Transfer Function Estimation Using Time-Frequency Analysis

by

Corinne Rachel Ilvedson

S.B., Aeronautics and Astronautics
Massachusetts Institute of Technology, 1996

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1998

© Massachusetts Institute of Technology 1998.
All rights reserved.

Author .....
Department of Aeronautics and Astronautics
August 21, 1998

Certified by .....
Steven R. Hall
Associate Professor
Thesis Supervisor

Accepted by .....
Jaime Peraire
Associate Professor
Chairman, Department Graduate Committee

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
SEP 22 1998
LIBRARIES

Transfer Function Estimation Using Time-Frequency Analysis

by

Corinne Rachel Ilvedson

Submitted to the Department of Aeronautics and Astronautics
on August 21, 1998, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

Abstract

Given limited and noisy data, identifying the transfer function of a complex aerospace system may prove difficult. In order to obtain a clean transfer function estimate despite noisy data, a time-frequency analysis approach to system identification has been developed. The method is based on the observation that for a linear system, an input at a given frequency should result in a response at the same frequency, and a time localized frequency input should result in a response that is nearby in time to the input. Using these principles, the noise in the response can be separated from the physical dynamics. In addition, the impulse response of the system can be restricted to be causal and of limited duration, thereby reducing the number of degrees of freedom in the estimation problem.

The estimation method consists of finding a rough estimate of the impulse response from the sampled input and output data. The impulse response estimate is then transformed to a two dimensional time-frequency mapping. The mapping provides a clear graphical method for distinguishing the noise from the system dynamics. The information believed to correspond to noise is discarded and a cleaner estimate of the impulse response is obtained from the remaining information. The new impulse response estimate is then used to obtain the transfer function estimate.

The results indicate that the time-frequency transfer function estimation method can provide estimates that are often less noisy than those obtained from other methods such as the Empirical Transfer Function Estimate and Welch's Averaged Periodogram Method.

Thesis Supervisor: Steven R. Hall
Title: Associate Professor

Acknowledgments

First of all, I'd like to thank Professor Steve Hall, my advisor, and Professor Feron for their technical support on this project.

My friends at MIT deserve a lot of credit for making the MIT memories that I want to remember... (I'm working on forgetting everything else - except maybe a few equations.) Robby, Atif, Sonia, Brian Schuler, Ernest (falafel!), Jean, and Dora were there to give encouragement and to listen to me go on and on sometimes.... (Brian and Jean – I'm going to stay away from the quicksand!) Thanks to Brian Bingham for keeping me company in athena - we made it! Malinda was there from day one to make the Aero/Astro Department a little more bearable and certainly a lot more fun.

Special thanks to Sharon for always looking out for the students. I'm glad someone is! And I certainly appreciate it.

Tons of thanks go to Professor Paul Lagace who gave so freely of his time. He was always there to provide encouragement and moral support, impart wisdom, give technical advice, shoot the breeze, and to argue about which one of us is the most.. uh... we'll just say orderly.

Mom, Dad, and Aunt Joie & Uncle Pete (the shoe police) each deserve a portion of this degree for being so supportive throughout all of MIT and especially grad school. I appreciate all the encouragement they've given me. And it just helped knowing that someone out there loves me!

Thanks to my housemate Becky who has had to live by herself for the past couple of months. Besides putting up with my stress, she was there to talk, bring me lunch, correct my grammar, and even put out the garbage Thursday night by herself!

Most of all, I owe a lot to Mike. I don't think I would have made it through the last 2 years without Mike. He helped me make the computers do what they were told, he rescued me from the clutches of MIT late at night, he made me dinner so I'd eat, he listened to me vent, he dealt with "The Piano" phenomenon, and he was just always there for me. Thanks so much.

This research was funded by the Department of Defense Graduate Fellowship Program.

Contents

List of Figures

List of Tables

Notation

$\hat{a}$	amplitudes of basis functions used in constructing $\hat{g}$
$a_1, \cdots, a_{n_a}$	polynomial coefficients of $A(q)$
A_{uu}	Toeplitz matrix of sample autocorrelation vector
$A(q)$	parameter polynomial (parametric estimation methods)
$B(q)$	parameter polynomial (parametric estimation methods)
$C(q)$	parameter polynomial (parametric estimation methods)
$D(q)$	parameter polynomial (parametric estimation methods)
$F(q)$	parameter polynomial (parametric estimation methods)
e	disturbance or measurement noise
$\boldsymbol{e}$	vector of sampled measurement noise
E	expected value
f	frequency (Hz)
f_R	frequency resolution of time-frequency mapping (Hz)
F_s	sampling frequency (Hz)
$\mathcal{F}$	Fourier transform
g	impulse response
$\boldsymbol{g}$	vector of sampled impulse response
$\hat{\boldsymbol{g}}$	impulse response estimate
G	input to output transfer function
$\hat{G}$	estimate of input to output transfer function
$\mathcal{G}$	time-frequency transform of impulse response, $g(t)$
H	disturbance to output transfer function
$\mathcal{I}$	information matrix
J_p	performance cost function which weights the difference in the points as opposed to the difference in the area under the curve
J_A	performance cost function which weights the difference in the area under the curve as opposed to the difference in the points
K	total number of points in a hanning window

m	number of previous data points the impulse response depends on (determines duration of the impulse response in time)
n	index indicating the n^{th} data point in a sampled signal
n_s	number of data points in a data block
N	total number of measurements or data points
N_{W}	number of windows or data blocks
p	index indicating the p^{th} of N_{W} windows
q	shift operator ($q = e^{j\omega}$)
s	Laplace transform ($s = j\omega$)
t	time (seconds)
t_R	time resolution of time-frequency mapping (seconds)
T	end time of data (seconds)
$\mathbf{T}$	transformation matrix of basis functions
u	system input
$\boldsymbol{u}$	vector of sampled system input
$\mathbf{U}$	convolution matrix of sampled inputs
$\mathbf{U}_e$	extra terms in input convolution matrix
U_N	Fourier transform of input u(t)
V_N	parametric estimation error criterion
w	window such as Hanning or boxcar
x	signal in the time domain
X	signal in the frequency domain
y	system output
$\boldsymbol{y}$	vector of sampled system output
$\boldsymbol{y}_e$	extra terms in vector of sampled system output
Y_N	Fourier transform of output y(t)
δ_{mn}	dirac delta function
$\Delta f_{\hat{G}}$	frequency resolution of transfer function estimate (Hz)
Δt	sampling interval (seconds)

θ	set of parameters to be estimated (parametric estimation methods)
σ	variance of noise
τ	time (seconds)
ϕ	basis function
ϕ_{uu}	sample autocorrelation vector of u
ϕ_{yu}	sample cross-correlation vector of y and u
Φ_{uu}	power spectral density of u
Φ_{yu}	cross-spectral density of y and u
χ	information vector
ω	frequency (rad/second)
$(\ )_s$	relating to symmetric data set
$(\ )_a$	relating to asymmetric data set

Chapter 1

Introduction

1.1 Motivation

When working with complex aerospace systems such as airplanes and helicopters, system identification is an important tool for identifying the system dynamics. Accurate identification is crucial in such procedures as safety flight testing or in designing controllers. However, these systems are often nonlinear and tend to operate in environments with large disturbances such as turbulence, which in addition to sensor noise, can lead to very noisy test data. Furthermore, flight testing and wind tunnel testing is extremely costly and because there is often limited time at a test facility, the amount of available data is limited. Even when a linear analysis is valid, given the limited and noisy data, obtaining an accurate estimate of a system transfer function can be difficult. Therefore, there is a need for a method that will quickly estimate the transfer function despite these limitations. An efficient and reliable method for more accurate estimation would not only save testing time but also reduce testing costs.

1.1.1 System Identification Techniques

Although there are many transfer function estimation techniques available, given data limitations, these may yield poor results. One such method is the Empirical Transfer Function Estimate (ETFE), which estimates the transfer function by taking the ratios

of the Fourier transforms of the output $y(t)$ and the input $u(t)$. The estimate is given by

$$\hat{G}(\omega) = \frac{\mathcal{F}(y(t))}{\mathcal{F}(u(t))} \tag{1.1}$$

If the data set is noisy, the resulting estimate is also noisy. Unfortunately, taking more data points does not help. The variance does not decrease as the number of data points increase because there is no feature of information compression. There are as many independent estimates as there are data points [1].

Parametric estimation methods are another class of system identification techniques. The motivation behind these methods is to be able to find an estimate or model of the system in terms of a small number (compared to the number of measurements) of numerical values or parameters, θ. A linear system is typically represented by

$$y(t) = G(q)u(t) + H(q)e(t) \tag{1.2}$$

where $e(t)$ is the disturbance, $G(q)$ is the transfer function from input to output, $H(q)$ is the transfer function from disturbance to output, and q is the shift operator ($q = e^{j\omega}$) used when dealing with discrete systems. The most generalized model structure is

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t) \tag{1.3}$$

where $A(q)$, $B(q)$, $C(q)$, $D(q)$, $F(q)$ are all parameter polynomials to be estimated. For example,

$$A(q) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \tag{1.4}$$

The coefficients of all the polynomials make up the set of parameters to be estimated, θ. By using some of the polynomials and setting the others equal to 1, popular estimation model structures such as ARX (Autoregressive Extra Input), ARMA (Autoregressive Moving Average), and OE (Output Error) can be obtained. However, when using these methods, the estimate can be poor at low frequencies. For example,

the ARX method is given by

$$A(q)y(t) = B(q)u(t) + e(t) \tag{1.5}$$

In estimating the parameters θ, the error criterion to be minimized is

$$V_N(\theta) = \sum_{\omega} \left| A(e^{jw})Y_N(\omega) - B(e^{jw})U_N(\omega) \right|^2 \tag{1.6}$$

where Y_N and U_N are Fourier transforms of $y(t)$ and $u(t)$. If the error criterion is rewritten as

$$V_N(\theta) = \sum_{\omega} \left| \frac{Y_N(\omega)}{U_N(\omega)} - \hat{G}(e^{jw}, \theta) \right|^2 |U_N(\omega)|^2 \left| A(e^{jw}) \right|^2 \tag{1.7}$$

it becomes evident that the minimization is being weighted by $|A(e^{jw})|^2$. Typically, system transfer functions roll off at higher frequencies. Conversely, the denominator $A(e^{jw})$ would increase at higher frequencies, as shown in Figure 1-1. Therefore, the error criterion is weighted more heavily at higher frequencies causing the transfer function to be estimated poorly at low frequencies [1, 2]. In addition, obtaining an estimate using these parametric estimation methods can be very time consuming for the engineer.

Because of problems using these methods to estimate transfer functions from noisy data sets, new methods are being explored.

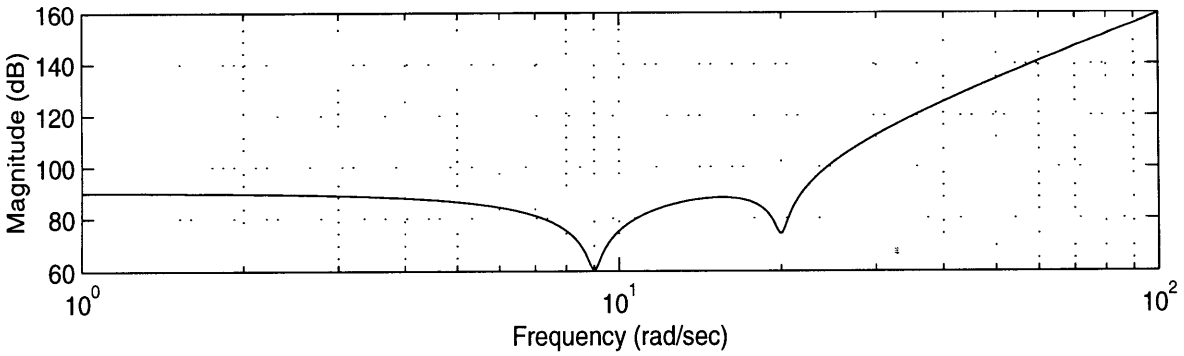


Figure 1-1: Typical transfer function denominator

Time-Frequency Analysis Techniques

In working to identify F-18 flutter boundaries, a new way of estimating transfer functions using a time-frequency analysis approach was developed recently by Paternot [3], Feron *et al.* [4], and Turevskiy [5]. The F-18 flight tests used a chirp excitation signal, a signal of sinusoidal shape with linear or logarithmic modulated frequency. In a time-frequency mapping, the chirp signal transforms very clearly, as shown in Figure 1-2. The darkest band corresponds to the frequency sweep, whereas the other bands are secondary harmonics. Any other spots are due to noise. A straight band indicates that the frequency sweep was linear in this data set. If the band were curved, this would indicate a logarithmic sweep. Since the system to be identified is assumed to be linear, any information in the output signal that occurred at a certain time and frequency must correspond to input information at the same time and frequency. If not, it is most likely an artifact of noise. Therefore, one can visually determine which information is due to system dynamics and which is due to noise. The input and output data is then cleaned of the identified noise, thereby allowing for an enhanced transfer function estimate. Feron *et al.* found that the time-frequency estimation method performed better than other system identification techniques such as Fourier analysis, Prediction Error Method (PEM), and subspace identification [4].

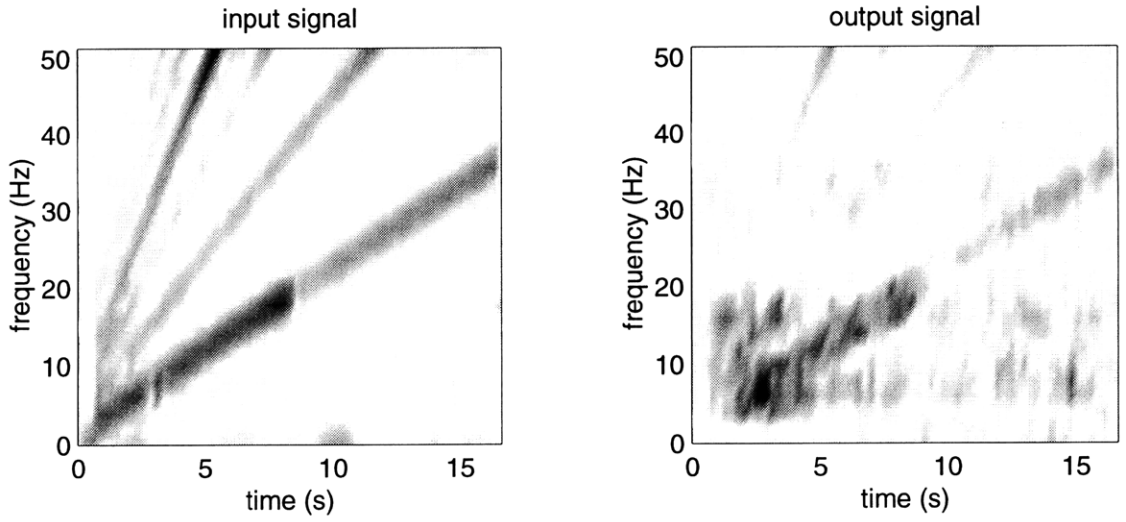


Figure 1-2: Time-frequency mapping of chirp signal and output

1.2 Thesis Objective and Overview

Feron *et al.* succeeded in obtaining cleaner transfer function estimates by making two important observations regarding the response of a linear system: an input at a given frequency should result in a response at the same frequency, and a time localized frequency input should result in a response that is nearby in time to the input. Therefore, any part of the response of a linear system that doesn't follow these observations must be due to the influence of noise. Based on these principles, they developed a time-frequency analysis method that utilized the structure of a chirp input signal to distinguish the system dynamics from the noise.

The objective of this thesis is to further the work of Feron *et al.* by developing a more generalized approach to time-frequency transfer function estimation that will accept data sets with any type of input signal. The method is demonstrated on a numerical example as well as experimental data both of which are identical to those used in the literature [3, 4, 5]. In addition, the performance is compared with other popular transfer function estimation techniques.

The subjects contained specifically in each chapter are as follows. Chapter 2 presents the theory behind the time-frequency analysis and demonstrates the method using a numerical example. A trade-off study is presented to give insight into how to set the method parameters. The results are compared with those obtained from other system identification methods. Chapter 3 demonstrates the method on experimental multi-input data. Finally, Chapter 4 discusses the results and makes suggestions for further improvements to the method. The time-frequency analysis method was implemented as a tool in MATLAB. The code and instructions can be found in the Appendices.

Chapter 2

Transfer Function Estimation

2.1 Time-Frequency Signal Analysis

A common and useful way to determine the frequency content of a sampled signal is to take the discrete Fourier transform

$$X(f) = \sum_{n=-\infty}^{\infty} x_n e^{-j2\pi f n \Delta t} \tag{2.1}$$

where $x_n = x(n\Delta t)$ and Δt is the sampling rate. Since test data sets are finite, it is assumed that the only non-zero data is during the test when $t = [0, T]$. Therefore, the discrete Fourier transform of the signal becomes

$$X(f) = \sum_{n=0}^{N-1} x_n e^{-j2\pi f n \Delta t} \tag{2.2}$$

where N is the total number of data points. By the nature of the time to frequency transformation, the resulting $X(f)$ only contains information about the frequency content of the signal x and therefore obscures the time behavior.

However, there are some situations where it is useful to have temporal information about a signal's frequency content. An example from everyday life is music. We perceive the music both in terms of time and frequency, notes of given duration and frequency. Analyzing such a signal purely in the time domain or purely in the

frequency domain surely misses some important information.

Estimating the transfer function and impulse response of a system from test data is another situation where is is useful to have simultaneous time and frequency information. When estimating the impulse response, it is reasonable to make the restriction that the impulse response be causal and time limited. In addition, the frequency content of the transfer function should only include those frequencies that are physical realizations of the system, as opposed to noise or outside disturbances. Therefore, in order to restrict both the time and frequency characteristics of the data, there is a need for a time-frequency representation.

The general representation of a time-frequency transform is

$$X(f,p) = \sum_{p=1}^{N_{\mathrm{W}}} \sum_{n=0}^{N-1} (x_n \cdot \mathrm{w}_{p_n}) e^{-j2\pi f n \Delta t} \tag{2.3}$$

where w_p is a set of N_{W} windows that filter sections of x. The set of windows essentially divide the signal into several segments or data blocks. The resulting representation of x is in matrix form, containing frequency content information for each windowed segment of the signal. (The details of the windowing method are discussed in Section 2.2.2.) This time-frequency transform can be used to provide simultaneous time and frequency information about a signal, thereby enabling the engineer to make the necessary restrictions for producing an enhanced transfer function estimate.

2.2 Transfer Function Estimation

The proposed method for system identification using a time-frequency analysis approach is briefly described below, followed by a detailed presentation of the theory.

The time-frequency analysis method starts by finding a rough estimate of the impulse response. This estimate is then mapped to a time-frequency representation using the transform of Equation 2.3. The mapping is used to distinguish the signal information that is due to system dynamics from that due to noise or disturbances. The information that is a physical realization of the system is retained, while infor-

mation believed to be due to noise is discarded. The remaining signal information is used to reconstruct a cleaner estimate of the impulse response, which in turn can be used to obtain a transfer function estimate.

2.2.1 Estimating the Impulse Response

The impulse response is related to the system input and output by

$$\begin{aligned} y(t) &= g(t) * u(t) + e(t) \\ &= \int_{-\infty}^{\infty} g(\tau)u(t-\tau)d\tau + e(t) \end{aligned} \tag{2.4}$$

where $*$ is the convolution operator, $y(t)$ is the output, $g(t)$ is the impulse response, $u(t)$ is the input, and $e(t)$ is the disturbance or noise. Because the input and output data is sampled and finite, the convolution integral of Equation 2.4 must be approximated in discrete time as the sum

$$y_n = \sum_{k=0}^{m-1} g_k u_{n-k} \Delta t + e_n \qquad n = m-1, \cdots, N-1 \tag{2.5}$$

where each output data point depends on m previous input data points, N is the total number of sampled data points and the noise, e_n, is assumed to be white noise with zero mean and variance σ^2. The convolution of Equation 2.5 can be rewritten in matrix form as

$$\begin{Bmatrix} y_{m-1} \\ y_m \\ y_{m+1} \\ \vdots \\ y_{N-1} \end{Bmatrix} = \Delta t \begin{bmatrix} u_{m-1} & u_{m-2} & \cdots & u_0 \\ u_m & u_{m-1} & \cdots & u_1 \\ u_{m+1} & u_m & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{N-1} & u_{N-2} & \cdots & u_{N-m} \end{bmatrix} \begin{Bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{m-1} \end{Bmatrix} + \begin{Bmatrix} e_{m-1} \\ e_m \\ e_{m+1} \\ \vdots \\ e_{N-1} \end{Bmatrix} \tag{2.6}$$

or

$$\boldsymbol{y} = \mathbf{U}\boldsymbol{g} + \boldsymbol{e}$$

An estimate of the impulse response $\hat{\boldsymbol{g}}$ that minimizes the squared estimation error $|\boldsymbol{y} - \mathbf{U}\hat{\boldsymbol{g}}|^2$ is desired. The least squares estimate is found by taking the pseudo-inverse, which is given by

$$\hat{\boldsymbol{g}} = (\mathbf{U}^T\mathbf{U})^{-1}(\mathbf{U}^T\boldsymbol{y}) \tag{2.7}$$

and can be rewritten as

$$\hat{\boldsymbol{g}} = (\mathcal{I})^{-1}\chi \tag{2.8}$$

where $\mathcal{I}$ and χ are the information matrix and vector respectively. They are defined as

$$\mathcal{I} = \mathbf{U}^T\mathbf{U} \tag{2.9}$$

$$\chi = \mathbf{U}^T\boldsymbol{y} \tag{2.10}$$

When the data sets (and therefore m and N) become large, the solution of Equation 2.7 becomes computationally expensive. In order to compute $\mathbf{U}^T\mathbf{U}$, it takes approximately N multiplications to calculate each of the m^2 terms. Reducing the number of multiplications would reduce the cost of the calculation, which could make working with large data sets more manageable. Writing out the terms of $\mathbf{U}^T\mathbf{U}$ as

$$\mathbf{U}^T\mathbf{U} = (\Delta t)^2 \sum_{i=m-1}^{N-1} \begin{bmatrix} u_iu_i & u_{i-1}u_i & \cdots & u_{i-(m-1)}u_i \\ u_iu_{i-1} & u_{i-1}u_{i-1} & \cdots & u_{i-(m-1)}u_{i-1} \\ \vdots & \vdots & \ddots & \vdots \\ u_iu_{i-(m-1)} & u_{i-1}u_{i-(m-1)} & \cdots & u_{i-(m-1)}u_{i-(m-1)} \end{bmatrix} \tag{2.11}$$

it is evident that $\mathbf{U}^T\mathbf{U}$ is almost Toeplitz. In fact, $\mathbf{U}^T\mathbf{U}$ would be a Toeplitz matrix if the limits of the sum in Equation 2.11 were changed to be

$$\mathbf{U}^T\mathbf{U} \approx A_{uu} \equiv (\Delta t)^2 \sum_{i=0}^{N-1} \begin{bmatrix} u_iu_i & u_{[i-1]}u_i & \cdots & u_{[i-(m-1)]}u_i \\ u_iu_{[i-1]} & u_{[i-1]}u_{[i-1]} & \cdots & u_{[i-(m-1)]}u_{[i-1]} \\ \vdots & \vdots & \ddots & \vdots \\ u_iu_{[i-(m-1)]} & u_{[i-1]}u_{[i-(m-1)]} & \cdots & u_{[i-(m-1)]}u_{[i-(m-1)]} \end{bmatrix} \tag{2.12}$$

where a bracket around an index indicates that the index is evaluated mod N. For

example, if $i = 1$ then $u_{[i-2]} = u_{-1} = u_{N-1}$. Equation 2.12 can be simplified to

$$A_{uu} = (\Delta t)^2 \sum_{i=0}^{N-1} \begin{bmatrix} u_i u_i & u_i u_{[i+1]} & \cdots & u_i u_{[i+(m-1)]} \\ u_i u_{[i+1]} & u_i u_i & \cdots & u_i u_{[i+(m-2)]} \\ \vdots & \vdots & \ddots & \vdots \\ u_i u_{[i+(m-1)]} & u_i u_{[i+(m-2)]} & \cdots & u_i u_i \end{bmatrix} \tag{2.13}$$

by a change of index variable for each term of the matrix. The first column of A_{uu} is recognized to be $N\Delta t$ times the sample autocorrelation vector of u, which is given by

$$\phi_{(uu)_n} = \frac{1}{N} \sum_{k=0}^{N-1} u_k u_{[n+k]} \Delta t \qquad n = 0, \cdots, N-1 \tag{2.14}$$

Therefore, it is possible that the sample autocorrelation vector could be used to calculate $\mathbf{U}^T\mathbf{U}$ more efficiently. Due to the symmetries of the Toeplitz matrix of Equation 2.13, only m terms would have to be calculated as opposed to m^2. However, a correction term is needed, since A_{uu} and $\mathbf{U}^T\mathbf{U}$ are not exactly equal. Rewriting Equation 2.6 to include the first $m-1$ terms of $\boldsymbol{y}$, the convolution matrix becomes

$$\begin{Bmatrix} y_0 \\ \vdots \\ y_{m-2} \\ \hline y_{m-1} \\ y_m \\ y_{m+1} \\ \vdots \\ y_{N-1} \end{Bmatrix} = \Delta t \left[\begin{array}{cccc} u_0 & u_N & \cdots & u_{N-m+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m-2} & u_{m-3} & \cdots & u_N \\ \hline u_{m-1} & u_{m-2} & \cdots & u_0 \\ u_m & u_{m-1} & \cdots & u_1 \\ u_{m+1} & u_m & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{N-1} & u_{N-2} & \cdots & u_{N-m} \end{array}\right] \begin{Bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{m-1} \end{Bmatrix} + \begin{Bmatrix} e_0 \\ \vdots \\ e_{m-2} \\ \hline e_{m-1} \\ e_m \\ e_{m+1} \\ \vdots \\ e_{N-1} \end{Bmatrix} \tag{2.15}$$

or

$$\begin{Bmatrix} \boldsymbol{y}_e \\ \boldsymbol{y} \end{Bmatrix} = \begin{bmatrix} \mathbf{U}_e \\ \mathbf{U} \end{bmatrix} \boldsymbol{g} + \boldsymbol{e}$$

where $\boldsymbol{y}_e$ and $\mathbf{U}_e$ denote the extra $m-1$ terms. The information matrix $\tilde{\mathcal{I}}$ corre-

sponding to Equation 2.15 is given by

$$\tilde{\mathcal{I}} = \begin{bmatrix} \mathbf{U}_e & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{U}_e \\ \mathbf{U} \end{bmatrix} = \mathbf{U}_e^T \mathbf{U}_e + \mathbf{U}^T \mathbf{U} \tag{2.16}$$

and turns out to be exactly equal to A_{uu} as given by Equation 2.13. Therefore, $\mathbf{U}^T\mathbf{U}$ can be related to A_{uu} by

$$\mathbf{U}^T\mathbf{U} = A_{uu} - \mathbf{U}_e^T\mathbf{U}_e \tag{2.17}$$

$\mathbf{U}^T\boldsymbol{y}$ can be calculated in a similar fashion using the sample cross-correlation vector, defined by

$$\phi_{(yu)_n} = \frac{1}{N}\sum_{k=0}^{N-1} y_k u_{[n+k]}\Delta t \qquad\qquad n = 0, \cdots, N-1 \tag{2.18}$$

The relationship between $\mathbf{U}^T\boldsymbol{y}$ and ϕ_{yu} is given by

$$\mathbf{U}^T\boldsymbol{y} = \phi_{yu} - \mathbf{U}_e^T\boldsymbol{y}_e \tag{2.19}$$

In computing A_{uu}, there are $N \cdot m$ multiplications and in computing $\mathbf{U}_e^T\mathbf{U}_e$ there are $m \cdot m^2$ multiplications, as opposed to the $N \cdot m^2$ multiplications in computing $\mathbf{U}^T\mathbf{U}$. Since N is generally at least an order of magnitude larger than m, computing the information matrix using Equation 2.17 can amount to substantial savings in computation time. Therefore, substituting Equations 2.17 and 2.19 into Equation 2.7 provides a more computationally inexpensive estimate of the impulse response.

