178
49

# EXPERIMENTAL DETERMINATION OF THE IMPULSE RESPONSE FUNCTION FOR ELASTIC VIBRATING SYSTEMS

by

Charles Mark Archibald

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for

the degree of

**MASTER OF SCIENCE**

in

Mechanical Engineering

APPROVED:

Alfred L. Wicks

A. L. Wicks, Chairman

L. D. Mitchell

L. D. Mitchell

F J Pierce

F. J. Pierce

EXPERIMENTAL DETERMINATION OF THE IMPULSE RESPONSE FUNCTION FOR ELASTIC VIBRATING SYSTEMS

by

CHARLES MARK ARCHIBALD

Committee Chairman:Alfred L. Wicks

Mechanical Engineering

(ABSTRACT)

An experimental method for determination and analysis of the impulse response function of linear, elastic, vibrating systems is developed. A deconvolution method is developed for estimation of the impulse response function. The estimator is shown to be unbiased in the presence of measurement noise. Modal parameters are extracted from impulse response estimates using a modification of the Pisarenko harmonic decomposition method. The advantages of a time-domain approach over traditional Fourier analysis procedures, including avoidance of leakage and enhanced statistical significance, are described. Several tests used to determine the performance of the impulse response estimator are described, and the results of these tests, are presented. It is shown that the method can provide accurate estimates of modal parameters even for short data sets or high noise levels.

## ACKNOWLEDGEMENTS

I would like to thank my major advisor, Dr. A.L. Wicks, for his generous support and encouragement. Dr. Wicks introduced me to modal analysis and signal processing. His tutorship during my research has been a tremendous help. I am very grateful for the guidance he provided while I was preparing this thesis.

In addition, I would like to thank Drs. L.D. Mitchell and F.J. Pierce for their recommendations and comments regarding this thesis. There advice was much appreciated.

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

**NOMENCLATURE**

| | |
|---|---|
| $f(t)$ | Continuous forcing function of time |
| $y(t)$ | Continuous response function of time |
| $Y(\omega)$ | Fourier transform of response function |
| $F(\omega)$ | Fourier transform of forcing function |
| $H(\omega)$ | Frequency response function |
| $h(t)$ | Impulse response function |
| $X$ | Discrete function of frequency |
| $x_k$ | Discrete function |
| $\omega$ | Circular frequency |
| $N$ | Data record sample length |
| $j$ | $\sqrt{-1}$ |
| $H_1$, $H_2$, $H^c$ | Frequency response function estimators |
| $\Delta$ | Sample period |
| $e_k$ | $k^{th}$ value of random noise series |
| $a_i$ | Autoregressive coefficient |
| $b_i$ | Moving average coefficient |
| $E[\ ]$ | Expected value operator |
| $[\phi]$ | Autoregressive system matrix |
| $[W]$ | Instrumental variable matrix |
| $[\ ]^T$ | Matrix transpose |
| $A_i$, $B_i$, $G_i$ | Trigonometric function model coefficients |
| $\alpha_i$ | Real part of exponent for complex exponential |
| $\Omega_i$ | Imaginary part of complex exponent |

| | |
|---|---|
| $v_i$ | Transfer function model coefficient |
| $N_k$ | Random noise term for transfer function model |
| [A] | System matrix for real eigenvalue method |
| [B] | Input matrix for real eigenvalue method |
| $m_f$ | Measured value of Input |
| $e_f$ | Input noise |
| $m_y$ | Measured value of output |
| $e_y$ | Output noise |
| f | True input |
| y | True output |
| s | Source series |
| $y(t)$, $y(t)$ | First and second derivatives of y(t) with respect to time |
| [M],[C],[K] | Mass, damping, and stiffness matrices |
| $\delta(t)$ | Delta function (unit impulse) |
| n | Number of modes of vibration |
| $c_i$ | Complex amplitude coefficient |
| $\zeta$ | Damping ratio |
| $\omega_n$ | Natural frequency |
| $\omega_d$ | Damped natura frequency |
| $\phi$ | Phase angle |
| $Q_k$ | Initial condition response component of $y_k$ |
| [S] | Source matrix |
| $RsQ_k$, $Rsy_k$, $Rsf_k$ | Crosscorrelations at lag k |
| $\hat{h}$ | Estimated value of impulse response |

| | |
|---|---|
| Z[ ] | Z transform operator |
| $X_Z$, $Y_Z$ | Z transforms of $x_k$, $y_k$ |
| z | Complex variable |

# CHAPTER 1: INTRODUCTION AND BACKGROUND

Experimental determination of the dynamic properties of vibrating systems is critically important in an age of increasing technological sophistication. The advent of powerful, inexpensive digital computers has given rise to rapid advances in both analytical and experimental analysis methods. On the analytic side, the finite element method has come to the forefront for use in dynamic analysis. On the experimental side, experimental modal analysis methods provide accurate vibration data for real systems and verification of finite element method results. This information is vital from several standpoints. The trend in equipment design is toward increased precision and performance. Often vibration characteristics are paramount to prediction of performance. Experimental data can provide the design engineer with data critical to the successful implementation of such designs. However, measurement data from dynamic systems must undergo considerable processing in order to extract useful information. Many special cases exist in which current signal processing algorithms either fail or perform poorly. Improved algorithms that fill these problem areas for data processing are thus extremely important to the experimentalist.

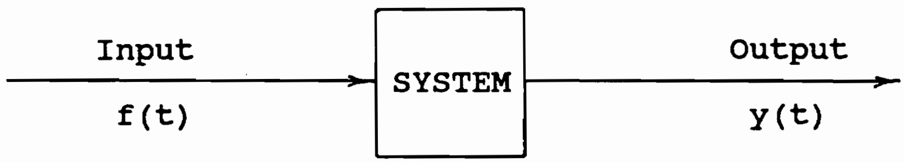


Figure 1

Input-Output System Model

A dynamic system can be modeled by an input-output relationship. Figure 1 depicts such a model. If an adequate model of the system exists, then the output of the system can be predicted for any given input. This implies that important dynamic properties of the system, such as resonance frequencies, are known. Thus, the problem confronting the experimentalist is to construct an appropriate model using the available data. In some cases, both input and output data are available. In other cases a structure is excited by an input force, which is then removed. Only the subsequent output, or free response, of the system is measured. In virtually all cases, the continuous signals are sampled to obtain discrete data. The resulting finite length data set, discretely sampled in time, is called a time series. The free response methods, in which only the output is measured, are called single series analysis procedures. Multiple time series methods can involve both the measured input and output series. Once data is available, an analysis method is selected. The traditional approach is to use Fourier transform methods. This involves transformation of the data measured in time to a new, related data set based in frequency. The subsequent analysis is said to be performed in the frequency-domain. Conversely, if the data is analyzed without any such transformation, the procedure is said to be performed in the time-domain. Often time-domain approaches, as a final step, yield frequency information, for example, the natural frequencies of a structure. This provides for easy comparison of results from frequency-domain methods with those obtained from time-domain methods. A modeling approach must be selected. Models can be either discrete or continuous. In general, since continuous data is usually sampled to give discrete data, discrete models are employed. A final modeling concern is measurement noise. Measurement noise consists of any component of the measured data that is not correlated with the true signal being measured. Noise can be modeled in several different ways. If an inappropriate noise model is employed, the quality of the resulting system model will be adversely impacted. Thus, care must be exercised in all modeling procedures. A successful model can provide a mathematical function relating the input signal to the output signal. If the system under investigation is linear, causal and shift invariant this function contains information sufficient to describe the system. In the frequency-domain, this function is called the frequency response function. In the time-domain, it is termed the impulse response function. Both functions contain the same information about the system, but in different forms. Modeling of a dynamic system thus entails decisions on single versus multiple time series approaches, discrete or continuous models, time-domain or frequency-domain procedures, and the appropriate assumptions pertaining to noise.

A new algorithm for estimating the impulse response function of a linear dynamic system is presented. Background information and a review of the literature are provided in the remainder of this Chapter. A number of time series analysis procedures are reviewed. The theoretical formulations for the algorithm are presented in Chapter 2. Chapter 3 presents an method for the analysis of the estimated impulse response function. This analysis will yield estimates of modal parameters, such as natural frequency, damping ratio, amplitude and phase. Chapter 4 includes description and results of experimental verification of the algorithm.

## FREQUENCY-DOMAIN METHODS

Frequency-domain methods are the traditional tools for dynamic systems analysis. The relationship between the input series and the output series in a dynamic system is multiplicative in the frequency-domain, whereas it is convoluted in the time-domain. The system represented in Fig. 1 can be described in the frequency-domain by the relation

$$Y(\omega) = H(\omega)\ F(\omega)$$

where $Y(\omega)$ and $F(\omega)$ are the Fourier transforms of the output and input series respectively, and $H(\omega)$ is the frequency response function. Thus the frequency response function is easily obtained from the transforms of the input and output series. The same relationship in the time-domain is given by the convolution integral

$$y(t) = \int_0^{\infty} h(\tau)\ f(t-\tau)\ d\tau \qquad 1.1$$

Convolution in the time-domain corresponds to multiplication in the frequency-domain. Because of the simplicity of the multiplicative relationship, frequency domain input-output analysis approaches are readily implemented.

The vast majority of frequency-domain procedures are performed using fast Fourier transform methods. The development of the fast Fourier transform algorithm by Cooley and Tukey in 1965 [1] and the development of high-speed digital computers have resulted in widespread use of these methods. Prior to this time, the fitting of data to a Fourier series was computationally so intense as to be essentially impractical. The fast Fourier transform is an algorithm to efficiently fit a Fourier series to a finite, discrete data set.

The discrete Fourier transform relates a discrete function of time, $x_k$, to a discrete function of frequency, $X_\omega$. The transformation is given by

$$X_\omega = \sum_{k=0}^{N-1} x_k \, e^{-j2\pi k\omega/N} \qquad 1.2$$

The transformation maps the time-domain series to a frequency-domain series. Any noise present in the time series is transformed along with the true signal. Thus, elimination of noise in the frequency domain requires that numerous data sets be transformed and subsequently averaged. Prior to the development of the fast Fourier transform algorithms the transformation of a series $x_k$ of length N required on the order of $N^2$ complex operations. The advent of the FFT, or fast fourier transform, reduced this computational overhead to only $N\log_2 N$ complex operations. For a data set of 1024 points, this results in a reduction of operations by a factor of more than 100.

The fast Fourier transform method is computationally efficient and widely implemented in hardware and software. However, it is subject to several limitations. The data set length is restricted to powers of two in most implementations. This restriction can become significant if the available data is limited, although for many applications it results in no limitations. Only a discrete set of frequencies are calculated. These frequencies are determined by the sample rate (for sampled data) and the size of the data set. If a signal exists at a frequency other than these discrete frequency values, it will be leaked into other nearby frequencies, thus smearing the signal over a frequency band. This problem can be minimized by using appropriate window functions. However, these can often lead to biased damping estimates [2]. A final restriction of fast Fourier transform techniques is elimination of noise. As mentioned above, to minimize random errors due to measurement noise, the frequency-domain data must be averaged. This requires a large amount of data. Again, if the data is readily obtained, this may impose no limitations. Conversely, if only a limited data set is available, one of the time domain methods might be a better analysis tool.

Frequency-domain analysis methods for dynamic systems involve calculation of the frequency response function (FRF) of the system. The frequency response function and the impulse response function of a system comprise a Fourier transform pair. The dynamic system under investigation can be represented by either function. The estimation of the frequency response function requires that the input and output data series be transformed, usually by a fast Fourier transform algorithm, into the frequency-domain. The frequency response function is then estimated from the resulting autospectra. Several estimators exit. The traditional FRF estimate, designated $H_1$, is computed by

$$H_1 = \frac{G_{xy}}{G_{xx}} \qquad 1.3$$

where $G_{xy}$ and $G_{xx}$ are the crossspectrum and input autospectrum, respectively. It is known that this estimate is biased if noise is present on the input series. An alternative estimate, known as $H_2$, is biased if output noise is present [3]. An unbiased frequency response function estimator is available, however [3]. If a third series is available, designated the source series, $s_k$, that is uncorrelated with the input and output measurement noise, then a philosophy similar to that employed with the instrumental variables approach can be use. The resulting FRF estimator, $H^c$, is given by

$$H^c = \frac{G_{sy}}{G_{sx}} \qquad 1.4$$

It has been shown that the $H^c$ estimator is unbiased in the presence of both input and output noise. In modal analysis practice, the source signal can be obtained directly from a signal generator used to drive a shaker. This methodology is effective and readily implemented.

## TIME-DOMAIN METHODS

A number of time-domain methods are available for analysis of dynamic systems. A diversity of approaches are used. Single time-series analysis procedures analyze systems for which only the output data is measured. These methods are also applied to the analysis of the impulse response function of a system which may be obtained by inverse Fourier transform of the frequency response function. Multiple time-series analysis procedures use measured input and output data to fit a system model.

## RELEVANT SINGLE SERIES AND FREE RESPONSE METHODS

Single series analysis methods are fundamental prerequisites for derivation and implementation of multiple time series analysis. An analysis procedure that involves only one time series is termed single series analysis, whereas multiple series analysis involves the simultaneous analysis of several time series. Several groups of single series methods are presented here. The first group of methods, known as difference equation models, describe a time series in terms of its previous values. Pisarenko harmonic decomposition is one example of a difference equation model. Autoregressive, moving average, and autoregressive-moving average models are additional examples. A modified form of Pisarenko harmonic decomposition will be used in the implementation of the impulse response function analysis presented in Chapters 3.

The last group of single series analysis methods involves modeling the data with exponentially weighted trigonometric functions. Essentially, these methods use a non-linear least squares method to fit a set of functions to the data. A technique known as Smith least squares [4] will be discussed.

## AUTOREGRESSIVE MODELS

Historically, the autoregressive analysis dates from 1927, when G. Udny Yule devised the method [5]. Central to his theory was the concept that noise can be modeled as a series of random shocks, each perturbing the corresponding value of the true series by an unknown amount. This idea of noise is contrary to the assumption that a process is purely deterministic, with an additive random component due to measurement inaccuracies. Yule presented a hypothetical pendulum oscillating as a result of some initial condition. It can be shown that the motion of a simple pendulum, in the absence of noise, can be completely described for all time by taking two measurements of the displacement of the pendulum. However, if, for example, some boys got into the room and pelted the pendulum from each side by peas, then the resulting motion is completely different. In this case, the random shocks caused by the peas contribute to the displacement of the pendulum. The motion at any particular time is thus determined by the past displacements and the present shock imparted by the peas. The underlying assumption is that measurements of the past position of the pendulum are known exactly, but the current position is affected to an unknown extent by the current force of the pea. It will be shown that this philosophy of noise is markedly different from the concept of additive noise, in which all measurements are assumed to consist of a true, deterministic component and an additive random component. The additive noise model will be described in the development of Pisarenko harmonic decomposition.

The motion of the simple pendulum can be described by a second order autoregressive equation. Assuming sampled data, the following notation is used to denote a time series

$$y_k = y(k\triangle)$$

where $\triangle$ is the sample interval, $y_k$ denotes the $k^{th}$ element in a discrete time series, and $y(k\triangle)$ is the value of a continuous time signal at time $k\triangle$. With this notation, the autoregressive equation for a simple pendulum is given by

$$y_k = a_1 y_{k-1} + a_2 y_{k-2} + e_k \qquad 1.5$$

Here $e_k$ is the $k^{th}$ random shock, and $a_1$ and $a_2$ are the autoregressive coefficients.

Since the days of Yule, the AR model has been applied to many different types of analysis in many disciplines. A more general form of the autoregressive model is given by

$$y_k = \sum_{i=1}^{p} a_i \, y_{k-i} + e_k \qquad 1.6$$

Thus, the autoregressive model describes a system in terms of p past values and a current random shock.

In contrast to this idea, the moving average (MA) model describes a system in terms of past random shocks alone. Thus, the output of a moving average model depends solely on the history of random shocks given to the system. The moving average model has the form

$$y_k = \sum_{i=1}^{m} b_i \, e_{k-i} \qquad 1.7$$

The autoregressive and moving average processes can be characterized by their correlations. Specifically, for an autoregressive process, the following relations hold

$$E[\, y_k \, y_{k+i} \,] \neq 0$$
$$E[\, e_k \, y_{k+i} \,] = 0 \qquad i \neq 0$$

Here E[ ] is the expectation operator. The above expressions simply state that an autoregressive process is correlated (to some extent) with all of its past values, that is, it is autocorrelated. Also, an autoregressive process is not correlated with the random shocks. Similar relations can be stated for a moving average process.

$$E[\ y_k\ y_{k+i}\ ] = 0 \qquad i > m$$

$$E[\ e_k\ y_{k+i}\ ] \neq 0 \qquad 0 < i < m$$

That is, a moving average process is not autocorrelated, but is correlated, over a certain range, with the random shocks.

The autoregressive and moving average models can be combined to form an autoregressive-moving-average model (ARMA). The general ARMA model is of the form

$$\sum_{i=0}^{p} a_i\ y_{k-i} = \sum_{i=0}^{m} b_i\ e_{k-i} \qquad 1.8$$

$$a_0 = b_0 = 0$$

The lefthand sum is known as the autoregressive term whereas the righthand sum is the moving average term. The autoregressive term describes the dependence of the output series, $y_k$ on its own previous values. The moving average term describes the dependence of the output series on previous and current values of the random shocks, $e_k$. A model in which only one term of the moving average sum is present in known as an autoregressive model. Conversely, a model in which only the first term of the lefthand sum is present is called a moving average model. In either case, the series $y_k$ is modeled as the output from a filter which has a white noise input, $e_k$. The $a_i$ and $b_i$ coefficients contain information regarding the nature of the output series. Thus, for a complex exponential series these coefficients contain frequency and damping information.

General ARMA models are described by Box and Jenkens [6]. Although the Box and Jenkens approach is not particularly conducive to analysis of vibrating systems, a good background in ARMA theory and methodology is provided. ARMA models as applied to dynamical analysis are described by a number of investigators, including Kay and Marple [7], [8], Pandit [9], and Patton [10]. These models often approximate the true system with fewer parameters than the autoregressive or moving average models alone. A disadvantage is that the solution of the general ARMA model, Eq. 1.8, is non-linear, and is not computationally efficient. Thus the reduced forms of the general ARMA model, as given in Eqs. 1.6 and 1.7, are often utilized. The autoregressive model is a special case in which only one non-zero moving average term is present. It is a much more computationally attractive model than Eq. 1.8. However, more terms may be required to adequately model a particular system. In theory, any time series can be modeled to an arbitrary degree of precision by an autoregressive model of sufficiently high order. However, the required order may be very large or infinite. In practice, parsimonious models, that is, models that adequately represent the data with only a few parameters, are desireable.

Several solution strategies are employed to solve the autoregressive model. The simplest involves a least squares solution of Eq. 1.6 for the autoregressive parameters. This is accomplished by using standard statistical regression analysis procedures on the time series. It has been shown that the least squares solution method leads to biased estimates of the autoregressive parameters if additive noise is present [11],[12]. The residuals, or the series of random shocks, obtained by autoregressive methods when additive noise is present, have been shown to be correlated [12]. The correlated residuals, due to inappropriate noise modeling, lead to the biased estimates of the autoregressive parameters. Overspecification of the autoregressive model order, that is, increasing the number of parameters in the model, reduces this bias, but introduces non-existent frequency content as well.

Several alternative formulations have been proposed to reduce or eliminate this bias. These include correlation-fit, double least squares, total least squares, and instrumental matrix with delayed observations methods. These method are compared, along with the least squares approach, in [11]. The correlation-fit method fits an autoregressive model in a similar manner as the least squares method, but by using additional correlations and omitting the first few lags. This method can eliminate the bias error inherent in autoregressive least squares, but difficulties exist in implementation. The double least squares method is a variant on the Ibrahim-time-domain method in which the time lag is negative. The total least squares method is similar to Pisarenko harmonic decomposition, discussed below. The instrumental matrix with delayed observations method is essentially a variation of the instrumental variable method in which the instrumental matrix is obtained from time- shifted values of the data.

The difficulties with correlated residuals can be avoided with the instrumental variable method [12]. This method attempts to eliminate the bias of the autoregressive least squares method by use of an additional series that is not correlated with the measurement noise. The method is effective if such a series can be found. Equation 1.6 can be written in matrix form as

$$\begin{Bmatrix} y_p \\ y_{p+1} \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{Bmatrix} = \begin{bmatrix} y_{p-1} & y_{p-2} & \cdots & y_0 \\ y_p & y_{p-1} & \cdots & y_1 \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ y_{N-1} & y_{N-2} & \cdots & y_{N-p} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_p \end{Bmatrix} + \begin{Bmatrix} e_p \\ e_{p+1} \\ \cdot \\ \cdot \\ \cdot \\ e_N \end{Bmatrix}$$

or

$$\{y\} = [\phi]\{a\} + \{e\} \qquad 1.9$$

The least squares solution to Eq. 1.9 involves premultiplication by $[\phi]^T$. The method assumes that the product $[\phi]^T\{e\}$ has an expected value of zero. In many cases, most notably when the actual noise is additive, this assumption is not true, and results in bias errors in the estimates of $\{a\}$. This bias can be eliminated if a matrix, designated the instrumental matrix $[W]$, can be found such that the following conditions are met

1. $E[\ [W]^T\{e\}\ ] = 0$
2. $[W]^T[\phi]$ nonsingular.

