er),
            S(creen), T(alaris),P(ostScript)
id.pltminfrq Minimum frequency of each window
id.pltmaxfrq Maximum frequency of each window

```

**FUNCTION: composite**

DESCRIPTION: Function to open, save and run a COMPOSITE case.  
 Function may be called with no arguments to return a template,  
 mostly blank structure.

```
[out] = composite(id,cmd,options)
```

INPUTS:

```

id          - either composite structure or string with case name
              Information on the details of the structure can
              be found with an empty call:
              >> composite
cmd         - command to have function perform a process
              1 - open COMPOSITE case
              2 - save a COMPOSITE case
              3 - save case and run batch job
options    - name-value pairs to set individual data
              fields (optional)
              e.g.: ..., 'casename', 'XVLATSWP', ...

```

OUTPUTS:

```

out        - return structure for composite data structure. A
              template structure can be returned using
              an empty call:
              >> out = composite

```

EXAMPLE CALLS:

```

[out] = composite
returns a mostly blank template composite structure

[out] = composite('TEST',1)
opens case named 'TEST', returns composite structure

```



```
[out] = composite('TEST',2,'casename','TEST2','winon',
                 {'*', '', '', '', ''})
    saves TEST as TEST2 and the window selection is
    changed; the structure is returned

composite('TEST2',3)
    sends case 'TEST2' to batch; nothing is returned
```

NOTES:

- A totally new case must be specified via a structure, not by the desired new name. (i.e., composite('NEWNAME',2 will NOT save a blank case entitled 'NEWNAME' into the database)

-----  
DESCRIPTION OF FIELDS IN INPUT STRUCTURE

SCREEN 2

id.casename	Composite Case name
id.caseid	Descriptive text for this case
id.casein	Frequency response prefix (input)
id.caseout	'Case' part of freq resp name for output
id.inpgm	Input program, 'FRE' or 'MIS'
id.source	Source for input frequency responses: 'F' for ASCII files; 'D' for database
id.savdb	Write results to the database? ('Y' or 'N')
id.plot	Generate plots? ('Y' or 'N')
id.outpts	Number of points in each resultant freq resp
id.controls	Control names
id.outputs	Output names
id.winon	Code for windows to combine (* indicates a window to include, ' indicates not to include a window)
id.winlen	Window length, in seconds

SCREEN 3

id.frall	Code indicating whether to clear/set ICOMP matrix
id.frcalc	Table indicating which responses to compute (* requests a new response computation, ' indicates not to calculate a response)

SCREEN 4

id.plotopt	Integer array indicating which plots to generate.
id.grid	Heavy grid on the plots? ('Y' or 'N')
id.lrgplot	Large plot? ('Y' or 'N')
id.plotdev	Selected plot device: Q(MS), C(omprs), V(er), S(creen), T(alaris), P(ostScript)

## Appendix D: Structure Field Specifics for Main Programs

This Appendix contains the information displayed through an empty call to any of the functions.

Default settings are shown within a curly brace.

```
>> frespid
```

Structure fields are:

```
casename: [ string, 8 characters { '' } ]
caseid: [ string, 60 characters { '' } ]
controls: [ cell array(10) of strings { '' } ]
outputs: [ cell array(20) of strings { '' } ]
caseout: [ string, 8 characters { '' } ]
crosscor: [ 'Y' | { 'N' } ]
savdb: [ { 'Y' } | 'N' ]
plot: [ 'Y' | { 'N' } ]
evntnum: [ scalar array(10) { 0 } ]
flghtnum: [ scalar array(10) { 0 } ]
strttim: [ positive scalar array(10) { 0 } ]
stoptim: [ positive scalar array(10) { 0 } ]
source: [ integer value { 1 } ]
thdt: [ positive scalar { 0 } ]
biasflag: [ { 'Y' } | 'N' ]
thfile: [ cell array(10) of strings { '' } ]
conchnl: [ cell array(10,5) of strings { '' } ]
conunit: [ cell array(10) of strings { '' } ]
conscfac: [ positive scalar array(10,5) { 1 } ]
outchnl: [ cell array(20,5) of strings { '' } ]
outunit: [ cell array(20) of strings { '' } ]
outscfac: [ positive scalar array(20,5) { 1 } ]
frall: [ 'Y' | { 'N' } ]
frcalc: [ cell array(20,10) of strings { '' } | '*' ]
conditioning: [ scalar array(2,10) { 0 } ]
condunit: [ cell array(10) of strings { '' } ]
savconth: [ 'Y' | { 'N' } | 'A' | 'U' ]
outpts: [ positive scalar { 1000 } ]
winon: [ cell array(5) of strings { '' } | '*' ]
winid: [ cell array(5) of strings { '' } ]
winlen: [ positive scalar array(5) { 0 } ]
wininpt: [ positive scalar array(5) { 0 } ]
winoutpt: [ positive scalar array(5) { 0 } ]
windec: [ positive scalar array(5) { 0 } ]
minfft: [ positive scalar array(5) { 0 } ]
maxfft: [ positive scalar array(5) { 0 } ]
plotopt: [ positive scalar array(12) { 0 } ]
plotdev: [ 'Q' | 'C' | 'V' | 'T' | { 'P' } ]
grid: [ { 'Y' } | 'N' ]
lrgplot: [ { 'Y' } | 'N' ]
plotdec: [ { 'Y' } | 'N' ]
```

```
>> misosa
```

```
Structure fields are:
```

```
casename: [ string, 8 characters {''} ]
caseid: [ string, 60 characters {''} ]
casein: [ string, 8 characters {''} ]
caseout: [ string, 8 characters {''} ]
source: [ { 'D' } | { 'F' } ]
savdb: [ { 'Y' } | { 'N' } ]
plot: [ 'Y' | { 'N' } ]
outpts: [ positive scalar { 1000 } ]
controls: [ cell array(10) of strings { '' } ]
outputs: [ cell array(20) of strings { '' } ]
winon: [ cell array(5) of strings { '' } | '*' ]
frall: [ 'Y' | { 'N' } ]
frcalc: [ cell array(20) of strings { '' } | '*' ]
plotopt: [ positive scalar array(12) { 0 } ]
grid: [ { 'Y' } | { 'N' } ]
lrgplot: [ { 'Y' } | { 'N' } ]
plotdev: [ 'Q' | 'C' | 'V' | 'T' | { 'P' } ]
pltminfrq: [ positive scalar array(5) { 0 } ]
pltmaxfrq: [ positive scalar array(5) { 0 } ]
```

```
>> composite
```

```
Structure fields are:
```

```
casename: [ string, 8 characters {''} ]
caseid: [ string, 60 characters {''} ]
casein: [ string, 8 characters {''} ]
caseout: [ string, 8 characters {''} ]
inpgm: [ { 'FRE' } | 'MIS' {''} ]
source: [ { 'D' } | { 'F' } ]
savdb: [ { 'Y' } | { 'N' } ]
plot: [ 'Y' | { 'N' } ]
outpts: [ positive scalar { 1000 } ]
controls: [ cell array(5) of strings { '' } ]
outputs: [ cell array(20) of strings { '' } ]
winon: [ cell array(5) of strings { '' } | '*' ]
winlen: [ positive scalar array(5) { 0 } ]
frall: [ 'Y' | { 'N' } ]
frcalc: [ cell array of strings { '' } | '*' ]
plotopt: [ positive scalar array(7) { 0 } ]
grid: [ { 'Y' } | { 'N' } ]
lrgplot: [ { 'Y' } | { 'N' } ]
plotdev: [ 'Q' | 'C' | 'V' | 'T' | { 'P' } ]
```

## Appendix E: Online Help for Analysis Utilities

This Appendix contains the information that is displayed when the 'help functionname' command is used in MATLAB.

### Function: cifrms

Description: This mex file allows the user to use the RMS utility of CIPHER. Function returns mean square value and root mean squared value as well as information about the frequency response if desired.