Note that the Fourier transforms of the correlations, ϕ_{yu} and ϕ_{uu}, are the cross-spectral density Φ_{yu} and the power spectral density Φ_{uu}. Since calculating the information matrix and vector of Equation 2.9 and Equation 2.10 is very close to calculating the correlations, ϕ_{yu} and ϕ_{uu}, it follows that estimating the impulse response in the time domain using

$$\hat{\boldsymbol{g}} = (\mathbf{U}^T\mathbf{U})^{-1}(\mathbf{U}^T\boldsymbol{y}) \tag{2.20}$$

is analogous to using spectral analysis in the frequency domain to calculate the Em-

pirical Transfer Function Estimate (ETFE) given by

$$\hat{G}(\omega) = \Phi_{uu}(w)^{-1}\Phi_{yu}(w) \tag{2.21}$$

Multi-Input Systems

The methodology for estimating the impulse response can be expanded to include multi-input systems. If

$$y(t) = g_1(t) * u_1(t) + g_2(t) * u_2(t) + e(t) \tag{2.22}$$

then Equation 2.6 becomes

$$\boldsymbol{y} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{Bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \end{Bmatrix} + \boldsymbol{e} \tag{2.23}$$

and Equation 2.7 becomes

$$\begin{Bmatrix} \hat{\boldsymbol{g}}_1 \\ \hat{\boldsymbol{g}}_2 \end{Bmatrix} = \begin{bmatrix} \mathbf{U}_1^T\mathbf{U}_1 & \mathbf{U}_1^T\mathbf{U}_2 \\ \mathbf{U}_2^T\mathbf{U}_1 & \mathbf{U}_2^T\mathbf{U}_2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{U}_1^T \\ \mathbf{U}_2^T \end{bmatrix} \boldsymbol{y} \tag{2.24}$$

Therefore, Equations 2.17 and 2.19 become

$$\begin{bmatrix} \mathbf{U}_1^T\mathbf{U}_1 & \mathbf{U}_1^T\mathbf{U}_2 \\ \mathbf{U}_2^T\mathbf{U}_1 & \mathbf{U}_2^T\mathbf{U}_2 \end{bmatrix} = \begin{bmatrix} A_{u_1u_1} & A_{u_1u_2} \\ A_{u_2u_1} & A_{u_2u_2} \end{bmatrix} - \begin{bmatrix} \mathbf{U}_{1e}^T\mathbf{U}_{1e} & \mathbf{U}_{1e}^T\mathbf{U}_{2e} \\ \mathbf{U}_{2e}^T\mathbf{U}_{1e} & \mathbf{U}_{2e}^T\mathbf{U}_{2e} \end{bmatrix} \tag{2.25}$$

$$\begin{bmatrix} \mathbf{U}_1^T \\ \mathbf{U}_2^T \end{bmatrix} \boldsymbol{y} = \begin{Bmatrix} \phi_{yu_1} \\ \phi_{yu_2} \end{Bmatrix} - \begin{bmatrix} \mathbf{U}_{1e}^T \\ \mathbf{U}_{2e}^T \end{bmatrix} \boldsymbol{y} \tag{2.26}$$

Substituting Equations 2.25 and 2.26 into Equation 2.24 provides the estimates of the impulse responses for multi-input systems.

2.2.2 Time-Frequency Decomposition

After obtaining an estimate of the impulse response, it is mapped from a one dimensional representation in time to a two dimensional representation in time and frequency. In order to achieve this, the time-frequency transform of Equation 2.3 is used. The transform of the impulse response is given by

$$\mathcal{G}(f,p) = \sum_{p=1}^{N_{\mathrm{W}}} \sum_{n=0}^{N-1} (g_n \cdot \mathrm{w}_{p_n}) e^{-j2\pi f n \Delta t} \tag{2.27}$$

The purpose of the set of windows, w_p, is to divide the impulse response into N_{W} segments or data blocks and to simultaneously filter each block.

Figure 2-1 illustrates the various ways the signal could be segmented. The basic decomposition method is to divide the signal into adjacent data blocks as shown in Figure 2-1a. In each block, any choice of filters or windowing techniques could be used. The simplest is a boxcar window. However, when the Fourier transform of each segment is taken, frequency spreading is introduced into the results due to the discontinuities at the ends of each segment. In order to smooth out this effect, a Hanning window may be used instead, as shown in Figure 2-1b. The Hanning window is described by

$$\mathrm{w}_{\mathrm{Hanning}}(k;K) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi k}{K}\right)\right] \qquad k = 0, 1, \cdots, K-1 \tag{2.28}$$

Because each segment of the signal decays continuously to zero at each side of the data block, the Fourier transform frequency spreading is eliminated. However, a Hanning window forces the signal to equal zero at the junctions of each block. In order to allow for non-zero values at these boundaries, a second set of overlapping segments can be added as shown in Figure 2-1c. Notice that the first and last blocks are only half the size of the others in order to allow for nonzero values at the beginning and end of the impulse response. For the time-frequency analysis used in this thesis, overlapping Hanning windows were used.

Using the expression for a Hanning window given in Equation 2.28, the set of

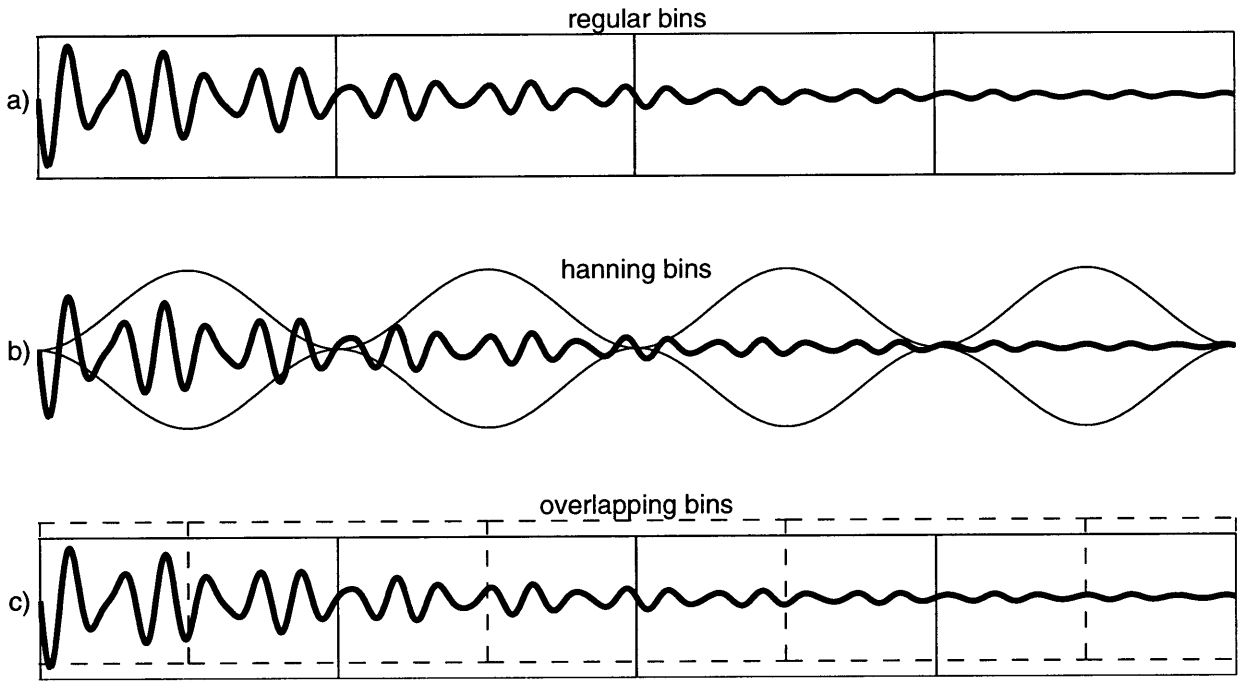


Figure 2-1: Division of impulse response into data blocks

overlapping windows can be represented mathematically by

$$
\begin{aligned}
&p = 1 \\
&\qquad \mathrm{w}_{p_n} = \begin{cases} \mathrm{w}_{\text{Hanning}}\left(n + \frac{n_s}{2}; n_s\right) & 0 \le n < \frac{n_s}{2} \\ 0 & \text{else} \end{cases} \\
&p = 2, \cdots, N_{\mathrm{W}} - 1 \\
&\qquad \mathrm{w}_{p_n} = \begin{cases} \mathrm{w}_{\text{Hanning}}\left(n - (p-2)\frac{n_s}{2}; n_s\right) & (p-2)\frac{n_s}{2} \le n < p\frac{n_s}{2} \\ 0 & \text{else} \end{cases} \\
&p = N_{\mathrm{W}} \\
&\qquad \mathrm{w}_{p_n} = \begin{cases} \mathrm{w}_{\text{Hanning}}\left(n - (N - \frac{n_s}{2}); n_s\right) & N - \frac{n_s}{2} \le n < N \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{2.29}
$$

where $n = 0, \cdots, N-1$. The number of data points in a segment, n_s, is determined by

$$
n_s = \begin{cases} \frac{m}{N_{\mathrm{W}}} & \text{if adjacent data blocks (Figure 2-1a,b)} \\ 2\left(\frac{m}{N_{\mathrm{W}}-1}\right) & \text{if overlapping data blocks (Figure 2-1c)} \end{cases}
\tag{2.30}
$$

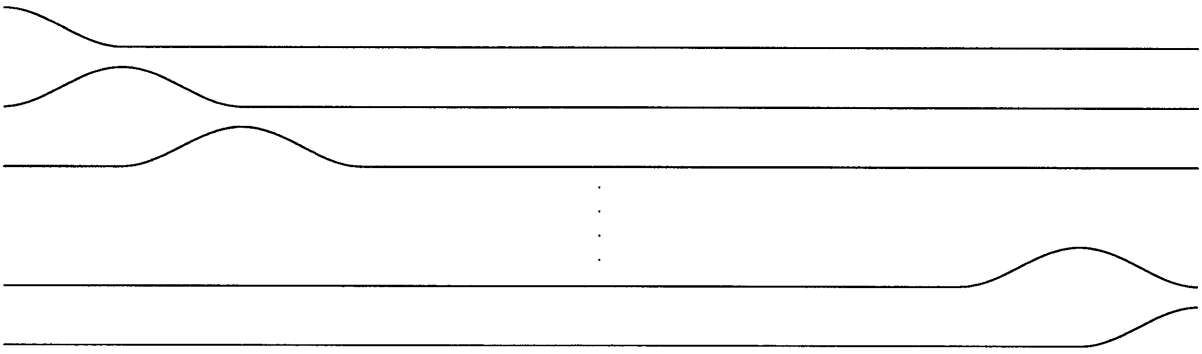


Figure 2-2: Set of Hanning windows used in the time-frequency decomposition

This set of overlapping Hanning windows is represented graphically in Figure 2-2.

The time-frequency transform may be represented graphically as shown in Figure 2-3, in which the magnitude of each time-frequency pair is represented by the intensity of that bin. The magnitude of each bin is given by the absolute value of the Fourier coefficient for that segment of the impulse response at that particular frequency. The time-frequency mapping provides an easy way to view the energy of the signal. For example, in Figure 2-3 there is energy around 5-10 Hz. Note that this band of bright squares looks organized and decays in an exponential fashion, which is what one would expect for an impulse response. The bright bins near the top look random, and are probably due to noise in the signal.

Each bin of the time-frequency mapping has a basis function associated with it. Regrouping the terms in Equation 2.27, the time-frequency transform can be rewritten as

$$\mathcal{G}_{k,p} = \sum_{p=1}^{N_W} \sum_{n=0}^{N-1} g_n(\mathrm{w}_{p_n} \cdot e^{-j\frac{2\pi kn}{N}}) \tag{2.31}$$

making it evident that the time-frequency transform projects the impulse response onto windowed sinusoidal basis functions. These basis functions are defined by

$$\phi_{k,p} = \mathrm{w}_{p_n} \cdot e^{j\frac{2\pi kn}{N}} \tag{2.32}$$

Figure 2-4 shows the first five basis functions of the second data block for the boxcar windowed and Hanning windowed segments of Figure 2-1.

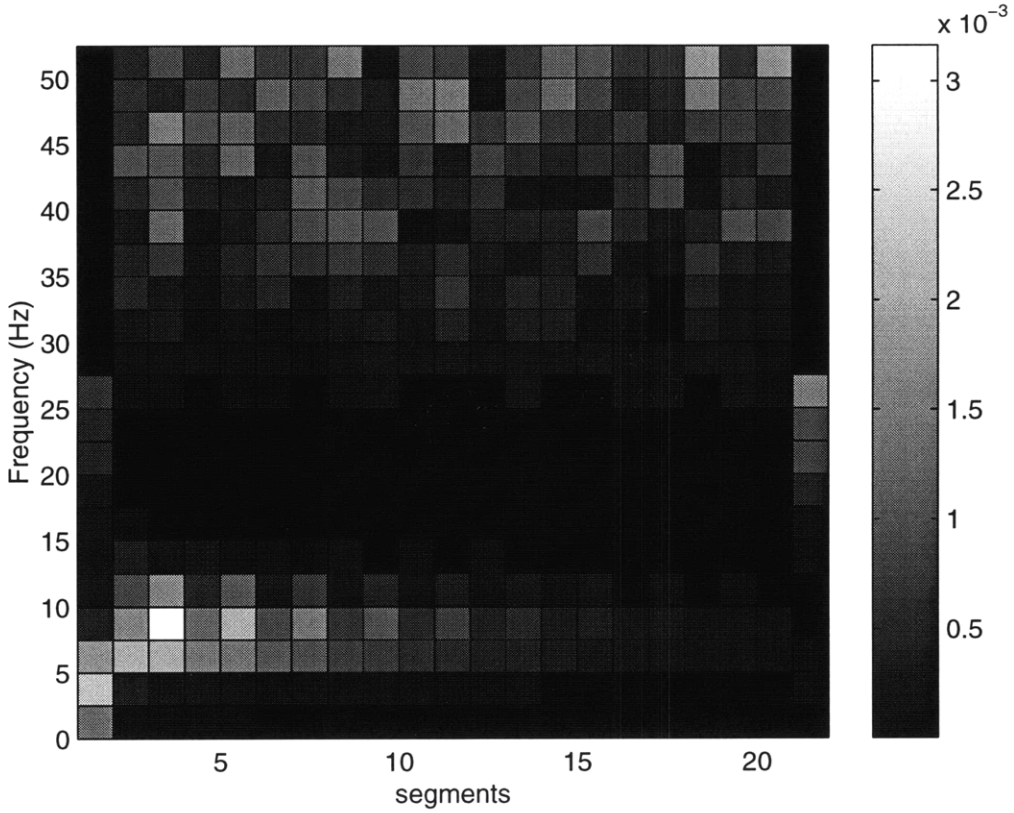


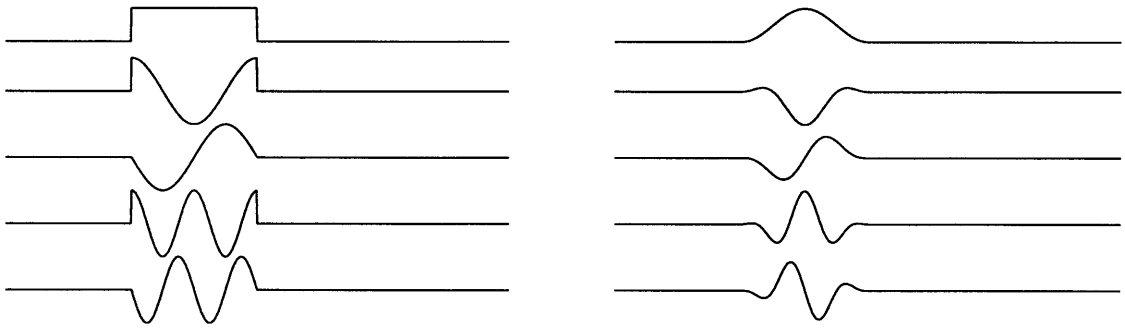
Figure 2-3: Two dimensional time-frequency mapping of a signal

2.2.3 Signal Noise Removal and Transfer Function Estimation

By looking at the time-frequency mapping of the signal, decisions can be made to determine which bins to keep and which to throw out. For example, in Figure 2-3, the band of bright bins at the bottom would be kept because the resonances there are most likely from the system dynamics, whereas the bright bins near the top would be thrown out because they are most likely due to noise.

After choosing which bins to keep, a new estimate impulse response must be reconstructed. By throwing out some of the degrees of freedom, the magnitudes of the Fourier coefficients for each bin will change along with the information matrix and vector, $\mathcal{I}$ and χ. The new magnitudes, information matrix and vector can be found from the old via a transformation matrix made up of the basis functions corresponding to the degrees of freedom that are kept. The transformation matrix is

$$\mathbf{T} = \begin{bmatrix} \phi_{n_1} & \phi_{n_2} & \cdots & \phi_{n_p} \end{bmatrix} \tag{2.33}$$



a) b)

Figure 2-4: Example basis functions: a) boxcar window b) Hanning window

where p is the number of basis functions used, and n_i are the indices of the retained basis functions. In the case of multi-input systems, the transformation matrix is given by

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_{g_1} & 0 \\ 0 & \mathbf{T}_{g_2} \end{bmatrix} \tag{2.34}$$

where $\mathbf{T}_{g_1}$ is a set of basis functions corresponding to g_1 and $\mathbf{T}_{g_2}$ is a set corresponding to g_2. The new information matrix and vector are then

$$\bar{\mathcal{I}} = \mathbf{T}^T \mathcal{I} \mathbf{T} \tag{2.35}$$

$$\bar{\chi} = \mathbf{T}^T \chi \tag{2.36}$$

and the new amplitudes are

$$\hat{a} = \bar{\mathcal{I}}^{-1} \bar{\chi} \tag{2.37}$$

Therefore, the new estimate of the impulse response is

$$\hat{\bar{g}} = \mathbf{T}\hat{a} \tag{2.38}$$

The estimate of the transfer function is then obtained by

$$\hat{G}(f) = \mathcal{F}\left(\hat{\bar{g}}(t)\right) \tag{2.39}$$

2.2.4 Specifying Method Parameters

In using the time-frequency method, there are two parameters to be set. The first parameter is m, the number of previous input data points each output data point depends on. In other words, m is the number of data points that the impulse response estimate will have, and therefore determines the duration of the impulse response. The second parameter is N_{W}, which determines the number of segments or data blocks of the impulse response. As defined in Equation 2.30, N_{W} also determines the number of data points, n_s, in each data block.

At the start of the estimation problem, there are as many degrees of freedom to estimate as there are data points in the sampled input and output data. After setting the parameter m, the duration of the impulse response is restricted to $m\Delta t$ and therefore, there are only m degrees of freedom in the estimation problem.

After setting N_{W}, the signal is split into data blocks and then transformed to the time-frequency mapping. When adjacent data blocks are used, the mapping will be a matrix of bins that is $\frac{n_s}{2}$ rows by N_{W} columns. When overlapping segments are used, there are N_{W} columns with $\frac{n_s}{4}$ rows in each column with the exception of the first and last which have only $\frac{n_s}{8}$ rows. This is because the first and last columns represent the half-size data blocks located at the beginning and end of the impulse response.

The frequency and time resolutions, f_R and t_R, of the time-frequency mapping are therefore determined by the choices of m and N_{W}. The resolutions are given by

$$f_R = \frac{F_s}{n_s} \qquad\qquad t_R = n_s \Delta t \tag{2.40}$$

where F_s is the sampling frequency and n_s is a function of m as defined in Equation 2.30. Note that in the time-frequency transformation, the number of degrees of freedom is preserved. The time-frequency mapping matrix has a total of $\frac{m}{2}$ bins where each bin represents two basis functions (a windowed sine and cosine) for a total of m degrees of freedom.

Therefore, in choosing m and N_{W}, there is a tradeoff between frequency resolution and time resolution. Which resolution to favor depends on the estimation problem.

For example, in estimating a transfer function that has two peaks that are close in frequency ω_n with small and approximately equal damping ζ, it may be better to favor frequency resolution. Because the peaks are close in frequency, a course frequency resolution would merge the information corresponding to each peak. In addition, the decay of the vibrations at each natural frequency is governed by $e^{-\zeta\omega_n t}$ and since the natural frequency and damping of both resonances are approximately the same, the vibrations at each frequency will decay at approximately the same rate. Therefore, a time resolution that captures the information corresponding to one of the peaks will also be able to capture the other.

However, if the transfer function to be estimated has peaks spaced further apart with a large difference in damping, it may be better to favor time resolution. Because the vibrations due to one of the resonances will decay much faster, inadequate time resolution could result in missing the second peak.

Two transfer functions like those described are shown in Figure 2-5. Below each transfer function are their time-frequency mappings using two data blocks and then six data blocks. The transfer function on the left has two peaks close in frequency with the same damping for each. When two data blocks are used, there is very good frequency resolution and it is easy to tell from the time-frequency mapping that there are two distinct peaks. However, when six data blocks are used, the frequency resolution is not as fine, making it hard to distinguish that there are two peaks. Therefore, it is better in this case to choose frequency resolution over time resolution.

The transfer function on the right has two peaks further apart in frequency, and the second peak has a higher damping ratio than the first. When two data blocks are used, the information corresponding to the second peak is missed, because the associated resonances decay so quickly. However, if less frequency resolution is used in favor of more time resolution, the information corresponding to the second peak becomes visible.

The choice of m also affects the frequency resolution of the transfer function estimate. The estimated impulse response, $\hat{\boldsymbol{g}}$, will have m data points. As defined in Equation 2.39, the transfer function estimate $\hat{G}$ is found by taking the Fourier

transform of the impulse response. The Fourier transform only provides frequency content information up to the Nyquist frequency. Therefore, the frequency resolution of the transfer function estimate is given by

$$\Delta f_{\hat{G}} = \frac{F_s}{m} \tag{2.41}$$

Due to these tradeoffs in time and frequency resolution, the user may need to iteratively adjust m and N_{W} in order to get the best estimate of the transfer function using the time-frequency analysis.

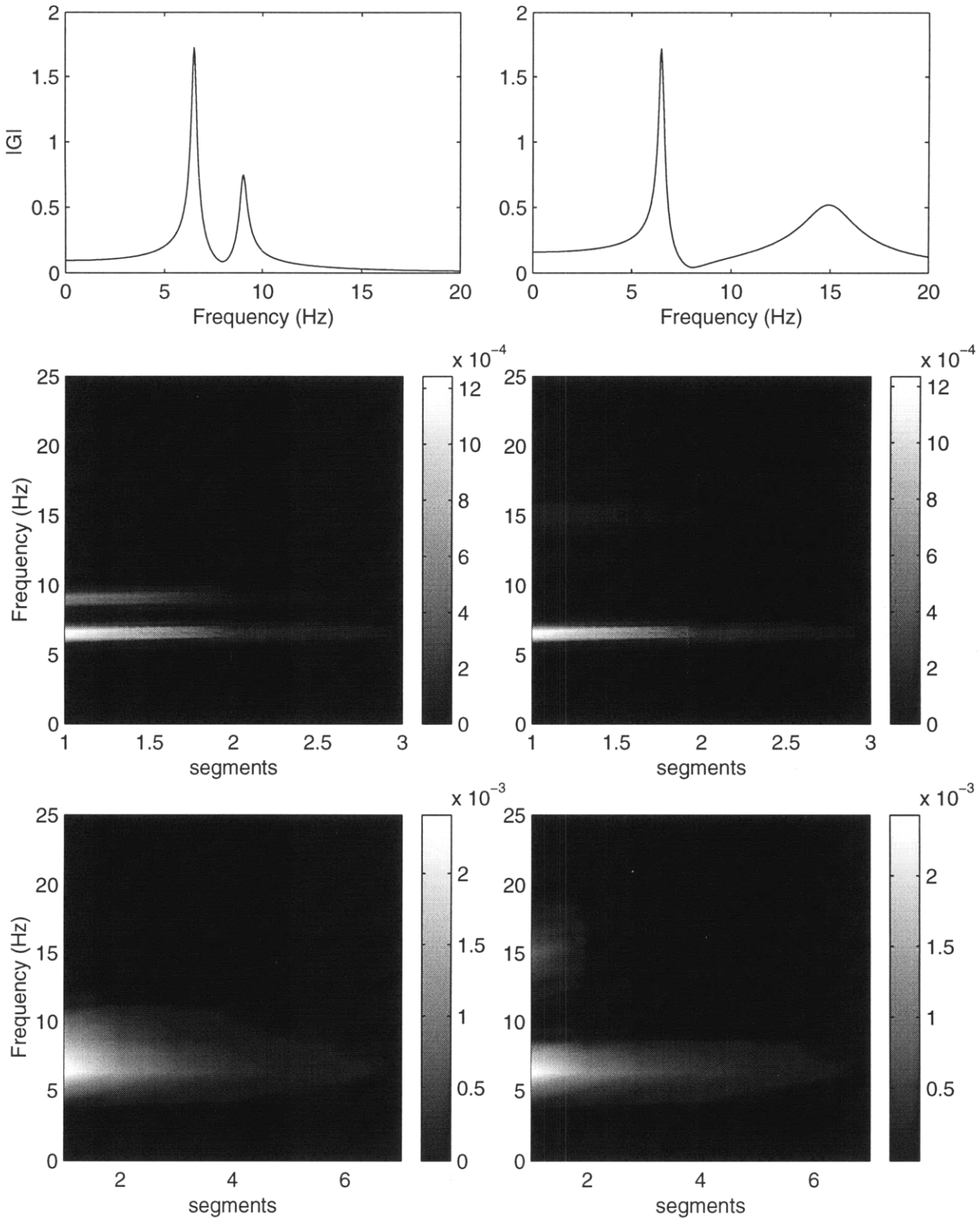


Figure 2-5: Comparison of time-frequency mappings based on choice of parameters m and N_{W}

2.3 Method Validation

In order to validate the method, as well as compare it to other transfer function estimation procedures, a numerical example was used as a trial case. This numerical example is the same example used by Turevskiy [5] and is a fourth-order system whose dynamics are very similar to the F-18 experimental data set of Chapter 3. The numerical example's transfer function is meant to match the F-18 experimental transfer function from right wing input to the left wing sensor. The transfer function is given by

$$G(s) = \frac{-200(s^2 + 2s(0.05)(50.26) + (50.26)^2)}{(s^2 + 2s(0.02)(40.85) + (40.85)^2)(s^2 + 2s(0.02)(56.56) + (56.56)^2)} \tag{2.42}$$

and is plotted in Figure 2-6. The system has a zero at 8 Hz and two lightly damped poles at 6.5 and 9 Hz.