That is, the instrumental variable W is not correlated with the residuals, e, and the product of the instrumental matrix transposed with the system matrix is a nonsingular matrix. An unbiased estimate of the autoregressive coefficients is possible. This estimate is obtained by premultiplication of Eq. 1.9 by the transpose of the instrumental matrix, $[W]^T$. The unbiased estimate of $\{a\}$ is then given by the solution of

$$[W]^T\{y\} = [W]^T[\phi]\{a\} \qquad 1.10$$

Where the term $[W]^T\{e\}$ has been replaced with its expected value of zero.

Additional time-domain methods in wide use include the complex-exponential or Prony, the polyreference, and the Ibrahim-time-domain methods. It has been shown that these methods are theoretically similar [13],[14],[15]. Each utilize data from either the free response or the impulse response of a vibrating system fit a model. Generally a least squares method is employed for model fitting. The complex-exponential method uses a single input, single output model, and is, thus, termed a local approach. The polyreference method is a generalization of the complex-exponential method to multiple outputs, and is hence termed a global approach. Data from all response locations is processed simultaneously, as opposed to the complex exponential method. The Ibrahim-time-domain method, also a global approach, utilizes two sets of response data temporally separated by a time lag. The two sets of response measurements are utilized to construct a transformation matrix. These methods are described in [16],[17] and [18].

Pisarenko harmonic decomposition (PHD) is an alternative difference equation model. The descriptive difference equation is similar to Eq. 1-8, except that the autoregressive and moving average coefficients are identical. Thus, the assumed model is

$$\sum_{i=0}^{p} a_i \, y_{k-i} = \sum_{i=0}^{p} a_i \, e_{k-i} \qquad \text{1-11}$$

$$a_0 = 0$$

Although some investigators have described Pisarenko harmonic decomposition as a special case ARMA model due to the similarities between Eq. 1-11 and Eq. 1-8 [7], the method differs in an important assumption regarding the way in which noise is treated. Unlike ARMA models, in which the time series is modeled as the output of a filter that is driven by noise, Pisarenko harmonic decomposition assumes that the noise is independent of the true series. That is, the time series is composed of the sum of two independent series: 1) a deterministic series composed of sinusoids of arbitrary frequency, and 2) a random noise series. The distinction is important. The ARMA models describe a process in which the noise affects the signal being measured, that is, the output at some time t is dependent on both its previous values and a noise term. Pisarenko harmonic decomposition, on the other hand, describes a process in which the deterministic signal is not affected by the noise. The deterministic output at some time t is dependent solely on its previous values, although the measured output consists of this value plus the current noise value. This assumption is more in line with actual measured signals. In the analysis of dynamical systems, the assumption is made that the underlying process is deterministic. However, such a process cannot be measured exactly. The process of measurement introduces errors, known as measurement noise. This measurement noise in no way affects the true signal. Therefore Pisarenko harmonic decomposition is a more applicable technique for dynamic analysis than the various ARMA methods. Figure 2 depicts the ARMA and PHD assumptions regarding noise.

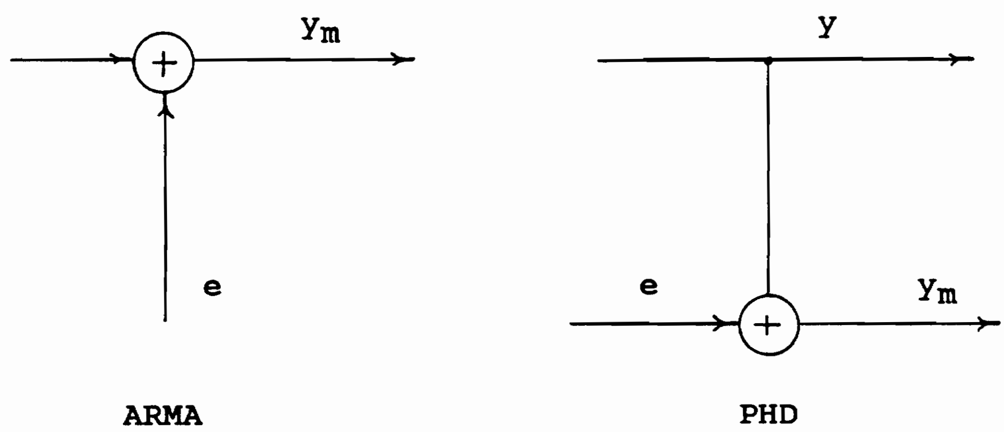


Figure 2

Comparison of ARMA and PHD noise assumptions

A thorough derivation of Pisarenko harmonic decomposition is given in [7] and [10]. Essentially, the formulation entails forming an eigenproblem from the difference equation. The minimum eigenvalue then corresponds to the variance of the additive noise, and the corresponding eigenvector contains the coefficients of the difference equation describing the process. The roots of this equation contain frequency and damping information. Amplitude and phase data can then be found from the original data. The method is described in more detail in Chapter 3.

A performance analysis of Pisarenko harmonic decomposition in comparison to several ARMA type approaches is presented in [10]. The results of this comparison indicated that Pisarenko harmonic decomposition performed better than the

ARMA models. Unlike the ARMA approaches, no overspecification of model order is required for accurate results.

**EXPONENTIALLY WEIGHTED TRIGONOMETRIC FUNCTION MODELS**

The class of models known as exponentially weighted trigonometric function models employ a different methodology. When the deterministic component of a signal is assumed to be comprised of exponentially weighted trigonometric functions, as is often the case in dynamical systems, an analytic model can be directly fitted to the data. W. R. Smith [4] presents such a method. This method is also known as Smith least squares. The following function is assumed

$$y_k = \sum_{i=1}^{p/2} e^{-\alpha_i k} \left[ A_i \sin \Omega_i k + B_i \cos \Omega_i k \right] \qquad 1.12$$

The parameters $A_i$, $B_i$, $\alpha_i$ and $\Omega_i$ are then fitted in a least squares sense to the data. Since the parameters are non linear in the function, an iterative approach must be used. Initial estimates, often generated from fast Fourier transforms, are provided for $\alpha_i$ and $\Omega_i$ . The parameters $A_i$ and $B_i$ are then calculated. The process is iterated until some convergence criteria is met.

The Smith least-squares approach provides unbiased estimates of parameters. A comparison of this method with autoregressive least squares and correlation fit is found in [14]. This comparison indicated that the Smith least squares procedure performed better than the other methods at high noise levels. However, convergence depended on the quality of the initial estimates. The number of modes need not be overestimated as in autoregressive least squares approaches. However, the computational intensity of the iterative approach can lead to excessive processing time. A full description of the method is given in [4]. An evaluation of the method is presented in [19].

**MULTIPLE TIME SERIES/FORCED RESPONSE METHODS**

The analysis of dynamical systems is often performed with multiple time-series analysis. A system is excited by some measured forcing function and the resulting response of the system is also measured. The characteristics of the system can be inferred by the relationship between the forcing function and the response function. As with single series analysis, a number of methods exist to process the data. Input-output models can be constructed using difference equations. Two types of difference equation models are considered. Autoregressive moving average models have been extended to multiple time series analysis. This is easily accomplished by utilizing the filter nature of the ARMA model. In addition, transfer function models are discussed.

Time domain deconvolution methods provide an alternative analysis technique. Measured inputs and outputs can be analyzed to obtain the system impulse response function (IRF), which is sufficient to describe the system. Modal parameters can then be extracted from the IRF.

Box and Jenkens [6] describe a general transfer function model to relate multiple time series. The relationship between the input series $x_k$ and the output series $y_k$ is given by

$$y_k = \sum_{i=1}^{\infty} v_i \, x_{k-1} + N_k \qquad 1.13$$

Where $N_k$ is a random noise term. As with the single series ARMA methods, the Box and Jenkens approach is not directly applicable to analysis of dynamic systems. Autoregressive-moving average models have been employed more often for multiple time series analysis. The data dependent systems approach is one such method [9],[20]. In this approach, Eq. 1.8 is used as a model for the system. The procedure first involves analysis of the free response of the system in order to obtain a suitable model. The forced response is then used to extract parameters of interest.

A different approach to multiple series analysis involves performing an eigen analysis on a system matrix constructed from input and output data [10], [20]. The real eigenvalue method [21] is one such method. The motion of a linear elastic dynamic system can be described in state vector form by the following matrix equation

$$\{y\} = [A]\{y\} + [B]\{f\} \qquad 1.14$$

where the matrices [A] and [B] contain the mass, stiffness and damping constants of the system. The real eigenvalue method presents a method that utilizes the state vectors {y}, and the force vector {f} to obtain an estimate of the matrix [A]. The eigenvalues of a submatrix of [A] correspond to the natural frequencies of the system. To obtain accurate damping estimates, however, the method resorts to use of a digital filter to remove all but one mode from the data. Then a single degree of freedom analysis is performed on each mode separately.

A final method for analysis of systems where input and output data is available is numerical deconvolution [22]. The output of a linear dynamic system can be determined by the convolution integral, given in Eq. 1.1. Note that this method assumes that 1) the system is linear and causal, and 2) that all initial conditions are zero. Deconvolution methods attempt to use measured input and output data to numerically solve Eq. 1.1 for the impulse response function. In the past, implementation of the method has proved difficult. Measurement noise and the presence of non-zero initial conditions have required trial and error or iterative solution strategies.

This thesis presents a new deconvolution method which is not subject to these limitations. The method is shown to be relatively insensitive to measurement noise and requires no iteration. The theoretical formulation for the method is provided in Chapter 2. A proof is provided that the method produces an unbiased estimate, regardless of input or output noise. Chapter 3 describes the analysis method used to extract modal parameters from the estimated impulse response. A modification of Pisarenko harmonic decomposition is used. Chapter 4 presents experimental results of tests of the impulse response function. The results of two experiments are presented. The first experiment involves testing the impulse response estimator with respect to five pertinent parameters. The data is obtained from a single degree of freedom electrical circuit. A second experiment investigates the performance of the system at estimating the natural frequencies of a vibrating beam.

# CHAPTER 2: THEORETICAL DEVELOPMENT

The theoretical development of an unbiased impulse response function estimator of a dynamical system is presented. The characterization of a system involves two independent steps. The first step, described in this chapter, is the estimation of the impulse response function. The assumptions made are stated and described. The impulse response function is defined and derived for a mechanical vibrating system. Convolution is defined and described. These concepts are prerequisites for understanding the impulse response function estimator. The estimation procedure itself is then developed and described. The unbiased nature of the estimator is proved. In addition, some pertinent theorems on Z-transforms are reviewed. Theory and methodology for analysis of the impulse response function, required in order to obtain the modal parameters of a system, is a separate, independent procedure and will be presented in Chapter 3.

## ASSUMPTIONS

Prior to development of theory, certain assumptions must be made with regard to the system under investigation. These assumptions are:

1. The system under investigation is linear, shift invariant and underdamped.

2. The system is either discrete or can be adequately represented by discrete sampling of a continuous process.

3. The system under investigation is a single input, single output system.

4. The input and output signals are composed of a deterministic component with additive measurement noise.

5. The measurement noise processes are assumed to be independent identically distributed random processes with zero mean.

6. The input and output noise processes are assumed to be uncorrelated with 1) the deterministic component of the series, and 2) each other.

7. An instrumental, or source, series is available.

Some of the above assumptions require additional comments or amplification. The restriction imposed by 3. is made for simplification. In general, a suitable expansion of the theory could result in valid results for multiple-input multiple-output systems.

Designating the measured input and output series by ${}^{m}f$ and ${}^{m}y$ respectively, assumption 4. can be summarized by

$$
\begin{aligned}
{}^{m}f &= f + e_f \\
{}^{m}y &= y + e_y
\end{aligned}
\qquad 2.1
$$

Where f and y represent the true, deterministic component, and $e_f$ and $e_y$ represent the random component of the input and output series, respectively. Likewise, assumption 6. can be summarized by

$$
\begin{aligned}
E[\, e_y\, y \,] &= 0 \\
E[\, e_f\, f \,] &= 0 \\
E[\, e_y\, f \,] &= 0 \\
E[\, e_f\, y \,] &= 0 \\
E[\, e_y\, e_f \,] &= 0
\end{aligned}
$$

where E[ ] is the expectation operator. Figure 3 provides a schematic diagram of the assumed dynamic system with additive measurement noise. The source signal referred to in assumption 7. will be defined and described in a later section.

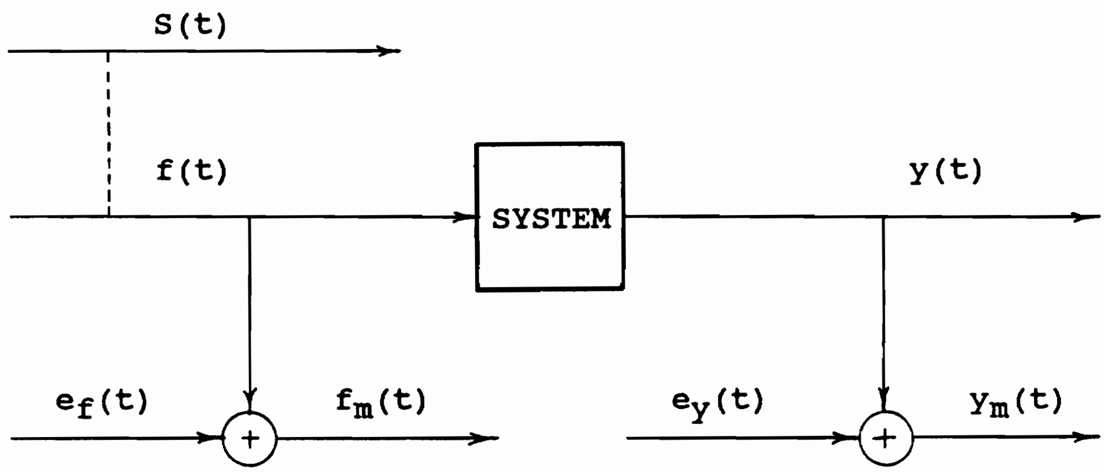


FIGURE 3

Dynamic System With Noise Model

**IMPULSE RESPONSE FUNCTION**

The impulse response function of a dynamical system is defined to be the response of the system, initially at rest, to a unit impulse force. More generally, for a multiple degree of freedom system, the impulse response function between the $i^{th}$ input and the $j^{th}$ output is the response at location j to a unit impulse at location i, with all other inputs and all initial conditions zero.

A linear, time-shift-invariant dynamical system can be completely characterized by its impulse response function. That is, if the impulse response function is known, the behavior of the system can be predicted for any particular forcing function. The impulse response and the frequency response of a system are related via Fourier transforms. The two functions form a Fourier transform pair. Each can characterize the system under investigation. The frequency response function is often utilized because of advantages in interpretation of a graph of the function and, in the past, had computational advantages. However, in many applications, particularly where data is limited, the experimental determination of the impulse response is preferable. These are the applications for which the impulse response function estimator was developed.

The impulse response function can have many mathematical forms, depending on the type of system involved. For mechanical, elastic, vibrating systems, the following matrix differential equation is often utilized.

$$[M]\{\ddot{y}(t)\} + [C]\{\dot{y}(t)\} + [K]\{y(t)\} = f(t) \qquad 2.2$$

Equation 2.2 describes a lumped mass, continuous time, multiple degree of freedom forced vibrational system. Here the vectors $\{\dot{y}(t)\}$ and $\{\ddot{y}(t)\}$ represent the first and second derivative of $\{y(t)\}$ with respect to time and [M], [C], and [K] represent the mass, damping, and stiffness matrices respectively. To illustrate the relevance of the impulse response function to the mathematical model of a system, the impulse response of the system described by Eq. 2.2 will be derived.

The impulse response function for the system described by Eq. 2.2 can be obtained for the $i^{th}$ forcing location and the $j^{th}$ response location by setting $f_i = \delta(t)$, the delta function or unit impulse, and $f_k = 0$, $k \neq i$. In addition let all initial conditions, $y(0)$ and $\dot{y}(0)$ be zero. With these conditions, the solution of Eq. 2.2 will yield the impulse response function between any response location j and the forcing location i. Note that for a system of equations such as Eq. 2.2, a family of impulse response functions exist. The pertinent forcing and response location must be specified to avoid ambiguity.

To facilitate the solution of the system of equations, the unit impulse at location i can be replaced by an appropriate initial velocity at that location. This can be seen from the impulse-momentum principle. Note that the unit impulse, $\delta(t)$ is defined such that

$$\delta(t) = \lim_{\Delta \to 0} \int_{t}^{t+\Delta} F(t)\, dt = 1$$

from the impulse momentum equation, this is equal to the change in momentum of the system. Thus

$$\delta(t) = 1 = m\dot{y}(0) - m\dot{y}(0^-) = m\dot{y}(0)$$

Therefore, the unit impulse can be replaced by the initial condition $\dot{y}_i(0) = 1/m_i$, and Eq. 2.2 is reduced to a homogenous matrix equation with simple initial conditions, that is

$$[M]\{\ddot{y}(t)\} + [C]\{\dot{y}(t)\} + [K]\{y(t)\} = 0$$

$$\begin{aligned} \{y(0)\} &= \{0\} && \\ \dot{y}_i(0) &= 0 && i \neq j \\ \dot{y}_i(0) &= 1/m_i && i = j \end{aligned} \qquad 2.3$$

The solution of this equation is obtained by application of matrix differential equation theory. The solution obtained represents a set of impulse response functions relating the $i^{th}$ forcing location to each response location. The particular impulse response function of interest, at the $j^{th}$ response location, is obtained by selecting the $j^{th}$ row of the solution to Eq. 2.3. In compliance with assumption 3. only single-input, single-output systems will be discussed henceforth. Thus, disregarding all references to forcing or response locations, the solution can be shown to be

$$y(t) = \sum_{l=1}^{n} \left[ c_l \, e^{(-\zeta_l \omega_{nl} + j\omega_{dl})t} + c_l^* \, e^{(-\zeta_l \omega_{nl} - j\omega_{dl})t} \right] \qquad 2.4$$

Where n is the number of modes in the system, the $c_i$'s are complex amplitudes, and $j = \sqrt{-1}$ . Note that Eq. 2.4 can be equivalently written as

$$h(t) = \sum_{i=1}^{n} e^{-\zeta_i \omega_{ni} t} \left[ G_i \cos \omega_{di} t - B_i \sin \omega_{di} t \right] \qquad 2.5a$$

or

$$h(t) = \sum_{i=1}^{n} A_i \, e^{-\zeta_i \omega_{ni} t} \sin \left( \omega_{di} t + \phi_i \right) \qquad 2.5b$$

Here the impulse response is designated by h(t). These are the forms of the impulse response function which will later be used to analyze the impulse response function estimate.

## CONVOLUTION

The impulse response function, for example Eq. 2.5b, relates a particular input to a particular output. If the actual forcing function is composed of a unit impulse, and the system is initially at rest with all initial conditions zero, then the impulse response function describes the actual response of the system. However, for a general forcing function, evaluation of the system response is more complicated.

Consider the system described by Eq. 2.2. In all cases, assume that the system is initially at rest. If the forcing function is comprised of a unit impulse at time t =0, then the response is given by Eq. 2.5. If the forcing function is comprised of a unit impulse at time $t = \tau$ , however, the response becomes

$$y(t) = h(t-\tau)$$

or

$$y(t) = \sum_{i=1}^{n} A_i \, e^{-\zeta_i \omega_{ni}(t-\tau)} \sin\left(\omega_{di}(t-\tau) + \phi_i\right)$$

If the magnitude of the impulse is not unity, for example f, then the response is simply scaled by the magnitude f

$$y(t) = f\ h(t-\tau)$$

Now consider the general forcing function. A general, continuous forcing function can be approximated by a series of constant forces acting over a short time period $\Delta$ . The magnitude of each such force is $f(\tau)$, which varies from one time period to the next. The strength of the impulse is the product $f(\tau)\Delta$ . The response of the system to a single such pulse at time is

$$y(t) = f(\tau)\,\Delta\, h(t-\tau)$$

For a linear system, the complete response is given by the sum of responses for each impulse, or

$$y(t) = \sum_{\tau=1}^{\infty} f(\tau)\ h(t-\tau)\Delta \qquad 2.6$$

In the limit as $\Delta \rightarrow 0$ the summation becomes an integral, and the result is the convolution integral

$$y(t) = \int_0^{\infty} f(\tau)\ h(t-\tau)d\tau \qquad 2.7$$

Equation 2.7 gives the system response to a general, continuous force input. It is worthwhile to note that the arguments $\tau$ and $(t-\tau)$ in Eqs. 2.6 and 2.7 can be reversed by a change of variables. Thus these equations can be equivalently written as

$$y_k = \sum_{i=0}^{\infty} h_i \, f_{k-i}$$

$$y(t) = \int_0^{\infty} h(\tau)\, f(t-\tau)\, d\tau$$

where discrete notation has replaced the continuous notation in Eq. 2.6. For continuous systems, Eq. 2.7 is appropriate. However, for sampled data systems, the continuous forcing function is replaced by a series of impulses of strength $f_k = f(k\Delta)$. In this case, it is appropriate to use Eq. 2.6. This equation is, thus, known as the discrete convolution summation. Since virtually all experimental data is sampled, this equation is very important. It will be used extensively in the remainder of this chapter.