```
[msv,rms,npts,lowfreq,highfreq,rmsfrq] = cifrms(in, varargin)
```

#### Inputs:

- in - input structure, fields defined below.  
Information on the details of the structure can be found with an empty call:  
>> cifrms
- options - name-value pairs to set individual data fields (optional) e.g.: ..., 'io', 'I', ...

#### Outputs:

- rms - optional output allows user to set up a template structure using this call with no inputs:  
>> out = cifrms

\*\*\* OR \*\*\*

- rms - root mean squared value
- msv - Mean square value (or full range RMS if percentage of range is specified)
- npts - number of points in response
- lowfreq - smallest value in response
- highfreq - highest value in response
- rmsfrq - frequency where RMS is desired fraction of full RMS value

#### NOTES:

This function can run with no outputs specified; the results will be displayed in the Matlab command window.

---

#### DESCRIPTION OF FIELDS IN INPUT STRUCTURE

- in.name - Name of Frequency Response (string)  
ex: name = 'XVLATSWP\_FRE\_A0000\_AIL\_P'
- in.source - Source: 'F' for file, 'D' for database
- in.io - 'I' to integrate input-auto, 'O' for output-auto
- in.spower - value for power of s correction, positive or negative integer (0 for none)
- in.minfreq - start frequency for calculations (0 for default)
- in.maxfreq - end frequency for calculations (0 for default)

NOTE: set minfreq equal to maxfreq for a fraction of the full range RMS.

- in.toscrn - turn printed screen output 'ON' or 'OFF'

**Function: cifhq**

Description: This function calls the first half of CIFER's utility 8, the handling quality calculations.

```
[output_s,output_n,npts,FR_info] = cifhq(in,options)
```

Inputs:

in - input structure, fields defined below.  
Information on the details of the structure can be found with an empty call:  
>> cifhq

options - name-value pairs to set individual data fields (optional) e.g.: ..., 'save', 'Y', ...

Outputs:

output\_s - optional output allows user to set up a template structure using this call with no inputs and a single output:  
>> out = cifhq

\*\*\* OR \*\*\*

output\_s - Cell array containing the resulting output from CIFER for handling qualities

output\_n - array containing numeric output from CIFER for handling qualities

npts - number of points in response

FRinfo - array containing information about the frequency response.

```
FRinfo = [sF,sM,sP,sFm  
          eF,eM,eP,sPm]
```

the elements are starting and ending (s,e) values for frequency, magnitude and phase (F,M,P). (m) denotes value after modification by correction factors.

NOTE: This function can be called with no outputs specified; the results will output to the command window.

-----  
DESCRIPTION OF FIELDS IN INPUT STRUCTURE

in.name - Name of Frequency Response (string)  
ex: name = 'XVLATSWP\_FRE\_A0000\_AIL\_P'  
in.source - Source: 'F' for file, 'D' for database  
in.minfreq - Minimum frequency for search range for handling qualities calculations. Enter 0 for the default range.  
in.cor\_list - Array with values as follows: (optional)