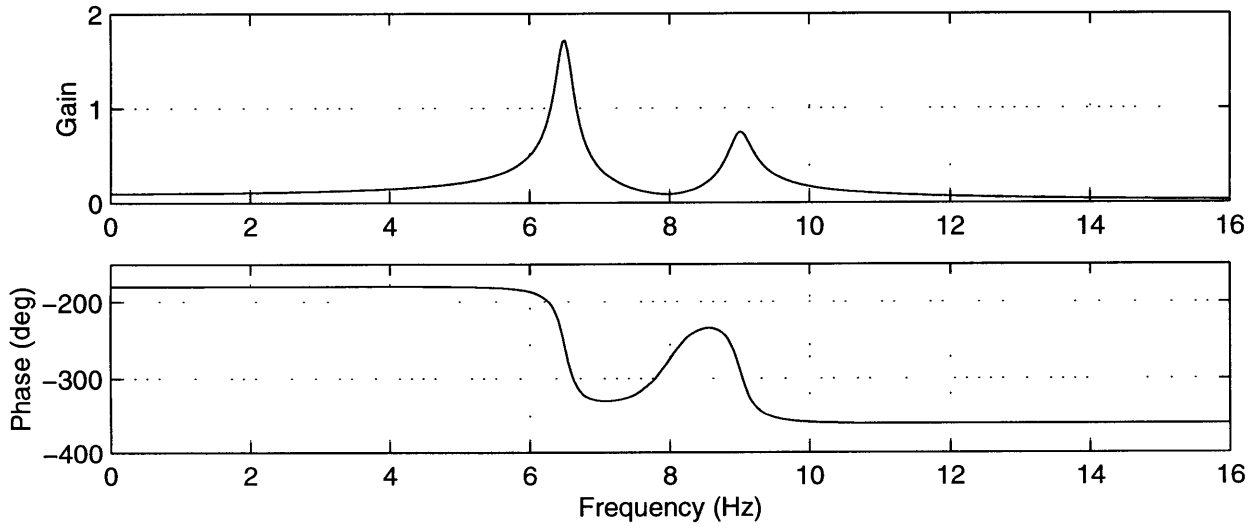


Figure 2-6: Transfer function of numerical example

In order to demonstrate the versatility of the time-frequency analysis method, the trial case was run using two different inputs to the system. Approximating the experimental data, the numerical system was first simulated with a linear sweep input signal given by

$$u_n = 1.5\sin(2.51(n\Delta t)^2) \tag{2.43}$$

The second run used white noise w_n with unit intensity for the input. For both runs, the system was simulated for 30 seconds with a 200 Hz sampling rate.

The output of the system is given by

$$y_n = g_n * u_n + e_n \tag{2.44}$$

where e_n represents sensor noise. It is assumed that $e(t)$ is a broadband random process with bandwidth much greater than the sampling frequency. Since the random process is highly uncorrelated, the expected value of e_n is defined as

$$E\left[e_m e_n\right] = \sigma^2 \delta_{mn} \tag{2.45}$$

where e_m and e_n are any two data points of the noise and δ_{mn} is the dirac delta function.

Transfer functions were estimated using both the chirp signal input and the white noise input for several levels of sensor noise as shown in Figure 2-7. The results are presented in Section 2.3.2 after a discussion of the estimation methods used for comparison.

2.3.1 Estimation Methods for Comparison

The results of the time-frequency estimation method are compared with several other system identification methods in order to give an indication of its performance. The system identification methods used for comparison are:

Empirical Transfer Function Estimate. This method, as described in Section 1.1.1, estimates the transfer function by taking the ratio of the Fourier transforms of the output and input. It is implemented using the `etfe.m` command in MATLAB's system identification toolbox.

Welch's Averaged Periodogram. This method is an improvement on the Empirical Transfer Function Estimate and has some similarities with the time-frequency analysis method. The variance of the ETFE can be reduced if the

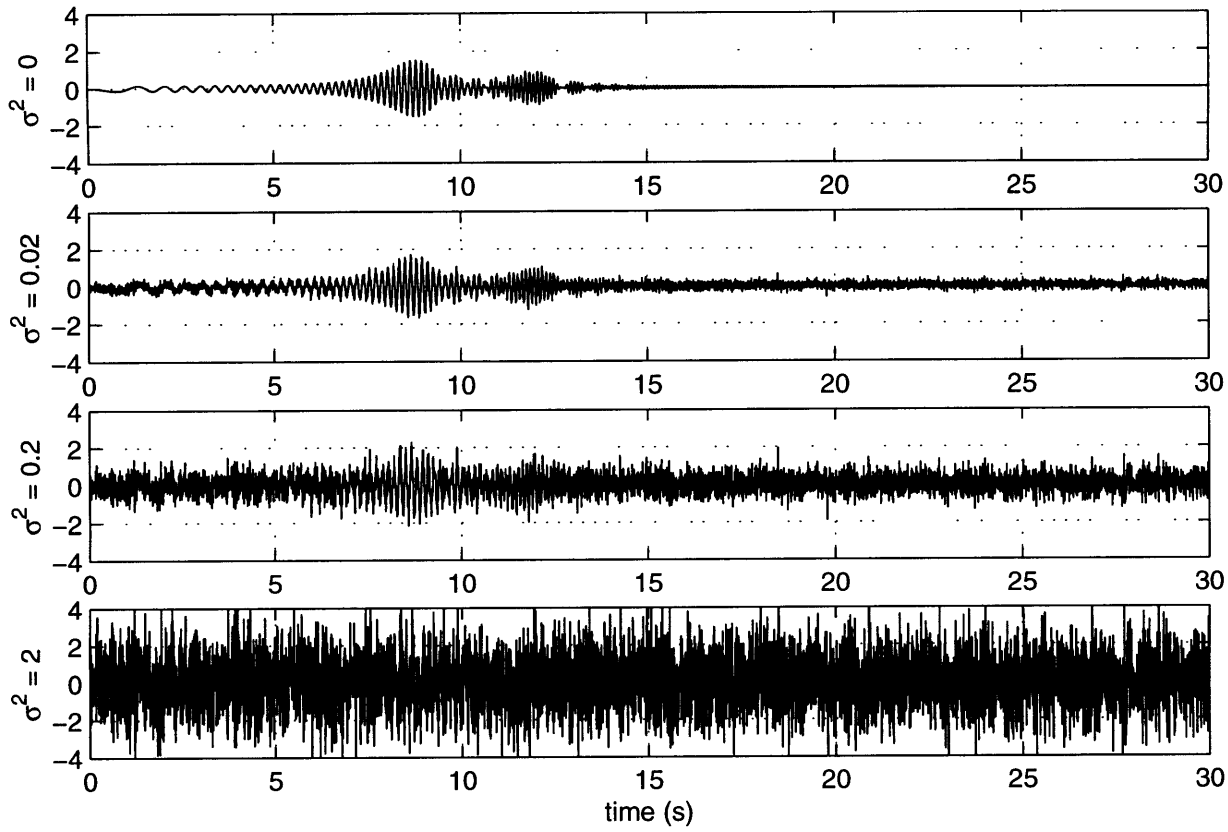


Figure 2-7: System output with sensor noise

signals, $u(t)$ and $y(t)$, are broken into sections or data blocks, and the periodograms of each section are averaged. The result is called an averaged periodogram. Welch proposed a variation to the averaged periodogram in which the data blocks are overlapped and a window, such as the Hanning window, is used to filter each block. By overlapping the blocks, usually by 50% or 75%, some extra variance reduction is achieved [6]. It is implemented using the `tfe.m` command in MATLAB's signal processing toolbox.

Note that Welch's method is different from the time-frequency method, in that it treats the signals $u(t)$ and $y(t)$ separately and then the ratio of their periodograms is taken, whereas the time-frequency method works directly with the impulse response, $g(t)$. Also, Welch's method reduces the effect of noise by averaging all the data blocks, whereas the time-frequency method reduces the effect of noise by throwing out the information in the bins corresponding to

noise. The information from each data block is not averaged, but rather used to reconstruct a better estimate of the impulse response.

Because this estimation problem is a numerical example, there is knowledge of the exact transfer function. Therefore, the estimated transfer functions can be compared against the transfer function given in Equation 2.42. With experimental data, this form of performance evaluation could not be used. The performance cost function to be minimized is

$$J = \int_{5\text{Hz}}^{12\text{Hz}} \left| G(f) - \hat{G}(f) \right|^2 df \tag{2.46}$$

and is evaluated over the frequency range of 5 to 12 Hz since this is where the resonances to be identified are located. Depending on how this cost function is computed, very different results may be obtained. One way of calculating the transfer function is to take the points of the estimated transfer function and compare them to the exact transfer function only at those same frequencies. This way of estimating the cost weights the actual points of the estimate more heavily than the shape of the curve formed by the points. Another way to calculate the cost is to take the points of the estimated transfer function and linearly interpolate between them such that curve of the estimate can be compared to the exact curve. This method weights the area under the curve more than the actual points in the curve. The first method will be denoted J_p since it uses the discrete points of the estimate to find the cost. The second method will be denoted J_A since it uses the area under the curve to find the cost.

To illustrate the difference between the two methods, Figure 2-8 shows a transfer function and two estimates, one represented by $\times$'s and the other by $\bullet$'s. The cost J_p would indicate that the estimate represented by $\times$'s is the better estimate whereas the cost J_A would indicate that the $\bullet$ estimate is the better one. The advantage of the cost J_A is that a transfer function estimate which misses a peak would perform poorly whereas the cost J_p would indicate good performance as long as each point was a good estimate. However, the cost J_A could also be misleading if for example, one peak was estimated to be much larger while another was estimated to be much smaller. The

added area under the curve from the large peak would make up for the missing area under the small peak and therefore J_A would still indicate good performance. In this case, however, J_p would be a good indicator of the performance because it would be comparing each point of the estimate with the actual transfer function. Because of these ambiguities, the cost defined in Equation 2.46 can only be used as an indicator of the value of each estimation method. In evaluating the estimation methods, both J_p and J_A were used. A very good estimate would be one that minimizes both costs.

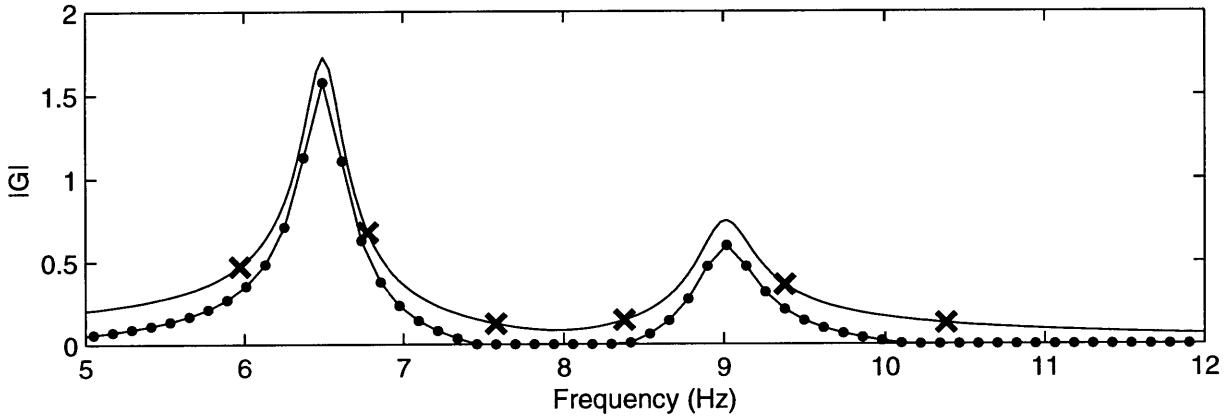


Figure 2-8: Methods of calculating the cost function

2.3.2 Numerical Example Results

The time-frequency analysis method was tested on six estimation problems. The input was either a chirp signal or white noise and the measurement noise had variance σ^2=.02, σ^2=.2 or σ^2=2.

Case 1: Chirp input signal, σ^2=0.02. The results for the case with a chirp signal input and sensor noise with variance σ^2=0.02 are reported in Table 2.1 and Figure 2-9. The best estimate using Welch's method was found by breaking the data into 3 data blocks. The time-frequency estimate was found by setting m=1200 and dividing the data into 2 data blocks. From the time-frequency mapping, it was very clear which bins were noise and at which frequencies the system resonances were located. Only the bins corresponding to these resonances were kept in the reconstruction of the impulse response. According to

the cost J_p, the time-frequency method performed well against the other methods. However, according to the cost J_A, the time-frequency estimate was the worst of the three. In terms of appearance, the time-frequency estimate looks to be the cleanest estimate of the transfer function.

Case 2: Chirp input signal, σ^2=0.2. The results for the case with a chirp signal input and sensor noise with variance σ^2=0.2 are reported in Table 2.2 and Figure 2-10. As the sensor noise was increased, all of the transfer function estimates, especially the ETFE, became less smooth than in the previous case. The estimate using Welch's method was found by breaking the data into 3 data blocks. The time-frequency method parameter m was set to 1200 and the data was divided into 2 data blocks. Again, the system resonances were distinguishable from the noise in the time-frequency mapping. Only the bins corresponding to the resonances were kept. The time-frequency estimate performed well by both cost functions indicating that the estimate is better than those by the other two methods. In addition, by visual inspection it appears cleaner than the other estimates.

Case 3: Chirp input signal, σ^2=2. The results for the case with a chirp signal input and sensor noise with variance σ^2=2 are reported in Table 2.3 and Figure 2-11. With large amounts of noise, the ETFE became very corrupted although it is still possible to guess at the resonances. Welch's method still produced a relatively clean transfer function by breaking the data into 5 segments. However, as a result, the transfer function is somewhat choppy. The time-frequency estimate was found by setting m=400 and dividing the data into 2 data blocks. This time, the time-frequency mapping was not able to pick out information corresponding to the physical system because the noise was dominating. However, if some a priori knowledge about the system is known, the time-frequency mapping can still be used to obtain a cleaned estimate of the transfer function. For example, in this case, it it was known that the system resonances were below 15 Hz. Therefore, all the information above this frequency could be thrown

out and attributed to noise. By keeping only the bins with frequency content below 15 Hz, the time-frequency estimate was obtained. The time-frequency estimate performed better than the ETFE but not as well as Welch's methods. Note that the time-frequency estimate is very choppy due to m being small. When the frequency resolution of the transfer function estimate is so poor, it is possible that the estimate could have missed a peak. Also, the time-frequency method did not estimate the phase well.

Case 4: Noise input signal, σ^2=0.02. The results for the case with a white noise input signal and sensor noise with variance σ^2=0.02 are reported in Table 2.4 and Figure 2-12. The Welch's method estimate was obtained by dividing the data into 2 segments. The time-frequency method was obtained using m=1200 and 2 data blocks. As with Case 1, the time-frequency mapping made it easy to distinguish the bins associated with the system resonances from those associated with noise. Only those believed to be associated with the system resonances were retained. The resulting transfer function estimate outperforms the other two in terms of the cost functions and in terms of appearance.

Case 5: Noise input signal, σ^2=0.2. The results for the case with a white noise input signal and sensor noise with variance σ^2=0.2 are reported in Table 2.5 and Figure 2-13. For this noise level, the ETFE does not provide a reliable transfer function estimate. The Welch's method estimate is fair but underestimates the second peak. It was found using 4 data blocks. The time-frequency estimate looks clean comparatively and performs better by both cost functions. This time-frequency estimate was found by setting m=1200 and using 2 data blocks. The time-frequency mapping still clearly distinguished the noise from the system dynamics.

Case 6: Noise input signal, σ^2=2. The results for the case with a white noise input signal and sensor noise with variance σ^2=2 are reported in Table 2.6 and Figure 2-14. In this case, neither the ETFE nor Welch's method performed well. The Welch's method estimate was found using 12 data blocks. The trans-

fer function estimate was found using m=800 and 2 data blocks. Even though there was a lot of noise in the time-frequency mapping, a band of bins corresponding to the resonances was still distinguishable. Keeping these bins only, the transfer function estimate was obtained. The estimate indicates at what frequencies the resonances occur, but underestimates the size of the first peak and overestimates the second. According to the cost functions, the time-frequency method performed better than the ETFE but not as well as Welch's Method. However, visual inspection of the estimates contradicts this result. This case is a good example of how the cost functions can only be used as indicators of performance as opposed to absolute measures.

Overall, the time-frequency estimation method performed well. In all cases, the estimate closely resembled the exact transfer function and was visually comparable or better than the other methods. In addition, the cost function performance was often better than the performance of ETFE or Welch's Method. An important result was also demonstrated by Case 3, which showed that it is not necessary to always be able to distinguish the noise from the dynamics in the time-frequency mapping. If the engineer has an idea of what frequency range the resonances should be in, then the bins outside of the range can be discarded, thereby eliminating some of the noise in the data.

Table 2.1: Performance of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=0.02.

u=chirp signal	σ^2=0.02	
	J_p	J_A
ETFE	0.0260	0.0173
Welch's Method	0.0214	0.0240
Time-Frequency Method	0.0078	0.0504

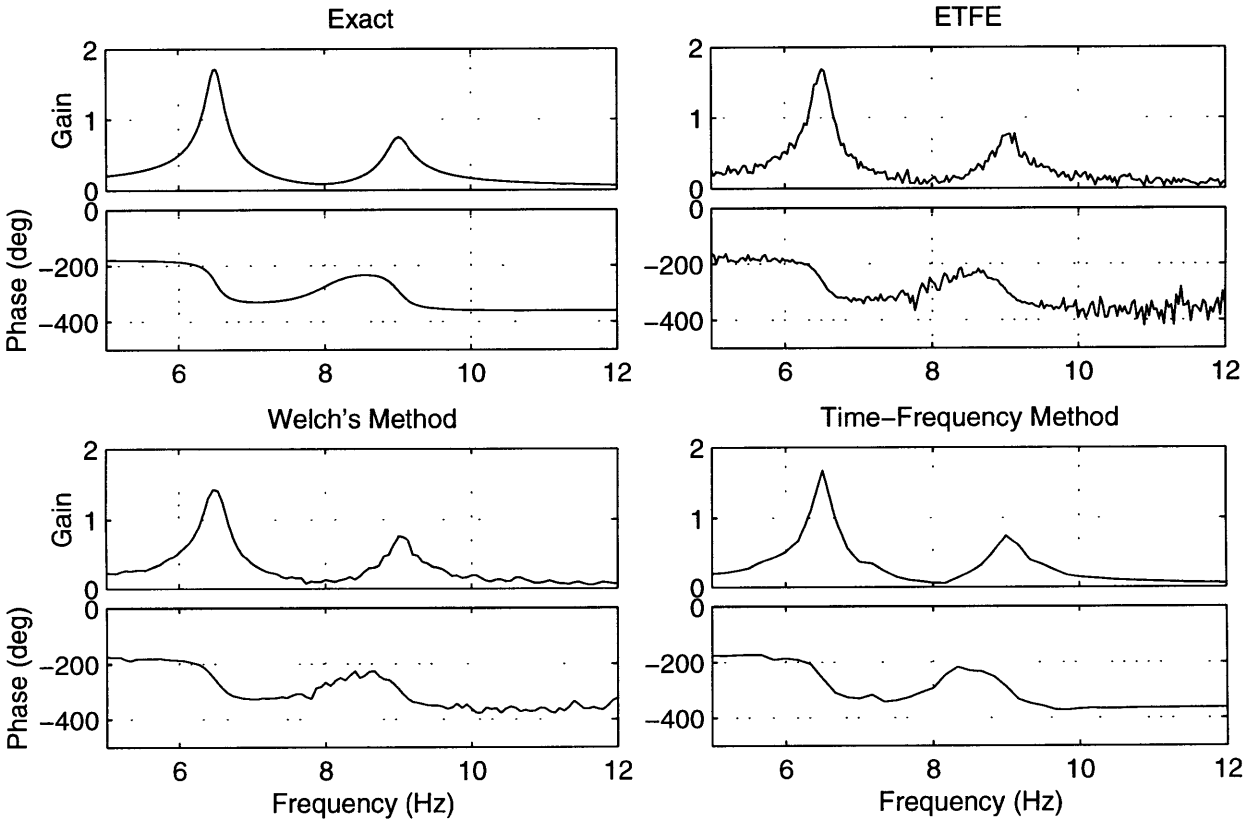


Figure 2-9: Comparison of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=0.02.

Table 2.2: Performance of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=0.2.

u=chirp signal	σ^2=0.2	
	J_p	J_A
ETFE	0.2579	0.1699
Welch's Method	0.0626	0.0497
Time-Frequency Method	0.0222	0.0406

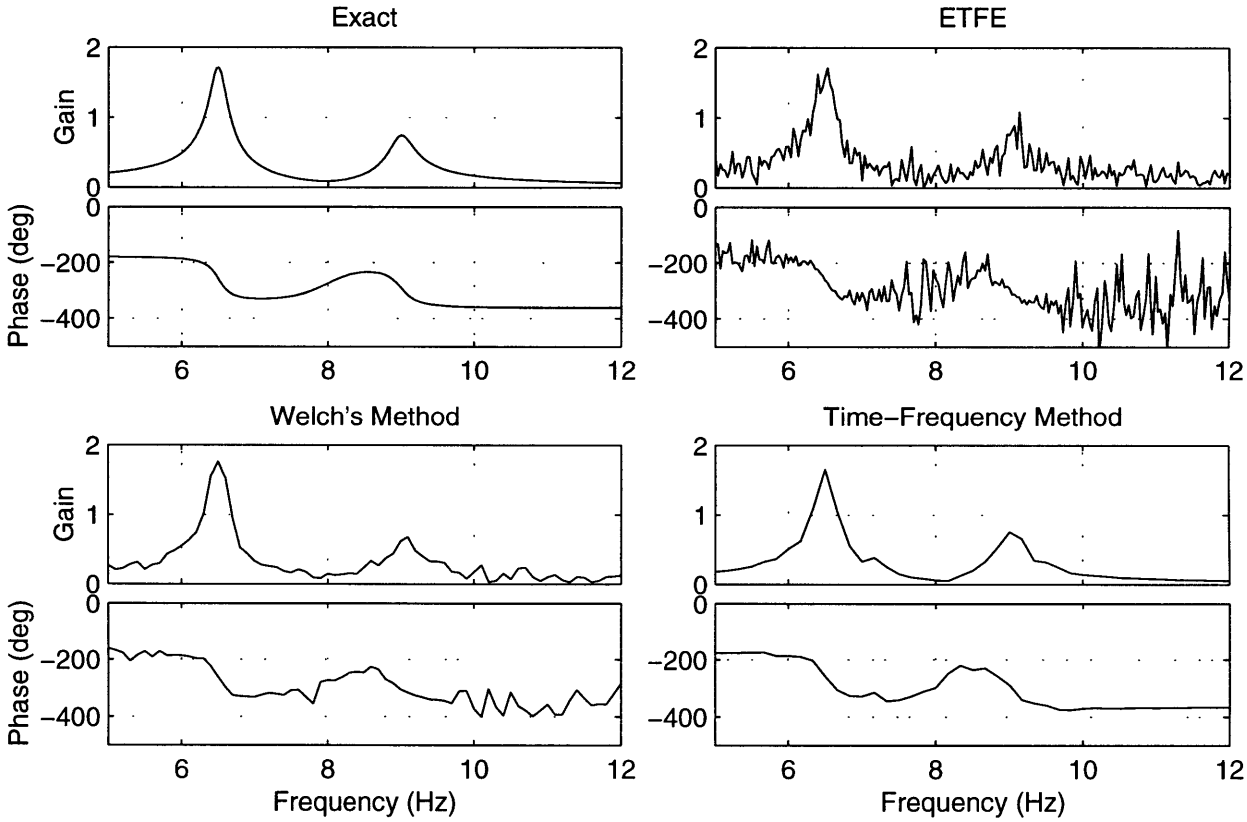


Figure 2-10: Comparison of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=0.2.

Table 2.3: Performance of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=2.

u=chirp signal	**σ^2=2**	
	J_p	J_A
ETFE	2.5759	1.6932
Welch's Method	0.2144	0.1972
Time-Frequency Method	0.2456	0.4212

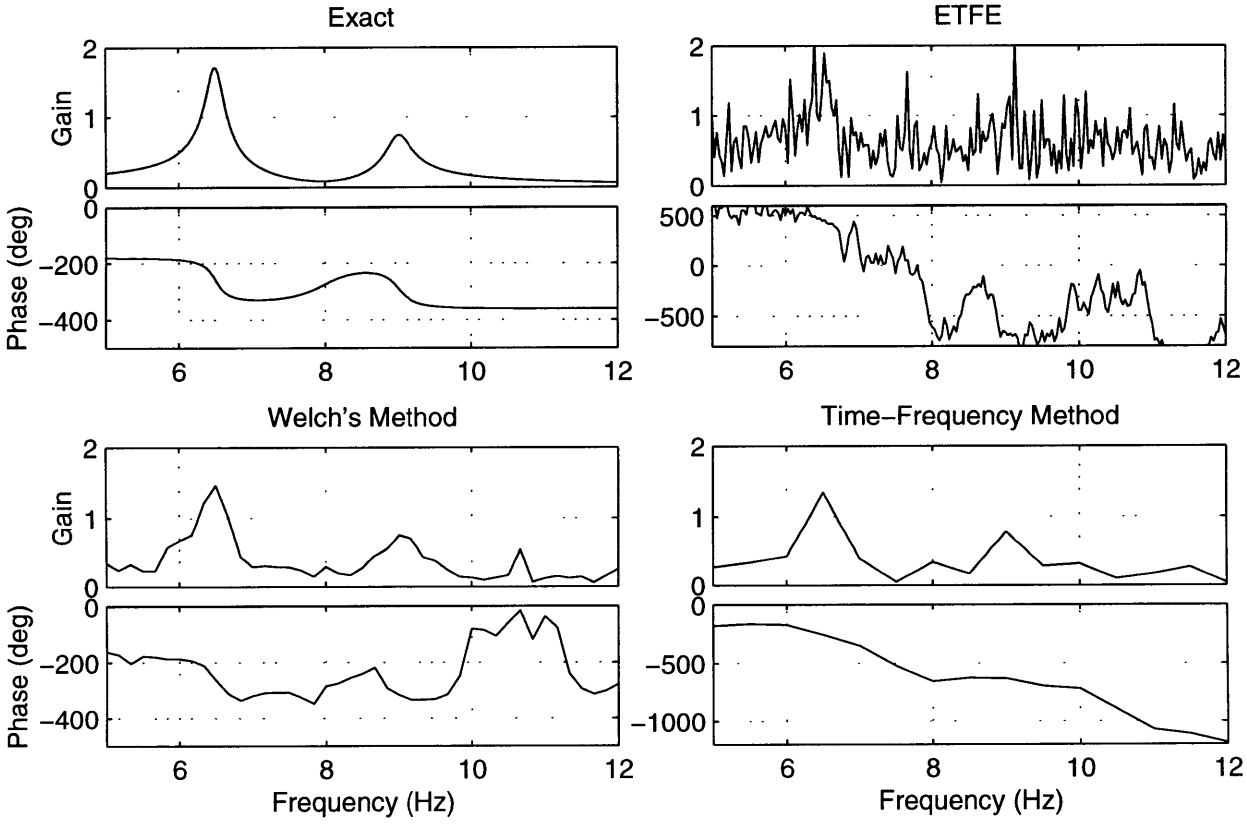


Figure 2-11: Comparison of the estimation methods for the case with a chirp signal input and measurement noise of variance σ^2=2.