The convolution summation is useful for analysis of dynamical systems. If the impulse response function is known, convolution may be used to predict the behavior of the system under the influence of an arbitrary forcing function. Conversely, it will be shown that if input and output sample records are known, the impulse response function can be accurately estimated using deconvolution techniques.

**IMPULSE RESPONSE FUNCTION ESTIMATION**

An accurate, unbiased estimate of the impulse response function can be obtained via deconvolution using Eq. 2.6. To facilitate the estimation, a sample record of the input and output functions is required. The sample interval must be known. In order to ensure unbiasedness, a source signal, described below, must also be available.

Deconvolution may be implemented directly by repetitively using Eq. 2.6. However, in the discussion on convolution, all initial conditions were assumed equal to zero. In general, the initial conditions will not be zero. In this case, Eq. 2.6 must be modified to include initial condition effects. Let the discrete function $Q_k$ be the response of a system to non-zero initial conditions. For dynamical systems, $Q_k$ is generally a function of complex exponentials and is dependent on initial velocities and displacements. In general, the contribution of this term on the total response $y_k$ is unknown. Equation 2.6 can be rewritten to include this term.

$$y_k = \sum_{i=1}^{k} h_i f_{k-i} + Q_k \qquad 2.8$$

The deconvolution technique will solve a set of simultaneous linear equations formed from repetitive use of Eq. 2.8 for a finite number of values of the impulse response function. Let q be this finite number of values for which a solution is sought. Then letting k = q in Eq. 2.8 yields an equation with q+1 unknown parameters, $h_1$ through $h_q$ and $Q_q$, and 2q known values of the input and output data set. The impulse response values for lags 1 through q are constant with respect to the input and output values used. However, the initial condition response term, $Q_q$, is dependent on the particular input/output data set used. If the data records are of length N, then N-q data sets are available, and hence N-q equations can be formed with use of Eq. 2.8. These equations are formed by writing Eq. 2.8 successively for each data set within the data record while holding the dependent variable, k, constant at the value k = q. Conceptually, this is equivalent to evaluating Eq. 2.8 while successively shifting the time origin from the first data point in the record to the (N-q)$^{th}$ data point. The resulting set of equations can be written in matrix form as

$$\{y\} = [F]\{h\} + \{Q\} \qquad 2.9$$

Where

$$\{y\} = \begin{Bmatrix} y_q \\ y_{q+1} \\ y_{q+2} \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{Bmatrix} \qquad \{h\} = \begin{Bmatrix} h_0 \\ h_1 \\ h_2 \\ \cdot \\ \cdot \\ \cdot \\ h_q \end{Bmatrix} \qquad \{Q\} = \begin{Bmatrix} Q_q \\ Q_{q+1} \\ Q_{q+2} \\ \cdot \\ \cdot \\ \cdot \\ Q_N \end{Bmatrix}$$

$$[f] = \begin{bmatrix} f_q & f_{q-1} & f_{q-2} & \cdot\ \cdot\ \cdot & f_0 \\ f_{q+1} & f_q & f_{q-1} & \cdot\ \cdot\ \cdot & f_1 \\ f_{q+2} & f_{a+1} & f_q & \cdot\ \cdot\ \cdot & f_2 \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ f_N & f_{N-1} & f_{N-2} & \cdot\ \cdot\ \cdot & f_{N-q} \end{bmatrix}$$

To solve for the vector {h}, a least squares procedure can be used. However, normal least squares procedures assume that the input is known exactly. If measurement noise is present on both the input and output data records, the resulting estimates are biased. This is easily seen if Eq. 2.9 is written in terms of the measured input and output series as opposed to the true series. Assumption 4. gives the relationship between the measured and true, or deterministic, series. Substituting this relationship into Eq. 2.9 gives

$$\{y_m\} - \{e_y\} = \Big[[f_m] - [e_f]\Big]\{h\} + \{Q\}$$

It is required that the noise terms, $\{e_y\}$ and $[e_f]$ be eliminated. The least squares estimate of $\{h\}$ involves premultiplication by $[f_m]^T$ and replacement of the terms $[f_m]^T\{e_y\}$ and $[f_m]^T[e_f]$ by their expected value of zero. If the expected values of these terms are in fact zero, then the estimate is unbiased. However, it is known that $E[\ [f_m]^T[e_f]\ ] \neq 0$. Thus a bias error is incurred in the estimation of $\{h\}$.

A bivariate least squares technique must be used to eliminate this bias. The instrumental variable method is an implementation of this approach. The method involves premultiplication of Eq. 2.9 by a matrix $[S]$ that has the properties

$$E[\ [S]^T\{e_y\}\ ] = 0$$

$$E[\ [S]^T[e_f]\ ] = 0$$

In addition, the matrix $[S]^T[f_m]$ should be non-singular.

Such a matrix can be constructed with a third series, known as the source series, S. The source series is any time series that meets the following criteria: a) The source series must be correlated with the input series (thus it is also correlated with the output series). b) The source series must be uncorrelated with the measurement noise on both the input and output series.

In the experimental analysis of dynamical systems, the forcing function is often applied to a structure by means of an electromagnetic shaker. A typical experimental setup involves a signal generator which produces the signal used to drive the shaker. The shaker is attached to the structure under investigation through a force transducer. An accelerometer measures the response of the structure. In such a case, a source signal is easily obtained by measuring the output of the signal generator. A sampled series obtained in such a manner will typically meet the source series requirements above.

Application of the instrumental variables method to Eq. 2.9 results in the following equation

$$[S]^T\{y\} = [S]^T[F]\{h\} + [S]^T\{Q\} \tag{2.10}$$

It will later proved that use of this procedure results in an unbiased estimate of {h}.

Inspection of Eq. 2.10 indicates that it is not yet in a form suitable for solution. The term $[S]^T\{Q\}$ must be eliminated. This can be accomplished by either of two methods. First, the parameter q can be chosen large enough such that all initial condition responses have decayed to approximately zero. Second, a random forcing function can be utilized.

The initial condition response term, $Q_k$, decays exponentially in a similar manner to the impulse response function, and both functions have the same time constants. If the parameter q is selected such that $q > 4\tau$, where $\tau$ is the dominant time constant of the system, then the response $Q_q$ will be approximately zero. At this time lag, Q will be 98% less than peak amplitude. In order to neglect the effects of the initial conditions, the amplitude of the term $Q_q$ must be less than the variance of the input measurement noise. If the actual amplitude of the signal were quite large, additional decay time could conceivably be required in order to achieve this level. That is, the value of q could potentially need to be increased. The advantage to this method is that it allows any type of forcing function to be used. There is no requirement that S and Q be uncorrelated at any lag. However, two disadvantages are apparent. First, the dominant time constant is unknown a-priori, and thus an iterative approach is required to guarantee that the initial condition effects are eliminated. Second, for lightly damped systems, a large value of q would be required, with corresponding additional computational effort and reduced statistical significance.

The second alternative is to specify an input function that assures that the source series is uncorrelated with the initial condition response. More specifically, $Q_k$ must be uncorrelated with the previous q values of the source series. In terms of crosscorrelations, this states that

$$RsQ_k = 0 \qquad k = 0,1,\ldots q$$

If the source series is white noise, this condition will be realized. Under this condition, the expected value of the term $[S]^T\{Q\}$ in Eq. 2.10 is zero. Replacing this term with the expected value allows ready solution for the impulse response function by the following system of equations

$$[S]^T\{y\} = [[S]^T[F]]\ \{h\} \qquad 2.11$$

This method has the advantage that the length of the vector {h} can be arbitrarily chosen. That is, Eq. 2.11 may be solved for any desired number q of values of the impulse response function. This can minimize the computational effort required for a solution. In addition, as the parameter q decreases, the statistical degrees of freedom increase, thus improved statistical significance is obtained for the estimate of {h}. The only restriction on this method is the limitation imposed on forcing functions. However, random excitation is often used in dynamic analysis to ensure all modes are excited. Thus, for dynamic analysis the input function limitation is not severe. In addition, if any arbitrary forcing function, not necessarily random, can be shown to meet the criterion above, this method is acceptable.

An alternative representation of Eq. 2.10 can be realized with the crosscorrelation function. This is advantageous from both computational and aesthetic standpoints. The crosscorrelation function of a real valued series is estimated by [23]

$$Rsy_k = \frac{1}{N-k} \sum_{i=1}^{N-k} s_i \, y_{i+k}$$

where $Rsy_k$ is the crosscorrelation function evaluated at lag k, and N is the data record length.

Examination of the matrices $[S]^T\{y\}$, $[S]^T[F]$, and $[S]^T\{Q\}$ in Eq. 2.10 indicates that the matrix elements are approximately equal to the crosscorrelations. For example, the $i^{th}$ element in the vector $[S]^T\{y\}$ is given by

$$\sum_{k=q+1-i}^{N-1-i} s_k \, y_{k-i-1}$$

As before, q represents the length of the impulse response function vector, or the width of the source matrix. The above quantity is approximately proportional to the value $Rsy_{i-1}$. The differences are the factor 1/(N-k) used in the crosscorrelation calculation and in the number of terms included in the summation. However, for large N the error due to summation tends to zero. For small q relative to N, the factors 1/(N-k) approximately cancel when Eq. 2.10 is constructed. Thus

$$[S]^T\{y\} = \{Rsy\} = \begin{Bmatrix} Rsy_0 \\ Rsy_1 \\ Rsy_2 \\ \cdot \\ \cdot \\ \cdot \\ Rsy_q \end{Bmatrix} \qquad [S]^T\{Q\} = \{RsQ\} = \begin{Bmatrix} RsQ_0 \\ RsQ_1 \\ RsQ_2 \\ \cdot \\ \cdot \\ \cdot \\ RsQ_q \end{Bmatrix}$$

$$[S]^T[F] = [Rsf] = \begin{bmatrix} Rsf_0 & Rsf_{-1} & Rsf_{-2} & . & . & . & Rsf_{-q} \\ Rsf_1 & Rsf_0 & Rsf_{-1} & . & . & . & Rsf_{-q+1} \\ Rsf_2 & Rsf_1 & Rsf_0 & . & . & . & Rsf_{-q+2} \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ Rsf_q & Rsf_{q-1} & Rsf_{q-2} & . & . & . & Rsf_0 \end{bmatrix}$$

Substitution of these approximations into Eq. 2.10 yields

$$\{Rsy\} = [Rsf]\{h\} + \{RsQ\} \qquad 2.12$$

Again, if Q and S are uncorrelated, the vector {RsQ} can be replaced with its expected value of zero. Equation 2.12 can then be readily solved for the unknown parameters {h}. Thus the estimate of {h} can be obtained from the solution of the following equation

$$\{Rsy\} = [Rsf]\{h\} \qquad 2.13$$

The crosscorrelation formulation avoids the necessity of bulky matrix operations. In addition, the matrix [Rsf] in Eq. 2.13 is a Toeplitz matrix, thus enabling rapid algorithms to be used for solution of that equation. A Toeplitz matrix is a square matrix with constant values along each (upper left to lower right) diagonal. Efficient algorithms are available to exploit this special form [24] in the solution of sets of linear equations. The total number of calculations to formulate the matrices in Eq. 2.13 is on the order of Nq, whereas for Eq. 2.11 it is on the order of $Nq^2$. The solution of a toeplitz system of equations such as Eq. 2.13 requires on the order of $q^2$ operations, whereas the more general case of Eq. 2.11 requires on the order of $q^3$ operations. Thus, considerable computational advantages are incurred with the use of the crosscorrelation formulation.

An additional, aesthetic advantage of the crosscorrelation approach is the ability to visualize the unbiasedness of the estimator. The diagonal elements of [Rsf] contain the crosscorrelation between the source and input series at lag zero. It is known from statistical theory that the autocorrelation function at lag zero of a series with random content contains the variance of that random content. Likewise, if two series contain correlated random content, the covariance of the random content of the series will appear in the zeroth lag of the crosscorrelation function. Since, by assumption, the source series and the input noise are uncorrelated, the covariance of the input noise to the source series is zero, and thus $Rsf_0$, on the main diagonal of [Rsf], is unbiased. However, had the univariate least squares approach been utilized, that is, had the source series not been available, then the matrix [Rsf] would be replaced with an autocorrelation matrix [Rff]. The autocorrelation matrix is constructed in an identical manner as the crosscorrelation matrix, except that autocorrelations Rff are used instead of crosscorrelations Rsf. Since the input series is known to have a random content, the variance of that random content is not equal to zero, and would appear in $Rff_0$. The main diagonal elements of [Rff] contain $Rff_0$. Thus, from the type of matrix, crosscorrelation or autocorrelation, it is obvious whether a bias error is present in the formulation.

**EXAMPLE**

An example is provided to illustrate the application of the impulse response function estimator given by Eq. 2.13. Consider a single degree of freedom system such as that described by Eq. 2.2. The input, output, and source signals have been sampled. The resulting series are

$f_k$ = { -62,-85,111,-111,94,41,96,-77,-3 }

$y_k$ = { 0,-28914,-44688,72317,-1940,-22677,20201,68041, -45891 }

$s_k$ = { -61,-86,110,-110,91,45,95,-78,-3 }

For this example N = 9. It is desired to solve for the first two values of the impulse response function. Thus, in this case q = 2. The first step is to form the matrices of Eq. 2.12. This is accomplished by calculating the necessary crosscorrelations. For example, the first entry in the vector {Rsy} is $Rsy_0$, or

$$Rsy_0 = \frac{1}{9} \sum_{i=1}^{9} s_i \; y_i = -468553$$

Likewise the remaining crosscorrelations are calculated. The resulting matrix equation is

$$\begin{Bmatrix} -468553 \\ 2833000 \\ -95962 \end{Bmatrix} = \begin{bmatrix} 6783 & -3183 & 1869 \\ -4539 & 6783 & -3183 \\ 1941 & -4539 & 6783 \end{bmatrix} \begin{Bmatrix} h_0 \\ h_1 \\ h_2 \end{Bmatrix}$$

Solving this Toeplitz set of equations gives the impulse response function vector as

$$\{h\} = [\ 151\ 717\ 422\ ]^T$$

To ensure proper scaling of sampled data, these values should be divided by the sample interval.

**UNBIASEDNESS OF ESTIMATOR**

The estimate of a parameter is said to be unbiased if the expected value of the estimate tends to converge to the true value of the parameter. For example, if the sample mean tends to converge to the true mean as the sample size tends to infinity then the estimate of the sample mean is said to be unbiased. The unbiasedness of the impulse response function estimator in the presence of both input and output noise is easily proved. The estimate of the impulse response function is given by rearranging Eq. 2.11

$$\{\hat{h}\} = [[S]^T[{}^mF]]^{-1}\ [S]^T\ \{{}^my\}$$

The superscripts have been added to emphasize that the quantities are measured, and thus include random content. The expected value of {h} is then given by

$$E[\{\hat{h}\}] = E[\ [[S]^T[{}^mF]]^{-1}\ [S]^T\ \{{}^my\}\ ]$$

Combining Eq. 2.9 and assumption 4. gives an expression for

$\{^m y\}$

$$\{^m y\} = [F]\{h\} + \{Q\} + \{e_y\}$$

where $\{e_y\}$ is a vector containing the random content of the output series $\{^m y\}$. Substituting for $\{^m y\}$ above gives

$$E[\{\hat{h}\}] = E[\ [[S]^T[^m F]]^{-1}\ [S]^T\ \{[F]\{h\} + \{Q\} + \{e_y\}\ \}\ ]$$

Again using assumption 4. with the matrix [F], and substituting gives

$$E[\{\hat{h}\}] = E[\ [[S]^T[^m F]]^{-1}[S]^T\{([^m F]-[e_f])\{h\} + \{Q\} + \{e_y\}\ \}]$$

Using the notation $[[S]^T[^m F]]^{-1}\ [S]^T = [S]^-$

$$E[\{\hat{h}\}] = E[\ [S]^-[^m F]\{h\} - [S]^-[e_f]\{h\} + [S]^-\{Q\} + [S]^-\{e_y\}]$$

$$= E[\ [I]\{h\}\ ] - E[\ [S]^-[e_f]\{h\}\ ] + E[\ [S]^-\{Q\}\ ] + E[\ [S]^-\{e_y\}\ ]$$

Where [I] is the identity matrix. Noting that $E[\ [I]\{h\}\ ] = \{h\}$, rearranging and returning to the original notation

$$E[\ \{\hat{h}\}\ ] = \{h\} + E[\ [[S]^T[^m F]]^{-1}\ ]\ (\ -E[\ [S]^T[e_f]\{h\}\ ] + E[\ [S]^T\{Q\}\ ]\ + E[\ [S]^T\{e_y\}\ ]\ )$$

From assumption 6. it is known that both $E[\ [S]^T[e_f]\{h\}\ ] = 0$ and $E[\ [S]^T\{e_y\}\ ] = 0$ since the input and output noise are uncorrelated with the source series. It can also be shown that the source series is uncorrelated with the vector $\{Q\}$. Thus, the quantity $E[\ [S]^T\{Q\}\ ] = 0$. Thus

$$E[\ \{\hat{h}\}\ ] = \{h\} + E[\ [[S]^T[{}^mF]]^{-1}\ ](\ 0\ ) = \{h\}$$

The expected value of the impulse response function estimate is equal to the true value. Thus, the estimator is unbiased in the presence of both input and output noise.

BASIC Z TRANSFORM THEORY

The analysis of the impulse response function requires a knowledge of Z transforms. The Z transform, also known as the discrete Laplace transform, is a generalization of the discrete Fourier transform. It is defined as

$$Z[f_k] = \sum_{i=0}^{\infty} f_k z^{-k} \qquad 2.14$$

This is the one-sided Z transform. It is valid only if $f_k = 0$ for $k < 0$. A detailed description of Z transform theory and properties is provided in [25]. Only a few properties will be treated here.

1. Linearity: If two sequences $x_k$ and $y_k$ exist such that

$$Z[x_k] = X_z$$
$$Z[y_k] = Y_z$$

then for two constants a and b the following relation holds

$$Z[ax_k + by_k] = aX_z + bY_z$$

2. Shifting Theorem: If a sequence $x_k$ exists such that

$$Z[x_k] = X_z$$

then the Z transform of the sequence shifted in time by k′, that is, $x_{(k + k')}$, is given by

$$Z[x_{(k + k')}] = z^{k'}X_z$$

These two theorems can be used to relate the difference equation describing a sequence to the characteristic polynomial of that sequence. Consider a sequence $y_k$, k = 0,...N, that can be described by the second-order difference equation

$$y_k = a_1y_{k-1} + a_2y_{k-2}$$

Taking the Z transform of both sides gives

$$Y_z = a_1 z^{-1} Y_z + a_2 z^{-2} Y_z$$

Canceling the factor $Y_z$ from both sides of the equation gives

$$1 = a_1 z^{-1} + a_2 z^{-2}$$

Multiplication of both sides by $z^2$ and subtracting the right hand terms then gives the characteristic equation

$$z^2 - a_1 z - a_2 = 0$$

Since the characteristic equation is obtained by transformation of the system difference equation, it contains pertinent information regarding the system represented by the original series. System parameters can be obtained from the roots of the characteristic equation. For example, in the analysis of the impulse response series, natural frequencies and damping ratios are obtained from the roots of the system characteristic equation.

# CHAPTER 3: IMPULSE RESPONSE FUNCTION ANALYSIS

The impulse response function contains sufficient information to characterize a dynamic system. Once the impulse response has been obtained, analysis can reveal system modal properties such as natural frequencies, damping ratios, amplitudes and phases. A number of methods are available to perform these analyses. Autoregressive least squares method, Smith least squares, and Pisarenko harmonic decomposition are examples that were introduced in chapter 1. In addition, the correlation-fit method is an adaptation of the autoregressive least squares technique. These methods are considered for analysis of the impulse response estimate obtained by the method presented in chapter 2. The criterion for a suitable analysis method is 1) robustness with respect to data record length, 2) unbiased estimation of modal parameters, 3) efficient implementation and computation, and 4) robustness with respect to additive noise. Based on these criterion, Pisarenko harmonic decomposition is selected as an appropriate analysis tool. Autoregressive least squares is biased in the presence of additive noise. Therefore that method is rejected. The correlation-fit method avoids the bias inherent in the autoregressive least squares formulation, but has been shown to have some implementation difficulties in terms of which autocorrelation lags to use

[11]. Likewise Smith least squares method yields unbiased estimates of the modal parameters, but considerable computation and iteration are involved. In addition, convergence criteria for the method is not well documented. Pisarenko harmonic decomposition is known to perform well with noise and for short data sets [10]. It is computationally straightforward to implement. No bias errors are incurred. Therefore, the analysis technique selected is Pisarenko harmonic decomposition, modified for damped sinusoids.