```
in.cor_list = [scor, gcor, ps, td]
```

scor - value for power of s correction  
gcor - value for gain correction, >= 0  
ps - value for phase shift  
td - value for time delay

NOTE: an entry of 0 for any of these values is assumed to mean no correction, shift, etc.

in.save - 'Y' to save response, 'N' not to save  
in.savename - name to save response as. (optional)  
ex: 'XVLATSWP\_BAN\_A0000\_AIL\_P'

NOTE: If save='Y' and savename is left blank, the program portion of the filename will automatically be changed to 'BAN'.

in.mpcplt - create a magnitude, phase, (coherence) plot  
in.mpcmin - minimum frequency for mpc plot  
in.mpcmax - maximum frequency for mpc plot  
in.mpcdev - output device Q(MS), C(omprs), V(er), S(screen),  
T(alaris), P(ostScript)  
in.lpcplt - create a linear phase and coherence plot  
in.lpcmin - minimum frequency for lpc plot  
in.lpcmax - maximum frequency for lpc plot  
in.lpcdev - output device Q(MS), C(omprs), V(er), S(screen),  
T(alaris), P(ostScript)  
in.lsfit - perform least squares fit?

NOTE: Must create linear phase and coherence plot in order to use least squares fitting.

in.lslow - lower fitting frequency  
in.lsup - upper fitting frequency  
in.lscoh - use coherence weighting?  
in.lsdev - output device Q(MS), C(omprs), V(er), S(screen),  
T(alaris), P(ostScript)  
in.toscrn - turn printed screen output 'ON' or 'OFF'

#### **Function: cifxover**

Description: This function calls the second half of CIFER's utility 8, the crossover calculations.

[X0db, Xn180db, FRrng] = cifxover(in, options)

#### Inputs:

in - input structure, fields defined below.  
Information on the details of the structure can be found with an empty call:  
>> cifxover  
options - name-value pairs to set individual data fields (optional) e.g.: ..., 'save', 'Y', ...

#### Outputs:

X0db - optional output allows user to set up a template structure using this call with no inputs and a single output:  
>> out = cifxover

\*\*\* OR \*\*\*

X0db - Array with results for 0 deg crossovers  
Xn180db - Array with results for -180 deg crossovers  
FRrng - array containing information about the frequency response. (optional)

FRrng = [minFreq, maxFreq]

NOTE: This function may be called with no outputs specified;

the results will be displayed in the command window.

-----  
DESCRIPTION OF FIELDS IN INPUT STRUCTURE

in.name - Name of Frequency Response (string)  
ex: name = 'XVLATSWP\_FRE\_A0000\_AIL\_P'  
in.source - 'F' for file, 'D' for database  
in.minfreq - Minimum frequency for search range for  
crossover calculations. Enter  
0 for the default range.  
in.maxfreq - Maximum frequency for search range for  
crossover calculations. Enter  
0 for the default range.

NOTE: Set minfreq equal to maxfreq to search for a desired  
fraction of full-range RMS

in.cor\_list - Array with values as follows: (optional)

cor\_list = [scor, gcor, ps, td]

scor - value for power of s correction  
gcor - value for gain correction, >= 0  
ps - value for phase shift  
td - value for time delay

NOTE: an entry of 0 for any of these values is assumed  
to mean no correction, shift, etc. Leaving this  
input out will default the entries to 0.

in.save - 'Y' to save response, 'N' not to save  
in.savename - name to save response as. (optional)  
ex: 'XVLATSWP\_BAN\_A0000\_AIL\_P'

NOTE: If save='Y' and savename is left blank, the  
program portion of the filename will automatically  
be changed to 'BAN'.

in.mpcplt - create a magnitude, phase, (coherence) plot  
in.mpcmin - minimum frequency for mpc plot  
in.mpcmax - maximum frequency for mpc plot  
in.mpcdev - output device Q(MS), C(omprs), V(er), S(creen),  
T(alaris), P(ostScript)  
in.toscrn - turn printed screen output 'ON' or 'OFF'

**Function: cifarith**

Description: This function allows the user to call CIFER  
utility 9 from the Matlab command screen. This utility  
performs arithmetic operations on frequency responses.  
The results are saved to a new frequency response.

[out] = cifarith(in,options)

Inputs:

in - input structure, fields defined below.  
Information on the details of the structure can  
be found with an empty call:  
>> cifarith  
options - name-value pairs to set individual data  
fields (optional) e.g.: ..., 'op', '/', ...

Outputs:  
 out - optional output allows user to set up a template structure using this call with no inputs:  
 >> out = cifarith

---

DESCRIPTION OF FIELDS IN INPUT STRUCTURE

SCREEN 1

in.names(2)	input names
in.scalefac(2)	scale factor
in.spower(2)	power of s
in.op	operation (*,/,,-,+)
in.outname	Resultant Response Name
in.outid	Resultant Response Description
in.cohopt	'N'->COH = 1.0, 'Y' -> COH = COH of first response
in.isource	Source of input responses ('D' - database, 'F' - file)
in.osource	Destination for output ('D' - database, 'F' - file)

SCREEN 2

in.minfreq	Minimum frequency to include (0 for default)
in.maxfreq	Maximum frequency to include (0 for default)
in.nvalue	Number of values in output response (0 for default)
in.unit	'RAD' or 'Hz' (0 for default)
in.toscrn	Turn printed screen output 'ON' or 'OFF'

**Function: cifplot**

Description: This function allows the user to call CIFER utility 19 from the Matlab command screen. This utility generates a 'canned' plot for frequency responses.

[out] = ciplot(in,options)

Inputs:

in	- input structure, fields defined below. Information on the details of the structure can be found with an empty call: >> cifplot
options	- name-value pairs to set individual data fields (optional) e.g.: ..., 'source', 'F', ...

Outputs:

out	- optional output allows user to setup a template structure using this call with no inputs: >> out = cifplot
-----	---

---

DESCRIPTION OF FIELDS IN INPUT STRUCTURE

SCREEN 1

in.array(5)	- which arrays (1:mag, 2:phas, 3:coh, 4:GXX, 5:GYG, 6:GXY, 7:err, 8:pcoh2, 9:pcoh3, 10:pcoh4, 11:pcoh5, 12:pcoh6, 13:pcoh7, 14:pcoh8, 15:pcoh9, 16:mcoh)
in.correct(5)	- use corrections (0 to skip, 1 to apply)
in.names(5)	- response names
in.gain(5)	- gain correction
in.phase(5)	- phase shift correction
in.spower(5)	- s power correction