Table 2.4: Performance of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=0.02.

u=noise σ^2=0.02		
	J_p	J_A
ETFE	3.3268	2.2411
Welch's Method	0.0806	0.0601
Time-Frequency Method	0.0237	0.0484

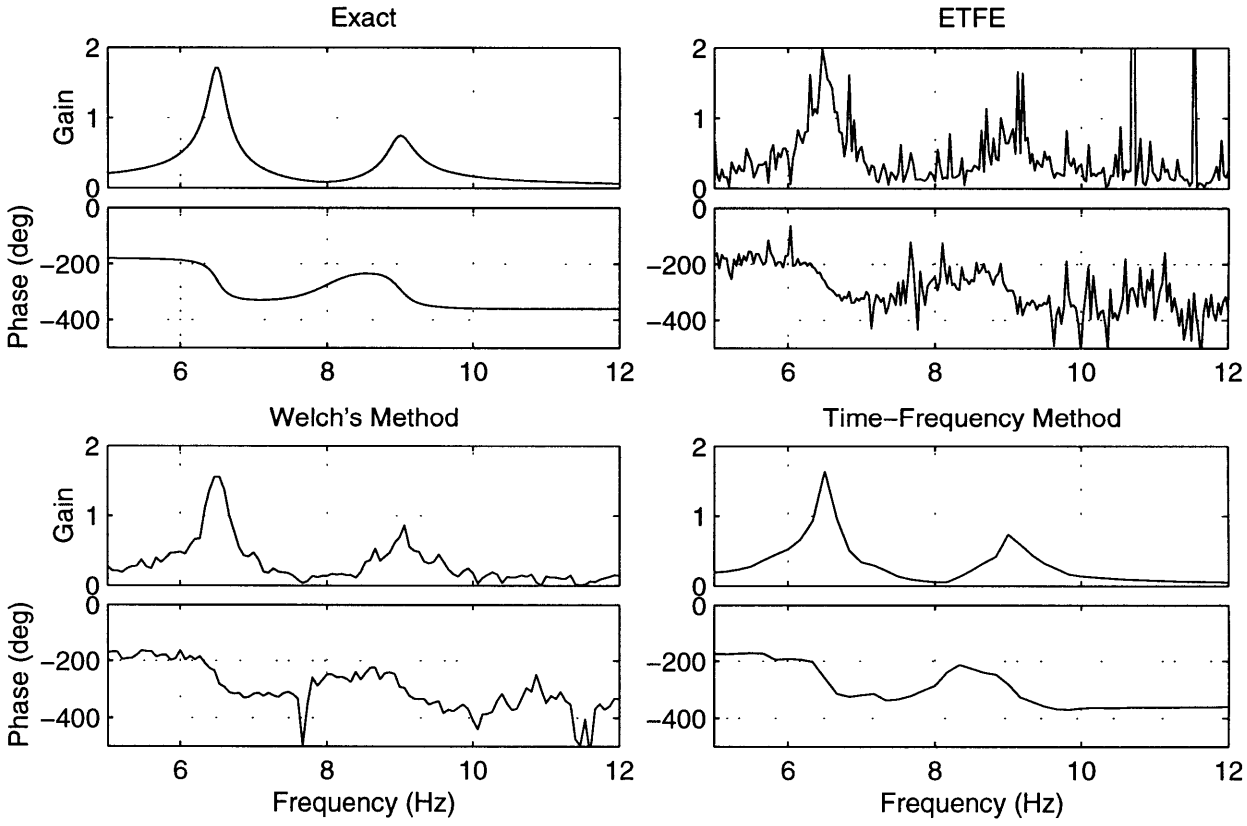


Figure 2-12: Comparison of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=0.02.

Table 2.5: Performance of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=0.2.

u=noise σ^2=0.2		
	J_p	J_A
ETFE	32.8913	22.1636
Welch's Method	0.2374	0.2235
Time-Frequency Method	0.1483	0.1454

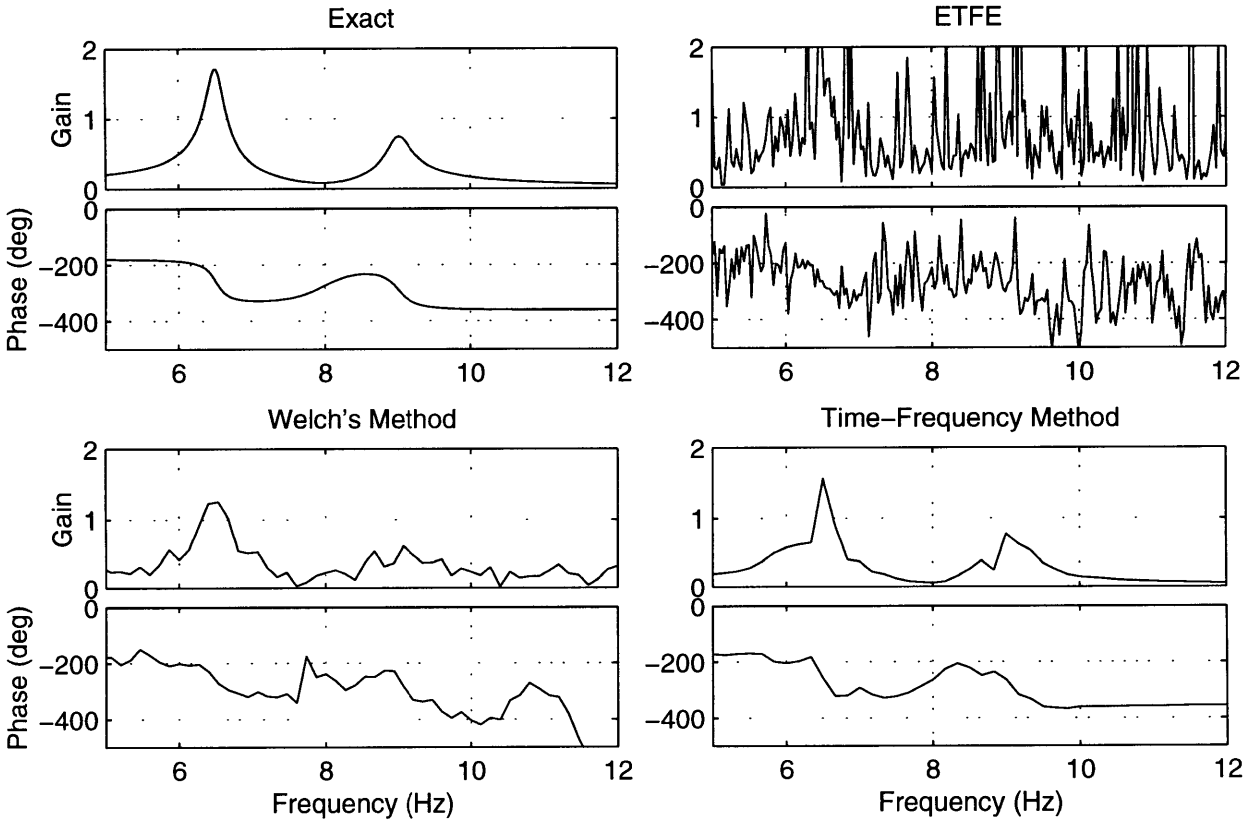


Figure 2-13: Comparison of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=0.2.

Table 2.6: Performance of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=2.

u=noise	σ^2=2	
	J_p	J_A
ETFE	328.4647	221.3725
Welch's Method	0.7812	0.7415
Time-Frequency Method	1.0644	0.8386

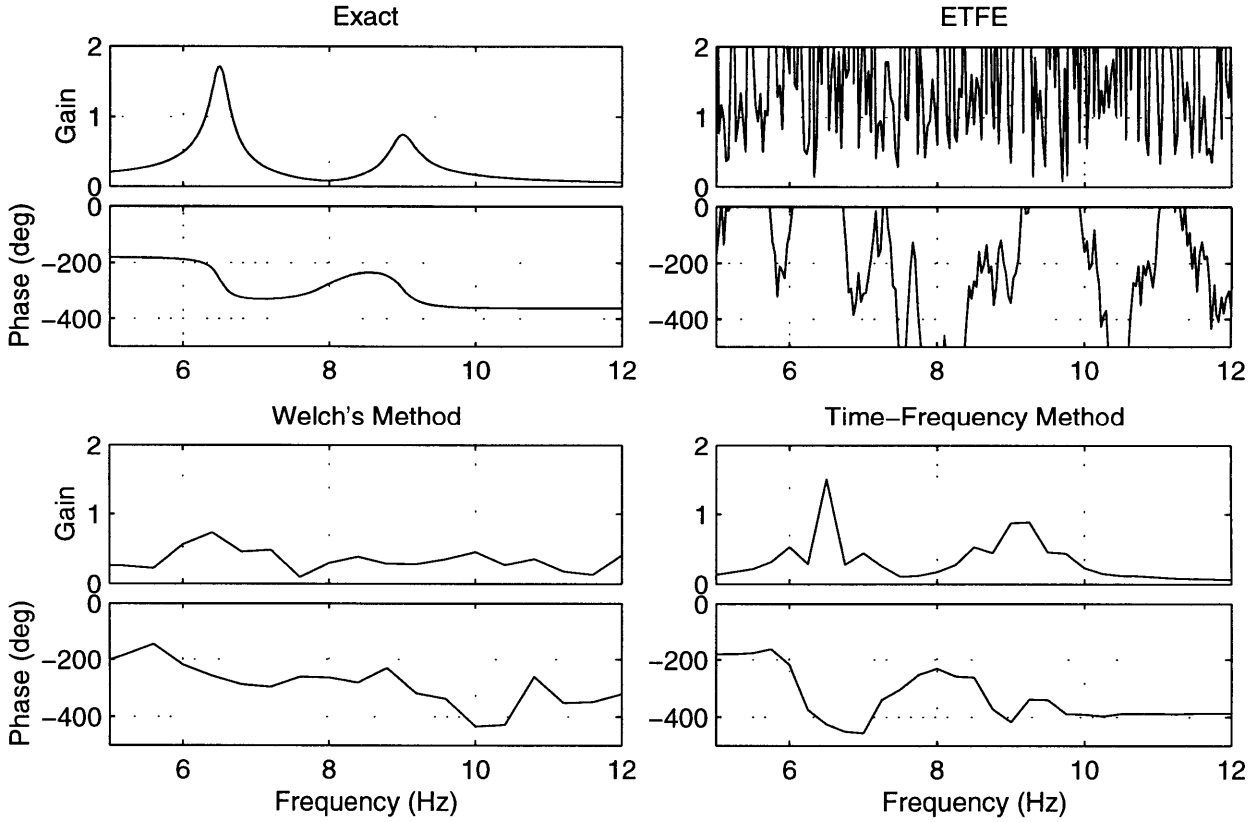


Figure 2-14: Comparison of the estimation methods for the case with a white noise input and measurement noise of variance σ^2=2.

Chapter 3

Application to Experimental Data

The proposed time-frequency estimation method is illustrated by applying it to the experimental data from the F18-SRA flight tests at the NASA Dryden Flight Research Center. This is the same data that was used by references [3, 4, 5] to develop the time-frequency analysis method for data sets with chirp input signals. For the purposes of this application, the F18 system can be considered a multi-input (2), single output system. An exciter was mounted on each wing of the F-18 and would produce sinusoidal variations in the force at the tip of the wing. These variations were modulated both linearly and logarithmically to provide a chirp or frequency sweep input signal. Two load sensors measured the input force on the wings. The output of interest is an accelerometer located on the forward left wingtip. Being consistent with the literature, the task is to estimate the transfer functions from the left and right exciters to the forward left wingtip accelerometer over the frequency range spanning from 5 to 12 Hz.

In order to distinguish the influence of the two inputs, it was necessary to have symmetrical tests where the inputs were the same, and asymmetrical tests where the inputs had a 180 degree phase difference. These cases are used together in identifying the transfer functions.

3.1 Single Input Data Set

Even though the experimental data is really multi-input, for the purpose of comparing the identified transfer function against the estimate given by ETFE and Welch's Method, a single input case was approximated by averaging the two inputs and averaging the two outputs from the symmetric test. The resulting input and output are shown in Figure 3-1. Note that the noise level on the output looks very similar to the numerical test case that was simulated with a sensor noise variance of σ^2=0.02.

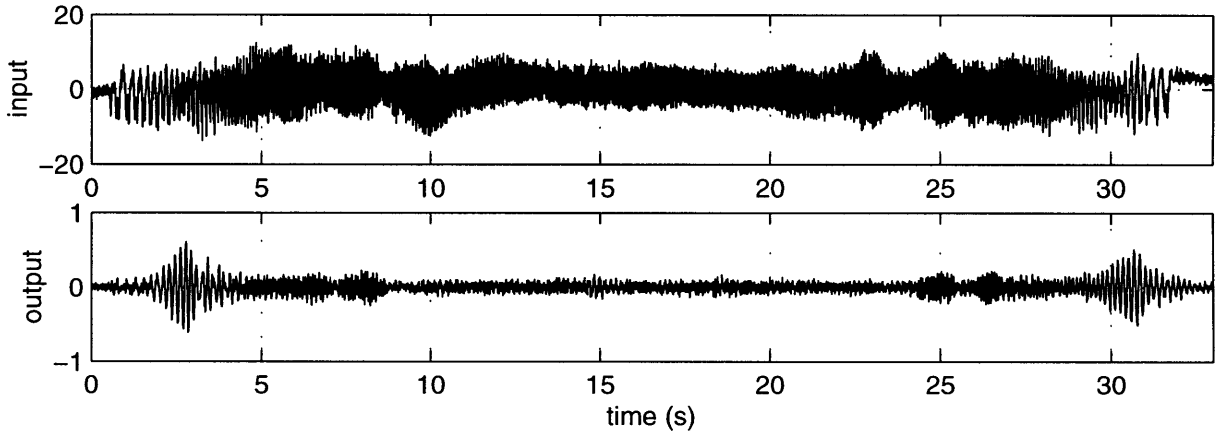


Figure 3-1: Input and output of SISO experimental data

The estimated transfer functions are shown in Figure 3-2. No performance costs of the estimates are given, of course, because the response is unknown. Because the two inputs and two outputs are averaged, only one of the two system peaks shows up in the estimated transfer function. This is because the second natural frequency is due to an asymmetric bending mode of the wings. Therefore, adding the two inputs makes the second mode unobservable.

According to the highest point in the magnitude estimate, the Empirical Transfer Function Estimate identifies a peak at 6.57 Hz. However, the general shape of the estimate suggests a resonance of about 6.4 Hz. The phase identified supports the magnitude estimate, in that there is a 180 degree phase shift at the same frequency. Unfortunately, the ETFE result is noisy.

The best estimate using Welch's method was obtained by dividing the impulse

response into 5 data blocks. The resulting estimate identified a 6.36 Hz resonance. The estimate is very clean, but because of the amount of averaging that was necessary, is also a little choppy. In fact, the peak appears to be lopsided to the left, which indicates that data points in the estimate have missed the tip of the peak. Therefore, the resonance is probably a little higher in frequency.

In estimating the transfer function using the time-frequency approach, the parameter m was set to 1312, and the signal was divided into 4 data blocks. The level of noise in the measurement is apparently low, as it was very clear which bins in the time-frequency mapping contained information corresponding to the system resonance. The time-frequency analysis method identified a peak at 6.4 Hz. The time-frequency estimate is smoother than Welch's Method, and cleaner than the ETFE, giving a better estimation of the dynamics of the system.

3.2 Multi-Input Data Set

In order to properly identify the multi-input F18 system, the symmetric and asymmetric data sets have to be used simultaneously. To do this, the information matrix and vector of each data set is calculated according to Equations 2.25 and 2.26. Information can be added, so the resulting information matrix and vector for the multi-data set is given by

$$\begin{aligned} \mathcal{I} &= \mathcal{I}_s + \mathcal{I}_a \\ \chi &= \chi_s + \chi_a \end{aligned} \tag{3.1}$$

where the subscripts $(\,)_s$ and $(\,)_a$ indicate information corresponding to the symmetric and asymmetric data sets, respectively. The impulse response estimate is then calculated according to Equation 2.8.

The transfer function estimates are shown in Figure 3-3. To obtain this estimate, m was set to 1600 and the data was divided into 4 data blocks. The left input to left output transfer was identified to have a peak at 6.38 Hz and 9.12 Hz. The right input to left output transfer function has a peak at 6.38 and 9.00 Hz. Because both the symmetric and asymmetric data sets were used simultaneously, both the symmetric

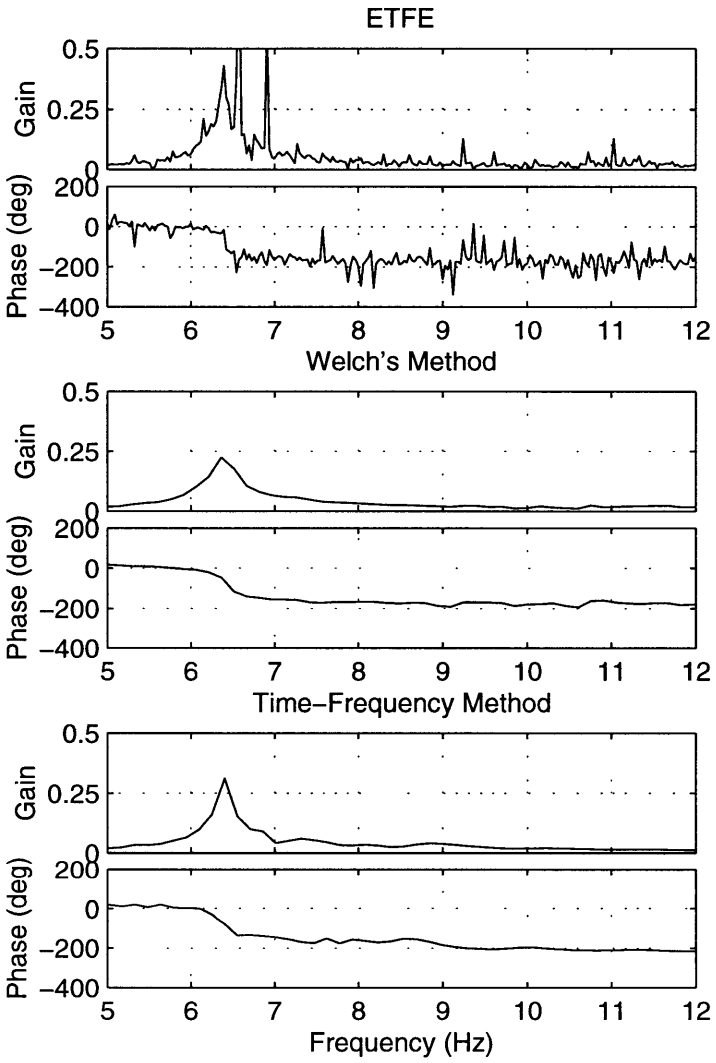


Figure 3-2: Transfer function estimates for single-input case

and asymmetric modes are observable and show up as two peaks in the transfer function estimate. Note that the transfer function from left input to left output has a zero between the two poles, as would be expected for a collocated measurement. Also, note that that because the inputs and outputs were added to create the single input case, the magnitudes of the single input transfer function are twice as large as the magnitudes for the two input case, as would be expected.

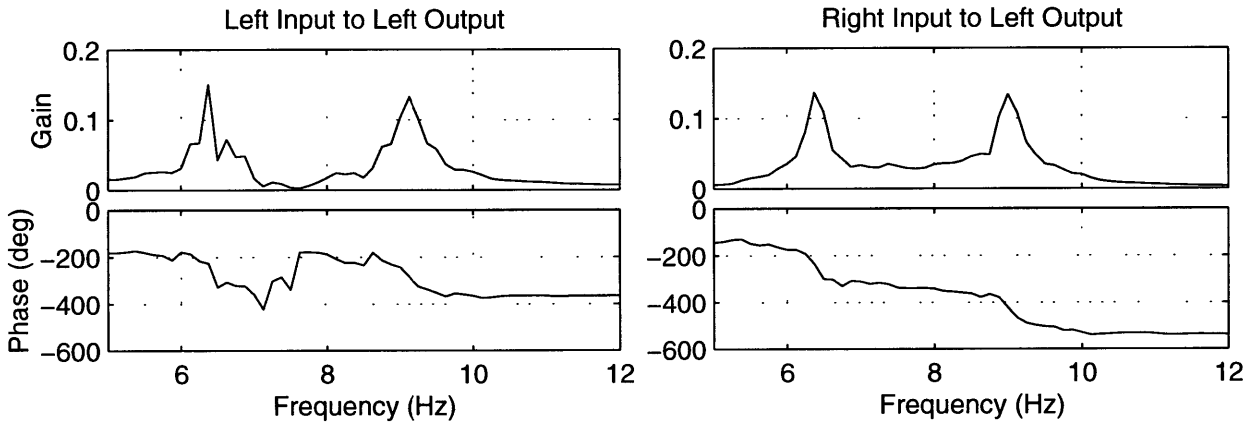


Figure 3-3: Transfer function estimates for multi-input case

Both the single input transfer function estimate and the multi-input transfer function estimate give a clean estimate of the system dynamics that match the results obtained by Feron *et al.* [4].

Chapter 4

Conclusions

4.1 Conclusions

Based on the work of Feron *et al.* [4], the time-frequency analysis method for transfer function estimation was generalized to include data sets with any type of input, rather than just chirp signals. The method is based on the observation that for a linear system, the output of the system should be at the same frequency as the input to the system and that the output should happen nearby in time to the corresponding input. In addition, the degrees of freedom of the estimation problem are reduced by restricting the impulse response to be causal and limited in duration. After using a time-frequency transform to map the impulse response of the system to a two dimensional representation in time and frequency, these principles were used to distinguish the noise in the response from the system's physical dynamics. By discarding the information in the impulse response that corresponds to the noise, a cleaner estimate of the impulse response was found and was then used to obtain the transfer function estimate.

The efficacy of the method was demonstrated on both a numerical example and experimental data from F18 flight testing. The time-frequency approach was shown to reduce the effect of noise in the transfer function estimate even in the presence of high levels of noise. In addition, it often performed better than the Empirical Transfer Function Estimate or Welch's Averaged Periodogram method.

4.2 Recommendations

There are several areas that could be explored to improve upon the time-frequency analysis method presented in this thesis. These areas are discussed below.

Using all output data in estimation method

Because the output depends on m number of previous input data points, the first m output data points cannot be used for estimating the transfer method unless a way to include them is developed. Currently, the only way the time-frequency tool can include those points is by assuming that the input and output are zero before the test. Under this assumption, the data can be padded with m zeros at the beginning. As a result, the first point of the sampled output data is dependent on the first sampled input data point plus $m - 1$ of the padded zeros. The results of the estimation method could possibly be improved if a more clever way of including the first m output data points were developed. One possibility would be to estimate the response of the system to the initial state using an auto-regressive estimation approach, while estimating the forced response using time-frequency analysis.

Windowing methods

It is possible that better results could be obtained using another type of window to filter the segments when performing the time-frequency decomposition. Using the windowing technique presented, there is constant frequency resolution, f_R, and time resolution, t_R, across the entire time-frequency mapping. A filtering method that allows variations in f_R and t_R over a time-frequency mapping might provide for better signal reconstruction.

Algorithm optimization

As the number of desired data points, m, in the impulse response estimate become large, so does the information matrix, $\mathcal{I}$, of Equation 2.9. The information matrix has m^2 elements when the data set has only one input and n^2m^2 elements for a data

set with n inputs. Storing this matrix takes a lot of memory, and manipulating the matrix takes many operations. Therefore, the size of m is limited by the memory of the computer.

Since the frequency resolution of the transfer function estimate is determined by m, the smoothness of the estimated transfer function is also limited by the memory of the computer. Therefore, it would be advantageous to look into ways to optimize the routines for more efficient data storage and manipulation.

Method combination

Finally, the time-frequency analysis method presented was shown to be effective in reducing the noise in a transfer function estimate. Further improvements in transfer function estimation might be obtained if the concepts of the time-frequency analysis were used in combination with other system identification techniques, such as parametric identification or subspace identification. For example, the time-frequency method could be used to estimate the system frequency response from the noise, and then a parametric technique could be used to fit a model to the system.

Appendix A

Using the MATLAB Time-Frequency Analysis Tool

A tool to implement the time-frequency estimation method was developed in MATLAB 5.1. The user interface of the tool is shown in Figure A-1. The tool first presents the time-frequency mapping of the impulse response. The user can then choose which bins to keep using the graphical user interface. After bins are chosen, the estimate of the impulse response and transfer function are calculated and plotted. The cost function is also evaluated and the performance of the current estimate is displayed.

The tool also allows the user to modify the bin choices. To aid in the modification, the tool will indicate with a $\diamond$ which bin to add or subtract in order to best reduce the cost function. The new estimate of the impulse response is then plotted and compared against the previous estimate. A record of the cost function for each iteration is displayed on the left side of the tool interface.