## PISARENKO HARMONIC DECOMPOSITION DEVELOPMENT

Pisarenko harmonic decomposition is a spectral estimation technique originally developed explicitly for sinusoids with additive noise. The extension to damped sinusoids with additive noise is straightforward. However, it will be shown that some of the computational efficiency of the original method is lost when this is done. The method has been shown to be very accurate [10], even for short data sets in the presence of noise.

## ASSUMPTIONS

The method assumes the following:

1. The time series under investigation, denoted $\hat{h}_k$, is comprised of a deterministic component, $h_k$, and an additive random component, $e_k$, such that

$$\hat{h}_k = h_k + e_k$$

2. The random component, $e_k$, is distributed $n(0,\sigma)$.

3. The deterministic component, $h_k$, is a linear combination of damped sinusoids, and that the damping ratios range from zero (no damping) to, but exclusive of, one (critical damping).

## DEVELOPMENT

For clarity, the initial description for the development will be based on a simplified, single-degree-of-freedom system. The results will then be generalized to multiple-degree-of-freedom systems. Assume that the deterministic component, $h_k$, has the following form:

$$h_k = A\, e^{-\alpha k} \sin \Omega k \qquad 3.1$$

Recall a trigonometric identity that states [26]:

$$\sin \Omega k = [2 \cos \Omega] \sin \Omega(k-1) - \sin \Omega(k-2) \qquad 3.2$$

Substitution of this identity into 3.1 and manipulating the exponential terms gives

$$h_k = [2e^{-\alpha} \cos \Omega] \; A \; e^{-\alpha(k-1)} \sin \Omega(k-1) - [e^{-2\alpha}] \; A \; e^{-\alpha(k-2)} \sin \Omega(k-2)$$

or

$$h_k = [2e^{-\alpha} \cos \Omega] \; h_{k-1} - [e^{-2\alpha}] \; h_{k-2} \qquad 3.3$$

Taking the Z-transform of this equation, multiplying through by $z^2$ and rearranging, yields the following

$$z^2 - [2e^{-\alpha} \cos \Omega] \; z + [e^{-2\alpha}] = 0 \qquad 3.4$$

This represents the characteristic equation of the system. The roots of this equation are

$$z = e^{-\alpha \pm j\Omega}$$

In general, a system will be more complex, that is, it will have more than one mode present. For the case wherein the impulse response function is composed of p/2 damped sinusoidal terms, Eqs. 3.3 and 3.4 are generalized to

$$h_k - \sum_{i=1}^{p} a_i h_{k-i} = 0 \qquad 3.5$$

$$z^p - \sum_{i=1}^{p} a_i z^{p-i} = 0 \qquad 3.6$$

Equation 3.6 has p/2 pairs of complex conjugate roots of the form

$$z_i = e^{-\alpha_i \pm j \Omega_i}$$

From these roots the natural frequencies and damping ratios can be extracted.

$$\zeta_i = \sqrt{\frac{\alpha_i^2}{\alpha_i^2 + \Omega_i^2}}$$

$$\omega_i = \frac{\alpha_i}{\zeta_i \Delta} \qquad 3.7$$

Now, designating the estimated value of the impulse response function as h, by assumption one,

$$h = \hat{h} - e$$

Substitution of this quantity into equation 3.5 gives

$$h_k - \sum_{i=1}^{p} a_i \left( \hat{h}_{k-i} - e_{k-i} \right) = 0$$

or, alternatively

$$\sum_{i=0}^{p} a_i \hat{h}_{k-i} = \sum_{i=0}^{p} a_i e_{k-i} \qquad 3.8$$

$$a_0 = 0$$

This can be written in matrix form as

$$[\hat{h}]^T\{a\} = [e]^T\{a\} \qquad 3.9$$

where

$$[\hat{h}]^T = \{ \hat{h}_i, \hat{h}_{i-1}, \ldots \hat{h}_{i-p} \}$$

$$\{a\}^T = \{ 1, a_1, \ldots \quad a_p \}$$

$$[e]^T = \{ e_i, e_{i-1}, \dots e_{i-p} \}$$

Premultiplying Eq. 3.9 by $[\hat{h}]$ and taking the expectation yields

$$E[ [\hat{h}][\hat{h}]^T ]\{a\} = E[ [\hat{h}][e]^T ]\{a\}$$

or

$$[C_{hh}]\{a\} = \lambda[I]\{a\} \qquad 3.10$$

Where $[C_{hh}] = E[ [\hat{h}][\hat{h}]^T ]$. The right hand side of 3.10 can be seen by expanding the term $[\hat{h}][e]^T$

$$E[ [\hat{h}][e]^T ] = E[ [h+e][e]^T ] = E[ [h][e]^T ] + E[ [e][e]^T ]$$

But $E[ [h][e]^T ] = 0$ because the noise is assumed to be uncorrelated with the deterministic signal, h. Also, $E[ [e][e]^T ] = \lambda[I]$ due to the assumption that the noise is independent, identically distributed. Thus,

$$E[ [\hat{h}][e]^T ] = \lambda[I]$$

and Eq. 3.10, which can be alternately written as

$$[C_{hh} - \lambda I]\{a\} = 0$$

an eigenproblem in which the minimum eigenvalue corresponds to the variance of the random noise content, and the corresponding eigenvector contains the autoregressive coefficients of the characteristic equation of the system. The problem lends itself to the inverse power method of solution since only the minimum eigenvalue and eigenvector are required. Note that if the deterministic component of the signal is purely sinusoidal, the matrix $[C_{hh}]$ becomes Toeplitz, thus increasing the efficiency of solution. However, for the present case in which the signal is known to contain damped sinusoids, the matrix is symmetric non-Toeplitz. Thus, greater computation is involved in the solution.

Once the autoregressive coefficients have been found, the characteristic polynomial of the system may be formed. The natural frequencies and damping ratios can be found from the roots of this polynomial by Eq. 3.7. The amplitudes and phases of each mode can then be found by appropriate curve fitting.

## AMPLITUDE AND PHASE DETERMINATION

The amplitudes and phases of the impulse response function can be determined with a least squares procedure once the natural frequencies and damping ratios have been determined. Equation 3.1 can be rewritten in a more general form as

$$h_k = \sum_{i=1}^{p/2} A_i \, e^{-\alpha_i k} \sin\left(\Omega_i k + \phi_i\right) \qquad 3.11$$

This form clearly indicates the amplitude and phase of each of the p/2 modes of the system. The equation can be linearized for the amplitudes and phases by rewriting it as

$$h_k = \sum_{i=1}^{p/2} e^{-\alpha_i k} \left[G_i \sin \Omega_i k + B_i \cos \Omega_i k\right] \qquad 3.12$$

or, equivalently as

$$h_k = \sum_{i=1}^{p/2} \left[G_i \left(e^{-\alpha_i k} \sin \Omega_i k\right) + B_i \left(e^{-\alpha_i k} \cos \Omega_i k\right)\right]$$

The amplitudes and phases in Eq. 3.11 are related to the coefficients $G_i$ and $B_i$ in Eq. 3.12 by the following relation

$$\phi_i = \tan^{-1}\left[\frac{B_i}{G_i}\right] \qquad 3.13$$

$$A_i = G_i \cos \phi_i + B_i \sin \phi_i$$

A least squares solution for the unknown coefficients $G_i$ and $B_i$ can be obtained by writing Eq. 3.12 repetitively for successive values of the time index k. In this manner, q equations in p unknowns can be formed, where q is the length of the impulse response vector obtained as in chapter 2, and p is the autoregressive model order of the system. These equations can be formulated into a matrix equation. For simplicity use the following nomenclature:

$$ {}^{i}s_k = e^{-\alpha_i k} \sin \Omega_i k $$

$$ {}^{i}c_k = e^{-\alpha_i k} \cos \Omega_i k $$

The following matrix equation can be formulated

$$
\begin{Bmatrix} h_0 \\ h_1 \\ \cdot \\ \cdot \\ \cdot \\ h_q \end{Bmatrix}
=
\left[\begin{array}{cccc|cccc}
{}^{1}s_0 & {}^{2}s_0 & \ldots & {}^{p/2}s_0 & {}^{1}c_0 & {}^{2}c_0 & \ldots & {}^{p/2}c_0 \\
{}^{1}s_1 & {}^{2}s_1 & \ldots & {}^{p/2}s_1 & {}^{1}c_1 & {}^{2}c_1 & \ldots & {}^{p/2}c_1 \\
\cdot & & & & \cdot & & & \\
\cdot & & & & \cdot & & & \\
\cdot & & & & \cdot & & & \\
{}^{1}s_q & {}^{2}s_q & \ldots & {}^{p/2}s_q & {}^{1}c_q & {}^{2}c_q & \ldots & {}^{p/2}c_q
\end{array}\right]
\begin{Bmatrix} G_1 \\ \cdot \\ \cdot \\ \cdot \\ G_{p/2} \\ \hline B_1 \\ \cdot \\ \cdot \\ \cdot \\ B_{p/2} \end{Bmatrix}
$$

or

$$ \{h\} = [S|C] \left\{\frac{G}{B}\right\} \qquad 3.14 $$

Implementation of a least squares solution of Eq. 3.14

involves premultiplication by the transpose of the exponential matrix, or $[S|C]^T$. The solution to the resulting system of equations

$$[S|C]^T\{h\} = [S|C]^T[S|C]\left\{\frac{G}{B}\right\}$$

represents the best estimates of the parameters $G_i$ and $B_i$ in a least squares sense. The amplitudes and phases of Eq. 3.11 can be found by application of Eq. 3.13.

**MODEL ORDER ESTIMATION**

A critical aspect of any time domain analysis method is correct estimation of the model order. A vibrating dynamic system having p/2 modes of vibration will require a model of order p to adequately describe the motion. Underspecification of the model order not only results in the omission of some of the modes in the system, but can distort the modal parameter estimates of the remaining modes. In addition to the computational penalty incurred, overspecification of model order results in the appearance of non-existent frequency content. Some analysis schemes, notably the autoregressive least squares methods, require such overspecification to minimize bias error inherent in these methods. Pisarenko harmonic decomposition is not subject to such bias errors. However, determination of a proper model order is still a paramount concern. A parsimonious model order is the smallest model order that accurately represents the actual system. In all cases it is important to determine a parsimonious model order.

Two methods are presented for estimation of a parsimonious model order for the impulse response function estimate. The first method utilizes the residual variance of the impulse response series. The second method employs a check of the fitted model against the estimated values of the impulse response. The latter is accomplished with a test variable known as the normalized mean square error.

The variance of the residuals can provide an indication of parsimony. As a model is fitted with successively increasing number of parameters, such as occurs when higher model orders are fitted, the residual variance decreases. However, when the proper model order is attained a marked decrease in residual variance occurs. As additional parameters are fitted, the residual variance decreases only slightly. This behavior is not always obvious, however. To enhance the ability to determine parsimony, ratios of variance of successive model orders are often calculated. A large value for this ratio, followed by relatively small values can indicate a parsimonious model. For example, if the residual variance at model order (p-2) is significantly larger than that at p, but the change in variance between model orders p and (p+2), and (p+4) are small, a model order of p is indicated.

Subjective inspection of residual variance and variance ratios is not desirable in many cases. A more rigorous and objective approach is required. A statistical test, called the F-test, provides such a method. Variance estimates follow a chi-square distribution. The ratio of two chi-square random variables, each divided by their respective degrees of freedom, follow an F distribution. Thus the variance ratios may be compared with an F statistic to test for equivalence. If the test indicates that the two variances are significantly different, parsimony has not been achieved. The correct model order of the system occurs when the F-test for additional increases in model order indicates that no significant difference exists between the variance for successive model orders.

The F-test is implemented with the following procedure. First, the residual variance for each postulated model order is calculated. With the Pisarenko harmonic decomposition method the variance is obtained from the minimum eigenvalue of the covariance matrix. These variance values are used for the test. Prior to the start of the test, a desired level of significance, for example 95%, must be chosen. This first step is mandatory for the test to have validity. Designate $V_p$ as the variance obtained from processing at model order p. Begin the process with p = 2, and proceed to increment model order up to the maximum postulated. The F-test is performed at each stage with the following steps.

1) Form the null hypothesis: $V_p = V_{p+2}$

2) Calculate the ratio of the two variances, designated F:

$$f = \frac{V_p}{V_{p+2}}$$

3. Determine the numerator and denominator degrees of freedom. The degrees of freedom are obtained by subtracting the model order, plus one, from the length of the impulse response vector. Thus, the numerator degrees of freedom is $m = (q-p-1)$, where q is the impulse response vector length. The denominator degrees of freedom is $n = (q-p-3)$.

4. Determine the value for an F-statistic with the appropriate degrees of freedom at the desired level of significance. This value can be obtained from statistical tables. This value is designated $F_{\alpha(m,n)}$.

5. Test the hypothesis. If

$$f > F_{\alpha(q-p-1,q-p-3)}$$

then a significant difference in variance is indicated. Reject the null hypothesis. Parsimony has not been attained. Increment p and proceed with step 1 as before.

On the other hand, if

$$f < F_{\alpha(q-p-1,q-p-3)}$$

then the two variances are statistically identical at the desired level of significance. This is indicative that the increased model order did not result in an improved model. Accept the null hypothesis. Parsimony has been attained. Process the data at model order p.

It should be noted that model order estimation is not always such a straightforward procedure. Despite the quantitative nature of the F-test, judgement is required for model order selection.

**NORMALIZED MEAN SQUARE ERROR MODEL ORDER ESTIMATION**

An alternative model order estimation method is also available. This method is based on a comparison of the fitted model against the impulse response estimates. The model order that provides the best fit is assumed to be correct. A measure of fit is provided by the normalized mean square error, or NMSE. The normalized mean square error is computed from the estimated impulse response function and the reconstructed impulse response function by the following formula

$$\text{NMSE} = \frac{1}{q} \sum_{i=1}^{q} \left[ \frac{h_i - h_i}{h_i} \right]^2 \qquad 3.15$$

where $h_i$ is the $i^{th}$ value of the estimated impulse response function and $h_i$ is the $i^{th}$ value of the impulse response function computed from the fitted modal parameters. The fitted impulse response values can be obtained by substituting the fitted modal parameters into Eq. 3.11. This is readily accomplished by using Eq. 3.14 with the estimated values of G and B.

The NMSE model order estimation method can provide useful information with regard to quality of model fit. However it has a drawback in terms of computational intensity. A model must be fitted for each postulated model order, and additional calculations are required to construct the fitted model impulse response values and to calculate the NMSE. In addition, the results obtained by this method are subject to a certain amount of ambiguity. For these reasons, the NMSE model order estimation method is best used as a verification or second check of model fit or model order.

# CHAPTER 4: EXPERIMENTAL RESULTS

Experimental verification is a critical step in the development of any theoretical algorithm. The theory developed in Chapters 2 and 3 is no exception. The impulse response function estimation and analysis methods must be shown to yield accurate estimates of the impulse response function and modal parameters. In addition, any factors, for example, data record length or signal-to-noise ratio, that affect the estimates need to be investigated, and the nature of the effect characterized.

Two experiments were performed in order to test the performance of the impulse response estimator. The first experiment entailed exciting an electrical circuit constructed so as to simulate a single degree of freedom oscillating system. The purpose of this experiment was to 1) verify that the impulse response estimator can provide accurate estimates of the impulse response and modal parameters, and 2) to determine the robustness of the estimator with respect to several pertinent parameters. The second experiment consisted of measuring several modes of a vibrating aluminum beam. The purpose of the beam experiment was to verify the effectiveness of the algorithms for multiple-degree-of-freedom systems, and to investigate model order estimation. The first experiment was designed to verify the method with a simple, relatively noise free system, whereas the second experiment attempted to evaluate performance under more realistic conditions. Computer source code, using the C programming language, was written to implement the time domain algorithms. This source code is listed in the appendix. This chapter describes and analyzes these experiments.

The results of the experiments indicate that the algorithms perform very well for lightly damped systems. In the single-degree-of-freedom (SDOF) experiment, accurate estimates of modal parameters were obtained for light damping even for very short data record lengths and high noise levels. More heavily damped cases resulted in an error in natural frequency estimation. However, a method was found for reducing this error, subject to certain restrictions. The multiple-degree-of-freedom (MDOF) experiment indicated that natural frequencies for several modes can be accurately estimated. However, overspecification of model order was required in some cases. In addition, natural frequency estimates were distorted for frequencies greater than 60% of the Nyquist frequency.

SINGLE-DEGREE-OF-FREEDOM EXPERIMENT

OBJECTIVES: The objectives of the first experiment were to determine:

1. if accurate modal parameter estimates are possible using the impulse response estimator

2. the performance of the impulse response estimator with respect to the following five parameters

   - Data Record Length
   - Noise Level (Signal to Noise Ratio)
   - Sample Rate (Sample Points/Period)
   - Length of Impulse Response Vector (q)
   - Overspecification of Covariance Matrix.

EXPERIMENTAL SETUP

Experimental verification of the impulse response algorithms required a relatively simple and well understood system. In order to facilitate such testing, an electrical circuit was constructed which simulated a single-degree-of-freedom, lumped-mass, mechanical vibrating system. An electrical circuit simulation was chosen because 1) the

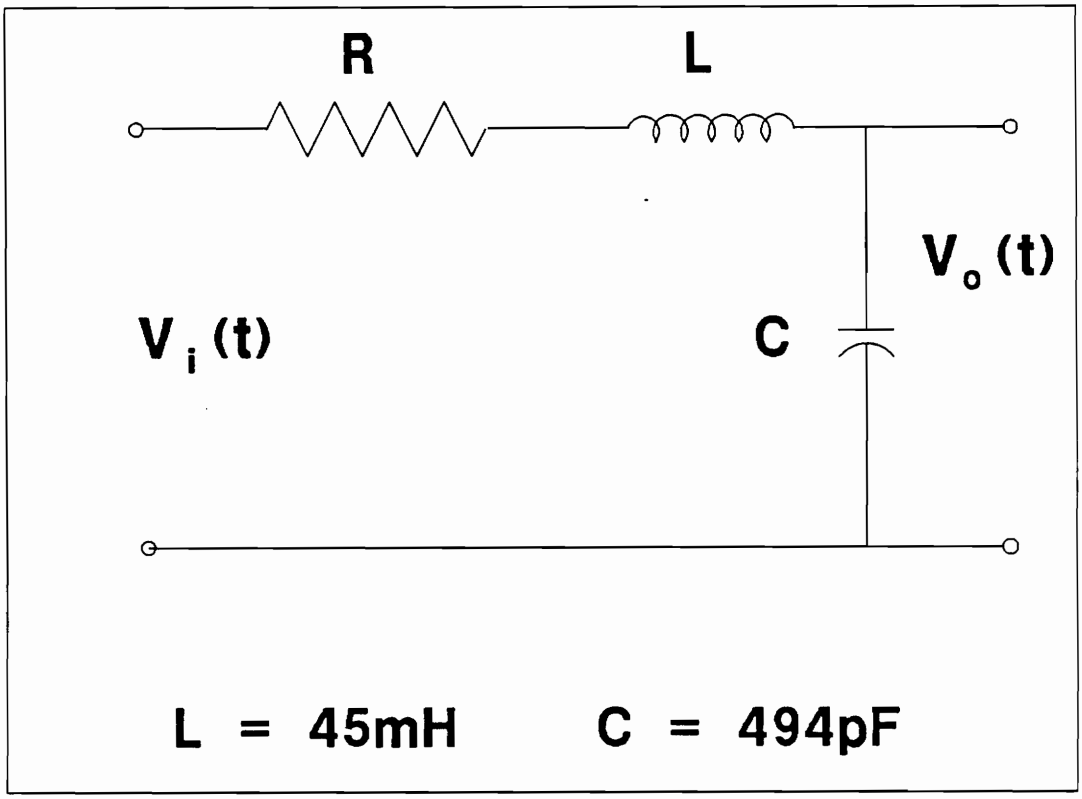


Figure 4

Single-Degree-Of-Freedom Electrical Circuit Diagram

circuit could be constructed so as to ensure the presence of only one mode 2) the measurements would be relatively noise free and 3) the simplicity of the experiment. The circuit consisted of a resistor, inductor, and capacitor connected in series. The applied voltage was measured as the input function, whereas the output was the measured voltage across the capacitor (See fig 4 ).