```

in.source      - source of data 'F' - File, 'D' - Database

SCREEN 2
in.pminfrq    - plot min freq (0 for default)
in.pmaxfrq    - plot max freq (0 for default)
in.pmin(1:5)  - plot y axis mins
in.pmax(1:5)  - plot y axis maxs
in.pinc(1:5)  - plot increments
in.device     - output device Q(MS),C(omprs),V(er),S(creen),
               T(alaris),P(ostScript)
in.grid       - grid lines ('Y' or 'N')
in.thick      - line thickness (1 - Default, 2 - double, etc.)
in.land_port  - L(andscape) or P(ortrait)
in.xaxis      - length of x axis
in.yaxis      - length of y axis

```

## Appendix F: Structure Field Specifics for Analysis Utilities

This Appendix contains the information displayed through an empty call to any of the functions.

Default settings are shown within a curly brace.

```
>> cifrms
```

Structure fields are:

```
name: [ string {''} ]
source: [ { 'D' } | 'F' ]
io: [ { 'I' } | 'O' ]
spower: [ scalar {0} ]
minfreq: [ positive scalar {0} ]
maxfreq: [ positive scalar {0} ]
toscrn: [ { 'ON' } | 'OFF' ]
```

```
>> cifhq
```

Structure fields are:

```
name: [ string {''} ]
source: [ { 'D' } | 'F' ]
minfreq: [ positive scalar {0} ]
cor_list: [ scalar array(4) { [0,0,0,0] } ]
save: [ { 'N' } | 'Y' ]
savename: [ string {''} ]
mpcplt: [ { 'N' } | 'Y' ]
mpcmin: [ positive scalar {0} ]
mpcmax: [ positive scalar {0} ]
mpcdev: [ { 'S' } | 'Q' | 'P' | 'T' | 'C' | 'V' ]
lpcplt: [ { 'N' } | 'Y' ]
lpcmin: [ positive scalar {0} ]
lpcmax: [ positive scalar {0} ]
lpcdev: [ { 'S' } | 'Q' | 'P' | 'T' | 'C' | 'V' ]
lsfit: [ { 'N' } | 'Y' ]
lslow: [ positive scalar {0} ]
lsup: [ positive scalar {0} ]
lscoh: [ { 'N' } | 'Y' ]
lsdev: [ { 'S' } | 'Q' | 'P' | 'T' | 'C' | 'V' ]
toscrn: [ { 'ON' } | 'OFF' ]
```