The code for the tool is presented in Appendix B. Before the tool can be run, the information matrix and information vector must be calculated. This is done using the codes:

`main_num.m`	single input numerical example
`main_siso.m`	single input experimental data
`main_miso.m`	multi-input experimental data

The information matrix and vector are stored in a `.mat` file and then loaded by the tool. The code for the tool is:

`tvf_tool.m`	single input numerical example or experimental data
`tvf_tool_miso.m`	multi-input experimental data

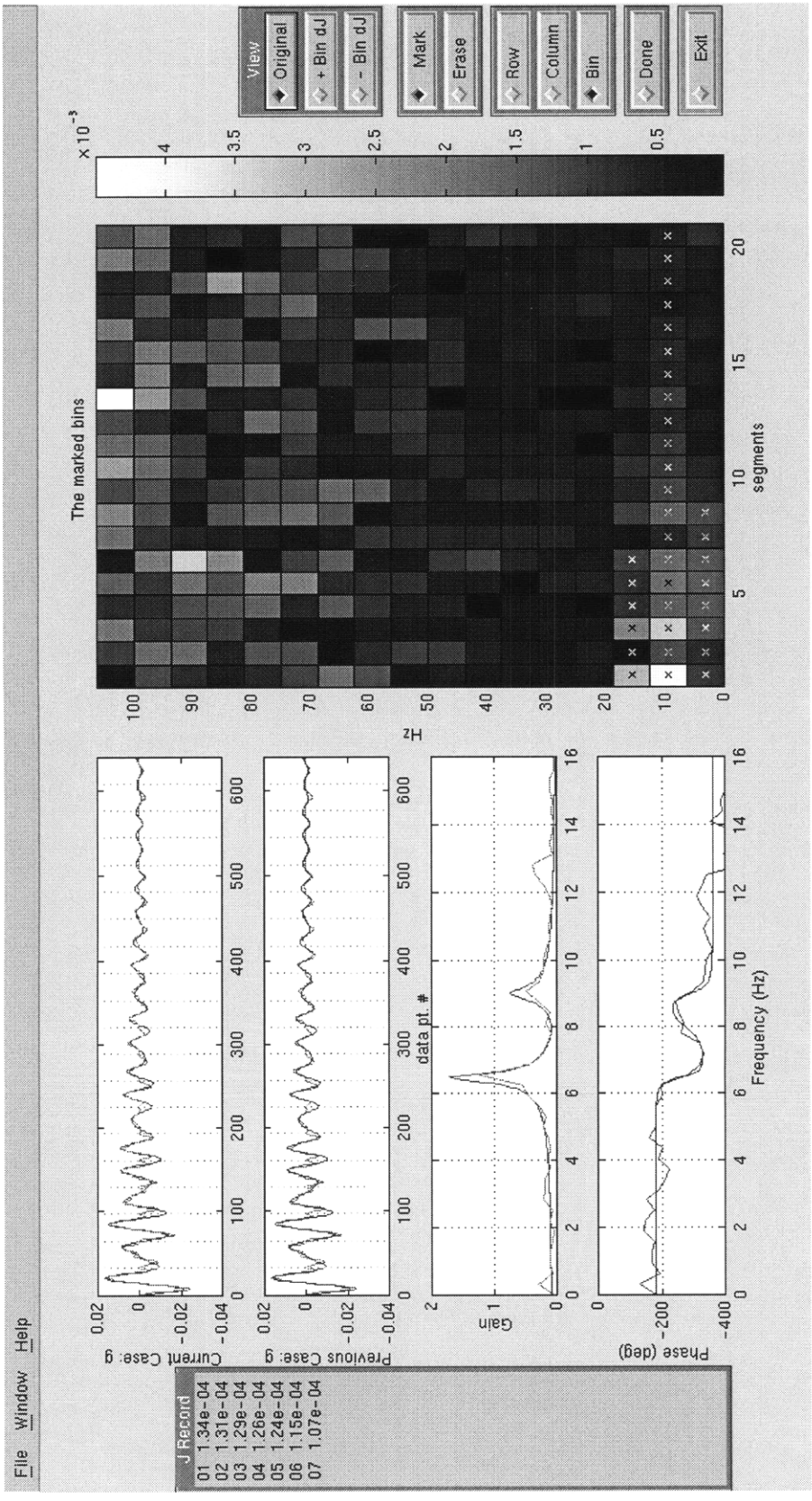


Figure A-1: Time-frequency analysis tool

Appendix B

MATLAB Code

```
%=========================================================================
% main_num.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/23/98
%
% Calculates the information matrix M from a numerical example
%                information vector V
%=========================================================================

close all
clear all
fig=1;

u_input='n'                          % c=chirp signal input
                                     % n=white noise input
plots=1;                             % 1=plot results
                                     % 0=don't plot results

%--------------------------------
% Code parameters
%--------------------------------
noise_variance = 4;                  % the variance of sensor noise
                                     % to add to the measurememement
                                     % (10^-(noise_variance))/dt
                                     % unless equal to zero then
                                     % variance equals zero

%--------------------------------
% Impulse Parameters
%--------------------------------
m=640;                               % # of previous input pts the
                                     % response depends on
```

```
%-------------------------------
% Define numerical system
%-------------------------------

w1=50.26;                          % zero frequency     (rad/s)
w2=40.85;                          % 1st peak frequency (rad/s)
w3=56.56;                          % 2nd peak frequency (rad/s)
num  = -200*[1 2*.05*w1 w1^2];     % transfer function numerator
den1 =      [1 2*.02*w2 w2^2];     % transfer function denominator part 1
den2 =      [1 2*.02*w3 w3^2];     % transfer function denominator part 2
den  =  conv(den1,den2);           % transfer function denominator

clear den1 den2 w1 w2 w3

%-------------------------------
% Plot exact transfer function of
% numerical example
%-------------------------------
w=linspace(0,16*2*pi,500);         % frequency (rad/s)
[mag,ph,w]=bode(num,den,w);        % magnitude and phase of numerical example
f=w/(2*pi);                        % frequency (Hz)

if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(211)
  plot(f,mag)
  axis([0 16 0 2])
  grid
  ylabel('Gain')
  title('Transfer Function of Numerical Example')

  subplot(212)
  plot(f,ph)
  axis([0 16 -400 -150])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')
end

%-------------------------------
% Create input and find system
% response to input
%-------------------------------
Ts=200;                                % sampling rate (Hz)
tend=30;                               % simulation end time (s)
n=tend*Ts+1;                           % total # of points
t = [linspace(0,tend,n)]';             % time vector (s)
dt=t(2);                               % time step   (s)

if strcmp(u_input,'n')                 % NOISE INPUT
  u=rand_input(n,dt,35*2*pi);          % create noise input
```

```
  y=lsim(num,den,u,t);                % system response

elseif strcmp(u_input,'c')            % CHIRP INPUT
  u=1.5*sin(2.51*t.^2);
  u=[zeros(m,1); u; zeros(m,1)];      % pad front with m zeros
  t=[t; t(2:2*m+1)+t(n)];             % adjust time vector accordingly

  n=length(u);                        % number of data points in data
                                      %    to consider
  y = lsim(num,den,u(m+1:n),t(1:n-m)); % system response
  y = [zeros(m,1); y];                % pad response w/zeros
end

N=n;                                  % number of data point in data

if noise_variance==0
  Nvar=0;
else
  Nvar=(1/dt)*10^(-noise_variance);   % variance of the noise
end
if Nvar ~= 0                          % create sensor noise
  randn('seed',42);                   % initialize random number generator
  v=randn(n,1)*sqrt(Nvar);            % sensor noise

  y=y+v;                              % add sensor noise to output
end

%--------------------------------
% Plot input and output
%--------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(211)
  plot(t,u)
  grid
  ylabel('INPUT u')

  subplot(212)
  plot(t,y)
  grid
  xlabel('time (s)')
  ylabel('OUTPUT y')
end

%--------------------------------
% Find exact impulse response
% using MATLAB's impulse.m
%--------------------------------
ttime=clock;                                      % time the calculation
  [g_matlab,t_matlab]=impulse(num,den,t(1:m));    % exact impulse response
  g_matlab=g_matlab';
```

```
tgone=etime(clock,ttime);
fprintf('Impulse Matlab     :  %2.4f s\n',tgone)

%-------------------------------
% Find information matrix, vector
% and impulse response using
% imp_est.m
%-------------------------------
ttime=clock;                                    % time the calculation
  [Minfo,Vinfo,g]=imp_est(y,u,m,N);             % info matrix, vector and
                                                % impulse resonse estimate
tgone=etime(clock,ttime);
fprintf('Impulse Estimate   :  %2.4f s\n',tgone)

%-------------------------------
% Plot exact and esimate impulse
% response
%-------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(211)
  plot(g_matlab,'r-.')
  ylabel('g, exact impulse response')
  title('Exact and Estimate Impulse Response')
  ax=axis;

  subplot(212)
  plot(g)
  xlabel('data pt. #')
  ylabel('g, estimate impulse response')
  ax2=axis;
  ax2=[ax(1) ax(2) ax2(3) ax2(4)];
  axis(ax2);
end

%-------------------------------
% estimate TF from g estimate
%-------------------------------
ttime=clock;                                    % time the calculation
  [mag_g,ph_g]=g2tf(g);                         % magnitude and phase
tgone=etime(clock,ttime);
fprintf('Estimate TF from g :  %2.4f s\n',tgone)

freq=[0:m-1]./(dt*m);                           % frequency (Hz)

%-------------------------------
% Plot TF estimate and compare to
% exact
%-------------------------------
if plots
  figure(fig)
```

```
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(211)
  plot(f,mag,freq,mag_g,'r--')
  axis([0 16 0 2])
  grid
  ylabel('Gain')
  title('Transfer Function Estimate')

  subplot(212)
  plot(f,ph,freq,ph_g-360,'r--')
  axis([0 16 -400 0])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')

  l1=legend('Exact TF','Estimate TF');
  clear l1
end


%--------------------------------
% Save information to .mat file
% for use with tvf_tool.m
%--------------------------------
filename=['main_m' num2str(m) '_' u_input num2str(noise_variance)]
eval(['save ' filename ...
' Minfo Vinfo dt g g_matlab t f mag ph y u N m n Nvar num den noise_variance u_input'])
```

```
%=====================================================================
% main_siso.m
%=====================================================================
% Corinne Ilvedson
%
% Calculates the information matrix M from siso data
%                information vector V
%=====================================================================

close all
clear all
fig=1;

load f18                                    % load data
plots=1;                                    % 1=plot results
                                            % 0=don't plot results

%-------------------------------
% Impulse Parameters
%-------------------------------
n                                           % # data points in data
N=6400                                      % # data points to look at
m=640;                                      % # of previous input pts the
                                            % response depends on
%-------------------------------
% Pad input and output data
%-------------------------------
u=u(1:N);                                   % keep first N data points
y=y(1:N);
t=t(1:N);

n=N;

pad =sum(u(1:100,1))*ones(m,1)/100;         % pad front of data w/ constant
ypad=sum(y(1:100,1))*ones(m,1)/100;         %   = avg value of data at beginning

u =[pad; u];
y =[ypad; y];
t=[t; t(2:m+1)+t(n)];

n=n+m;

N=n;

clear pad ypad

%-------------------------------
% Plot input and output data
%-------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;
```

```
  subplot(211)
  plot(t,u)
  grid
  ylabel('INPUT u')

  subplot(212)
  plot(t,y)
  grid
  xlabel('time (s)')
  ylabel('OUTPUT y')
end

%--------------------------------
% Find information matrix, vector
% and impulse response using
% imp_est.m
%--------------------------------
ttime=clock;                              % time the calculation
  [Minfo,Vinfo,g]=imp_est(y,u,m,N);       % info matrix, vector and
tgone=etime(clock,ttime);                 % impulse resonse estimate
fprintf('Impulse 2          :  %2.4f s\n',tgone)

%--------------------------------
% Plot impulse response
%--------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(311)
  plot(g)
  xlabel('data pt. #')
  ylabel('g, impulse response')
  title('Impulse Response and Transfer Functions')
end

%--------------------------------
% estimate TF from g estimate
%--------------------------------
ttime=clock;                              % time the calculation
  [mag_g,ph_g]=g2tf(g);                   % magnitude and phase
tgone=etime(clock,ttime);
fprintf('Estimate TF g      :  %2.4f s\n',tgone)

freq=[0:m-1]./(dt*m);                     % frequency (Hz)

%--------------------------------
% Plot TF estimate
%--------------------------------
if plots
  subplot(312)
  plot(freq,mag_g)
  axis([0 16 0 .4])
```

```
  grid
  ylabel('Gain')

  subplot(313)
  plot(freq,ph_g-180)
  axis([0 16 -400 200])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')
end

%-------------------------------
% Save information to .mat file
% for use with tvf_tool.m
%-------------------------------
filename = ['main15_exp' num2str(m)]
eval(['save /var/tmp/' filename ' Minfo Vinfo dt g t y u N m n'])
```

```
%=====================================================================
% main_miso.m
%=====================================================================
% Corinne Ilvedson
%
% Calculates the information matrix M from miso data
%                information vector V
%=====================================================================

close all
clear all
fig=1;

load f18_mimo2_trend2                          % load data
plots=1;                                       % 1=plot results
                                               % 0=don't plot results


%--------------------------------
% Impulse Parameters
%--------------------------------
N=n;                                           % # data points to look at
m=320;                                         % # of previous input pts the
                                               % response depends on
%--------------------------------
% Pad input and output zeros with
% constant equal to average
% of beginning data points.
%--------------------------------

pad=[sum(u(1:100,1))*ones(m,1)/100    sum(u(1:100,2))*ones(m,1)/100];
pad2=[sum(u2(1:100,1))*ones(m,1)/100  sum(u2(1:100,2))*ones(m,1)/100];

ypad=sum(y(1:100,1))*ones(m,1)/100;
ypad2=sum(y2(1:100,1))*ones(m,1)/100;

u =[pad; u];
u2=[pad2; u2];

y =[ypad; y];
y2=[ypad2; y2];

t=[t; t(2:m+1)+t(n)];

n=n+m;
N=n;

clear pad pad2 ypad ypad2
%--------------------------------
% Plot input and output
%--------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;
```

```
  subplot(311)
  plot(t,u(:,1))
  grid
  ylabel('INPUT u_{left}')
  title('Symmetric')

  subplot(312)
  plot(t,u(:,2))
  grid
  ylabel('INPUT u_{right}')

  subplot(313)
  plot(t,y)
  grid
  xlabel('time (s)')
  ylabel('OUTPUT y_{left}')

  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(311)
  plot(t,u2(:,1))
  grid
  ylabel('INPUT u_{left}')
  title('Asymmetric')

  subplot(312)
  plot(t,u2(:,2))
  grid
  ylabel('INPUT u_{right}')

  subplot(313)
  plot(t,y2)
  grid
  xlabel('time (s)')
  ylabel('OUTPUT y_{left}')
end

%--------------------------------
% Find information matrix, vector
% and impulse response
%--------------------------------
save /var/tmp/temp1 y u y2 u2 m N            % do this to clear out memory
save /var/tmp/temp2                          % for calculating Minfo and Vinfo
clear all
pack
load /var/tmp/temp1

ttime=clock;                                 % time the calculation
  [Minfo,Vinfo,g]=imp_miso(y,u,y2,u2,m,N);   % info matrix, vector and
tgone=etime(clock,ttime);                    % impulse resonse estimate
fprintf('Impulse 2          :  %2.4f s\n',tgone)
```

```
load /var/tmp/temp2

%--------------------------------
% Plot impulse response
%--------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(211)
  plot(g(:,1))
  xlabel('data pt. #')
  ylabel('g, impulse response (L to L)')
  ax2=axis;

  subplot(212)
  plot(g(:,2))
  xlabel('data pt. #')
  ylabel('g, impulse response (L to R)')
  ax2=axis;
end

%--------------------------------
% estimate TF from g estimate
%--------------------------------
[mag_g1,ph_g1]=g2tf(g(:,1));              % magnitude and phase
[mag_g2,ph_g2]=g2tf(g(:,2));

freq=[0:m-1]./(dt*m);

%--------------------------------
% Plot TF estimate
%--------------------------------
if plots
  figure(fig)
  set(fig,'Position',[548 389 560 440])
  fig=fig+1;

  subplot(221)
  plot(freq,mag_g1)
  axis([0 16 0 0.2])
  grid
  ylabel('Gain')
  title('Left Input to Output')

  subplot(223)
  plot(freq,ph_g1)
  axis([0 16 -900 100])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')
```

```
  subplot(222)
  plot(freq,mag_g2)
  axis([0 16 0 0.2])
  grid
  ylabel('Gain')
  title('Right Input to Output')

  subplot(224)
  plot(freq,ph_g2)
  axis([0 16 -900 100])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')
end

%--------------------------------
% Save information to .mat file
% for use with tvf_tool_miso.m
%--------------------------------

filename = ['miso_m' num2str(m)];

eval(['save /var/tmp/' filename ' Minfo Vinfo dt g t y u m N n'])
eval(['save ' filename '_tf ph_g1 ph_g2 mag_g1 mag_g2 freq'])
```

```
%=========================================================================
% rand_input.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/13/98
%
% Generate White Noise Input with a frequency rolloff at a
% specified corner frequency.
%
% INPUT ARGUMENTS   n :  # of points desired in input vector
%                   dt:  time step
%                   wc:  corner frequency for rolloff (rad/s)
% OUTPUT            u :  input vector
%
% function [u]=rand_input(n,dt,wc)
%=========================================================================

function [u]=rand_input(n,dt,wc)

randn('seed',0)
u=randn(n,1);                          % random numbers

w=[-n/2:n/2-1]'*2*pi/(n*dt);           % frequency (rad/s)

%----------------------------
% Create filter with rolloff
% at wc rad/s
%----------------------------

gain_num=1;                            % the filter
gain_den=[1/wc 1];
gain=nyquist(gain_num,gain_den,w);
gain=fftshift(gain);

f=[0:n-1]'/(n*dt);                     % in case want to plot filter

u=real(ifft(fft(u).*gain));            % filter the random numbers
                                       % to get input
```

```
%=========================================================================
% imp_est.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/9/98
%
% Finds the impulse response, g via an information matrix/vector method
% This code follows the math of Ilvedson MIT MS Thesis
%                   "Transfer Function Estimation Using Time-Frequency
%                    Analysis" 1998         Section 2.2.1
%
% INPUT ARGUMENTS   y :  system output
%                   u :  system input
%                   m :  y depends on m previous u data points
%                   N :  will only consider first N data points of y and u
%                        N >= m
% OUTPUTS           Minfo: information matrix
%                   Vinfo: information vector
%                   g    : impulse response estimate
%
% function [Minfo, Vinfo, g] = imp_est(y,u,m,N)
%=========================================================================

function [Minfo, Vinfo, g] = imp_est(y,u,m,N)

u_temp=u(1:N);
phiuu=zeros(m,1);                       % autocorrelation of u
phiuy=zeros(m,1);                       % cross correlation of u and y

%--------------------------------------
%Calculate U'U and U'y the fast way
%--------------------------------------
for i=1:m
  phiuu(i)=u(1:N)'*u_temp;
  phiuy(i)=y(1:N)'*u_temp;
  u_temp=[u_temp(N); u_temp(1:(N-1))];   % shift by one data point
end

A=toeplitz(phiuu);

%--------------------------------------
% Calculate extra terms
%--------------------------------------
u_temp=flipud(u(1:N));
for i=1:m-1;
  u_temp=[u_temp(N); u_temp(1:(N-1))];   % shift by one data point
  W(i,:)=u_temp(1:m)';
end

Minfo=A-W'*W;                           % information matrix
Vinfo=phiuy-W'*y(1:m-1);                % information vector

g=Minfo\Vinfo;                          % least squares estimate of
                                        % impulse response
```

```
%=========================================================================
% imp_miso.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/28/98
%
% Finds the impulse response, g via an information matrix/vector method
% Adds the information from two different data sets.
%
% This code follows the math of Ilvedson MIT MS Thesis
%                    "Transfer Function Estimation Using Time-Frequency
%                     Analysis" 1998         Section 2.2.1
%
% INPUT ARGUMENTS   y  :  system output from 1st data set
%                   u  :  system input from 1st data set
%                   y2 :  system output from 2nd data set
%                   u2 :  system input from 2nd data set
%                   m  :  y depends on m previous u data points
%                   N  :  will only consider first N data points of y and u
%                         N >= m
% OUTPUTS           Minfo: information matrix
%                   Vinfo: information vector
%                   g    : impulse responses (mx2 matrix)
%
% function [Minfo, Vinfo,g] = imp_miso(y,u,y2,u2,m,N)
%=========================================================================

function [Minfo, Vinfo,g] = imp_miso(y,u,y2,u2,m,N)

p=2;                                            % number of inputs

%==============================
% Symmetric
%==============================
ttime=clock;                                    % time the calculation

%Calculate U'U and U'y the fast way
u_temp=u(1:N,:);
phiuu=zeros(m,p,p);                             % autocorrelation of u
phiuy=zeros(p*m,1);                             % cross correlation of u and y
A=zeros(p*m);                                   % Toeplitz of phiuu

for i=1:m
  for j=1:p
    for k=1:p
      phiuu(i,j,k)=u(1:N,j)'*u_temp(:,k);
    end
    phiuy((i+m*(j-1)),1)=y(1:N)'*u_temp(:,j);
  end
  u_temp=[u_temp(N,:); u_temp(1:(N-1),:)];      % shift by one data point
end

for i=1:p
  for j=1:p
```

```
    A((1:m)+m*(i-1),(1:m)+m*(j-1)) = toeplitz(phiuu(:,i,j));
  end
end

% Calculate extra terms
u_temp=flipud(u(1:N,:));
for i=1:m-1;
  u_temp=[u_temp(N,:); u_temp(1:(N-1),:)];     % shift by one data point
  Ue(i,:)=u_temp(1:m,1)';
  We(i,:)=u_temp(1:m,2)';
end

Msubtract=[Ue'*Ue Ue'*We;                      % extra terms
           We'*Ue We'*We];
Vsubtract=[Ue'*y(1:m-1);
           We'*y(1:m-1)];

MinfoS=A-Msubtract;                            % information matrix
VinfoS=phiuy-Vsubtract;                        % information vector

clear Msubtract Vsubtract Ue We u_temp A phiuu phiuy u y

tgone=etime(clock,ttime);
fprintf('M V symm :  %2.4f s\n',tgone)
%============================
% Anit-symmetric
%============================
ttime=clock;                                   % time the calculation
u=u2;
y=y2;

%Calculate U'U and U'y the fast way
u_temp=u(1:N,:);
phiuu=zeros(m,p,p);                            % autocorrelation of u
phiuy=zeros(p*m,1);                            % cross correlation of u and y
A=zeros(p*m);                                  % Toeplitz of phiuu

for i=1:m
  for j=1:p
    for k=1:p
      phiuu(i,j,k)=u(1:N,j)'*u_temp(:,k);
    end
    phiuy((i+m*(j-1)),1)=y(1:N)'*u_temp(:,j);
  end
  u_temp=[u_temp(N,:); u_temp(1:(N-1),:)];     % shift by one data point
end

for i=1:p
  for j=1:p
    A((1:m)+m*(i-1),(1:m)+m*(j-1)) = toeplitz(phiuu(:,i,j));
  end
end

% Calculate extra terms
```

```
u_temp=flipud(u(1:N,:));
for i=1:m-1;
  u_temp=[u_temp(N,:); u_temp(1:(N-1),:)];     % shift by one data point
  Ue(i,:)=u_temp(1:m,1)';
  We(i,:)=u_temp(1:m,2)';
end

clear u_temp phiuu u u2 y2

Msubtract=[Ue'*Ue Ue'*We;                      % extra terms
           We'*Ue We'*We];
Vsubtract=[Ue'*y(1:m-1);
           We'*y(1:m-1)];

MinfoA=A-Msubtract;                            % information matrix
VinfoA=phiuy-Vsubtract;                        % information vector

clear Msubtract Vsubtract Ue We A phiuy y

tgone=etime(clock,ttime);
fprintf('M V asym :  %2.4f s\n',tgone)

%============================
% Impulse
%============================
Minfo=MinfoS+MinfoA;                           % add information from each
Vinfo=VinfoS+VinfoA;                           % data set

clear MinfoS MinfoA VinfoS VinfoA

ttime=clock;                                   % time the calculation
  g=Minfo\Vinfo;                               % least squares estimate of
tgone=etime(clock,ttime);                      % impulse response
fprintf('inv(M)V :  %2.4f s\n',tgone)

g=[g(1:m) g(m+1:2*m)];
```

```
%=========================================================================
% g2tf.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/29/98
%
% Computes transfer function from impulse response
% INPUT ARGUMENTS   g :  impulse response
% OUTPUT            m :  magnitude
%                   p :  phase (degrees)
%
% function [m,p]=g2tf(g)
%=========================================================================

function [m,p]=g2tf(g)

tf_g=fft(g);                    %transfer function real and imag parts

m = abs(tf_g);                  % magnitude
p = unwrap(angle(tf_g));        % phase (rad)
p = p*180/pi;                   % phase (degrees)
```

```
%=========================================================================
% tvf_tool.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Transforms the impulse response to time frequency mapping and
% presents a tool for user to distinguish noise from physical dynamics
% of the system.  A cleaned estimate of the transfer function is
% reconstructed and then used to obtain the transfer function estimate.
%
% Code loads a .mat file created by main_num.m or main_siso.m
%=========================================================================

close all
clear all
fig=1;

exp=1;                              % 1 = experimental data
                                    % 0 = numerical example data
%--------------------------------
% Impulse Parameters
%--------------------------------
if exp==1
  load /var/tmp/main15_exp640       % load .mat file created by main_num.m
else
  load main_m640_n4                 % load .mat file created by main_num.m
end

segments=5;                         % number of adjacent data blocks to
                                    %       break impulse response into
han = 1;                            % if 1, use hanning windows
                                    % if 0, use boxcar windows
lap = 1;                            % if 1, create additional overlapping
                                    %       data blocks
cost = 'G';                         % type of cost function to use
                                    % if y, J = integral |y - g*u|^2 dt
                                    % if g,
                                    %  J = integral |g exact - g est|^2 dt
                                    % if G,
                                    %  J = integral |G exact - G est|^2 df
if exp==1
  cost = 'y';                       % This is the only choice for experimental
end                                 % data


%There must be a better way than these global variables but after hours
%of fighting with matlab, had to resort to them...
global ADDRD MODE PBDONE DJTEXT DJ EXIT JINFO JINFO2
global frameJ frame1 frame2 frame3 frame4 frame5
global JmatA JmatS track a

%--------------------------------
% Create window for tool
```

```
%--------------------------------
figure(fig)
set(fig,'Position',[32         328        1120         500])
fig=fig+1;

subplot(4,21,1:10)
if exp==1
  p1=plot(g);                          % rough initial estimate of impulse
else                                   %       response
  p1=plot(g_matlab*dt);                % exact impulse response
end
set(p1,'erasemode','xor')
ylabel('g, impulse response')
ax=axis;
ax=[0 m ax(3) ax(4)];
axis(ax)

%--------------------------------
% Divide g into data blocks
%--------------------------------
ns=m/segments;                         % # of data points in each data block

hold on
vline=[];                              % draw divisions between data blocks
hline=[];
for i=1:segments-1
  vline=[vline [ax(3);       ax(4)]];
  hline=[hline [i*ns+1; i*ns+1]];
end
p2=plot(hline,vline,'r:');
set(p2,'erasemode','xor')

%--------------------------------
% Find t vs f mapping matrix
%--------------------------------

if lap==0
  [a,ac,F] = tvf(g,segments,dt,han);
else
  [a,ac,F] = tvf_lap(g,segments,dt,han);
end

[hh,ww]=size(a);

revision=0;                            % number of times going through the
                                       %    iterations of reconstructing g
exit=0;                                % stop iterations when exit=1

while exit==0
  %--------------------------------
  % Plot t vs f matrix
  %--------------------------------
  if revision==0
    track=[];                          % matrix that keeps track of which bins to
```

```
                                %    keep and which to throw away.
  JmatA=[];                     % matrix that looks at change in cost
                                %    function if any one of discarded
                                %    bins was added to those being kept
  JmatS=[];                     % matrix that looks at change in cost
                                %    function if any one of bins kept
                                %    was discarded.
end
track=tvf_mark(F,lap,revision,JmatA,JmatS);  % choose bins to keep using
                                             % graphical user interface
disp('processing bin choices')
exit=get(EXIT,'val');           % check on value of exit

if exit==0
  %--------------------------------
  % Process bins choices and then
  % find new information matrix
  % and vector
  %--------------------------------
  [T,Tkeep,unT,Ttoss]=Tbasis(track,ns,segments,han,lap);
                                % find transformation matrix

  Minfo2=T'*Minfo*T;            % new information matrix
  Vinfo2=T'*Vinfo;              % new information vector

  a_hat=Minfo2\Vinfo2;          % coefficients corresponding to the
                                % the basis functions kept

  g_hat=T*a_hat;                % new estimate of impulse response

  %--------------------------------
  % Plot new impulse response estimate
  %--------------------------------
  subplot(4,21,1:10)
  if revision==0
    p3=plot(g_hat,'r-');        % estimate impulse response
    set(p3,'erasemode','xor')
    ylabel('Current Case: g')
    axis(ax)
  else
    set(p3,'YData',g_hat)
  end
  gmax=max(g_hat);              % set axis of plot
  if gmax>ax(4)
    ax2=[ax(1) ax(2) ax(3) gmax];
    if isequal(axis,ax2)==0
      axis(ax2)
    end
  else
    if isequal(axis,ax)==0
      axis(ax)
    end
  end
```

```
%-------------------------------
% Plot old impulse response estimate
%-------------------------------
subplot(4,21,22:31)
if revision~=0
  if revision==1
    if exp==1
      p4=plot(g_hat,'r-');      % current estimate of impulse response
    else
      p4=plot(g_matlab*dt);     % exact impulse response
    end
    set(p4,'erasemode','xor')
    hold on
    p5=plot(g_old,'g-');        % old esimate of impulse response
    set(p5,'erasemode','xor')
    p6=plot(hline,vline,'r:'); % division between data block
    set(p6,'erasemode','xor')
    xlabel('data pt. #')
    ylabel('Previous Case: g')
  else
    if exp==1
      set(p4,'YData',g_hat)     % current estimate of impulse response
    end
    set(p5,'YData',g_old)       % old estimate of impulse response
  end
  gmax=max(g_old);              % set axis of plot
  if gmax>ax(4)
    ax2=[ax(1) ax(2) ax(3) gmax];
    if isequal(axis,ax2)==0
      axis(ax2)
    end
  else
    if isequal(axis,ax)==0
      axis(ax)
    end
  end
end

%-------------------------------
% Find and Plot tf while comparing
% to exact transfer function
%-------------------------------

[mag_g_hat,ph_g_hat]=g2tf(g_hat); % transfer function estimate
freq=[0:m-1]./(dt*m);             % corresponding frequency vector (Hz)

subplot(4,21,43:52)
if revision==0
  if exp~=1
    p7=plot(f,mag);                 % exact transfer function
    set(p7,'erasemode','xor')
    hold on
  end
  p8=plot(freq,mag_g_hat,'r');    % estimate
```

```
  set(p8,'erasemode','xor')
  hold on
  ylabel('Gain')
  if exp==1
    axis([0 16 0 .4])
  else
    axis([0 16 0 2])
  end
  grid
elseif revision==1
  set(p8,'YData',mag_g_hat)       % estimate
  p9=plot(freq,m_old,'g');        % old estimate
  set(p9,'erasemode','xor')
else
  set(p8,'YData',mag_g_hat)       % estimate
  set(p9,'YData',m_old)           % old estimate
end

subplot(4,21,64:73)
if revision==0
  if exp~=1
    p10=plot(f,ph);                 % exact transfer function
    set(p10,'erasemode','xor')
    hold on
  end
  p11=plot(freq,ph_g_hat,'r');    % esimate
  set(p11,'erasemode','xor')
  hold on
  axis([0 16 -400 0])
  grid
  xlabel('Frequency (Hz)')
  ylabel('Phase (deg)')
  %l2=legend('tf','tf_{hat}');
elseif revision==1
  set(p11,'YData',ph_g_hat)       % estimate
  p12=plot(freq,p_old,'g');       % old estimate
  set(p12,'erasemode','xor')
  %l2=legend('tf','tf_{hat}','prev tf_{hat}');
else
  set(p11,'YData',ph_g_hat)       % estimate
  set(p12,'YData',p_old)          % old estimate
end

g_old=g_hat;                      % old estimates
m_old=mag_g_hat;
p_old=ph_g_hat;

%--------------------------------
% Cost Function
%--------------------------------
Jy=ycost(y,Vinfo2,a_hat,m,N,dt);  % type y cost function
if exp~=1
  Jg=gcost(g_matlab,g_hat,m,dt);  % type g cost function
  JG=Gcost_a(mag_g_hat,ph_g_hat,freq,num,den,5,12); %type G cost function
```

```
end
if revision+1<10                    % form vector of cost functions for each
                                    %      iteration
  Jyinfo_mat(revision+1,:)=...
      ['0' num2str(revision+1) '  ' num2str(Jy,'%1.2e')];
  if exp~=1
    Jginfo_mat(revision+1,:)=...
      ['0' num2str(revision+1) '  ' num2str(Jg,'%1.2e')];
    JGinfo_mat(revision+1,:)=...
      ['0' num2str(revision+1) '  ' num2str(JG,'%1.2e')];
  end
else
  Jyinfo_mat(revision+1,:)=...
      [num2str(revision+1) '  ' num2str(Jy,'%1.2e')];
  if exp~=1
    Jginfo_mat(revision+1,:)=...
      [num2str(revision+1) '  ' num2str(Jg,'%1.2e')];
    JGinfo_mat(revision+1,:)=...
      [num2str(revision+1) '  ' num2str(JG,'%1.2e')];
  end
end
if strcmp(cost,'y')
  J=Jy;
  Jinfo_mat=Jyinfo_mat;
elseif strcmp(cost,'g')
  J=Jg;
  Jinfo_mat=Jginfo_mat;
elseif strcmp(cost,'G')
  J=JG;
  Jinfo_mat=JGinfo_mat;
end
set(JINFO2,'str',Jinfo_mat)
%Tkeep                              % display numbers of bins being kept
dof(revision+1)=length(T(1,:));     % number of dof's kept

titleJ = sprintf('J = %4.4e',J);
subplot(4,21,1:10)
title(titleJ)

%--------------------------------
% Change in cost function if add
% or subract a bin?
%--------------------------------
if exp==1
  g_matlab=[];
  num=[];
  den=[];
end
[JmatA] = add_bin(J,Ttoss,T,unT,Minfo,Vinfo,Minfo2,Vinfo2,...
    y,dt,N,m,segments,ns,F,lap,g_matlab,cost,num,den);

if strcmp(cost,'g') | strcmp(cost,'y') %sub_bin doesn't calculate G cost
  [JmatS] = sub_bin(J,Tkeep,T,Minfo,Vinfo,y,dt,N,m,segments,...
    ns,F,lap,g_matlab,cost);
```

```
    else
      JmatS=zeros(size(JmatA));
    end
  end
  revision=revision+1;
  %exit=1;                                % uncomment this line if you don't
                                          % want to iterate
end

Jinfo=str2num(Jinfo_mat(:,5:end));        % vector of cost function over the
                                          % iterations
%--------------------------------
% Plot cost vs # dof's
%--------------------------------
if revision>1
  figure
  plot(dof,Jinfo)
  xlabel('number of dof''s')
  ylabel('J')
end

%--------------------------------
% Save reconstruction to .mat file
%--------------------------------
if exp==1
  filename = ['main15_exp' num2str(m)]
else
  filename = ['m' num2str(m) '_s' num2str(segments) '_n' ...
      num2str(noise_variance) '_' u_input];
end

eval(['save ' filename ' dof Jinfo g_hat dt freq mag_g_hat ph_g_hat Tkeep'])

%--------------------------------
% Compare transfer function
% estimate with exact transfer
% function
%--------------------------------
if 1==1
  if exp~=1
    [zz,i4a]=min(abs(freq-5));          % find freq pt closest to 5 Hz
    [zz,i4b]=min(abs(freq-12));         % find freq pt closet to 12 Hz
    clear zz

    fprintf('f(%d)=%4.4f Hz\n',i4a,freq(i4a))
    fprintf('f(%d)=%4.4f Hz\n',i4b,freq(i4b))

    Jp=Gcost_p(mag_g_hat(i4a:i4b),ph_g_hat(i4a:i4b),freq(i4a:i4b),num,den)
    Ja=Gcost_a(mag_g_hat,ph_g_hat,freq,num,den,5,12)
  end
end
```

```
%=========================================================================
% tvf_tool_miso.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Transforms the impulse response to time frequency mapping and
% presents a tool for user to distinguish noise from physical dynamics
% of the system.  A cleaned estimate of the transfer function is
% reconstructed and then used to obtain the transfer function estimate.
%
% Code loads a .mat file created by main_miso.m
%=========================================================================

close all
clear all
fig=1;

%-------------------------------
% Impulse Parameters
%-------------------------------
load /var/tmp/miso_m320             % load .mat file created by main_miso.m