The electrical circuit effectively simulates a single-degree-of-freedom mechanical system. This is easily seen by inspection of the differential equation describing the output voltage. This expression

$$(LC)\ddot{V}_0(t) + (RC)\dot{V}_0(t) + V_0(t) = V_i(t) \qquad 4.1$$

has a form identical to that of a spring - mass - damper vibrating system. Solution of Eq. 4.1 for the particular case in which $V_i$ is a unit impulse gives the impulse response function,

$$h(t) = \frac{1}{LC\,\omega_d} e^{-\zeta\omega_n t} \sin \omega_d t \qquad 4.2$$

where

$$\omega_n = \sqrt{\frac{1}{LC}} \qquad 4.2a$$

$$\zeta = \frac{RC}{2\sqrt{LC}} \qquad 4.2b$$

$$\omega_d = \omega_n \sqrt{1 - \zeta^2} \qquad 4.2c$$

This is directly analogous to an elastic vibrating mechanical spring-mass-damper system with mass replaced by LC, viscous damping replaced by RC and stiffness equal to unity. As in the case of the mechanical system, the continuous electrical system can be sampled and modeled on a discrete basis.

The actual electrical circuit was constructed as in fig 4, with the exception that the single resistor was replaced with a switched bank of parallel resistors. This allowed various damping ratios to be utilized in the experiment. Four damping ratios, ranging from light to moderately heavy, were used. The net resistance values and corresponding theoretical damping values for each data set are presented in Table 1. The theoretical natural frequency of the system, independent of the resistance value, was 1067.5 Hz. A random (white noise) input signal was generated with a Bruel & Kjaer random signal generator. The electrical circuit was excited and both input and output voltages were measured and recorded using a Rapid Systems 8-bit PC-based data acquisition system. The signals were filtered prior to sampling to minimize aliases. Due to hardware limitations, a source signal had to be synthesized from the input data. This was accomplished by 1) computing the variance of the input data 2) generating a random sequence with a variance of 0.5% of the input variance and 3) adding the imput series and the resulting random series to the input series.

| DAMPING CASE | MEASURED RESISTANCE (OHMS) | DAMPING RATIO |
|---|---|---|
| 1 | 12.4 | 0.021 |
| 2 | 20.0 | 0.033 |
| 3 | 60.0 | 0.099 |
| 4 | 110.2 | 0.183 |

Table 1

Measured Resistance and Corresponding Theoretic Damping Ratios

Twenty data sets were collected. The data sets are tabulated in Table 2. To investigate the robustness of the impulse response estimator with respect to damping ratios, four different damping values were used. To investigate performance with respect to sample rate, data was collected at five sample frequencies. After termination of the experiment it was discovered that defective data was obtained for several of the data sets. The data record lengths were in some instances truncated, in essence reducing the length of the data sets concerned. This discrepancy did not, in most cases, seriously impact the analysis. The particular tests that were affected are annotated in the results section.

| SAMPLE RATE (kHz) | DAMPING CASE 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5 | t1d5k | t2d5k | t3d5k | t4d5k |
| 10 | t1d10k | t2d10k | t3d10k | t4d10k |
| 20 | t1d20k | t2d20k | t3d20k | t4d20k |
| 50 | t1d50k | t2d50k | t3d50k | t4d50k |
| 100 | t1d100k | t2d100k | t3d100k | t4d100k |

TABLE 2

DATA SETS FOR SINGLE-DEGREE-OF-FREEDOM EXPERIMENT

**NOMENCLATURE**

The following discussion and plots refer to particular data sets. The nomenclature used for the data sets indicates both the damping case and the sample rate for that data set. The format for the data set names consists of: 1) a prefix t to indicate time base data, 2) the damping case number (see Table 1) followed by the designator t, 3) the sample rate in kilohertz, followed by the designator k. Thus, the data set named t2d5k refers to time domain data collected from the circuit set at damping case 2 and sampled at 5kHz. The theoretical damping ratio as found from Table 1 for case 2 is 0.033.

## COMPARISON WITH FREQUENCY DOMAIN PROCESSING

The performance of the impulse response estimation algorithms was evaluated by comparing results with frequency-domain results. Modal parameters were estimated by both the time-domain method and by more traditional frequency-domain methods. The frequency domain analysis entailed computing the $H^c$ estimate of the frequency response function. The $H^c$ estimator was chosen due to similarities with the impulse response estimator in respect to noise assumptions. Both estimators assume noise is present on input and output signals, and both estimators eliminate biases by using a source series. Modal parameters were estimated in the frequency-domain by use of a curve fitting method based on Forsythe polynominals [27]. The resulting modal parameter estimates from each method were compared with each other and with the theoretical values. The resulting comparison is presented in Table 3. This process was only performed for the 5kHz sample rate data. For this analysis, eight averages of 1024 point data sets were used for computing $H^c$. A Hanning window was applied to the data to minimize leakage effects. Data sets t1d5k and t2d5k were truncated, having only 700 and 3400 data points, respectively. Thus the FRF for data set t1d5k was computed by one average of 512 data points, with no window function applied. Data set t2d5k was processed with 3 averages of

1024 data points, and data sets t3d5k and t4d5k each were processed with eight averages. The time domain processing, however, used only 1000 data points for each test, with the exception of data set t1d5k, for which 700 data points were used. Tests, described below, demonstrated that the impulse response estimator provides good estimates with this data record length, so the test was considered valid.

During processing, it was noted that the estimated natural frequencies obtained from Pisarenko harmonic decomposition were biased high. The bias was significant for moderate to heavy damping ratios. The natural frequency estimates could be corrected by subtracting the mean value of the estimated impulse response function from each term in the series prior to modal parameter analysis. The damping ratios, amplitudes, and phase angles were not substantially affected by this procedure. Table 3 contains a column for both the corrected and uncorrected parameter estimates. If sampling frequency is high and the impulse response vector length is small, substantial errors can be incurred by using this correction method. In addition, the correction method works best for high damping ratios. However, tests indicate that application of the correction method can lead to instability in frequency estimation. The benefits of the correction method are lost if impulse response vector length is not chosen correctly. Since the optimum vector

| DATA SET | | THEORY | TIME DOMAIN UNCORRECTED | CORRECTED | FREQUENCY DOMAIN |
|---|---|---|---|---|---|
| t1d5k | f | 1067.5 | 1077.7 | 1076.4 | 1071.3 |
| | $\zeta$ | 0.021 | 0.023 | 0.023 | 0.017 |
| | A | 6707 | 6367 | 6349 | 5680 |
| | $\phi$ | 0 | 4.84 | 3.70 | |
| t2d5k | f | 1067.5 | 1078.3 | 1076.0 | 1075.1 |
| | $\zeta$ | 0.033 | 0.034 | 0.031 | 0.037 |
| | A | 6707 | 6359 | 6166 | 6694 |
| | $\phi$ | 0 | 1.47 | 0.095 | |
| t3d5k | f | 1067.5 | 1087.4 | 1074.2 | 1075.4 |
| | $\zeta$ | 0.099 | 0.095 | 0.092 | 0.103 |
| | A | 6708 | 6389 | 6272 | 6788 |
| | $\phi$ | 0 | 2.97 | 0.040 | |
| t4d5k | f | 1067.5 | 1093.1 | 1074.6 | 1075.7 |
| | $\zeta$ | 0.183 | 0.166 | 0.162 | 0.185 |
| | A | 6708 | 6299 | 6133 | 6816 |
| | $\phi$ | 0 | 2.05 | 0.21 | |

Table 3

Comparison of Time-Domain Versus Frequency-Domain Results

Notes: f = Natural Frequency, Hz; $\phi$ = Phase Angle, Degrees

$\zeta$ = Damping Ratio A = Amplitude

length is unknown prior to testing, and no method is readily available for predicting proper vector length, it is possible that application of the correction method can compound frequency errors, rather than alleviating them. In addition, the damping ratio estimates are not improved by application of the correction method. Thus the correction method, despite possible improvements in frequency estimation, is not recommended. Both corrected and uncorrected modal parameter estimates are included in Table 3.

Inspection of Table 3 indicates that the natural frequency estimates obtained from the corrected time domain method agree closely to the values obtained from frequency domain estimates. The theoretical value is somewhat lower than actual due to inaccuracies in the stated inductance and capacitance values. Damping estimates indicate a difference between frequency-domain and time-domain approaches on the order of 10%, with larger errors occurring for the heavier damped cases. Recall, however, that application of a Hanning window can skew damping estimates obtained from frequency-domain methods [2]. The time-domain method produced amplitude estimates on the order of 5% below the frequency-domain estimates. The errors increased slightly with increasing damping ratio. Since the amplitude and phase estimates are based on frequency and damping ratio estimates, if the latter parameters are distorted, the former must necessarily be in error. The underestimation of damping ratios, coupled with the need for using a corrected algorithm for high damping, is considered significant.

**IMPULSE RESPONSE FUNCTION ANALYSIS EVALUATION**

The quality of the impulse response function estimates should be checked against the modal parameter estimates. This was accomplished on a qualitative basis by plotting the estimated impulse response along with impulse response function values computed based on the fitted parameters. To compute the reconstructed impulse response function, the fitted parameters were substituted into Eq. 3.11, and the impulse responses evaluated at times $k$ , $k = 1,...q$. For this test, data sets collected at 20kHz were used. A high sample rate was desirable because of the smoother curve possible with the greater number of points per period of the signal. For visual plots, a smooth curve is advantageous. Higher sample rates yield increased errors of modal parameter estimates (described below). Thus, the 20kHz data sets were chosen for the plots. The estimates were obtained using data record lengths of 1000 and solving for 40 values of the impulse response function.

| DATA SET | NATURAL FREQUENCY | DAMPING RATIO | AMPLITUDE | PHASE (DEG) |
|---|---|---|---|---|
| t1d20k | 1080.5 | 0.054 | 7564 | 7.903 |
| t2d20k | 1090.0 | 0.053 | 6908 | 0.907 |
| t3d20k | 1095.0 | 0.122 | 7354 | 3.275 |
| t4d20k | 1088.0 | 0.199 | 7067 | 1.111 |

Table 4

Modal Parameter Estimates Used For Fitted Model Plots in Figures 5,6,7 and 8

The estimated versus fitted impulse responses are plotted in figs 5,6,7,and 8. Table 4 lists the modal parameters used to calculate the fitted model impulse response values. These values were obtained from the uncorrected modal parameter estimates. Comparison of Table 4 with Table 3 indicates that the damping errors are greater than those obtained from the 5kHz data sets. Frequency errors are comparable.

Three facts can be inferred by inspection of fig 5, the plot for data set t1d20k. First, the overall smoothness of the impulse response estimate is apparent. This attests to the ability of the impulse response estimator to eliminate noise effects. Second, the fitted model plot indicates that the fitted damping estimate is markedly biased high.

Comparison of Tables 3 and 4 confirm the high damping estimate. However, fig 5 indicates that the damping may be due to errors in the Pisarenko harmonic decomposition rather than the impulse response estimation. Thirdly, close examination of the zero crossings indicate that the fitted frequency estimate is higher than the frequency of the estimated impulse response. Again, the Pisarenko harmonic decomposition produces somewhat skewed estimates of the natural frequency.

Inspection of figs 6,7,and 8 confirm the observations made from fig 5. In each case, the estimated impulse response is remarkably smooth. The damping and frequency of the fitted models are slightly distorted. Based on these qualitative observations, two conclusions can be drawn. First, that the impulse response estimation algorithm can produce apparently smooth, accurate estimates. Secondly, that the Pisarenko harmonic decomposition produces slightly distorted estimates of the modal parameters.

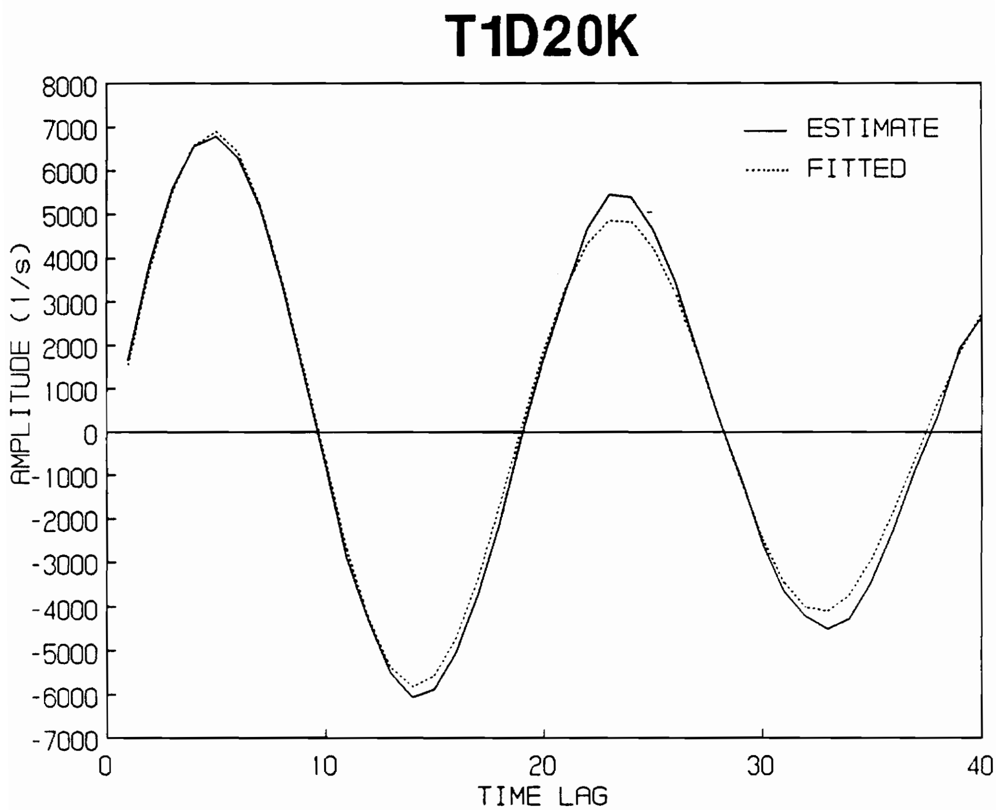


**Figure 5**

**Estimated and Fitted IRF for Data Set t1d20k**

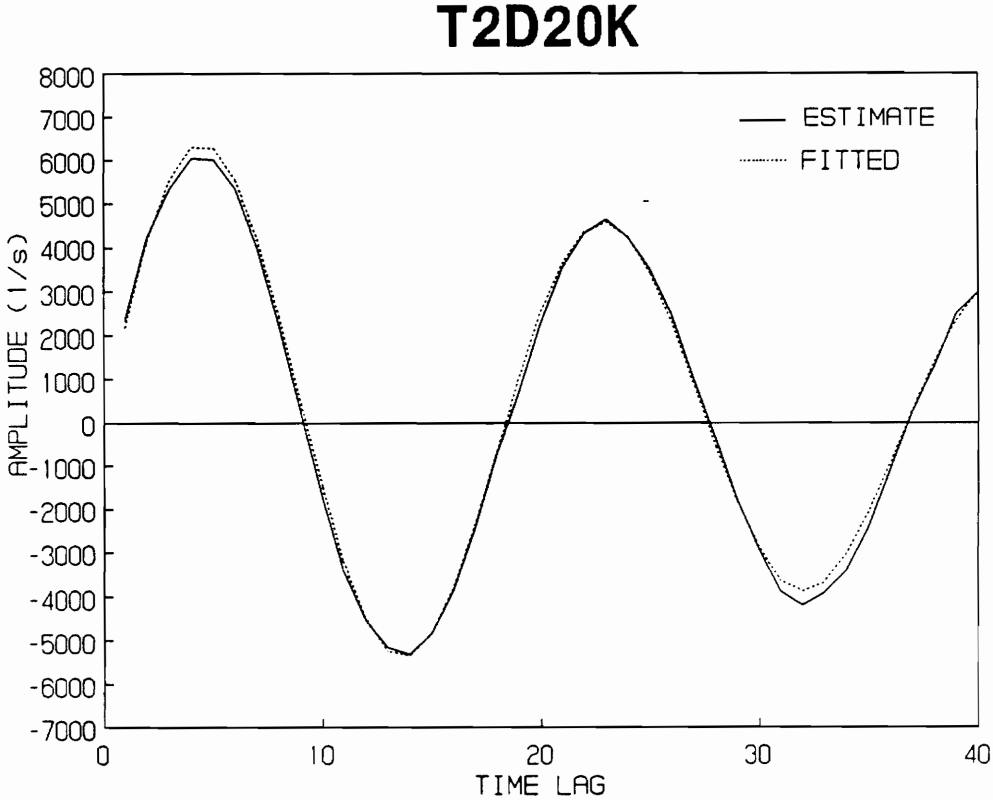


Figure 6

Estimated and Fitted IRF for Data Set t2d20k

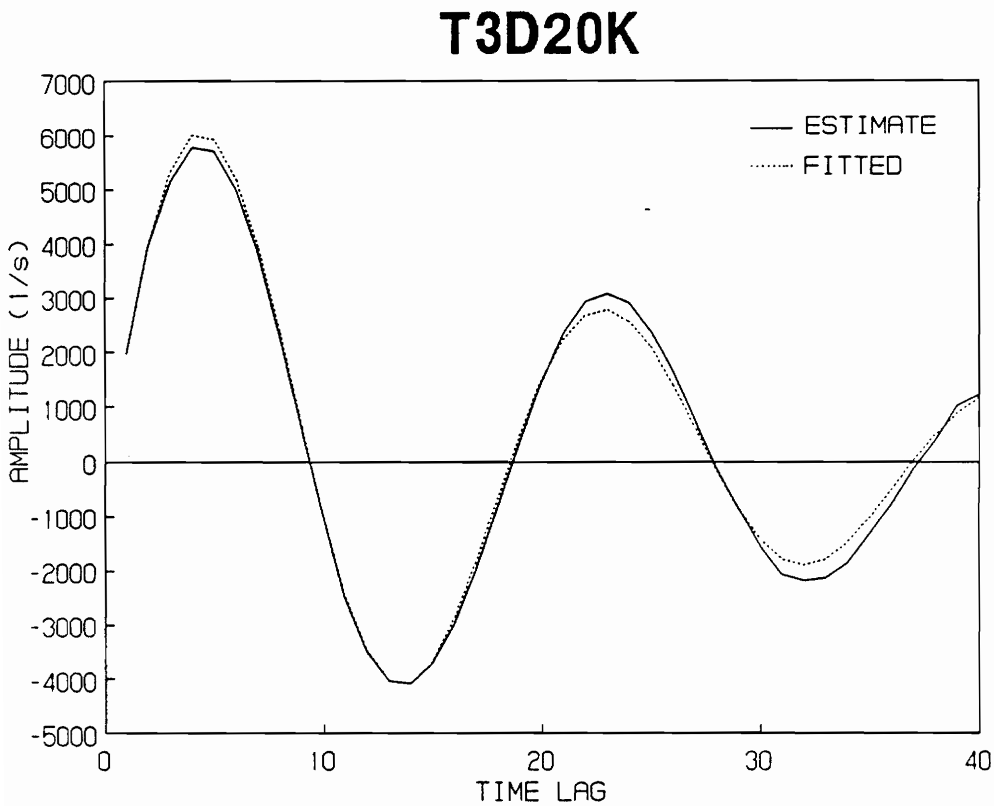


Figure 7

Estimated and Fitted IRF for Data Set t3d20k

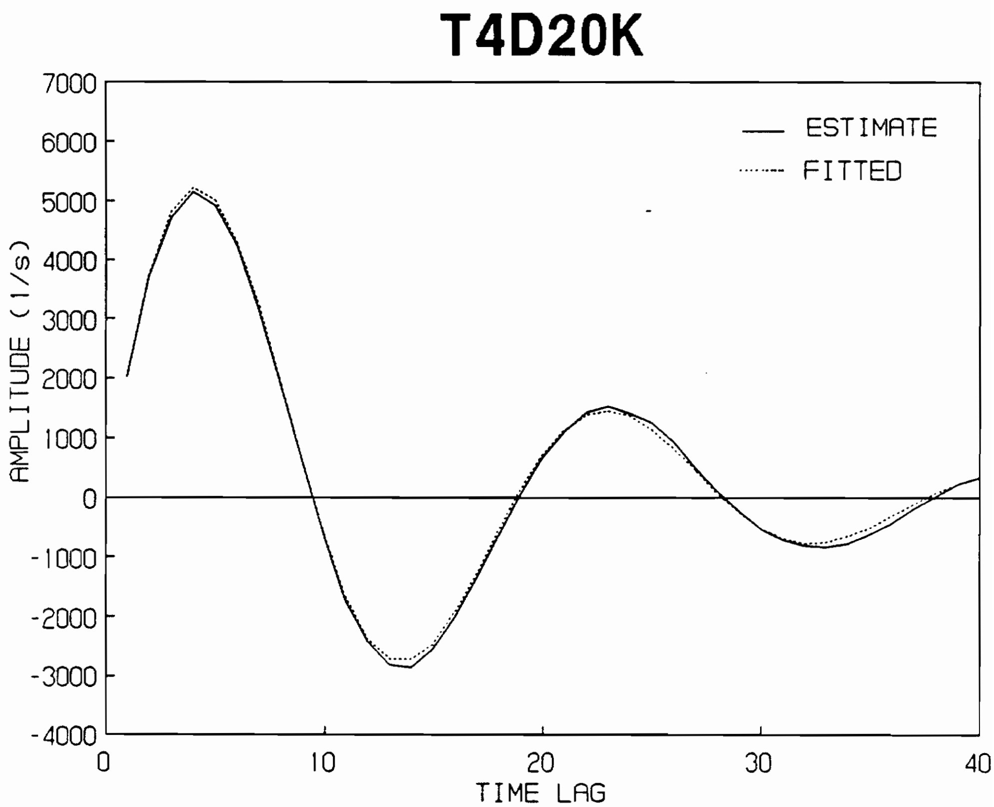


**Figure 8**

**Estimated and Fitted IRF for Data Set t4d20k**

## PERFORMANCE EVALUATION

It was desired to determine the performance of the time-domain algorithms with respect to several important parameters. Five tests were performed on the single-degree-of-freedom electric circuit data . The tests investigated the performance of the impulse response function estimation and analysis algorithms with respect to the following parameters: sample record length, impulse response vector length, measurement noise, sample rate and covariance matrix dimension. These provide important information on both the robustness of the algorithms and on practical implementation considerations. The results of these tests indicate that the impulse response estimator is robust with respect to record length, impulse response vector length, and particularly robust with respect to measurement noise. Excessively high sample rates cause a degradation in performance. At moderate to high damping ratios, excessive overspecification of the covariance matrix dimension also results in poor performance.