```
>> cifxover
```

Structure fields are:

```
name: [ string {''} ]
source: [ { 'D' } | 'F' ]
minfreq: [ positive scalar {0} ]
maxfreq: [ positive scalar {0} ]
cor_list: [ scalar array(4) { [0,0,0,0] } ]
save: [ { 'N' } | 'Y' ]
savename: [ string {''} ]
mpcplt: [ { 'N' } | 'Y' ]
mpcmin: [ positive scalar {0} ]
mpcmax: [ positive scalar {0} ]
mpcdev: [ { 'S' } | 'Q' | 'P' | 'T' | 'C' | 'V' ]
toscrn: [ { 'ON' } | 'OFF' ]
```

```
>> cifarith
```

```
Structure fields are:
```

```
names: [ cell array(2) of strings {''} ]
scalefac: [ scalar array(2) {1} ]
spower: [ scalar array(2) {0} ]
op: [ {'*'} | {'/'} | {'-'} | {'+' } ]
outname: [ string {''} ]
outid: [ string {''} ]
cohopt: [ {'N'} | {'Y'} ]
isource: [ {'D'} | {'F'} ]
osource: [ {'D'} | {'F'} ]
minfreq: [ scalar {0} ]
maxfreq: [ scalar {0} ]
nvalue: [ positive scalar {0} ]
unit: [ {0} | 'RAD' | 'Hz' ]
toscrn: [ {'ON'} | {'OFF'} ]
```

```
>> cifplot
```

```
Structure fields are:
```

```
array: [ positive scalar array(5) {0} ]
correct: [ integer array(5) {0} | 1 ]
names: [ cell array(5) of strings {''} ]
gain: [ scalar array(5) {1} ]
phase: [ scalar array(5) {0} ]
spower: [ scalar array(5) {0} ]
source: [ {'D'} | {'F'} ]
pminfrq: [ positive scalar {0} ]
pmaxfrq: [ positive scalar {0} ]
pmin: [ positive scalar array(5) {0} ]
pmax: [ positive scalar array(5) {0} ]
pinc: [ positive scalar array(5) {0} ]
device: [ {'S'} | {'Q'} | {'C'} | {'V'} | {'T'} | {'P'} ]
grid: [ {'Y'} | {'N'} ]
thick: [ positive scalar { 1 } ]
land_port: [ {'L'} | {'P'} ]
xaxis: [ positive scalar { 0 } ]
yaxis: [ positive scalar { 0 } ]
```

## Appendix G: Online Help for Support Functions

This Appendix contains the information that is displayed when the 'help functionname' command is used in Matlab.

### Function: getfr

Description: This function allows the user to access the CIPHER frequency response database and retrieve information from the arrays stored there.

```
[frq,mag,pha,coh,gxx,gyy,gxy,rel,img,err] = getfr(name)
```

#### Inputs:

```
name - Name of Frequency Response (string)
      ex: name = 'XVLATSWP_FRE_A0000_AIL_P'
```

#### Outputs:

```
Arrays of values: (lengths depend on database)
```

```
frq - frequency
mag - magnitude
pha - phase
coh - coherence
gxx - gxx
gyy - gyy
gxy - gxy
rel - real
img - imaginary
err - error
```

### Function: writefr

Description: The user is allowed to place frequency response data back into the CIPHER database.

#### Minimum Input Requirements:

```
writefr(name, frq, mag)
```

#### Maximum Input:

```
writefr(name, frq, mag, pha, coh, gxx, gyy, gxy, rea, ima, err)
```

#### Inputs:

```
name - Name of Frequency Response (string)
      ex: name = 'XVLATSWP_FRE_A0000_AIL_P'
```