load miso_m320_tf                   % load .mat file created by main_miso.m

segments=4;                         % number of adjacent data blocks to
                                    %       break impulse response into
han = 1;                            % if 1, use hanning windows
                                    % if 0, use boxcar windows
lap = 1;                            % if 1, create additional overlapping
                                    %       data blocks

%-------------------------------
% Plot new and old estimates of g
%-------------------------------
figure(fig)
orient landscape
set(fig,'Position',[548 389 560 440])
fig=fig+1;

subplot(311)
plot(g(:,1))
ylabel('g, impulse response')
titleg='LEFT Impulse Response : Reconstruction Using T Matrix';
title(titleg)
ax=axis;

figure(fig)
orient landscape
set(fig,'Position',[682   493   560   440])
fig=fig+1;

subplot(311)
plot(g(:,2))
ylabel('g')
```

```
titleg2='RIGHT Impulse Response : Reconstruction Using T Matrix';
title(titleg2)
ax2=axis;

%-------------------------------
% Divide g into data blocks
%-------------------------------
ns=m/segments;                      % # of data points in each data block

vline=[];                           % draw divisions between data blocks
hline=[];
vline2=[];
hline2=[];
for i=1:segments-1
  vline=[vline [ax(3);       ax(4)]];
  hline=[hline [i*ns+1; i*ns+1]];

  vline2=[vline2 [ax2(3);     ax2(4)]];
  hline2=[hline2 [i*ns+1; i*ns+1]];
end

figure(fig-2)
subplot(311)
hold on
plot(hline,vline,'r:')

figure(fig-1)
subplot(311)
hold on
plot(hline2,vline2,'r:')

%-------------------------------
% Find t vs f mapping matrix
%-------------------------------
if lap==0
  [a,ac,F]    = tvf(g(:,1),segments,dt,han);
  [a2,ac2,F2] = tvf(g(:,2),segments,dt,han);
else
  [a,ac,F]    = tvf_lap(g(:,1),segments,dt,han);
  [a2,ac2,F2] = tvf_lap(g(:,2),segments,dt,han);
end

[hh,ww]=size(a);

%-------------------------------
% Plot t vs f matrix and have
% user select bins
%-------------------------------

global ADDRD MODE PBDONE
[fig,track]=tvf_mark_miso(fig,a,F,lap);
[fig,track2]=tvf_mark_miso(fig,a2,F2,lap);

%-------------------------------
```

```
% Process bins choices and then
% find new information matrix
% and vector
%--------------------------------
% find transformation matrices
[T,Tkeep,unT,Ttoss]=Tbasis(track,ns,segments,han,lap);
[T2,Tkeep2,unT2,Ttoss2]=Tbasis(track2,ns,segments,han,lap);

T=[T             zeros(size(T2));   % transformation matrix
  zeros(size(T)) T2];

Minfo2=T'*Minfo*T;                  % new information matrix
Vinfo2=T'*Vinfo;                    % new information vector

a_hat=Minfo2\Vinfo2;                % coefficients corresponding to the
                                    % the basis functions kept

g_hat=T*a_hat;                      % new estimate of impulse response
g_hat=[g_hat(1:m) g_hat(m+1:2*m)];

%--------------------------------
% Plot new impulse response estimates
%--------------------------------
figure(fig-4)
subplot(312)
plot(g_hat(:,1))
hold on
plot(hline,vline,'r:')
ylabel('g_{hat}')
axis(ax)

figure(fig-3)
subplot(312)
plot(g_hat(:,2))
hold on
plot(hline2,vline2,'r:')
ylabel('g_{hat}')
axis(ax2)

figure(fig-4)
subplot(313)
plot(g(:,1))
hold on
plot(g_hat(:,1),'m--')
plot(hline,vline,'r:')
l=legend('original','reconstructed');
xlabel('data pt. #')
ylabel('g')
axis(ax)

figure(fig-3)
subplot(313)
plot(g(:,2))
hold on
```

```
plot(g_hat(:,2),'m--')
plot(hline2,vline2,'r:')
l=legend('original','reconstructed');
xlabel('data pt. #')
ylabel('g')
axis(ax2)

clear l

%--------------------------------
% Plot tf estimate
%--------------------------------

[mag_g_hat1,ph_g_hat1]=g2tf(g_hat(:,1));
[mag_g_hat2,ph_g_hat2]=g2tf(g_hat(:,2));
freq=[0:m-1]./(dt*m);

figure(fig)
set(fig,'Position',[682   493   560   440])
fig=fig+1;

subplot(221)
plot(freq,mag_g_hat1)
axis([0 16 0 .2])
grid
ylabel('Gain')
title('Left Input to Output')

subplot(222)
plot(freq,mag_g_hat2)
axis([0 16 0 .2])
grid
ylabel('Gain')
title('Right Input to Output')

subplot(223)
plot(freq,ph_g_hat1)
axis([0 16 -800 200])
grid
xlabel('Frequency (Hz)')
ylabel('Phase (deg)')

subplot(224)
plot(freq,ph_g_hat2)
axis([0 16 -800 200])
grid
xlabel('Frequency (Hz)')
ylabel('Phase (deg)')

%--------------------------------
% Cost Function
%--------------------------------
if 1==0
  J=ycost(y,Vinfo2,a_hat,m,N,dt);              % type y cost function
```

```
  title_add = sprintf('  :  J = %4.4e',J);

  figure(fig-1)
    subplot(221)
    hold on
    title([titletf title_add])
  figure(fig-2)
    subplot(211)
    hold on
    title([titletf title_add])
  figure(fig-3)
    subplot(221)
    hold on
    title([titletf title_add])
  figure(fig-4)
    subplot(221)
    hold on
    title([titletf title_add])
  figure(fig-5)
    title(title_add(6:length(title_add)))
  figure(fig-6)
    title(title_add(6:length(title_add)))
  figure(fig-7)
    subplot(311)
    hold on
    title([titleg2 title_add])
  figure(fig-8)
    subplot(311)
    hold on
    title([titleg title_add])
end


filename=['m' num2str(m) '_s' num2str(segments) '_miso'];

eval(['save ' filename ...
     ' mag_g_hat1 mag_g_hat2 ph_g_hat1 ph_g_hat2 freq Tkeep Tkeep2 dt t u y g_hat'])
```

```
%=========================================================================
% tvf.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/13/98
%
% Divides impulse response into adjacent data blocks and then transforms
% to time frequency mapping
%
% INPUT ARGUMENTS   g        :  impulse response
%                   segments :  number of data blocks to divide g into
%                   dt       :  time step
%                   win      :  type of window to use
%                                  0 = boxcar, 1 = hanning
% OUTPUTS           ac : matrix of fft coefficients (complex)
%                   a  : matrix of magnitude of fft coefficients (real)
%                   F  : corresponding frequency vector (Hz)
%
% function [a,ac,F] = tvf(g,segments,dt,win)
%=========================================================================

function [a,ac,F] = tvf(g,segments,dt,win)

%--------------------------------
% Divide g into segments
%--------------------------------
m=length(g);                        % # of previous input pts the
                                    % response depends on

ns=m/segments;                      % # of data points in each data block

%--------------------------------
% Find t vs f matrix
%--------------------------------
ac=[];                              % matrix of fft coefficients (complex)
han=[0; hanning(ns-1)];             % hanning window
for i=1:segments
  temp=g((1:ns)+((i-1)*ns));        % a data block of g
  if win==1                         % apply window to data block
    temp=temp.*han;
  end
  temp=fft(temp)/ns;                % fourier coefficients of that data block
  temp=temp(1:ns/2+1);              % throw away extra fft info
                                    %        above nyquist freq
  ac=[ac temp];                     % matrix of fft coefficients (complex)
end

a=abs(ac);                          % matrix of magnitude of fft
                                    %        coefficients (real)
F=[0:ns/2]./(dt*ns);                % corresponding freq vector (Hz)
```

```
%=========================================================================
% tvf_lap.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/21/98
%
% Divides impulse response into overlapping data blocks and then
% transforms to time frequency mapping
%
% INPUT ARGUMENTS   g        :  impulse response
%                   segments :  number of segments to divide g into
%                   dt       :  time step
%                   win      :  type of window to use
%                                  0 = boxcar, 1 = hanning
% OUTPUTS           ac : matrix of fft coefficients (complex)
%                   a  : matrix of magnitude of fft coefficients (real)
%                   F  : corresponding frequency vector (Hz)
%
% function [a,ac,F] = tvf_lap(g,segments,dt,win)
%=========================================================================

function [a,ac,F] = tvf_lap(g,segments,dt,win)

%-------------------------------
% Divide g into segments
%-------------------------------
m=length(g);                          % # of previous input pts the
                                      % response depends on

ns=m/segments;                        % # of data points in each data block

%-------------------------------
% Find t vs f matrix
%-------------------------------
ac=[];                                % matrix of fft coefficients (complex)
han=[0; hanning(ns-1)];               % hanning window
for i=1:2*segments+1;
  if i==1 | i==(2*segments+1)         % if the 1st or last data block (1/2 size)
    if i == 1
      temp=g(1:ns/2);                 % 1st data block of g (1/2 size)
      if win==1                       % apply window to data block
        temp=temp.*han(ns/2+1:ns);
      end
    else
      temp=g(m-ns/2+1:m);             % last data block of g (1/2 size)
      if win==1                       % apply window to data block
        temp=temp.*han(1:ns/2);
      end
    end
    temp=fft(temp)/ns;                % fourier coefficients of that data block
    temp=temp(1:ns/8+1);              % throw away extra fft info
                                      %        above nyquist freq
    temp=[temp; zeros(ns/8,1)];       % because data block is 1/2 size there are
                                      %        only 1/2 as many fourier coef's
```

```
                                    %        so have to pad with zeros.
  else                              % if all other data blocks...
    temp=g((1:ns)+((i-2)*ns/2));    % a data block of g
    if win==1                       % apply window to data block
      temp=temp.*han;
    end
    temp=fft(temp)/ns;              % fourier coefficients of that data block
    temp=temp(1:ns/4+1);            % throw away extra fft info
  end                               %        above nyquist freq
  ac=[ac temp];                     % matrix of fft coefficients (complex)
end

a=abs(ac);                          % matrix of magnitude of fft
                                    %        coefficients (real)
F=[0:ns/4]./(dt*ns);                % corresponding freq vector (Hz)

   % Note that F would normally be [0:ns/2]./(dt*ns) however have to throw
   % away more fft coefficients in order to preserve number of degrees of
   % freedom in estimation problem.  This is due to the overlapping bins.
   % Can have a max of m degrees of freedom in estimating g.
```

```
%=========================================================================
% Tbasis.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/27/98
%
% Find the transformation matrix and related info.
%
% INPUT ARGUMENTS   track    : matrix that keeps track of which bins to
%                              keep and which to throw away
%                   ns       : number of pts in each data block
%                   segments : number of data blocks to divide g into
%                   win      : type of window to use
%                              0 = boxcar, 1 = hanning
%                   lap      : 0 = adjacent bins
%                              1 = overlapping bins
% OUTPUTS           T        : transformation matrix of basis function
%                              corresponding to bins being kept
%                   Tkeep    : list of bins being kept
%                   unT      : matrix of basis functions corresponding
%                              to bins being discarded
%                   Ttoss2   : list of bins being discarded
%
% function [T,Tkeep2,unT,Ttoss2] = Tbasis(track,ns,segments,win,lap)
%=========================================================================

function [T,Tkeep2,unT,Ttoss2] = Tbasis(track,ns,segments,win,lap)

m=ns*segments;                                  % y depends on m previous u data
                                                %           points

%---------------------------------------
% Find set of e^j(theta) basis functions
%---------------------------------------
basis = fft(eye(ns));                           % set of basis functions
if win==1
  window=[0; hanning(ns-1)];                    % hanning window
else
  window = ones(ns,1);                          % boxcar window
end

%---------------------------------------
% Convert to a matrix of sin and cos
% basis functions filtered by chosen
% window.  This is the set of basis
% functions corresponding to one data
% block of g
%---------------------------------------
nn=1;
  Tsmall(:,nn) = real(basis(:,1)).*window;      % DC basis function ( cos(0) )
  nn=nn+1;
if lap==1
  for i=2:ns/4
    Tsmall(:,nn) = real(basis(:,i)).*window; % cos basis function
```

```
    nn=nn+1;
    Tsmall(:,nn) = imag(basis(:,i)).*window; % sin basis function
    nn=nn+1;
  end
else
  for i=2:ns/2
    Tsmall(:,nn) = real(basis(:,i)).*window; % cos basis function
    nn=nn+1;
    Tsmall(:,nn) = imag(basis(:,i)).*window; % sin basis function
    nn=nn+1;
  end
end
  Tsmall(:,nn) = real(basis(:,i+1)).*window; % differential basis function
                                             % (-1 1 -1 1 -1 1 etc...)
%---------------------------------------
% Find basis functions for 1/2 data
% blocks at beginning and end of g
%---------------------------------------
if lap==1
  Thalf  = Tsmall(:,1:ns/4);
  Thalf2 = [zeros((m-ns/2),ns/4);       Thalf(1:ns/2,:)];
  Thalf  = [Thalf(ns/2+1:ns,:);  zeros((m-ns/2),ns/4)];
end

%---------------------------------------
% Make the big transformation matrix
% corresponding to all data blocks of g.
% This contains all m possible basis
% functions.
%---------------------------------------
if segments==1
  Tbig=Tsmall;
else
  col=Tsmall;
  for i=2:segments;
    col=[col; zeros(size(Tsmall))];
  end
  if lap==1
    Tbig=zeros(m,ns/2*(2*segments-1));
    for i=1:2*segments-1;
      Tbig(:,(i-1)*ns/2+1:i*ns/2)=col;
      col=[col(m-ns/2+1:m,:); col(1:m-ns/2,:)];
    end
  else
    Tbig=zeros(m);
    for i=1:segments;
      Tbig(:,(i-1)*ns+1:i*ns)=col;
      col=[col(m-ns+1:m,:); col(1:m-ns,:)];
    end
  end
end

if lap==1
  Tbig=[Thalf Tbig Thalf2];
```

```
end

%-------------------------------------------------------
% Make list of all the bins to be kept.
%
% Tkeep2 and Ttoss2 contain the index
% of the bins being kept or discarded
% 1 = bin in lower left corner of time-frequency mapping
% 2 = bin directly above bin 1,
% etc.
% In the time-frequency mapping plot, this index increases as
% you go bottom to top, left to right.
%
% Each bin represents two basis functions (sin & cos) except
% first row (DC) and last row (differential).  Therefore the
% index corresponding to the basis functions to be kept
% (Tkeep) or discarded (Ttoss) is slightly different
% than Tkeep2 and Ttoss2.
%-------------------------------------------------------
Tkeep=[];                                   % list of basis functions to keep
Tkeep2=[];                                  % list of bins to keep
Ttoss=[];                                   % list of basis functions to discard
Ttoss2=[];                                  % list of bins to discard
tk=1;
tk2=1;
tt=1;
tt2=1;
if lap==1                                   % if overlapping bins
  for i=1:2*segments+1
    for j=1:ns/4+1
      if i == 1 | i == 2*segments+1         % if first or last column of t-f map
                                            %        (1/2 size bin)
        if j<=ns/8+1                        % if first column
          if j == 1 | j == ns/8+1           % if DC or differential row
            if track(j,i)==1
              Tkeep  = [Tkeep  tk];
              Tkeep2 = [Tkeep2 tk2];
            else
              Ttoss  = [Ttoss  tt];
              Ttoss2 = [Ttoss2 tt2];
            end
            tk=tk+1;
            tk2=tk2+1;
            tt=tt+1;
            tt2=tt2+1;
          else
            if track(j,i)==1
              Tkeep  = [Tkeep tk tk+1];
              Tkeep2 = [Tkeep2 tk2];
            else
              Ttoss =  [Ttoss tt tt+1];
              Ttoss2 = [Ttoss2 tt2];
            end
            tk=tk+2;
```

```
          tk2=tk2+1;
          tt=tt+2;
          tt2=tt2+1;
        end
      end
    else
      if j == 1 | j == ns/4+1
        if track(j,i)==1
          Tkeep  = [Tkeep  tk];
          Tkeep2 = [Tkeep2 tk2];
        else
          Ttoss  = [Ttoss  tt];
          Ttoss2 = [Ttoss2 tt2];
        end
        tk=tk+1;
        tk2=tk2+1;
        tt=tt+1;
        tt2=tt2+1;
      else
        if track(j,i)==1
          Tkeep  = [Tkeep tk tk+1];
          Tkeep2 = [Tkeep2 tk2];
        else
          Ttoss =  [Ttoss tt tt+1];
          Ttoss2 = [Ttoss2 tt2];
        end
        tk=tk+2;
        tk2=tk2+1;
        tt=tt+2;
        tt2=tt2+1;
      end
    end
  end
else                                          % if adjacent bins
  for i=1:segments
    for j=1:ns/2+1
      if j == 1 | j == ns/2+1                 % if DC or differential row
        if track(j,i)==1
          Tkeep  = [Tkeep tk];
          Tkeep2 = [Tkeep2 tk2];
        else
          Ttoss  = [Ttoss  tt];
          Ttoss2 = [Ttoss2 tt2];
        end
        tk=tk+1;
        tk2=tk2+1;
        tt=tt+1;
        tt2=tt2+1;
      else
        if track(j,i)==1
          Tkeep  = [Tkeep tk tk+1];
          Tkeep2 = [Tkeep2 tk2];
        else
```

```
          Ttoss =  [Ttoss tt tt+1];
          Ttoss2 = [Ttoss2 tt2];
        end
        tk=tk+2;
        tk2=tk2+1;
        tt=tt+2;
        tt2=tt2+1;
      end
    end
  end
end

%---------------------------------------
% Find transformation matrices based on
% which bins were kept and which were
% discarded.
%---------------------------------------
  T=Tbig(:,Tkeep);                          % transformation matrix of basis
                                            % function corresponding to bins
                                            % being kept

  unT=Tbig(:,Ttoss);                        % transformation matrix of basis
                                            % function corresponding to bins
                                            % being discarded
```

```
%====================================================================
% tvf_mark.m
%====================================================================
% Corinne Ilvedson
% Last Modified 8/3/98
%
% Plot the time vs. frequency map.  Create tool allowing user to
% select which bins to keep and discard.
%
% INPUT ARGUMENTS   F       : corresponding frequency vector (Hz)
%                   lap     : 0 = adjacent bins
%                             1 = overlapping bins
%                   revision: number of iteration currently on
%                   JmatA   : a matrix containing the change in the
%                             cost J for each bin if it was added
%                             to the bins being kept
%                   JmatS   : a matrix containing the change in the
%                             cost J for each bin if subtraced
%                             from the bins being kept
% OUTPUTS           track   : a matrix indicating which bins were
%                             kept.  Is the same size as matrix [a]
%                             and contains 0's and 1's where 1
%                             means that bin was kept.
%
% function [track]=tvf_mark(F,lap,revision,JmatA,JmatS)
%====================================================================

function [track]=tvf_mark(F,lap,revision,JmatA,JmatS)

global ADDRD MODE PBDONE DJTEXT DJ EXIT JINFO JINFO2
global frameJ frame1 frame2 frame3 frame4 frame5
global pc JmatA_all JmatS_all a_all Fall track a Fall

%-------------------------------------------------------------
% get matrices ready for plotting with pcolor.m
%-------------------------------------------------------------
[h,w]=size(a);                          % [a] is the matrix of coefficient
                                        %      magnitudes corresponding to
                                        %      basis functions
seg=1:w;                                % vector of data block index number

if revision==0
  track=zeros(size(a));
end

a_all=a;                                % pcolor.m leaves off last row and
a_all(h+1,:)=zeros(1,w);                %   column of matrix it plots so have
a_all(:,w+1)=zeros(h+1,1);              %   to add a bogus row and column for
                                        %   purpose of plotting [a]
if revision~=0
  [Amax,Aindex]=max(JmatA);             % max value in JmatA
  [Amax,Aindex2]=max(Amax);

  JmatA_all=JmatA;                      % add bogus row and column for plotting
```

```
  JmatA_all(h+1,:)=zeros(1,w);
  JmatA_all(:,w+1)=zeros(h+1,1);

  JmatS_all=JmatS;                    % add bogus row and column for plotting
  JmatS_all(h+1,:)=zeros(1,w);
  JmatS_all(:,w+1)=zeros(h+1,1);
end

seg_all=[seg seg(w)+1];               % add bogus term for plotting
Fall=[F F(h)+F(2)];                   % add bogus term for plotting

%-------------------------------------------------------------
% will be marking bins w/ x's but need to make sure the color of
% the x show up against pcolor plot.  The following values
% determine if the x is white or black.
%-------------------------------------------------------------
if revision==0
  a_range=max(max(a))-min(min(a));
  a_white=min(min(a))+.25*a_range;
  a_white2=max(max(a))-.1*a_range;
else
  a_range=max(max(JmatA))-min(min(JmatA));
  a_white=min(min(JmatA))+.25*a_range;
  a_white2=max(max(JmatA))-.1*a_range;
end

if revision~=0
  A_range=max(max(JmatA))-min(min(JmatA));
  A_white=min(min(JmatA))+.25*A_range;
  A_white2=max(max(JmatA))-.1*A_range;

  S_range=max(max(JmatS))-min(min(JmatS));
  S_white=min(min(JmatS))+.25*S_range;
  S_white2=max(max(JmatS))-.1*S_range;
end


%-------------------------------------------------------------
% Plot either a or JmatA depending on iteration number
%-------------------------------------------------------------
subplot(1,21,12:21)
if revision == 0
  pc=pcolor(seg_all,Fall,a_all);
else
  pc=pcolor(seg_all,Fall,JmatA_all);
end
  set(pc,'EraseMode','none');
  colorbar
  ylabel('Hz')
  xlabel('segments')
if revision==0
  title('Mark the bins you would like to keep')
else
  title('\DeltaJ if add bin (+ = reduction in cost) bins marked orig')
```

```
end
hold on
%-----------------------------------------------------------------
% Mark areas of plot which don't contain info (only when have
% overlapping bins
%-----------------------------------------------------------------
if lap==1
  xout =[1 2];
  xout2=[w w+1];
  yout=[Fall((h+3)/2) Fall(h+1)];
  plot(xout,yout,'w')
  plot(fliplr(xout),yout,'w')
  plot(xout2,yout,'w')
  plot(fliplr(xout2),yout,'w')
end

%-----------------------------------------------------------------
% Mark the bins that were kept in previous iteration
%-----------------------------------------------------------------
if revision~=0
  for i=1:h
    for j=1:w
      if track(i,j)==1
        if a(i,j)<=a_white | a(i,j)>=a_white2
          plot(j+.5,(Fall(2)/2+Fall(i)),'wx', 'erasemode', 'none')
        else
          plot(j+.5,(Fall(2)/2+Fall(i)),'kx', 'erasemode', 'none')
        end
      end
      if i==Aindex(Aindex2) & j==Aindex2
        if JmatA(i,j)<=A_white | JmatA(i,j)>=A_white2
          plot(j+.5,(Fall(2)/2+Fall(i)),'wd', 'erasemode', 'none')
        else
          plot(j+.5,(Fall(2)/2+Fall(i)),'kd', 'erasemode', 'none')
        end
      %elseif JmatA(i,j)>0     %This will mark bins that have positive values
      %  if JmatA(i,j)<=A_white | JmatA(i,j)>=A_white2
      %    plot(j+.5,(Fall(2)/2+Fall(i)),'ws', 'erasemode', 'none')
      %  else
      %    plot(j+.5,(Fall(2)/2+Fall(i)),'ks', 'erasemode', 'none')
      %  end
      end
    end
  end
end

%-----------------------------------------------------------------
% Colormap information
%-----------------------------------------------------------------
[cmin cmax]=caxis;
cm_length=length(colormap);
cm=colormap;

%-----------------------------------------------------------------
```

```
% General callback used by most of the gui's
% Makes radio buttons mutually exclusive
%-------------------------------------------------------------
if revision<=1
  call=[...
      'me=get(gcf,''CurrentObject'');,',...
      'if(get(me,''Value'') == 1),',...
      'set(get(me,''UserData''),''Value'',0),',...
      'else,',...
      'set(me,''Value'',1),',...
      'end'];
end

%-------------------------------------------------------------
% Create gui's
%-------------------------------------------------------------
if revision==0
  frameJ=uicontrol(gcf,...
    'Sty','frame',...
    'Pos',[0 60 95 420],...
    'BackgroundColor',[.3 .3 1]);

  JINFO=uicontrol(gcf,...
    'Sty','text',...
    'Pos',[5 465 85 15],...
    'String','J Record',...
    'hor','center',...
    'BackgroundColor',[.3 .3 1],...
    'ForegroundColor',[1 1 1]);

  JINFO2=uicontrol(gcf,...                % displays cost info
    'Sty','text',...
    'Pos',[5 65 85 400],...
    'String','',...
    'hor','center');

  frame1=uicontrol(gcf,...
    'Sty','frame',...
    'Pos',[1035 245 85 65],...
    'BackgroundColor',[.3 .3 1]);

  ADDRD(1) = uicontrol(gcf,...            % Mark bins
    'Sty','radio',...
    'Pos',[1040 280 75 25],...
    'String','Mark',...
    'Value',1);

  ADDRD(2) = uicontrol(gcf,...            % Erase bins
    'Sty','radio',...
    'Pos',[1040 250 75 25],...
    'String','Erase',...
    'Value',0);

  for i=1:2
```

```
  set(ADDRD(i),'UserData',ADDRD(:,[1:(i-1),(i+1):2]))
end

frame2=uicontrol(gcf,...
  'Sty','frame',...
  'Pos',[1035 145 85 95],...
  'BackgroundColor',[.3 .3 1]);

MODE(1) = uicontrol(gcf,...              % Row mode
  'Sty','radio',...
  'Pos',[1040 210 75 25],...
  'String','Row',...
  'Hor','left',...
  'Value',1);

MODE(2) = uicontrol(gcf,...              % Column mode
  'Sty','radio',...
  'Pos',[1040 180 75 25],...
  'String','Column',...
  'Hor','left',...
  'Value',0);

MODE(3) = uicontrol(gcf,...              % Bin mode
  'Sty','radio',...
  'Pos',[1040 150 75 25],...
  'String','Bin',...
  'Hor','left',...
  'Value',0);

for i=1:3
  set(MODE(i),'UserData',MODE(:,[1:(i-1),(i+1):3]))
end

set(ADDRD,'CallBack',call);
set(MODE,'CallBack',call);

frame3=uicontrol(gcf,...
  'Sty','frame',...
  'Pos',[1035 105 85 35],...
  'BackgroundColor',[.3 .3 1]);

PBDONE = uicontrol(gcf,...               % Done
  'Style','radio',...
  'Pos',[1040 110 75 25],...
  'String','Done',...
  'Value',0,...
  'Callback','set(PBDONE,''Value'',1)');

frame5=uicontrol(gcf,...
  'Sty','frame',...
  'Pos',[1035 65 85 35],...
  'BackgroundColor',[1 .3 .3],...
  'Visible','off');
```

```
  EXIT = uicontrol(gcf,...                % Exit
      'Sty','radio',...
      'Pos',[1040 70 75 24],...
      'Hor','center',...
      'String','Exit',...
      'Value',0,...
      'Callback','set(EXIT,''Value'',1)',...
      'Visible','off');
else
  set(MODE(1),'val',0)
  set(MODE(3),'val',1)
  set(PBDONE,'val',0)
  set(frame5,'vis','on')
  set(EXIT,'vis','on')
end

if revision==1
  frame4=uicontrol(gcf,...
      'Sty','frame',...
      'Pos',[1035 320 85 110],...
      'BackgroundColor',[.3 .3 1]);

  DJTEXT=uicontrol(gcf,...
      'Sty','text',...
      'Pos', [1040 410 75 15],...
      'String','View',...
      'hor','center',...
      'BackgroundColor',[.3 .3 1],...
      'ForegroundColor',[1 1 1]);

  DJ(1) = uicontrol(gcf,...
      'Sty','radio',...
      'Pos',[1040 385 75 25],...          % plot bin magnitude
      'String','Original',...
      'Hor','left',...
      'Value',0);

  DJ(2) = uicontrol(gcf,...               % plot change in cost if add bin
      'Sty','radio',...
      'Pos',[1040 355 75 25],...
      'String','+ Bin dJ',...
      'Hor','left',...
      'Value',1);

  DJ(3) = uicontrol(gcf,...               % plot change in cost if subtract bin
      'Sty','radio',...
      'Pos',[1040 325 75 25],...
      'String','- Bin dJ',...
      'Hor','left',...
      'Value',0);

  for i=1:3
    set(DJ(i),'UserData',DJ(:,[1:(i-1),(i+1):3]))
  end
```

```
  calldj1=[call ',',...
      'dj1,',...
      'title(''The marked bins'')'];
  calldj2=[call ',',...
      'dj2,',...
      'title(''\DeltaJ if add bin (+ = reduction in cost)'')'];
  calldj3=[call ',',...
      'dj3,',...
      'title(''\DeltaJ if subtract bin (+ = reduction in cost)'')'];

  set(DJ(1),'CallBack',calldj1);
  set(DJ(2),'CallBack',calldj2);
  set(DJ(3),'CallBack',calldj3);
elseif revision>1
  set(DJ(1),'Value',0)
  set(DJ(2),'Value',1)
  set(DJ(3),'Value',0)
end

%-----------------------------------------------------------------
% User chooses bins
%-----------------------------------------------------------------
while get(PBDONE,'Value')==0 & get(EXIT,'Value')==0
  if get(MODE(1),'Value')==1              % Row mode
    [x,y]=ginput(1);                      % Get user input from mouse
    y=y/F(2)+1;
    if y>1 & y<h+1 & x>1                  % if user clicks outside of plot
      if x>w+1                            % may have clicked done
        pause(1)                          % pause so system sets done=1 if
      end                                 % needs to
      if revision~=0
        if get(DJ(1),'Value')==1          % Determines color to plot x
          mat=a;                          %             ^
          white=a_white;                  %             |
          white2=a_white2;                %             |
        elseif get(DJ(2),'Value')==1      %             |
          mat=JmatA;                      %             |
          white=A_white;                  %             |
          white2=A_white2;                %             |
        elseif get(DJ(3),'Value')==1      %             |
          mat=JmatS;                      %             |
          white=S_white;                  %             |
          white2=S_white2;                %             |
        end                               %             |
      else                                %             |
        mat=a;                            %             |
        white=a_white;                    %             |
        white2=a_white2;                  %             V
      end
      if get(PBDONE,'Value')==0
        if x<=w+1
          y=floor(y);
          for i=1:w
```

```
          if get(ADDRD(1),'Value')==1   % Mark w/ 'x'
            track(y,i)=1;
            if mat(y,i)<=white | mat(y,i)>=white2
              plot(i+.5,(y-.5)*F(2),'wx', 'erasemode', 'none')
            else
              plot(i+.5,(y-.5)*F(2),'kx', 'erasemode', 'none')
            end
          else                          % Erase the 'x'
            track(y,i)=0;               % (actually just plots over x with
                                        % same color as bin)
            index=fix((mat(y,i)-cmin)/(cmax-cmin)*cm_length) + 1;
            if index>cm_length; index=cm_length; end
            p=plot(i+.5,(y-.5)*F(2),'wx', 'erasemode', 'none');
            set(p,'Color',cm(index,:))
          end
        end
      end
    end
  end
end

if get(MODE(2),'Value')==1              % Column mode (see Row mode comments)
  [x,y]=ginput(1);
  y=y/F(2)+1;
  if y>1 & y<h+1 & x>1
    if x>w+1
      pause(1)
    end
    if revision~=0
      if get(DJ(1),'Value')==1
        mat=a;
        white=a_white;
        white2=a_white2;
      elseif get(DJ(2),'Value')==1
        mat=JmatA;
        white=A_white;
        white2=A_white2;
      elseif get(DJ(3),'Value')==1
        mat=JmatS;
        white=S_white;
        white2=S_white2;
      end
    else
      mat=a;
      white=a_white;
      white2=a_white2;
    end
    if get(PBDONE,'Value')==0
      if x<=w+1
        x=floor(x);
        for i=1:h
          if get(ADDRD(1),'Value')==1
            track(i,x)=1;
            if mat(i,x)<=white | mat(i,x)>=white2
```

```
            plot(x+.5,(i-.5)*F(2),'wx', 'erasemode', 'none')
          else
            plot(x+.5,(i-.5)*F(2),'kx', 'erasemode', 'none')
          end
        else
          track(i,x)=0;
          index=fix((mat(i,x)-cmin)/(cmax-cmin)*cm_length) + 1;
          if index>cm_length; index=cm_length; end
          p=plot(x+.5,(i-.5)*F(2),'wx', 'erasemode', 'none');
          set(p,'Color',cm(index,:))
        end
      end
    end
  end
end

if get(MODE(3),'Value')==1              % Bin mode (See row mode comments)
  [x,y]=ginput(1);
  y=y/F(2)+1;
  if y>1 & y<h+1 & x>1
    if x>w+1
      pause(1)
    end
    if revision~=0
      if get(DJ(1),'Value')==1
        mat=a;
        white=a_white;
        white2=a_white2;
      elseif get(DJ(2),'Value')==1
        mat=JmatA;
        white=A_white;
        white2=A_white2;
      elseif get(DJ(3),'Value')==1
        mat=JmatS;
        white=S_white;
        white2=S_white2;
      end
    else
      mat=a;
      white=a_white;
      white2=a_white2;
    end
    if get(PBDONE,'Value')==0
      if x<=w+1
        x=floor(x);
        y=floor(y);
        if get(ADDRD(1),'Value')==1
          track(y,x)=1;
          if mat(y,x)<=white | mat(y,x)>=white2
            plot(x+.5,(y-.5)*F(2),'wx', 'erasemode', 'none')
          else
            plot(x+.5,(y-.5)*F(2),'kx', 'erasemode', 'none')
          end
```

```
          else
            track(y,x)=0;
            index=fix((mat(y,x)-cmin)/(cmax-cmin)*cm_length) + 1;
            if index>cm_length; index=cm_length; end
            p=plot(x+.5,(y-.5)*F(2),'kx', 'erasemode', 'none');
            set(p,'Color',cm(index,:))
          end
        end
      end
    end
  end
end
if lap==1                                    % set parts of track that correspond
  track((h+3)/2:h,1)=zeros((h-1)/2,1);       % to nonexistant basis functions to
  track((h+3)/2:h,w)=zeros((h-1)/2,1);       % zero even if user chose those bins
end
```

```
%=========================================================================
% tvf_mark_miso.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 7/16/98
%
% Plot the time vs. frequency map.  Create tool allowing use to select
% which bins to keep and discard.
%
% INPUT ARGUMENTS   fig   : current figure number
%                   a     : matrix of magnitude of fft coefficients (real)
%                   F     : corresponding frequency vector (Hz)
%                   lap   : 0 = adjacent bins
%                           1 = overlapping bins
% OUTPUTS           fig   : updated figure number
%                   track : matrix that keeps track of which bins to
%                                   keep and which to throw away
%
% function [fig,track]=tvf_mark_miso(fig,a,F,lap)
%=========================================================================

function [fig,track]=tvf_mark_miso(fig,a,F,lap)