In order to facilitate evaluation of performance of the algorithms, a suitable criterion must be chosen. The Pisarenko harmonic decomposition analysis initially computes natural frequencies and damping ratios, whereas the amplitudes and phases are computed in a secondary calculation. For this reason, either frequency or damping ratio should be chosen as a suitable criteria. The theoretical natural frequency of the circuit is independent of the damping ratio. Thus, the natural frequency is the same for all data sets. For this reason, the theoretical natural frequency of the system was chosen as the basis for an comparison. It is known from inspection of Table 3 that the theoretical natural frequency is slightly lower than the actual value due to uncertainty in the theoretical values of the electrical components in the circuit. This error will lead to an offset bias in the results. However, the offset bias should be constant for all data sets and trends can be readily ascertained. Using the theoretical natural frequency as a base value, an performance criterion, termed the normalized frequency, can be defined as

$$\text{normalized frequency} = \frac{f_{calculated}}{f_{theoretical}}$$

A value of unity is expected for error-free frequency estimates. Based on the frequency domain and corrected time domain estimates, the actual natural frequency is approximately 0.7% higher than the theoretical value. Thus, a normalized frequency value of 1.007 is considered correct. Thus, for most data sets the normalized frequency converges to a value slightly greater than one.

| TEST: | Fixed Parameters | | | | |
|---|---|---|---|---|---|
| | RECORD LENGTH | VECTOR LENGTH | NOISE ADDED | SAMPLE RATE (kHz) | COVAR. DIMENSION |
| RECORD LEN. | -- | 40[1] | NO | 5 | 3 |
| VECTOR LEN. | 1000 | -- | NO | 5 | 3 |
| NOISE | 995 | 40 | -- | 5 | 3 |
| SAMPLE RATE | 995 | 45 | NO | -- | 3 |
| COV. DIM. | 1000 | 40 | NO | 5 | -- |

Table 5

Values of Fixed Parameters for Single-Degree-Of-Freedom Tests

1. For data record lengths of less than 250 data points, the IRF vector length was optimized for frequency estimation

The five tests are interdependent. The values of the parameters not being tested impact the results of the test for any one test parameter. Much preliminary work was therefore required to determine the optimum values for the parameters not being tested. Table 5 contains a matrix indicating the values of the fixed parameters for each test. Exceptions made on a case to case basis are noted in the text.

## TEST RESULTS

## RECORD LENGTH

A principal advantage of time domain analysis methods is the enhanced accuracy possible with small data sets. In many applications data may be limited. For these reasons a test was performed to determine the robustness of the impulse response estimation and analysis algorithms with respect to data record length. The test involved four data sets, damping cases 1 through 4 each sampled at 5kHz. The data was processed for sample record lengths consisting of 10,25,50,75,100,125,250,500,675,and 1000 data points. Due to truncation of data set t1d5k, the maximum record length for the first damping case was omitted. The upper limit of 1000 data points was chosen because the rapid convergence of the method indicated that acceptable estimates are possible with 1000 or less data points. For this test the dimension of the covariance matrix was set equal to three, the correct size for a second-order system such as the single-degree-of-freedom electric circuit. The length of the impulse response vector was optimized for each record length less than 250 points, and set equal to 40 for record length of 250 or more. The rationale for these values is based on results obtained from the impulse response vector length test and judicious pre-testing. It was found that for sample record lengths greater than 250, the estimated frequencies stabilize for impulse response vector lengths greater than approximately 40.

The results of the sample record length test are plotted in fig 9. Only uncorrected data is presented. It can be seen that the data exhibits an offset error with higher damping ratios. At the very lowest value, 10 data points, the behavior of the estimation algorithms are unpredictable. With as few as 25 samples, however, all frequency estimates are within 15% of the theoretical value. At 50 data points, all frequencies are within 5% of the theoretical value. The normalized frequency for all data sets is essentially stabilized at 125 data points.

These results indicate that the algorithm is very robust with respect to data record length. The frequency estimates converge rapidly with increasing sample record length. Caution should be exercised, however, in generalizing the numbers obtained from this test to other situations. The data taken from the electric circuit is well behaved in the sense that 1) it is known that only one mode is present, and 2) the data is relatively noise free. However, in general it can be concluded that the method performs well for short data sets.

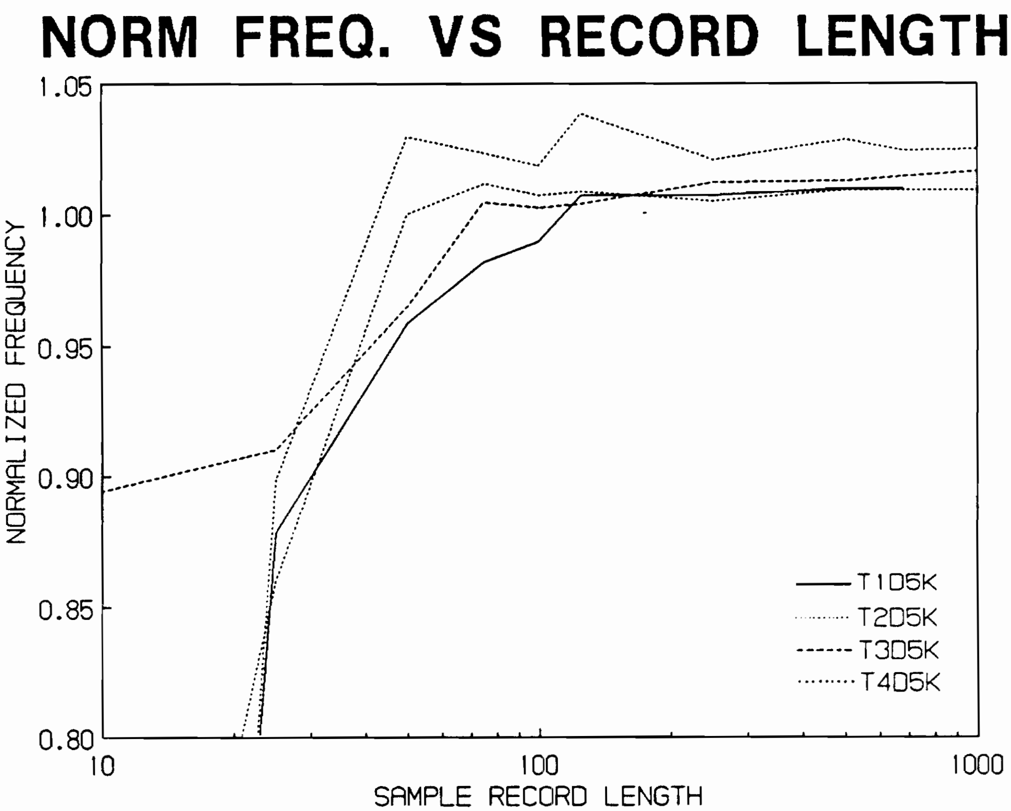


**Figure 9**

**Normalized Frequency Versus Data**

**Sample Record Length**

## IMPULSE RESPONSE VECTOR LENGTH

The impulse response estimation algorithm allows for an arbitrary length of the impulse response vector length to be chosen. That is, it is possible to solve for an arbitrary number of impulse response values. Incorrect choices can impact frequency estimation. A test was performed to determine the nature and extent of that effect. The test involved data sets t1d5k, t2d5k, t3d5k and t4d5k, that is, the four damping cases sampled at 5kHz. The data was processed with a covariance matrix dimension of three. A data record length of 1000 points was used for all except the first damping case, which was processed with 675 data samples. The impulse response vector length was incremented from 5 to 204. The normalized frequency was calculated for each case.

The impulse response vector length affects both the impulse response estimation and analysis algorithms. As the vector length increases, fewer statistical degrees of freedom are available for each element of the impulse response vector. Thus, if the length of the impulse vector is large with respect to data record length, poor statistical estimates of the impulse response are obtained. Conversely, to obtain statistically sound estimates of the modal parameters, the impulse response vector should be relatively large with respect to the number of modes present. Thus, a balance must be struck between the requirements to obtain significant estimates of the impulse response and the modal parameters. This predicament is alleviated somewhat by the fact that as the quality of the impulse response estimates increase, so does the estimates of the modal parameters. Thus, to an extent, a smaller vector length is advantageous. Fortunately, experimental results have indicated a relatively broad band of acceptable values of impulse response vector length, at least for longer data record lengths.

The results of the test of impulse response vector length is plotted in fig 10. For short vector lengths a moderate amount of scatter is apparent in the normalized frequency. This is expected due to the lack of statistical degrees of freedom available for the modal parameter extraction. It is worth noting that the heavier damping cases exhibit less scatter and become stabilized at shorter vector lengths than the light damping cases. For example, the data set t4d5k, with heaviest damping, exhibits stable behavior beyond a vector length of 20. However, the lightly damped data sets t1d5k and t2d5k do not stabilize until roughly a vector length of 50. As in other cases, the bias error in the frequency estimation increases with increasing damping ratio.

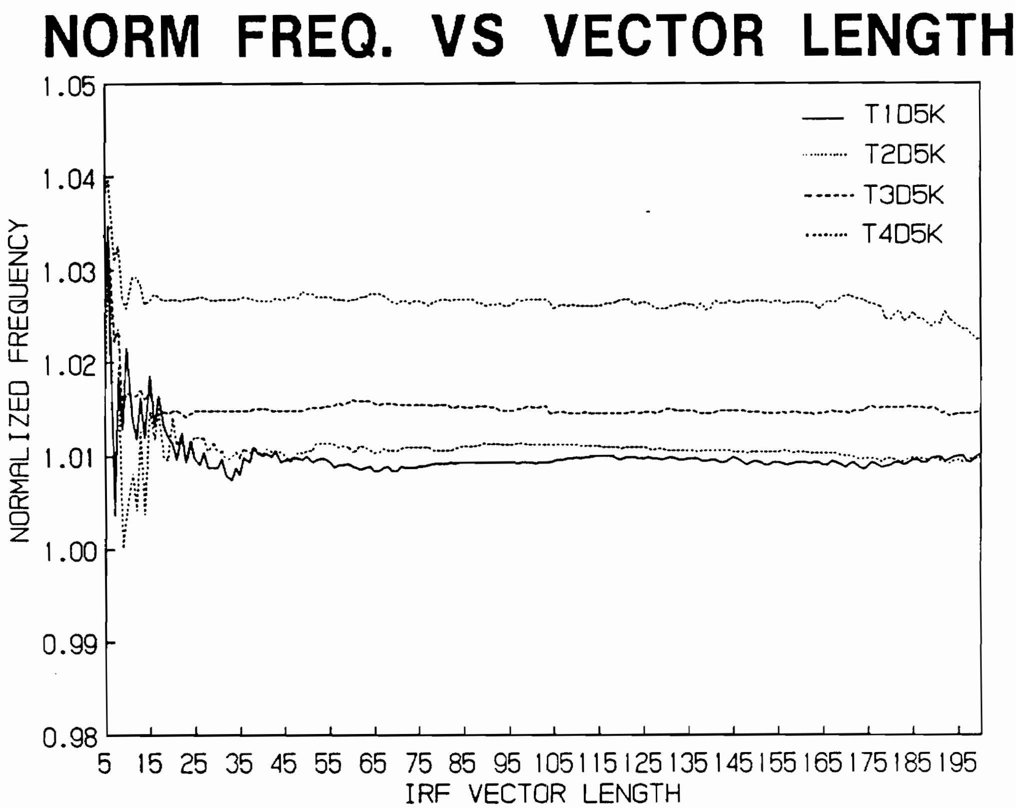


**Figure 10**

**Normalized Frequency Versus IRF Vector Length**

**MEASUREMENT NOISE**

All experimental data is prone to measurement noise. Noise from many sources can contaminate data. Quantization error, analog signal processing, sensor inaccuracy, sensor noise, and many other noise sources can be present. For an estimator to be effective, the noise must be eliminated or at least minimized. The impulse response estimator is designed to perform well in the presence of both input and output noise, since in real data acquisition systems, all channels are equally prone to noise. The purpose of this test was to evaluate the performance of the algorithm with additive input and output noise. As in previous tests, the 5kHz data sets, that is, t1d5k, t2d5k, t3d5k, and t4d5k, were used. For this test, data record length was 670 samples for t1d5k and 995 samples for the remaining data sets. The length of the impulse response vector was fixed at 40. The dimension of the covariance matrix was three. The test was implemented by generating two independent normally distributed random data sets. The random data was then added to the input and output series. Eight levels of additive noise were tested. Since white noise was used as the true input to the system, the signal-to-noise ratio (SNR) was computed based on the variance ratio of the true input to that of the additive noise. The variance of the white noise input was used as signal strength. The signal to noise ratio was then calculated by

$$\text{SNR (dB)} = 10 \,\text{Log}_{10} \left[ \frac{\text{input variance}}{\text{noise variance}} \right] \tag{4.3}$$

Signal-to-noise ratios of 0, 5, 10, 15, 20, 25, and 30 dB were used for the test. In addition, the case with no synthesized noise was included. The procedure involved first calculating the variance of the input series. The noise variance for each desired signal to noise ratio was then calculated using Eq. 4.3. Two random series were then generated using this value of noise variance. Finally, these noise series were added to the input and output data series respectively. The resulting noise contaminated data was then analyzed. Since the source series was not altered, the requirements set forth in Chapter 2 regarding the source signal were met in all cases.

The results of the measurement noise test are plotted in fig 11. The algorithms perform extremely well in the presence of noise. It can be observed that even with signal-to-noise ratios as low as 0dB, the maximum frequency deviation is less than 4% of the theoretical value. In addition, the maximum deviation of any individual data set is approximately 3%. The lightly damped cases in particular exhibit very small deviations. This may be due to the fact that the impulse response function dies out more rapidly for heavier damping, thus resulting in lower impulse

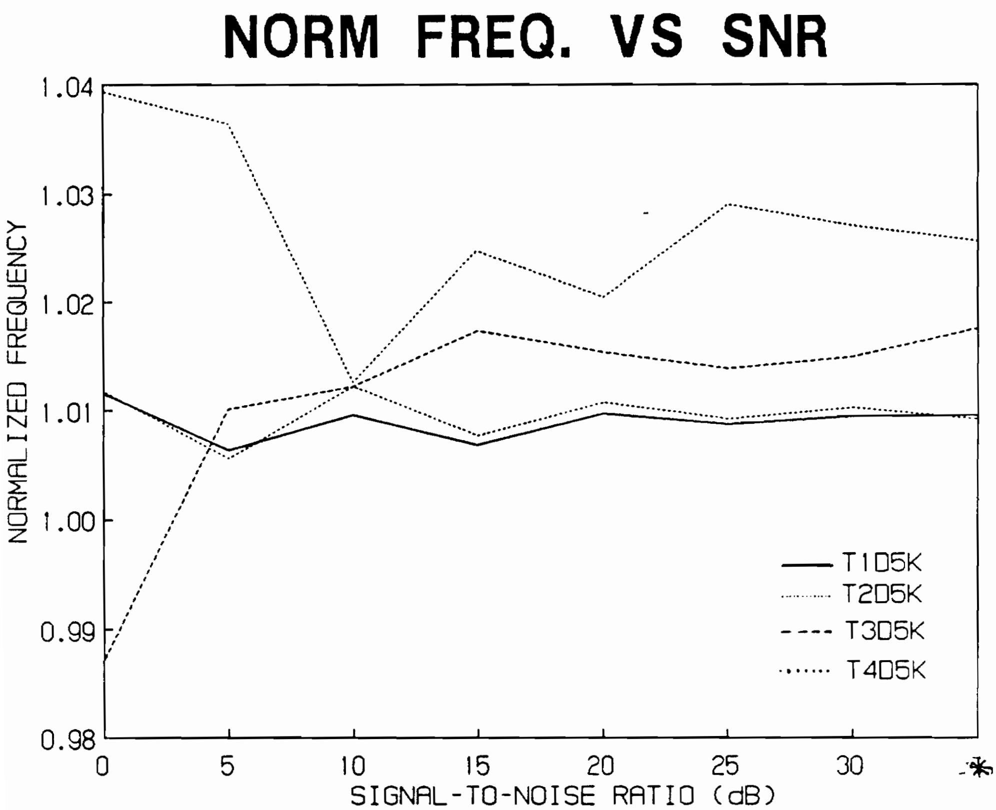


Figure 11

Normalized Frequency Versus Signal to Noise Ratio

* No additive noise

response signal to noise ratios. At 30dB, all data sets achieved essentially the same frequency value as the uncontaminated case. The rapid convergence of frequency estimation with high noise levels indicate the effectiveness of using a source series to obtain unbiased estimates of the impulse response.

**SAMPLE RATE**

In the analysis of sampled signals, it is often assumed that higher sampling rates yield more accurate results. However, this is not necessarily true. As sample rates become increasingly large relative to signal frequency, the differences between successive values of the deterministic component of the signal become small. However, measurement noise may be independent of sample rate. This is strictly true if the noise is white, as was assumed in Chapter 2. Furthermore, some types of noise are band limited to higher frequencies, thus exacerbating the noise problem at high sample rates. Thus, when the sample rate is much higher than the signal frequency, the sampled series can become noise dominant, with a corresponding degradation in frequency estimation. To determine the effect of sample rate on the impulse response estimation and analysis algorithms, a test was performed utilizing data collected at five sample rates. The data record length was set to

1000 data points for all data sets except: 1) data set t1d5k used 675 data points, 2) all data sets sampled at 10kHz were restricted to 550 data points, and 3) data set t2d10k was the most severely restricted data set; only 49 data points were available. These exceptions were due to anomalies in these data sets that resulted in truncated series. Despite these limitations, the test was considered a valid indication of algorithm performance with respect to sample rate. This is because sufficient data was available to obtain useful results even with the limited data record lengths.

The results of the sample rate test are plotted in fig 12. Note that for generality, the actual sample rate has been scaled by the natural period of the signal. Thus the horizontal axis represents sample points per natural period of the signal frequency. It can be seen that for low numbers of samples per period, below 18.7 points per period (5 to 20 kHz actual sample rate), there is little variation in frequency estimation. Above 18.7 sample points per signal period increasing scatter is observed in the frequency estimate of all data sets. This is likely due to the increased effect of measurement noise at high sample rates.