```
arrays to put into database:
```

```
frq - frequency
mag - magnitude
pha - phase
coh - coherence
gxx - gxx
gyy - gyy
gxy - gxy
rea - real
ima - imaginary
err - error
```

NOTES: There must be between 3 and 11 inputs. Frequency

and magnitude must be included. Other arrays can be included, but they must be included in the order shown.

**Function: caselist**

Purpose: Allow user to list present CIFER cases by program

names = caselist(pgm)

Inputs:

pgm - program to check cases for

- 1 - FRESPID
- 2 - MISOSA
- 3 - COMPOSITE
- 4 - DERIVID
- 5 - model
- 6 - F matrix
- 7 - G matrix
- 8 - tau matrix
- 9 - H matrix
- 10 - M matrix
- 11 - VERIFY
- 12 - DERIVID results
- 13 - VERIFY results
- 14 - frequency response

Outputs:

names - cell array containing names of cases for specified program

## Appendix H: Mass-Spring-Damper Case Example

This example is for a simple second order mass-spring-damper system modeled in Simulink as a transfer function. A frequency sweep simulating noise was run through the model and the input and output data recorded into a time history file. One case was set up entirely in CIPHER and then a second was set up and run from MATLAB. The commands shown below were used to run the case from MATLAB. Calls to the CIPHER-MATLAB interface functions have been highlighted in red. This example is provided to give a very simple scenario to set up cases using the MATLAB interface. CIPHER does not come with this example so the simulation would have to be created and run in order to generate the data necessary.

```
% Assign a blank frespid structure
f_in = frespid;
thename = 'MASSSPRG';

% Fill in all the necessary information to make the case
f_in.casename = thename;
f_in.comments = 'mass spring system';
f_in.caseout = thename;
f_in.crosscor = 'Y';
% Time history selection parameters:
f_in.source = 5;
f_in.evntnum(1) = 1;
f_in.flghtnum(1) = 1;
f_in.thdt = 0.01;
f_in.thfile(1) = {'massspring.CIFERTEXT'};
% channel definition parameters:
f_in.controls(1) = {'IN'};
f_in.outputs(1) = {'OUT'};
f_in.conchnl(1,1) = {'IN'};
f_in.outchnl(1,1) = {'OUT'};
% Frequency response selection parameters
f_in.frncalc(1,1) = {'*'};

% NOTE: The file 'massspring.CIFERTEXT' was created for this example,
% in order to run the example this file must be created.

% Save the structure into the database
frespid(f_in,2);
% Change the window sizes and turn them on.

frespid('MASSSPRG',2,'winlen',[30,25,20,15,10],'winon',{'*','*','*','*','*'},
'*'),
'maxfft',[125,125,125,125,125]);

% Set up blank composite case
c_in = composite;

% Fill in appropriate values
c_in.casename = thename;
c_in.comments = 'mass spring system';
c_in.casein = thename;
c_in.caseout = thename;
c_in.inpgm = 'FRE';
c_in.controls(1) = {'IN'};
```

```

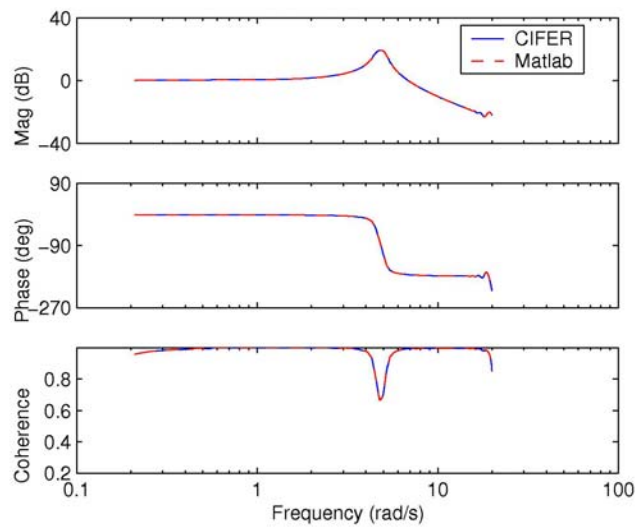
c_in.outputs(1)          = {'OUT'};
c_in.frcalc(1,1)         = {'*'};
c_in.winon               = {'*', '*', '*', '*', '*'};