%-------------------------------------------------------------
% get matrices ready for plotting with pcolor.m
%-------------------------------------------------------------
[h,w]=size(a);                          % [a] is the matrix of coefficient
                                        %      magnitudes corresponding to
                                        %      basis functions
seg=1:w;                                % vector of data block index number

track=zeros(size(a));

a_all=a;                                % pcolor.m leaves off last row and
a_all(h+1,:)=zeros(1,w);                %   column of matrix it plots so have
a_all(:,w+1)=zeros(h+1,1);              %   to add a bogus row and column for
                                        %   purpose of plotting [a]

seg_all=[seg seg(w)+1];                 % add bogus term for plotting
Fall=[F F(h)+F(2)];                     % add bogus term for plotting

%-------------------------------------------------------------
% Plot a
%-------------------------------------------------------------
figure(fig)
set(fig,'Pos',[548 389 560 440])
fig=fig+1;

subplot(1,11,2:11)
pcolor(seg_all,Fall,a_all)
colorbar
ylabel('Hz')
xlabel('segments')
title('Mark the bins you would like to keep')
```

```
hold on
%-------------------------------------------------------------
% Mark areas of plot which don't contain info (only when have
% overlapping bins
%-------------------------------------------------------------
if lap==1
  xout =[1 2];
  xout2=[w w+1];
  yout=[Fall((h+3)/2) Fall(h+1)];
  plot(xout,yout,'w')
  plot(fliplr(xout),yout,'w')
  plot(xout2,yout,'w')
  plot(fliplr(xout2),yout,'w')
end

%-------------------------------------------------------------
% Colormap information
%-------------------------------------------------------------
[cmin cmax]=caxis;
cm_length=length(colormap);
cm=colormap;

%-------------------------------------------------------------
% Create gui's
%-------------------------------------------------------------
done=0;                                 % This value is set to 1 after user
                                        % chooses bins to keep
global ADDRD MODE PBDONE

frame1=uicontrol(gcf,...
    'Sty','frame',...
    'Pos',[5 265 70 65],...
    'BackgroundColor',[.3 .3 1]);

ADDRD(1) = uicontrol(gcf,...            % Mark bins
    'Sty','radio',...
    'Pos',[10 300 60 25],...
    'String','Mark',...
    'Value',1);

ADDRD(2) = uicontrol(gcf,...            % Erase bins
    'Sty','radio',...
    'Pos',[10 270 60 25],...
    'String','Erase',...
    'Value',0);

for i=1:2
  set(ADDRD(i),'UserData',ADDRD(:,[1:(i-1),(i+1):2]))
end

frame2=uicontrol(gcf,...
    'Sty','frame',...
    'Pos',[5 165 70 95],...
    'BackgroundColor',[.3 .3 1]);
```

```
MODE(1) = uicontrol(gcf,...              % Row mode
    'Sty','radio',...
    'Pos',[10 230 60 25],...
    'String','Row',...
    'Hor','left',...
    'Value',1);

MODE(2) = uicontrol(gcf,...              % Column mode
    'Sty','radio',...
    'Pos',[10 200 60 25],...
    'String','Column',...
    'Hor','left',...
    'Value',0);

MODE(3) = uicontrol(gcf,...              % Bin mode
    'Sty','radio',...
    'Pos',[10 170 60 25],...
    'String','Bin',...
    'Hor','left',...
    'Value',0);

for i=1:3
  set(MODE(i),'UserData',MODE(:,[1:(i-1),(i+1):3]))
end

% General callback used by most of the gui's
% Makes radio buttons mutually exclusive
call=[...
      'me=get(gcf,''CurrentObject'');,',...
      'if(get(me,''Value'') == 1),',...
      'set(get(me,''UserData''),''Value'',0),',...
      'else,',...
      'set(me,''Value'',1),',...
      'end'];

set(ADDRD,'CallBack',call);
set(MODE,'CallBack',call);

frame3=uicontrol(gcf,...
    'Sty','frame',...
    'Pos',[5 125 70 35],...
    'BackgroundColor',[.3 .3 1]);

PBDONE = uicontrol(gcf,...               % Done
    'Style','radio',...
    'Pos',[10 130 60 25],...
    'String','Done',...
    'Value',0,...
    'Callback','set(PBDONE,''Value'',1)');

%-----------------------------------------------------------------
% will be marking bins w/ x's but need to make sure the color of
% the x show up against pcolor plot.  The following values
```

```
% determine if the x is white or black.
%-------------------------------------------------------------
a_range=max(max(a))-min(min(a));
a_white=min(min(a))+.25*a_range;
a_white2=max(max(a))-.1*a_range;

%-------------------------------------------------------------
% User chooses bins
%-------------------------------------------------------------
while get(PBDONE,'Value')==0
  if get(MODE(1),'Value')==1              % Row mode
    [x,y]=ginput(1);                      % Get user input from mouse
    y=y/F(2)+1;
    if y>1 & y<h+1 & x<w+1                % if user clicks outside of plot
      if x<1                              % may have clicked done
        pause(1)                          % pause so system sets done=1 if
      end                                 % needs to
      if get(PBDONE,'Value')==0
        if x>=1
          y=floor(y);
          for i=1:w
            if get(ADDRD(1),'Value')==1   % Mark w/ 'x'
              track(y,i)=1;
              if a(y,i)<=a_white | a(y,i)>=a_white2
                plot(i+.5,(y-.5)*F(2),'wx', 'erasemode', 'none')
              else
                plot(i+.5,(y-.5)*F(2),'kx', 'erasemode', 'none')
              end
            else                          % Erase the 'x'
              track(y,i)=0;               % (actually just plots over x with
                                          % same color as bin)
              index=fix((a(y,i)-cmin)/(cmax-cmin)*cm_length) + 1;
              if index>cm_length; index=cm_length; end
              p=plot(i+.5,(y-.5)*F(2),'wx', 'erasemode', 'none');
              set(p,'Color',cm(index,:))
            end
          end
        end
      end
    end
  end

  if get(MODE(2),'Value')==1              % Column mode (see Row mode comments)
    [x,y]=ginput(1);
    y=y/F(2)+1;
    if y>1 & y<h+1 & x<w+1
      if x<1
        pause(2)
      end
      if get(PBDONE,'Value')==0
        if x>=1
          x=floor(x);
          for i=1:h
            if get(ADDRD(1),'Value')==1
```

```
            track(i,x)=1;
            if a(i,x)<=a_white | a(i,x)>=a_white2
              plot(x+.5,(i-.5)*F(2),'wx', 'erasemode', 'none')
            else
              plot(x+.5,(i-.5)*F(2),'kx', 'erasemode', 'none')
            end
          else
            track(i,x)=0;
            index=fix((a(i,x)-cmin)/(cmax-cmin)*cm_length) + 1;
            if index>cm_length; index=cm_length; end
            p=plot(x+.5,(i-.5)*F(2),'wx', 'erasemode', 'none');
            set(p,'Color',cm(index,:))
          end
        end
      end
    end
  end

  if get(MODE(3),'Value')==1              % Bin mode (See row mode comments)
    [x,y]=ginput(1);
    y=y/F(2)+1;
    if y>1 & y<h+1 & x<w+1
      if x<1
        pause(2)
      end
      if get(PBDONE,'Value')==0
        if x>=1
          x=floor(x);
          y=floor(y);
          if get(ADDRD(1),'Value')==1
            track(y,x)=1;
            if a(y,x)<=a_white | a(y,x)>=a_white2
              plot(x+.5,(y-.5)*F(2),'wx', 'erasemode', 'none')
            else
              plot(x+.5,(y-.5)*F(2),'kx', 'erasemode', 'none')
            end
          else
            track(y,x)=0;
            index=fix((a(y,x)-cmin)/(cmax-cmin)*cm_length) + 1;
            if index>cm_length; index=cm_length; end
            p=plot(x+.5,(y-.5)*F(2),'kx', 'erasemode', 'none');
            set(p,'Color',cm(index,:))
          end
        end
      end
    end
  end
end
if lap==1                                 % set parts of track that correspond
  track((h+3)/2:h,1)=zeros((h-1)/2,1);    % to nonexistant basis functions to
  track((h+3)/2:h,w)=zeros((h-1)/2,1);    % zero even if user chose those bins
end
```

```
%=================================================================
% ginput.m
%=================================================================
% Corinne Ilvedson's modification of MATLAB's ginput.m
%
% This is very similar to the matlab command - but had to change the
% line marked by:         *****
% in order for it to look good with tool.
%
%=================================================================
function [out1,out2,out3] = ginput(arg1)
%GINPUT Graphical input from mouse.
%   [X,Y] = GINPUT(N) gets N points from the current axes and returns
%   the X- and Y-coordinates in length N vectors X and Y.  The cursor
%   can be positioned using a mouse (or by using the Arrow Keys on some
%   systems).  Data points are entered by pressing a mouse button
%   or any key on the keyboard except carriage return, which terminates
%   the input before N points are entered.
%
%   [X,Y] = GINPUT gathers an unlimited number of points until the
%   return key is pressed.
%
%   [X,Y,BUTTON] = GINPUT(N) returns a third result, BUTTON, that
%   contains a vector of integers specifying which mouse button was
%   used (1,2,3 from left) or ASCII numbers if a key on the keyboard
%   was used.

%   Copyright (c) 1984-97 by The MathWorks, Inc.
%   $Revision: 5.19 $  $Date: 1997/04/08 06:55:47 $

out1 = []; out2 = []; out3 = []; y = [];
c = computer;
if ~strcmp(c(1:2),'PC') & ~strcmp(c(1:2),'MA')
    tp = get(0,'TerminalProtocol');
else
    tp = 'micro';
end

if ~strcmp(tp,'none') & ~strcmp(tp,'x') & ~strcmp(tp,'micro'),
    if nargout == 1,
        if nargin == 1,
            eval('out1 = trmginput(arg1);');
        else
            eval('out1 = trmginput;');
        end
    elseif nargout == 2 | nargout == 0,
        if nargin == 1,
            eval('[out1,out2] = trmginput(arg1);');
        else
            eval('[out1,out2] = trmginput;');
        end
        if  nargout == 0
            out1 = [ out1 out2 ];
        end
```

```
    elseif nargout == 3,
        if nargin == 1,
            eval('[out1,out2,out3] = trmginput(arg1);');
        else
            eval('[out1,out2,out3] = trmginput;');
        end
    end
else

    fig = gcf;
    figure(gcf);

    if nargin == 0
        how_many = -1;
        b = [];
    else
        how_many = arg1;
        b = [];
        if  isstr(how_many) ...
            | size(how_many,1) ~= 1 | size(how_many,2) ~= 1 ...
            | ~(fix(how_many) == how_many) ...
            | how_many < 0
            error('Requires a positive integer.')
        end
        if how_many == 0
            ptr_fig = 0;
            while(ptr_fig ~= fig)
                ptr_fig = get(0,'PointerWindow');
            end
            scrn_pt = get(0,'PointerLocation');
            loc = get(fig,'Position');
            pt = [scrn_pt(1) - loc(1), scrn_pt(2) - loc(2)];
            out1 = pt(1); y = pt(2);
        elseif how_many < 0
            error('Argument must be a positive integer.')
        end
    end

    pointer = get(gcf,'pointer');
%    set(gcf,'pointer','fullcrosshair');                    %*****
    set(gcf,'pointer','crosshair');                         %*****
    fig_units = get(fig,'units');
    char = 0;

    while how_many ~= 0
        % Use no-side effect WAITFORBUTTONPRESS
        waserr = 0;
        eval('keydown = wfbp;', 'waserr = 1;');
        if(waserr == 1)
      if(ishandle(fig))
        set(fig,'pointer',pointer,'units',fig_units);
        error('Interrupted');
      else
        error('Interrupted by figure deletion');
```

```
  end
    end

    ptr_fig = get(0,'CurrentFigure');
    if(ptr_fig == fig)
        if keydown
            char = get(fig, 'CurrentCharacter');
            button = abs(get(fig, 'CurrentCharacter'));
            scrn_pt = get(0, 'PointerLocation');
            set(fig,'units','pixels')
            loc = get(fig, 'Position');
            pt = [scrn_pt(1) - loc(1), scrn_pt(2) - loc(2)];
            set(fig,'CurrentPoint',pt);
        else
            button = get(fig, 'SelectionType');
            if strcmp(button,'open')
                button = b(length(b));
            elseif strcmp(button,'normal')
                button = 1;
            elseif strcmp(button,'extend')
                button = 2;
            elseif strcmp(button,'alt')
                button = 3;
            else
                error('Invalid mouse selection.')
            end
        end
        pt = get(gca, 'CurrentPoint');

        how_many = how_many - 1;

        if(char == 13) % & how_many ~= 0)
            % if the return key was pressed, char will == 13,
            % and that's our signal to break out of here whether
            % or not we have collected all the requested data
            % points.
            % If this was an early breakout, don't include
            % the <Return> key info in the return arrays.
            % We will no longer count it if it's the last input.
            break;
        end

        out1 = [out1;pt(1,1)];
        y = [y;pt(1,2)];
        b = [b;button];
    end
end

set(fig,'pointer',pointer,'units',fig_units);

if nargout > 1
    out2 = y;
    if nargout > 2
        out3 = b;
```

```
        end
    else
        out1 = [out1 y];
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function key = wfbp
%WFBP   Replacement for WAITFORBUTTONPRESS that has no side effects.

% Remove figure button functions
fprops = {'windowbuttonupfcn','buttondownfcn', ...
          'windowbuttondownfcn','windowbuttonmotionfcn'};
fig = gcf;
fvals = get(fig,fprops);
set(fig,fprops,{'','','',''})

% Remove all other buttondown functions
ax = findobj(fig,'type','axes');
if isempty(ax)
    ch = {};
else
    ch = get(ax,{'Children'});
end
for i=1:length(ch),
    ch{i} = ch{i}(:)';
end
h = [ax(:)',ch{:}];
vals = get(h,{'buttondownfcn'});
mt = repmat({''},size(vals));
set(h,{'buttondownfcn'},mt);

% Now wait for that buttonpress, and check for error conditions
waserr = 0;
eval(['if nargout==0,', ...
      '   waitforbuttonpress,', ...
      'else,', ...
      '   keydown = waitforbuttonpress;',...
      'end' ], 'waserr = 1;');

% Put everything back
if(ishandle(fig))
  set(fig,fprops,fvals)
  set(h,{'buttondownfcn'},vals)
end

if(waserr == 1)
  error('Interrupted');
end

if nargout>0, key = keydown; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%=========================================================================
% dj1.m
%=========================================================================
% Corinne Ilvedson
%
% Callback function for push buttons of tvf_tool.m
%
% function []=dj1()
%=========================================================================

function []=dj1()

global pc a a_all Fall track

cmin=min(min(a));
cmax=max(max(a));

[h,w]=size(a);

range=cmax-cmin;
white=cmin+.25*range;
white2=cmax-.1*range;

for i=1:h
  for j=1:w
    if track(i,j)==1
      if a(i,j)<=white | a(i,j)>=white2
        plot(j+.5,(Fall(2)/2+Fall(i)),'wx', 'erasemode', 'none')
      else
        plot(j+.5,(Fall(2)/2+Fall(i)),'kx', 'erasemode', 'none')
      end
    end
  end
end

set(pc,'cdata',a_all)
caxis([cmin cmax])
colorbar
```

```
%=========================================================================
% dj2.m
%=========================================================================
% Corinne Ilvedson
%
% Callback function for push buttons of tvf_tool.m
%
% function []=dj2()
%=========================================================================
function []=dj2()

global pc JmatA_all JmatA Fall track

cmin=min(min(JmatA));
cmax=max(max(JmatA));

[h,w]=size(JmatA);

range=cmax-cmin;
white=cmin+.25*range;
white2=cmax-.1*range;

[Amax,index]=max(JmatA);
[Amax,index2]=max(Amax);

for i=1:h
  for j=1:w
    if track(i,j)==1
      if JmatA(i,j)<=white | JmatA(i,j)>=white2
        plot(j+.5,(Fall(2)/2+Fall(i)),'wx', 'erasemode', 'none')
      else
        plot(j+.5,(Fall(2)/2+Fall(i)),'kx', 'erasemode', 'none')
      end
    end
    if i==index(index2) & j==index2
      if JmatA(i,j)<=white | JmatA(i,j)>=white2
        plot(j+.5,(Fall(2)/2+Fall(i)),'wd', 'erasemode', 'none')
      else
        plot(j+.5,(Fall(2)/2+Fall(i)),'kd', 'erasemode', 'none')
      end
    %elseif JmatA(i,j)>0       %This will mark bins that have positive values
    %  if JmatA(i,j)<=white | JmatA(i,j)>=white2
    %    plot(j+.5,(Fall(2)/2+Fall(i)),'ws', 'erasemode', 'none')
    %  else
    %    plot(j+.5,(Fall(2)/2+Fall(i)),'ks', 'erasemode', 'none')
    %  end
    end
  end
end

set(pc,'cdata',JmatA_all)
caxis([cmin cmax])
colorbar
```

```
%=========================================================================
% dj3.m
%=========================================================================
% Corinne Ilvedson
%
% Callback function for cost push buttons of tvf_tool.m
%
% function []=dj3()
%=========================================================================
function []=dj3()

global pc JmatS_all JmatS Fall track

cmin=min(min(JmatS));
cmax=max(max(JmatS));
if cmin == 0 & cmax == 0    % for the case when cost = G, there is no
  cmin=-1e-14;              % JmatS info.  Therefore JmatS is a matrix of
  cmax=1e-14;               % zeros.  However, cmin and cmax cannot be equal
end                         % so will set to arbitray value to make matlab
                            % happy.
[h,w]=size(JmatS);

range=cmax-cmin;
white=cmin+.25*range;
white2=cmax-.1*range;

for i=1:h
  for j=1:w
    if track(i,j)==1
      if JmatS(i,j)<=white | JmatS(i,j)>=white2
        plot(j+.5,(Fall(2)/2+Fall(i)),'wx', 'erasemode', 'none')
      else
        plot(j+.5,(Fall(2)/2+Fall(i)),'kx', 'erasemode', 'none')
      end
    end
  end
end

set(pc,'cdata',JmatS_all)
caxis([cmin cmax])
colorbar
```

```
%=========================================================================
% add_bin.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Calculates the change in cost due to adding back just one of the bins
% that had been discarded.  Will tell you which is the most important bin
% to add in order to further mimimize the cost.
%
%
%INPUT ARGUMENTS    J     : current value of cost function
%                   Ttoss : a matrix of the containg the index of the bins
%                           being discarded
%                   T     : the current transformation matrix
%                   unT   : matrix of basis functions not being used in
%                           transformation matrix
%                   Minfo : original information matrix
%                   Vinfo : original information vector
%                   Minfo2: new information matrix
%                   Vinfo2: new information vector
%                   y     : output
%                   dt    : time step (sec)
%                   N     : number of pts of input output being kept
%                   m     : number of pts in impulse response
%                   segments: number of data blocks
%                   ns    : number of pts in each data block
%                   F     : frequecny vector corresponding to time-frequency
%                           mapping
%                   lap   : if 1 --> overlapping bins, if 0 --> adjacent bins
%                   g_matlab : exact impulse response
%                   cost  : type of cost function (y, g or G)
%                   num   : numerator of exact transfer function
%                   den   : denominator of exact transfer function
%
% OUTPUTS           Jmat : a matrix containing the change in the cost J for
%                          each bin if it was added to the bins being kept
%
% function [Jmat] = ...
%    add_bin(J,Ttoss,T,unT,Minfo,Vinfo,Minfo2,Vinfo2,y,dt,N,m,...
%    segments,ns,F,lap,g_matlab,cost,num,den)
%=========================================================================

function [Jmat] = ...
    add_bin(J,Ttoss,T,unT,Minfo,Vinfo,Minfo2,Vinfo2,y,dt,N,m,...
    segments,ns,F,lap,g_matlab,cost,num,den)

bins_left=length(Ttoss);                        % # of bins being discarded
if lap==1                                       % create matrix to keep track
  Jtrack=zeros(1,(segments*2+1)*(ns/4+1));      %   of the change in cost due
else                                            %   to adding any particular bin
  Jtrack=zeros(1,segments*(ns/2+1));
end
```

```
% Use Matrix Inversion Lemma to make quick calculation of new information
% matrix and vector after adding one bin (and therefore 2 basis functions)

A = Minfo2;
Ai = inv(A);

Bpart = T'*Minfo;

kk=1;
for i=1:bins_left
  if lap==1         % USE THIS PART FOR OVERLAPPING BINS
    if Ttoss(i)<=ns/8+1                          % if bin is in first column
                                                 % corresponding to 1/2 size bin
      if mod(Ttoss(i),ns/8+1)==0 | mod((Ttoss(i)-1),ns/8+1)==0
                                                 % if DC or differential freq
        phi = unT(:,kk);                         % basis function to add
        kk=kk+1;
      else
        phi = unT(:,[kk kk+1]);                  % basis functions (sin and cos)
        kk=kk+2;                                 %       to add
      end
    else
      if mod(Ttoss(i)-ns/8-1,ns/4+1)==0 | ...
          mod((Ttoss(i)-ns/8-2),ns/4+1)==0 | ...
          Ttoss(i)==(segments*2-1)*(ns/4+1)+2*(ns/8+1)
                                                 % if DC or differential freq bin
        phi = unT(:,kk);                         % basis function to add
        kk=kk+1;
      else
        phi = unT(:,[kk kk+1]);                  % basis func's to add (sin&cos)
        kk=kk+2;
      end
    end
    if Ttoss(i)<=ns/8+1
      sect=1;                                    % the section or column where
                                                 % the added bin is located in
                                                 % the time-frequency map
      index=1:ns/2;                              %   using sect, can figure
                                                 % out where the basis function
                                                 % is non-zero and where it is
                                                 % zero.  Then we can use the
                                                 % structure of the basis
                                                 % functions to speed up the
                                                 % calculations
    else
      sect=ceil((Ttoss(i)+ns/8)/(ns/4+1));
      if sect==2*segments+1
        index=m-ns/2+1:m;
      else
        index=(1:ns)+((sect-2)*ns/2);
      end
    end
  else          % USE THIS PART FOR ADJACENT BINS
    if mod(Ttoss(i),ns/2+1)==0 | mod((Ttoss(i)-1),ns/2+1)==0
```

```
      phi = unT(:,kk);                          % basis function to add
      kk=kk+1;
    else
      phi = unT(:,[kk kk+1]);                   % basis func's to add (sin&cos)
      kk=kk+2;
    end
    sect=ceil(Ttoss(i)/(ns/2+1));
    index=(1:ns)+((sect-1)*ns);
  end
  Tnew=[T phi];                               % new transfomation matrix now that
                                              % we have add one more bin
  % More Matrix Inversion Lemma
  phi=phi(index,:);                           % only keep non-zero part of basis
                                              % function in order to speed up
                                              % calculation

  phiT=phi';
  CDpart=phiT*Minfo(index,:);

  B = Bpart(:,index)*phi;
  C = CDpart*T;
  D = CDpart(:,index)*phi;

  CABD = inv(C*Ai*B-D);

  Mi1=Ai*B*CABD;
  Mi=[Ai-Mi1*C*Ai  Mi1;
      CABD*C*Ai   -CABD];

  Vnew=[Vinfo2; phiT*Vinfo(index,:)];    % new info vector including extra bin

  a_new=Mi*Vnew;                              % new basis function coefficients
  if strcmp(cost,'y')                         % find new cost
    J2=ycost(y,Vnew,a_new,m,N,dt);
  elseif strcmp(cost,'g')
    g_new=Tnew*a_new;
    J2=gcost(g_matlab,g_new,m,dt);
  elseif strcmp(cost,'G')
    g_new=Tnew*a_new;
    [mag_g_new,ph_g_new]=g2tf(g_new);
    freq=[0:m-1]./(dt*m);
    J2=Gcost_a(mag_g_new,ph_g_new,freq,num,den,5,12);
  end
  if lap==1                                 % compare new and old cost
    if Ttoss(i)<=ns/8+1
      Jtrack(Ttoss(i))=J-J2;
    else
      Jtrack(Ttoss(i)+ns/8)=J-J2;
    end
  else
    Jtrack(Ttoss(i))=J-J2;
  end
end
```

```
if lap==1
  Jmat=reshape(Jtrack,ns/4+1,segments*2+1);  % map delta cost into matrix
else                                         % corresponding to time-frequency
  Jmat=reshape(Jtrack,ns/2+1,segments);      % matrix of bins
end
```

```
%=========================================================================
% sub_bin.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Calculates the change in cost due to subtracting just one of the bins
% that had been kept.  Will tell you which is the most important bin
% to subtract in order to further mimimize the cost.
%
%
%INPUT ARGUMENTS    J     : current value of cost function
%                   Tkeep : a matrix of the containg the index of the bins
%                           being kept
%                   T     : the current transformation matrix
%                   Minfo : original information matrix
%                   Vinfo : original information vector
%                   y     : output
%                   dt    : time step (sec)
%                   N     : number of pts of input output being kept
%                   m     : number of pts in impulse response
%                   segments: number of data blocks
%                   ns    : number of pts in each data block
%                   F     : frequecny vector corresponding to time-frequency
%                           mapping
%                   lap   : if 1 --> overlapping bins, if 0 --> adjacent bins
%                   g_matlab : exact impulse response
%                   cost  : type of cost function (y, g)
%
% OUTPUTS           Jmat : a matrix containing the change in the cost J for
%                          each bin if subtracted from the bins being kept
%
% function [Jmat] = sub_bin(J,Tkeep,T,Minfo,Vinfo,y,dt,N,m,segments,ns,F,...
%    lap,g_matlab,cost)
%=========================================================================

function [Jmat] = sub_bin(J,Tkeep,T,Minfo,Vinfo,y,dt,N,m,segments,ns,F,...
    lap,g_matlab,cost)

bins_used=length(Tkeep);                    % # of bins being kept
if lap==1                                   % create matrix to keep track
  Jtrack=zeros(1,(segments*2+1)*(ns/4+1)); %    of the change in cost due to
else                                        %    subtracting any particular bin
  Jtrack=zeros(1,segments*(ns/2+1));
end

[z,B]=size(T);                              % size of transformation matrix

kk=1;
for i=1:bins_used
  if lap==1         % USE THIS PART FOR OVERLAPPING BINS
    if Tkeep(i)<=ns/8+1                     % if bin is in first column
                                            % corresponding to 1/2 size bin
      if mod(Tkeep(i),ns/8+1)==0 | mod((Tkeep(i)-1),ns/8+1)==0
```

```
                                        % if DC or differential freq
      Tnew=[T(:,1:kk-1) T(:,kk+1:B)];   % new transformation matrix (old
                                        %   minus dc or diff basis
      kk=kk+1;                          %   function)
    else
      Tnew=[T(:,1:kk-1) T(:,kk+2:B)];   % index of basis functions to be
                                        % new transformation matrix (old
      kk=kk+2;                          %   minus sin&cos basis function)
    end
  else
    if mod(Tkeep(i)-ns/8-1,ns/4+1)==0 | ...
        mod((Tkeep(i)-ns/8-2),ns/4+1)==0 | ...
        Tkeep(i)==(segments*2-1)*(ns/4+1)+2*(ns/8+1)
                                        % if DC or differential freq bin
      Tnew=[T(:,1:kk-1) T(:,kk+1:B)];
      kk=kk+1;
    else
      Tnew=[T(:,1:kk-1) T(:,kk+2:B)];
      kk=kk+2;
    end
  end
else         % USE THIS PART FOR ADJACENT BINS
  if mod(Tkeep(i),ns/2+1)==0 | mod((Tkeep(i)-1),ns/2+1)==0
    Tnew=[T(:,1:kk-1) T(:,kk+1:B)];
    kk=kk+1;
  else
    Tnew=[T(:,1:kk-1) T(:,kk+2:B)];
    kk=kk+2;
  end
end

Mnew=Tnew'*Minfo*Tnew;                  % new info matrix
Vnew=Tnew'*Vinfo;                       % new info vector

a_new=Mnew\Vnew;                        % new basis function coef's

if strcmp(cost,'y')                     % new cost
  J2=ycost(y,Vnew,a_new,m,N,dt);
elseif strcmp(cost,'g')
  g_new=Tnew*a_new;
  J2=gcost(g_matlab,g_new,m,dt);
end
if lap==1                               % compare new and old cost
  if Tkeep(i)<=ns/8+1
    Jtrack(Tkeep(i))=J-J2;
  else
    Jtrack(Tkeep(i)+ns/8)=J-J2;
  end
else
  Jtrack(Tkeep(i))=J-J2;
end
end

if lap==1
```

```
  Jmat=reshape(Jtrack,ns/4+1,segments*2+1); % map delta cost into matrix
else                                        % corresponding to time-frequency
  Jmat=reshape(Jtrack,ns/2+1,segments);     % matrix of bins
end
```

```
%=========================================================================
% ycost.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Calculates cost that minimizes |y-g*u|^2 (convolution) divided by number
% of data points

% INPUT ARGUMENTS   y      :  output
%                   Vinfo2 : new information vector
%                   a_hat  : magnitude coefficients corresponding to basis
%                            functions kept in reconstruction of g
%                   m      : number of points in impulse response g
%                   N      : number of data points keeping from input & output
%                   dt     : time step
% OUTPUT            J      : cost
%
% function J=ycost(y,Vinfo2,a_hat,m,N,dt)
%=========================================================================

function J=ycost(y,Vinfo2,a_hat,m,N,dt)

y2=y(m:N)'*y(m:N);                      % output squared
J=abs(y2-Vinfo2'*a_hat)*dt/(N-m+1);     % cost
```

```
%=========================================================================
% gcost.m
%=========================================================================
% Corinne Ilvedson
% Last Modified 8/4/98
%
% Calculates performance cost that minimizes |g_exact - g_est|^2
%
% INPUT ARGUMENTS   g_matlab :  exact impulse response via impulse.m
%                   g_hat    :  estimate of impulse response
%                   m        :  number of points in impulse response
%                   dt       :  time step
% OUTPUT            J        :  cost
%
% function J=gcost(g_matlab,g_hat,m,dt)
%=========================================================================

function J=gcost(g_matlab,g_hat,m,dt)

integrand=abs(g_matlab*dt-g_hat').^2;
J=(sum(integrand)-integrand(1)/2-integrand(m)/2)*dt;
```

```
%====================================================================
% Gcost_p.m
%====================================================================
% Corinne Ilvedson
% Last Modified 8/11/98
%
% Calculates performance cost which minimizes the difference between
% the exact transfer function and the estimate.  Cost calculated using
% methods that weights points of estimate more heavily than shape of
% curve.
%
% See Section 2.3.1 of Ilvedson MIT MS Thesis, "Transfer Function
% Estimation Using Time-Frequency Analysis"
%
% INPUT ARGUMENTS   m   :  magnitude of estimate
%                   p   :  phase of estimate       (degrees)
%                   f   :  freq vector of estimate (Hz)
%                          (cost function is evaluated over this
%                          frequency span)
%                   num :  numerator of exact transfer function
%                   den :  denominator of exact transfer function
% OUTPUT            J   :  cost integral of |G_exact - G_est|^2 df
%
% function J=Gcost_p(m,p,f,num,den)
%====================================================================

function J=Gcost_p(m,p,f,num,den)

[a_tru,b_tru]=nyquist(num,den,f*2*pi);    % real and imag part of exact tf

a_est=m.*cos(p*pi/180);                   % real part of tf estimate
b_est=m.*sin(p*pi/180);                   % imag part of tf estimate

Gtru=a_tru+i*b_tru;                       % exact tf
Gest=a_est+i*b_est;                       % estimate tf

integrand=abs(Gtru-Gest).^2;              % integrand of cost function

J=int_sum(integrand,f);                   % cost
```

```
%=====================================================================
% Gcost_a.m
%=====================================================================
% Corinne Ilvedson
% Last Modified 8/12/98
%
% Calculates performance cost which minimizes the difference between
% the exact transfer function and the estimate.  Cost calculated using
% methods that weights shape of curve more heavily than the actual points of
% curve.  Obtains shape by linearly interpolating between pts of curve.
%
% See Section 2.3.1 of Ilvedson MIT MS Thesis, "Transfer Function
% Estimation Using Time-Frequency Analysis"
%
% INPUT ARGUMENTS   m   :  magnitude of estimate
%                   p   :  phase of estimate       (degrees)
%                   f   :  freq vector of estimate (Hz)
%                   num :  numerator of exact transfer function
%                   den :  denominator of exact transfer function
%                   f1  :  lower limit of integral
%                   f2  :  upper limit of integral
% OUTPUT            J   :  cost integral of |G_exact - G_est|^2 df
%
% function J=Gcost_a(m,p,f,num,den,f1,f2)
%=====================================================================

function J=Gcost_a(m,p,f,num,den,f1,f2)

w2=linspace(f1*2*pi,f2*2*pi,5000);        % freq pts of curve (rad/s)
f2=w2/(2*pi);                             % (Hz)

[a_tru,b_tru]=nyquist(num,den,w2);        % real and imag part of exact tf

a_est=m.*cos(p*pi/180);                   % real part of tf estimate
b_est=m.*sin(p*pi/180);                   % imag part of tf estimate

a_est=a_est(:);                           % make sure all are column vectors
b_est=b_est(:);
a_tru=a_tru(:);
b_tru=b_tru(:);

a_est2=interp1(f,a_est,f2);               % linearly interpolate to find curve
b_est2=interp1(f,b_est,f2);               %     of tf estimate

a_est2=a_est2(:);                         % make sure all are column vectors
b_est2=b_est2(:);

Gtru=a_tru+i*b_tru;                       % exact tf
Gest=a_est+i*b_est;                       % estimate tf
Gest2=a_est2+i*b_est2;                    % curve of estimate tf

integrand=abs(Gtru-Gest2).^2;             % integrand of cost function

J=int_sum(integrand,f2);                  % cost
```

```
%====================================================================
% int_sum.m
%====================================================================
% Corinne Ilvedson
% Last Modified 8/11/98
%
% Numerical Integration
% INPUT ARGUMENTS   x :  signal to be integrated
%                   t :  t or freq vector over which to be integrated
% OUTPUT            y :  value of integral
%
% function y=int_sum(x,t)
%====================================================================

function y=int_sum(x,t)

l=length(x);

y=0;
for i=1:(l-1)
  x_ave=(x(i)+x(i+1))/2;
  dt=t(i+1)-t(i);
  y=y+x_ave*dt;
end
```