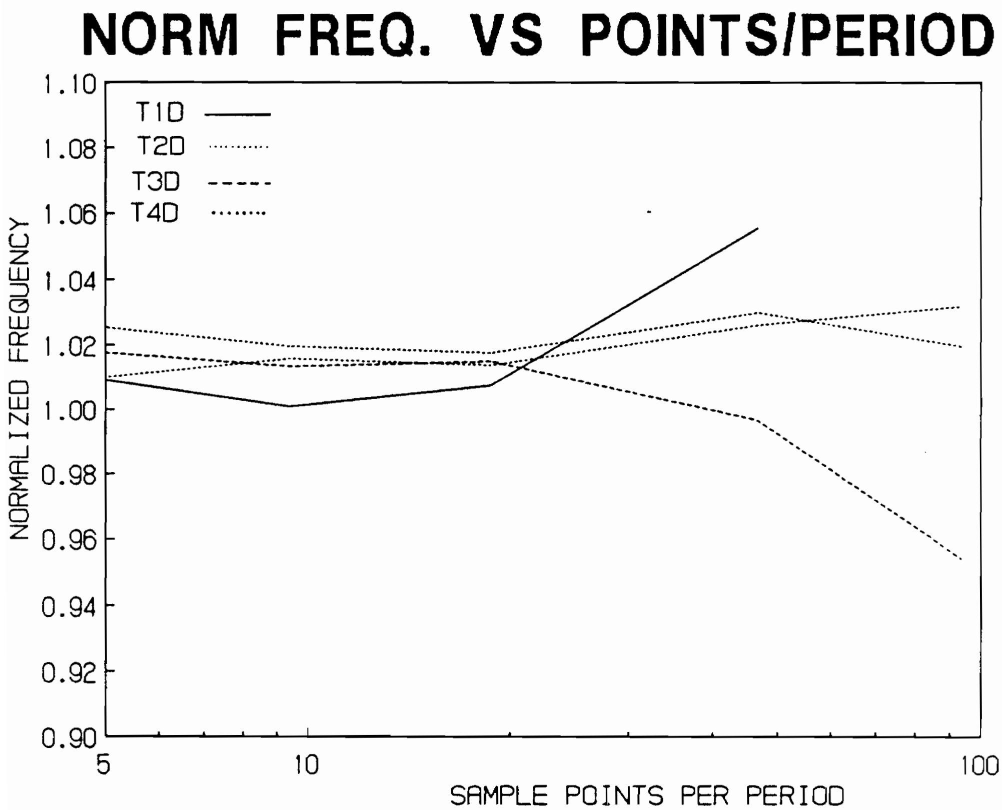


Figure 12

Normalized Frequency Versus Sample Rate

**COVARIANCE MATRIX DIMENSION:** The final test involved determining the effect of overspecification of the covariance matrix on frequency estimation. In the application of Pisarenko harmonic decomposition, a typical methodology for determining model order consists of forming the covariance matrix for the largest postulated model order. The matrix is then partitioned, forming a submatrix comprised of the upper lefthand corner of the covariance matrix. The submatrix is taken to be the covariance matrix for the lowest possible model order. The eigen analysis is performed on the submatrix, and the minimum eigenvalue is saved. The covariance matrix is again partitioned and solved for the second smallest postulated model order. The process is repeated until the entire range of postulated model orders have been analyzed. A parsimonious model order is then selected based on the minimum eigenvalues of the submatrices. This method has considerable computational advantages in that the covariance matrix need only be computed once. For pure sinusoids in additive noise the covariance matrix is Toeplitz and hence the method is effective. However, for damped sinusoids, the covariance matrix is not Toeplitz, and thus errors could be incurred by partitioning the covariance matrix.

To determine the extent of these errors, a final test was performed. For this test, the covariance matrix was constructed successively for orders 3,5,7,9,11,13,15. Recall that the dimension of the covariance matrix must be one larger than the largest postulated model order, hence the odd numbers. Data sets t1d5k, t2d5k, t3d5k, and t4d5k, were used for this test. The length of the impulse response vector was set to 40 in all cases. The data record length for the first damping case was 700, for all other cases, 1000. All processing was performed for model order 2. Thus the covariance matrix was overspecified for dimensions 5 and greater.

Test results for the covariance matrix dimension are plotted in fig 13. The results are as expected. The light and moderately damped cases, data sets t1d5k, t2d5k and t3d5k, exhibit little sensitivity to overspecification of the covariance matrix. However, as the damping ratio increases, the frequency estimate becomes significantly low with overspecification of the covariance matrix dimension. Data set t4d5k, with a theoretical damping ratio of 0.183, indicates a slight error at covariance matrix dimension of 9 and significant error by dimension 15. In light of this observation, care should be utilized to form the covariance matrix to the proper dimension prior to extraction of modal parameter. This is particulary true for moderate to heavy damping.

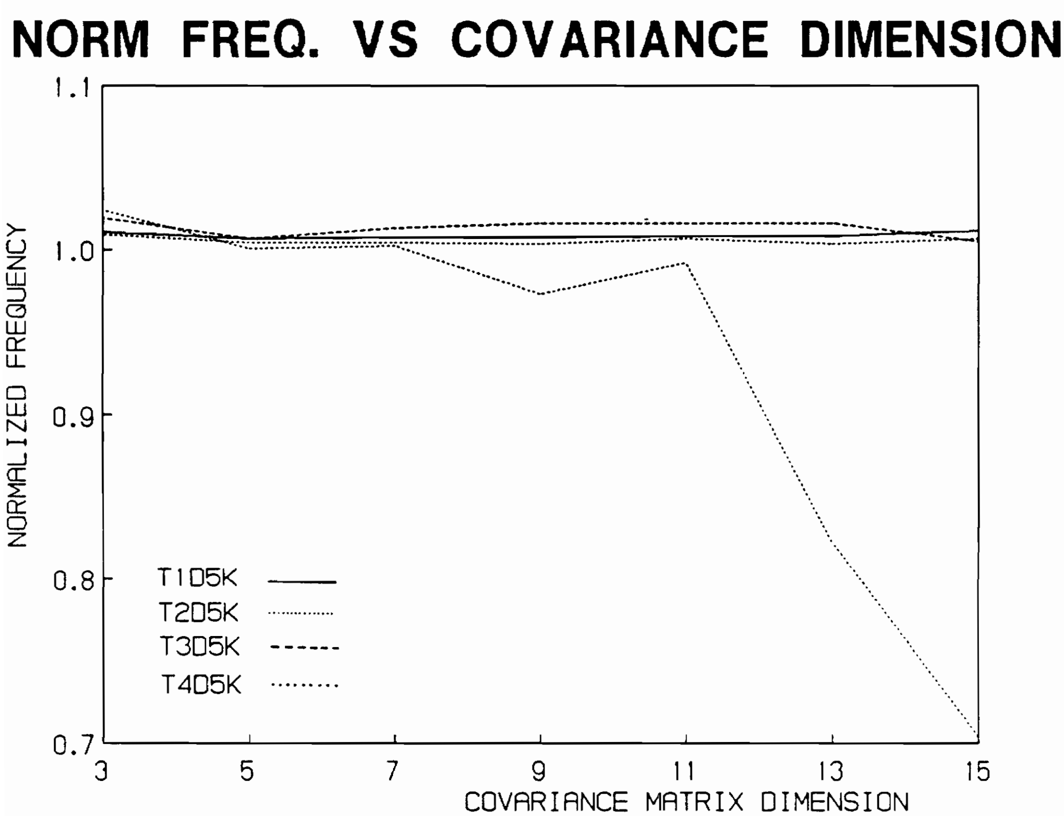


**Figure 13**

**Normalized Frequency Versus Covariance Matrix Dimension**

## MDOF EXPERIMENT

### OBJECTIVE

Any practical impulse response estimator must perform satisfactorily for multiple-degree-of-freedom systems. The second experiment was performed in order to evaluate the performance of the estimator with respect to such a system. Particular goals of the experiment were to evaluate:

1) The ability of the impulse response function estimator to identify each mode in a multiple degree of freedom system

2) The performance of model order estimation methods

### EXPERIMENTAL SETUP

In order to implement the objectives of the experiment, a multiple-degree-of-freedom system suitable for testing had to be chosen. The criteria used to determine the nature of the test system include ease and reliability of measurement and applicability to realistic dynamical analysis. Measuring the bending modes of a vibrating beam met these criteria. Thus, an aluminum beam was chosen for implementation of this experiment. Figure 14 depicts the

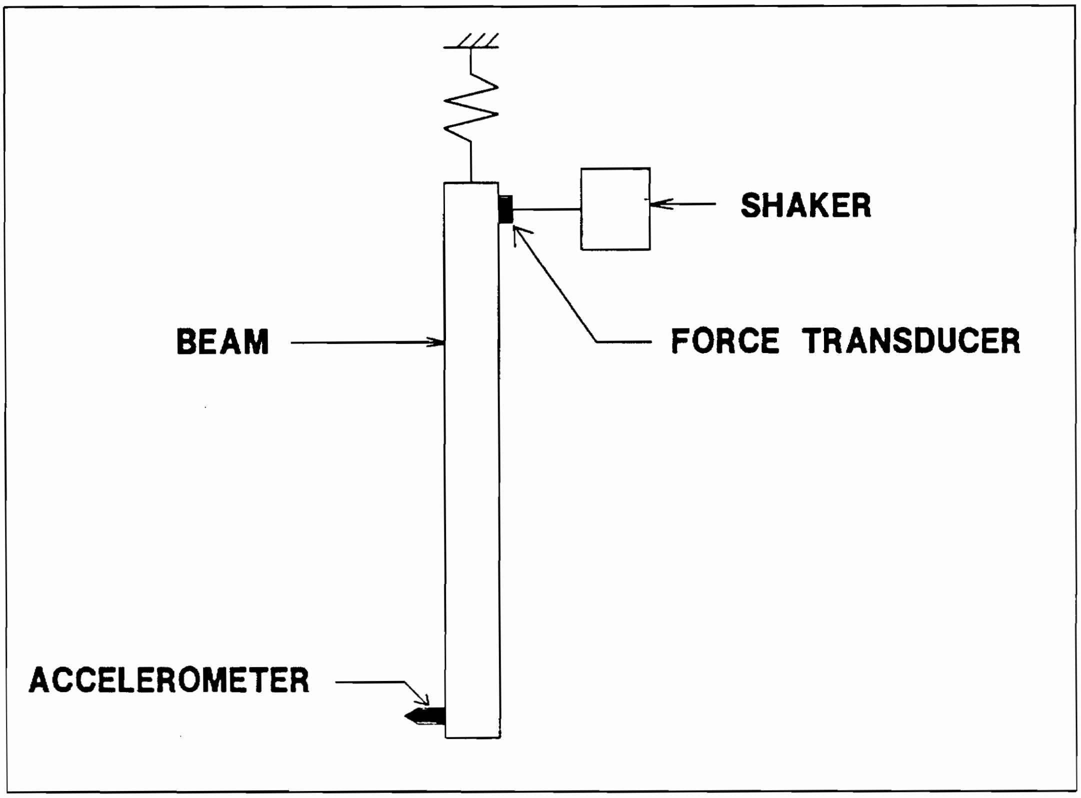


Figure 14

MDOF Experimental Setup

experimental setup. The beam was suspended on one end with an elastic support so as to simulate free-free boundary conditions. The input force was applied by an electromagnetic shaker mounted at the upper end of the beam. A force transducer mounted between the shaker and beam measured the input force. The output was measured by an accelerometer. For this test, only one response location was necessary to meet the objectives outlined above. The lower terminus of the beam the selected response location so as to minimize the possibility of placing the accelerometer near a node, thus missing a mode of vibration of the beam. A random, white noise input function was used. The input was generated by a Bruel & Kjaer type 1405 random signal generator. Input and output signals were filtered and amplified to minimize aliasing and scale the signals. A Stanford Research Systems, Inc. Model SR 640 low pass anti-aliasing filter was used. The signals were then digitized by a PC-based data acquisition system utilizing a Data Translation DT2818 12-bit analog-to-digital converter.

Preliminary testing of the beam using frequency-domain methods indicated that the first three bending modes occurred at approximately 73, 202 and 396 Hz. These compare to theoretical values of 79.9, 220, and 431 Hz. The theoretical natural frequencies were computed using formulas from Blevins [28]. Based on these frequencies, four data sets were collected. Two data sets were sampled at 768 and 1280 Hz so as to contain respectively, two and three vibration modes of the beam. Two additional data sets were taken at higher sample rates of 900 and 1800 Hz to determine the effect of oversampling on frequency estimation. The anti-aliasing filter cutoff frequency for both the normally sampled and the oversampled cases was identical. A contiguous data block consisting of 5400 data

| DATA SET | SAMPLE FREQUENCY (Hz) | FILTER CUTOFF (Hz) | RECORD LENGTH |
|---|---|---|---|
| 1 | 768 | 300 | 5400 |
| 2 | 1280 | 500 | 5400 |
| 3 | 900 | 300 | 5400 |
| 4 | 1800 | 500 | 5400 |

Table 6

Data Acquisition Parameters for MDOF Experiment

points was recorded for each data set. Table 6 summarizes this information.

## COMPARISON WITH FREQUENCY DOMAIN PROCESSING

Data collected from the vibrating beam was processed in both the time-domain and the frequency-domain. The natural frequency and damping ratio estimates obtained from each method were compared. All data points were used for time domain processing. The time domain method involved estimation of 60 values of the impulse response function. Five data blocks of 1024 points each were averaged for frequency domain estimates. A Hanning window was applied prior to transforming the data. The frequency response function was obtained from the $H^c$ estimator. As in the previous experiment, modal parameters were extracted by frequency domain curve fitting using Forsythe polynomials.

Table 7 compares the natural frequency and damping ratio estimates obtained from the time-domain and frequency-domain methods. Four observations can be made from data sets 1 and 2. First, the natural frequency estimates obtained from the time domain method are slightly higher ( 1-2% ) than the corresponding frequency-domain estimates. This is consistent with observations made for the SDOF experiment for uncorrected data. Second, model order for the time domain method appears to be overspecified. However, The low frequency modes may represent rigid-body modes of the beam. The heavy damping ratios for these modes lend support to this hypothesis. The frequency-domain curve fitting procedure did not attempt to fit rigid-body modes. Thus, the model order may in fact be correct. Thirdly, the highest frequency estimate from the time-domain algorithm is severely distorted in both data sets. In both cases, the estimated frequency is well below that obtained from frequency domain methods. Fourth, damping estimates differ by an order of magnitude between the time-domain and frequency-domain methods. It is known, however, that application of a Hanning window distorts damping estimates in the frequency domain [2]. Thus no definitive statements can be made regarding damping errors.

| | | NATURAL FREQUENCY | | DAMPING RATIO | |
|---|---|---|---|---|---|
| DATA SET | Theory | Time Domain | Freq Domain | Time Domain | Freq Domain |
| 1 | | 10.4 | | 0.177 | |
| | 79.9 | 71.5 | 73.1 | 0.016 | 0.0019 |
| | 220.1 | 181.6 | 201.1 | 0.016 | 0.0012 |
| 2 | | 14.9 | | 0.128 | |
| | 79.9 | 74.4 | 73.1 | 0.032 | 0.0069 |
| | 220.1 | 204.1 | 201.3 | 0.017 | 0.0029 |
| | 431.4 | 243.0 | 395.6 | 0.018 | 0.0017 |
| 3 | | -50.1 | | | |
| | | -14.1 | | | |
| | 79.9 | 72.2 | 73.2 | 0.017 | 0.0007 |
| | 220.1 | 202.3 | 201.0 | 0.015 | 0.0004 |
| 4 | | -124.3 | | | |
| | 79.9 | 75.0 | 72.7 | 0.044 | 0.0013 |
| | | 107.0 | | 0.437 | |
| | 220.1 | 203.8 | 201.2 | 0.017 | 0.0038 |
| | | 214.9 | | 0.015 | |
| | | 241.1 | | 0.011 | |
| | 431.4 | 399.3 | 394.9 | 0.010 | 0.0002 |

Table 7

Comparison of Time Domain Vs Frequency Domain, MDOF

Inspection of data sets 3 and 4 indicate a definite model order overspecification for the oversampled cases. Non-existent frequency content is present in the form of negative frequencies as well as spurious positive frequencies. However, the time-domain approach successfully identified the highest frequency mode present in the data. Thus, Table 7 indicates that the time-domain impulse response estimator is capable of estimating the natural frequencies of a multiple-degree-of-freedom system subject to the restriction that either 1) frequencies greater than approximately 60% of the Nyquist frequency, can not be resolved, or 2) the data must be oversampled and model order overspecified in order to estimate the high frequency component. For the latter case, spurious frequencies must be distinguished from the actual frequency content of the system.

## MODEL ORDER ESTIMATION

Parsimonious model order was determined by using ratios of the residual variance. A measure of the residual variance was obtained from the minimum eigenvalue of the covariance matrix. Actual variance of the residual series is found by dividing the minimum eigenvalue by (q-p-1), where q is the length of the impulse response vector and p is model order. Model order estimation proceeded as described in Chapter 3.

The minimum eigenvalue and minimum eigenvalue ratios for two of the data sets are presented in Table 8. Data sets 2 and 4 illustrate the model order estimation procedure. Model order estimation for data set 2 is unambiguous. The minimum eigenvalues and minimum eigenvalue ratios behave exactly as predicted by theory. The ratios for this case follow the typical pattern, increasing to a large value of 15.6 at model order 8, followed by a sharp decrease to 1.47 at model order 10. This indicates that model order equal to 8 is appropriate. Application of an F-test provides additional verification. From tabulated values, at the 95% significance level an F-statistic is given by

$$F_{(51,49)} = 1.6$$

It is apparent from inspection of Table 8 that variance ratios between model orders 4 and 6 and orders 6 and 8 are significant. The variance ratio between model orders 8 and 10 is 1.47. Since

$$F_{(51,47)} = 1.6 > 1.47$$

The conclusion is that the change in variance between model orders 8 and 10 is insignificant. This is interpreted as indicating that the model of order 10 is not an improvement over the model of order 8. Therefore model order 8 is chosen.

| DATA SET/ SAMPLE RATE/ FILTER | MODEL ORDER | MINIMUM EIGENVALUE | EIGENVALUE RATIO |
|---|---|---|---|
| 2 | 2 | 1680671 | |
| 1280Hz | 4 | 1244482 | 1.35 |
| 500Hz | 6 | 179699 | 6.92 |
| | 8 | 11490 | 15.64 |
| | 10 | 7787 | 1.47 |
| 4 | 2 | 5841325 | |
| 1800Hz | 4 | 3432689 | 1.70 |
| 500Hz | 6 | 2191250 | 1.57 |
| | 8 | 1177581 | 1.86 |
| | 10 | No Convergence | |
| | 12 | 358804 | 3.28 |
| | 14 | 20728 | 17.3 |
| | 16 | 11093 | 1.87 |

Table 8

Minimum Eigenvalues and Ratios for Beam Data

A similar analysis was performed on all data sets. Note that data set 4 in Table 8 displays some of the problems that can arise in model order estimation. The eigenanalysis in the Pisarenko harmonic decomposition failed to converge at model order 10. In addition, while inspection of minimum eigenvalue ratios clearly indicates a model order of 14, application of an F-test at the 95% confidence level indicates that a significant difference still exists between model orders 14 and 16. In this instance, the results of the F-test were rejected in light of the large variance ratio between model orders 12 and 14. Model order 14 was selected. Recall that model order estimation is not an exact procedure, and some jugdment is required of the investigator.

# CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

This thesis has presented a time-domain algorithm for experimental determination of the impulse response function of linear elastic vibrating systems. The method is based on deconvolution of measured input-output data. A source series, in conjunction with the instrumental variables technique, provides for unbiased impulse response estimates. Modal parameters such as natural frequency, damping ratio, amplitude and phase can be extracted from the estimated impulse responses by a separate procedure. Pisarenko harmonic decomposition is the procedure used to determine these parameters. Two experiments have verified the validity of the algorithms, as well as illuminated some problems which require further investigation. The major conclusions and resulting recommendations are summarized below.

The experiments described in Chapter 4 of this thesis demonstrate that the impulse response estimation algorithm can produce accurate estimates of the impulse response function for lightly damped systems and for signal frequencies well below the Nyquist frequency. Several specific comments can be made. Modal parameter estimates obtained from both the time domain and frequency domain methods were compared. The results indicate that comparable esimates are possible. For the tested single-degree-of-freedom system, natural frequency estimates between the two methods agreed within 2%. For heavily damped systems, modal parameter estimates are slightly distorted. However, qualitative review of plots of the estimated impulse responses, as compared with impulse response values obtained from the modal parameters, indicate that the distortion may be due to the modal parameter estimation process rather than the impulse response estimation process. Thus, the impulse response estimation algorithm may be more accurate than the Pisarenko harmonic decomposition procedure. Estimates of signal frequencies above 60% of the Nyquist frequency are distorted. This distortion can be eliminated by oversampling the data. However, oversampling results in the appearance of spurious frequencies.

The impulse response estimator was also tested for performance with respect to sample record length, impulse response vector length, signal-to-noise ratio, sample rate, and covariance matrix dimension. The method was found to be robust with respect to sample record lengths. Accurate estimates were obtained from very short data sample records. The impulse response estimates were shown to be stable with respect to impulse response vector length. Although for very short vector lengths frequency estimation errors can occur, these errors quickly die out with increasing vector length. Performance of the method in the presence of additive noise was particulary good. At signal to noise ratios as low as 0dB, modal estimates varied less than 3% from the noise free cases. Very high sample rate, exceeding approximately 50 samples per natural period, led to some distortion of modal parameter estimates. Overspecification of covariance matrix dimension had little effect on lightly damped systems. However, for excessively large matrix dimensions, the natural frequency estimates for heavily damped systems was distorted.

## RECOMMENDATIONS

The above observations lead to several recommendations for further study.

1. Qualitative observations of the impulse response estimates, compared with fitted model impulse responses indicate that the modal parameter estimation errors may not be due to impulse response estimation errors. The impulse response estimation algorithm should be quantitatively tested independently of the modal parameter estimation (Pisarenko harmonic decomposition) algorithm.

2. The Pisarenko harmonic decomposition algorithm, as modified for damped sinusoids, should be rigorously

investigated to determine the reason for inaccurate modal estimates. Alternative analysis methods, such as those described in Chapter 1 could be considered if improvements with the Pisarenko method are not feasible. Comparisons of the various methods, tailored to the specific needs of the impulse response analysis, should be made.