% Save case into database
composite(c_in,2);

% Run both cases
frespid('MASSSPRG',3)
composite('MASSSPRG',3)

```

The figure below shows the results from the analysis. The response is very clean with a drop in coherence at the mode. Both the MATLAB and CIFER results overlay closely. There is a small difference due to machine precision; however this error tends to be on the order of a thousandth of a percent or less.



## Appendix I: XVLATSWP Case Example

This example shows the MATLAB commands used to fully set up and run the XVLATSWP sample case provided with installations of CIFER.

```
% Assign a blank frespid structure
f_in = frespid;
thename = 'XVLATSP2';

% Fill in all the necessary information to make the case
f_in.casename      = thename;
f_in.comments      = 'Matlab-created XVLATSWP case, new
functions';
f_in.caseout       = thename;
f_in.crosscor      = 'Y';
f_in.plot          = 'N';
% Time history selection parameters:
f_in.source        = 1;
f_in.evntnum(1:2)  = [883,884];
f_in.flghtnum(1:2) = [150,150];
f_in.thdt          = 0.004;
% channel definition parameters:
f_in.controls(1:2) = {'AIL', 'RUD'};
f_in.conunit(1:2)  = {'deg', 'deg'};
f_in.outputs(1:4)  = {'P', 'R', 'AY', 'VDOT'};
f_in.outunit(1:4)  = {'deg/s', 'deg/s', 'ft/sec2',
                    'ft/sec2'};
f_in.conchnl(1:2,1) = {'D645', 'D284'};
f_in.outchnl(1:4,1) = {'V012', 'V014', 'A300', 'VDOT'};
f_in.outscfac(1:3,1) = [0.0175, 0.0175, 32.1740];
% Conditioning parameters
f_in.conditioning(1,1:2) = [3,2];
f_in.conditioning(2,1:2) = [4,25];
% Frequency response selection parameters
f_in.frcalc(1:4,1:2) = {'*'};
% Window Parameters
f_in.winid           = {'45 SECOND WINDOW'
                        '40 SECOND WINDOW'
                        '30 SECOND WINDOW'
                        '20 SECOND WINDOW'
                        '15 SECOND WINDOW'};
f_in.winlen         = [45,40,30,20,15];
f_in.winon(1:5)     = {'*'};
% Save the structure into the database
frespid(f_in,2);

% Set up blank misosa case
m_in = misosa;

% Fill in appropriate values
m_in.casename      = thename;
m_in.comments      = 'Matlab-created XVLATSWP case';
m_in.casein        = thename;
m_in.caseout       = thename;
m_in.controls(1:2) = {'AIL', 'RUD'};
m_in.outputs(1:4)  = {'P', 'R', 'AY', 'VDOT'};
m_in.winon(1:5)    = {'*'};
m_in.frcalc(1:4)   = {'*'};

% save case to database
misosa(m_in,2);
```



```

% Set up blank composite case
c_in = composite;

% Fill in appropriate values
c_in.casename       = thename;
c_in.comments       = 'Matlab-created XVLATSWP case';
c_in.casein        = thename;
c_in.caseout       = thename;
c_in.inpgm         = 'MIS';
c_in.controls(1)   = {'AIL'};
c_in.outputs(1:4)  = {'P', 'R', 'AY', 'VDOT'};
c_in.winon(1:5)    = {'*'};
c_in.frcalc(1,1)   = {'*'};

% Save case into database
composite(c_in,2);

% Run both cases
frespid('XVLATSP2',3);
misosa('XVLATSP2',3);
composite('XVLATSP2',3);

```

The following figure shows a plot of the original XVLATSWP case as created and run in CIFER overlaid by the same case set up and run from MATLAB. There is no appreciable difference between the two results.