Bibliography

[1] Ljung, L., *System Identification*, Prentice Hall, Inc., 1987.

[2] Crawley, E. F. and S. R. Hall, "The Dynamics of Controlled Structures," Tech. Rep. SERC #10-91-I, MIT Space Engineering Research Center, July 1991.

[3] Paternot, X., *Signal Processing via Wavelet and Identification*, Master's thesis, Swiss Federal Institute of Technology, 1996. written at The Department of Aeronautics and Astronautics, Massachusetts Institute of Technology.

[4] Feron, E., M. Brenner, J. Paduano, and A. Turevskiy, "Time-Frequency Analysis for Transfer Function Estimation and Application to Flutter Clearance," *Journal of Guidance, Control and Dynamics*, Vol. 21, No. 3, 1998, pp. 375–382.

[5] Turevskiy, A., *Flutter Boundary Prediction Using Experimental Data*, Master's thesis, Massachusetts Institute of Technology, 1998.

[6] Kay, S. M., *Modern Spectral Estimation: Theory and Application*, Prentice-Hall, 1988.

[7] Karu, Z. Z., *Signals and Systems Made Ridiculously Simple*, ZiZi Press, Cambridge MA, 1995.

[8] Ljung, L., *System Identification Toolbox For Use with MATLAB*, The MathWorks, Inc., Natick, Massachusetts, 1993.

[9] Krauss, T. P., L. Shure, and J. N. Little, *Signal Processing Toolbox For Use with MATLAB*, The MathWorks, Inc., Natick, Massachusetts, 1993.

[10] Wirsching, P. H., T. L. Paez, and K. Ortiz, *Random Vibrations: Theory and Practice*, John Wiley and Sons, 1995.