3. The cause for inaccurate frequency estimates for high frequency signals should be investigated. In many cases, frequency content above 60% of the Nyquist frequency may exist in a signal. The failure of the algorithms to identify these frequencies is a significant discrepancy. An analysis should be performed to isolate the reason for the failure of the algorithm at these frequencies.

4. A rigorous statistical analysis of the method should be performed. A sound statistical description of the impulse response estimates should be available. Statistical descriptions of the input and output noise series need to be developed. These statistical descriptors are needed to quantify the quality of the impulse response estimates.

5. In order to transform the present algorithms into useful modal analysis tools, the theory and algorithms need to be expanded to include multiple input-multiple output systems.

## REFERENCES

1. Robinson, Enders A., Historical Perspective of Spectrum Estimation, Proc. IEEE, vol. 70, pp. 885-906, Sept. 1982.

2. Wicks, A.L., Personal communication, Mar. 1989, Virginia Polytechnic Institute and State University, Blacksburg, VA.

3. Mitchell, L.D., et al., An Unbiased Frequency Response Function Estimator, Int. J. of Ana. and Exp. Modal Ana., vol. 3, pp. 12-18, Jan. 1988.

4. Smith, W.R., Least-Squares Time-Domain Method for Simultaneous Identification of Vibration Parameters from Multiple Free-Response Records, AIAA Paper 81-0530, 1981.

5. Yule, G.U., On a Method of Investigation Periodicities in Disturbed Series, With Special Reference to Wolfer's Sunspot Numbers, Phil. Trans. Roy. Soc. London, A, vol. 226, pp. 267-298, 1927.

6. Box, G.E.P., and Jenkins, G.M., Time Series Analysis: Forcasting and Control, Holden-Day, San Francisco, 1976.

7. Kay, S.M., and Marple, S.L., Spectrum Analysis -- A Modern Perspective, Proc. IEEE, vol. 69, pp. 1380-1419, Nov. 1981.

8. Kay, S.M., Modern Spectral Estimation: Theory and Application, Prentice-Hall, Englewood Cliffs NJ, 1988.

9. Pandit, Sudhakar M., and Mehta, Nakul P., Data Dependent Systems Approach to Modal Parameter Identification, Proc. 2nd Int. Modal Ana. Conf., Orlando FL, pp. 35-43, 1984.

10. Patton, K.B., Analysis of Parametric Model Signal Processing Techniques For Signature Analysis, Masters Thesis, Virginia Polytechnic Institute and State University, Mechanical Engineering Dept., June 1988.

11. Cooper, J.E., Comparison of Some Time-domain-system Identification Techniques Using Approximate Data Correlations, Int. J. of Ana. and Exp. Modal Analysis, vol. 4, pp. 51-57, April 1989.

12. Astrom, K.J., and Eykhoff, P., System Identification--A Survey, Automatica, vol. 7, pp.123-162, 1971.

13. Fullekrug, U., Structural-dynamics Identification in the Time Domain: Estimation of Modal Parameters Based on Forced Vibrations, Int. J. of Ana. and Exp. Modal Analysis, vol. 4, pp. 58-67, April 1989.

14. Cooper, J.E., and Wright, J.R., comparison of Some Time Domain System Identification Methods for Free Response Data, Proc. 2nd Int. Symp. on Aeroelasticity and Struct. Dynamics, Aachen, W. Germany, 1985.

15. Pi, Yong Lin, and Mickleborough, N.C., Modal Identification of a Vibrating Structure in the Time Domain, Computers and Structures, vol. 32, pp. 1105-1115, 1989.

16. Ewins, D.J., Modal Testing: Theory and Practice, Research Studies Press Ltd., Letchworth, England, 1986.

17. Ibrahim, S.R., and Mikulcik, E.C., A Method for the Direct Identification of Vibration Parameters from the Free Response, Shock and Vibration Bulletin, vol. 47, pp. 183-198, 1977.

18. Ibrahim, S.R., and Mikulcik, E.C., The Experimental Determination of Vibration Parameters from Time Responses, Shock and Vibration Bulletin, vol. 46. pp. 187-195, 1976.

19. Smith, Kevin E., An Evaluation of a Least-Squares Time-Domain Parameter Identification Method for Free-Response Measurements, Proc. 2nd Int. Modal Analysis Conf., Orlando, FL, pp. 610-615, 1984.

20. Pandit, Sudhakar M., et al., Model Estimation of Lumped Parameter Systems Using Data Dependent System Models, Proc. 4th Int. Modal Analysis Conf., Los Angles, CA, pp. 414-421, 1986.

21. Udea, Tetsuhiko, and Iwasaki, Kazuo, Vibration Analysis in the Time Domain by the Real Eigenvalue Method, AIAA Paper 87-0945, 1987.

22. Cuenod, M., and Sage, A.P., Comparison of Some Methods Used for Process Identification, Automatica, vol. 4, pp. 235-269, 1968.

23. Bendat, J.S., and Piersol, A.G., Random Data: Analysis and Measurement Proceedures, John Wiley and Sonbs, New York, 1986.

24. Press, William H., et al, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, Cambridge, 1988.

25. Oppenheim, Alan V., and Schafer, Ronald W., Digital Signal Processing, Prentice-Hall, Englewood Clifts, New Jersey, 1975.

26. Beyer, William H., Chairman, CRC Standard Mathematical Tables, 28th Edition, CRC Press, Boca Raton, FL, 1987.

27. Han, Man-Cheol, and Wicks, A.L., On the Application of Forsythe Orthogonal Polynomials for Global Modal Parameter Estimation, Proc. 7th Int. Modal Analysis Conf., Los Vegas, NV, pp 625-630, 1989.

28. Blevins, Robert D., Formulas for Natural Frequency and Mode Shape, Van Nostrand Reinhold Co., 1979.

APPENDIX - COMPUTER PROGRAM SOURCE CODE

```
/*                  TIME DOMAIN IRF ESTIMATION PROGRAM

main tdirfest --> Main function to estimate impulse
                  response values and plot IRF.

Written by: Charles Mark Archibald

Required Functions:

vector() --> Creates a float vector
corr()   --> Computes crosscorrelations/autocorrelations
toeplz() --> Solves a Toeplitz set of equations
setmodehires() --> Sets video mode to high resolution
barplot()--> Plots impulse response function

Important Variables:

freq  --> Sample frequency
*x    --> Pointer to array of force function values
*y    --> Pointer to array of response values
*s    --> Pointer to array of source values
*rsy  --> Ptr to response crosscorrelation vector
*rsx  --> Ptr to forc crosscorrelation vector
*h    --> Ptr to estimated impulse response vector

*/

#include <nrutil.h>
#include <math.h>
#include <stdio.h>
#include <graph.h>
#include <string.h>
#include <time.h>

FILE *input,*output ;
char ext1[5]=".dat", ext2[5]=".irf";
char *filein, *fileout, base1[16], base2[16] ;

main()
{
    float     freq,gain1,gain2,gain3,gain4,*x,*y,*s ;
    float     *rsy, *rsx, *h, hmax, hmin, ax, ay, as;
    float     corr();
    void      toeplz();
    int       p,pmin,pmax,q,N ;
    int       ch,nm, i,j,k;
    char      ans, time[10], date[10] ;
    void      barplot();
    struct videoconfig setmodehires(), config;
```

```
    _clearscreen( _GCLEARSCREEN );
    printf("\n\n                                    TIME DOMAIN IRF
ESTIMATION PROGRAM \n");
    printf("\n\n  Input Data Filename (No ext): ");
    scanf( "%s", base1 );
    printf( "\n  Input q: ");
    scanf( "%d", &q );

    strcpy( base2, base1 );
    filein  = strcat( base1,ext1 );
    fileout = strcat( base2,ext2 );

    if( (input = fopen( filein, "r" )) == NULL ){
        printf("\n\n  Could not open datafile...\n\n");
        exit(0);
    }

    fscanf( input, "%f", &freq );             /*   Read Data
*/
    fscanf( input, "%d", &ch );
    fscanf( input, "%f", &gain1 );
    fscanf( input, "%f", &gain2 );
    fscanf( input, "%f", &gain3 );
    fscanf( input, "%f", &gain4 );
    fscanf( input, "%d", &nm );

    printf( "\n   %4d Data Points Available - Input N: ",
            600*nm );
    scanf( "%d", &N );

    x   = vector(1,N);  /* Create Vectors and Matrix    */
    y   = vector(1,N);
    s   = vector(1,N);
    h   = vector(1,q);
    rsy = vector(1,q);
    rsx = vector(1,2*q-1);

    printf("\n\n Reading Data...\n");
    for(i=1;i<=N;i++)       /*   Read Time Series Data  */
        fscanf( input, "%f %f %f", &x[i],&y[i],&s[i] );

    fclose( input );

    printf(" Removing Mean...\n"); /* Remove Mean Value */
    as = ax = ay = 0.0;
    for(i=1;i<=N;i++){
        ax += x[i];
        ay += y[i];
        as += s[i];
    }
    ax /= N;
    ay /= N;
    as /= N;
```

```
    for(i=1;i<=N;i++){
        x[i] -= ax;
        y[i] -= ay;
        s[i] -= as;
    }

    printf(" Filling Correlation Vectors...\n");
    for(k=1;k<=q;k++)  /* Populate Correlation Vectors  */
        rsy[k] = corr( s,y,k,N );

    for(i=1;i<=2*q-1;i++)
        rsx[i] = corr( s,x,i-q,N );
     printf(" Calculating IRF...\n");
    toeplz( rsx,h,rsy,q );       /* Solve for IRF   */

    _clearscreen( _GCLEARSCREEN );
    printf("\n\n     Impulse Response Function: \n");
    printf("   k     h[k] \n");
    hmin = hmax = 0.0;
    for(i=1;i<=q;i++){
        h[i] *= freq;
        if( h[i] > hmax )
            hmax = h[i];
        if( h[i] < hmin )
            hmin = h[i];
        printf("  %2d  %9.4f \n", i, h[i] );
        if( i == 21 || i == 46 )
            ans = getch();
    }
    ans = getch();

    config = setmodehires();                          /*
Plot IRF       */
    barplot( h,hmax,hmin,config,q );
    ans = getch();
    _setvideomode( _DEFAULTMODE );

    printf("\n  Print to Output file? (Y/N): ");
    ans = getch();
    if( ans == 89 || ans == 121 ){
        if( (output = fopen( fileout, "w" )) == NULL ){
            printf("\n\n    Could  not  open  Output
file...\n\n");
            exit(0);
        }
        _strtime(time);
        _strdate(date);
        fprintf( output,"%s
%s\n",base1,date);
        fprintf( output,"Impulse Response Function
%s\n",time);
        fprintf( output," %2d  %f  %d \n",q,1.0/freq,N );
        for(i=1;i<=q;i++)
```

```
            fprintf( output,"  %2d  %9.4f \n", i, h[i] );
        fclose( output );
    }
    _clearscreen( _GCLEARSCREEN );
}


float corr( x,y,k,n )
float x[], y[];
int   k,n;

/*          Calculates crosscorrelation or autocorrelation
function. For autocorrelation, call with x = y.

Written by: Mark Archibald

The Arguments are
             x,y -> Time Series Data
             k     -> Lag for which function is to be
computed
             n    -> Length of vectors x and y
   */

{
        int   i, K;
        float sum = 0.0;
        if( k < 0){
                K = -k;
                for(i=1;i<=n-K;i++)
                        sum += y[i]*x[i+K];
                sum /= (n-K);
                return sum;
        }
        else{
                for(i=1;i<=n-k;i++)
                        sum += x[i]*y[i+k];
                sum /= (n-k);
                return sum;
        }
}

/* These functions from Numerical Recipes in C */

#include <malloc.h>

void nrerror(error_text)
char error_text[];
{
        /*          void exit();    */

        fprintf(stderr,"Numerical  Recipes  run-time
error...\n");
        fprintf(stderr,"%s\n",error_text);
```

```
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}


float *vector(nl,nh)

int nl,nh;
{
        float *v;

        v=(float  *)malloc((unsigned)
(nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");

        return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double  *)malloc((unsigned)
(nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");

        return v-nl;
}



float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float  **)  malloc((unsigned)
(nrh-nrl+1)*sizeof(float*));
        if  (!m)  nrerror("allocation  failure  1  in
matrix()");
        m -= nrl;
```

```
        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned)
(nch-ncl+1)*sizeof(float));




                if (!m[i]) nrerror("allocation failure 2 in
matrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned)
(nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in
dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned)
(nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in
dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i,**m;

        m=(int **)malloc((unsigned)
(nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in
imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int *)malloc((unsigned)
(nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in
imatrix()");
                m[i] -= ncl;
```

```
        }
        return m;
}

float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned)
(oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in
submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++)
m[j]=a[i]+oldcl-newcl;

        return m;
}



void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}



void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;
```

```
        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}



void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}



float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m   =   (float   **)   malloc((unsigned)
(nrow)*sizeof(float*));
        if  (!m)  nrerror("allocation  failure  in
convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;

        return m;
}
```

```
void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}


#    d    e    f    i    n    e       F    R    E    E    R    E    T    U    R    N
{free_vector(h,1,n);free_vector(g,1,n);return;}

void toeplz(r,x,y,n)
float r[],x[],y[];
int n;
{
        int j,k,m,m1,m2;
        float pp,pt1,pt2,qq,qt1,qt2,sd,sgd,sgn,shn,sxn;
        float *g,*h,*vector();
        void nrerror(),free_vector();

        g=vector(1,n);
        h=vector(1,n);
        if (r[n] == 0.0) FREERETURN
        x[1]=y[1]/r[n];
        if (n == 1) FREERETURN
        g[1]=r[n-1]/r[n];
        h[1]=r[n+1]/r[n];
        for (m=1;m<=n;m++) {
                m1=m+1;
                sxn = -y[m1];
                sd = -r[n];
                for (j=1;j<=m;j++) {
                        sxn += r[n+m1-j]*x[j];
                        sd += r[n+m1-j]*g[m-j+1];
                }
                if (sd == 0.0) nrerror("TOEPLZ-1 fails with
singular principal minor");
                x[m1]=sxn/sd;
                for (j=1;j<=m;j++) x[j] -= x[m1]*g[m-j+1];
                if (m1 == n) FREERETURN
                sgn = -r[n-m1];
                shn = -r[n+m1];
                sgd = -r[n];
                for (j=1;j<=m;j++) {
                        sgn += r[n+j-m1]*g[j];
                        shn += r[n+m1-j]*h[j];
                        sgd += r[n+j-m1]*h[m-j+1];
                }
                if (sd == 0.0 || sgd == 0.0)
```

```
nrerror("TOEPLZ-2 singular principal minor");
                g[m1]=sgn/sgd;
                h[m1]=shn/sd;
                k=m;
                m2=(m+1) >> 1;
                pp=g[m1];
                qq=h[m1];
                for (j=1;j<=m2;j++) {
                        pt1=g[j];
                        pt2=g[k];
                        qt1=h[j];
                        qt2=h[k];
                        g[j]=pt1-pp*qt2;
                        g[k]=pt2-pp*qt1;
                        h[j]=qt1-qq*pt2;
                        h[k--]=qt2-qq*pt1;
                }
        }
        nrerror("TOEPLZ - should not arrive here!");
}

#undef FREERETURN



/*   Function setmodehires --> Sets video mode to maximum
resolution
                                    and     returns
videoconfiguration.

        */
struct videoconfig setmodehires( )
{
    short mod;
    struct videoconfig config;

    _getvideoconfig( &config);
    mod = 0;
    if(config.adapter == _CGA)
        mod = _setvideomode(_HRESBW);
    if(config.adapter == _HGC)
        mod = _setvideomode(_HERCMONO);
    if(config.adapter == _EGA)
        mod = _setvideomode(_ERESCOLOR);
    if(config.adapter == _VGA)
        mod = _setvideomode(_VRES16COLOR);
    if(mod == 0){
        printf("\n\n  GRAPHICS  MODE  NOT  SUPPORTED  BY
HARDWARE. \n");
        exit(0);
```

```
    }
    _getvideoconfig( &config );
    return( config );
}

/*      Function barplot --> Plots a bargraph of a float
array.

Written by: Mark Archibald

Arguments are as follows:

          xx     -> float - Pointer to array of values
                            to be plotted  xx[1...n]
          xmax   -> float - Maximum value in xx
          xmin   -> float - Minimum value in xx
          config -> struct videoconfig - Videoconfiguration

          n      -> int   - Number of values of xx to be
                            plotted                       */

void barplot( xx, xmax, xmin, config, n )
float                  xx[], xmax, xmin ;
int                    n ;
struct videoconfig config ;
{

    short x,y, x1,y1, x2,y2, z, w, ws ;
    float s ;
    int   i ;

    /*   Plot Bar Chart        */

    x = config.numxpixels-1;
    y = config.numypixels-1;

    /*   Draw Boundry for Plot    */

    _moveto(0,0);
    _lineto(0,y);
    _lineto(x,y);
    _lineto(x,0);
    _lineto(0,0);
    x1 = .2*x;
    y1 = .9*y;
    _moveto(x1,0);

    _lineto(x1,y1);
    _lineto(x,y1);

    /*    Determine Width of Bars and Zero Location    */

    w  = 0.6*(x-x1)/n;
```

```
    ws = 0.4*(x-x1)/n;
    if( xmin >= 0 ){
        z = y1;
        s = y1/(xmax*1.2);
    }
    else{
        s = y1/(xmax*1.2 - xmin*1.2);
        z = xmax*1.2*s;
    }

    /*        Plot Bars        */

    _moveto( x1,z );
    _lineto( x,z );

    x2 = x1-(w+ws/2);
    for(i=1;i<=n;i++){
         x2 += w+ws;
         y2 = z - xx[i]*s ;
         _rectangle( _GFILLINTERIOR, x2,y2, x2+w,z );
    }
}
```

```
*/        IMPULSE RESPONSE ANALYSIS PROGRAM

main irfest --> Estimates modal parameters from impulse
response function estimates. Uses Pisarenko Harmonic
decomposition.

Written by: Charles Mark ARchibald

Required Functions:

dmatrix()    --> Creates Double Matrix
dvector()    --> Creates Double Vector
ntcov_mat() --> Fills Non-Toeplitz covariance matrix
luphd()      --> Pisarenko Harmonic decomposition using
                 LU decomposition
solvewz()    --> Solves for nat. freq. and Damp ratio
solveAp()    --> Solves for amplitude and phase

Important Variables:

cov    --> Covariance matrix
h      --> Impulse Response Estimates
a      --> Autoregressive coefficients
dt     --> Sample Interval
al_amp -> Amplitudes
om_phs -> Phases
w      --> Natural Frequencies
```

```
z     --> Damping Ratios
Var   --> Minimum Eigenvalues     */

#include <stdio.h>
#include <dcomplex.h>
#include <sig_proc.h>
#include <math.h>
#include <graph.h>
#include <string.h>
#include <nrutil.h>

FILE *irf_in, *zwA_out;
char ext1[5] = ".IRF", ext2[5] = ".zwA";
char *irf_file, *out_file, base1[16], base2[16];

main()
{
    double **cov, *h, *a, dt, *w, *z, *al_amp, *om_phs;
    double ave, *var, **amat, **ccov, Var, *tmp, two_pi;
    int    N, k, k1, k2, p, q, i, j, r, c, scratch;
    char      name[17],   date[10],   time[10],
imp[10],res[10],fun[10];
    char   ans;

    _clearscreen( _GCLEARSCREEN );



    printf("\n\n\n                            ");
    printf("IRF ANALYSIS PROGRAM \n\n");
    printf("Method:           Pisarenko  Harmonic
Decomposition\n");
    printf("                  with non-toeplitz covariance
matrix\n\n");
    printf("Input IRF data file (No Extension): ");
    scanf("%s",base1);
    printf("\nInput Initial and Final Model Order: (k1,k2):
");
    scanf("%d,%d",&k1,&k2);

    if(k2 > k1)
        k = k2+1;
    else{
        k  = k1+1;
        k1 = k2;
        k2 = k-1;
    }

    strcpy(base2,base1);
    irf_file = strcat(base1,ext1);
    out_file = strcat(base2,ext2);
```

```
    if((irf_in = fopen(irf_file, "rt")) == NULL){
        printf("\a\nCould not open file %s!
\n\n",irf_file);
        exit(1);
    }

    fscanf(irf_in, "%s %s",name,date);
    fscanf(irf_in, "%s %s %s %s",imp,res,fun,time);
    fscanf(irf_in, "%d %lf %d",&q,&dt,&N);

    j = (k2-k1)/2 + 1;

    cov     = dmatrix(1,k,1,k);         /*    Create Vectors
and Matrix   */
    ccov    = dmatrix(1,k,1,k);
    tmp      = dvector(1,k);                      /*   Try to
eliminate tmp */
    h       = dvector(1,q);
    var     = dvector(1,j);
    amat    = dmatrix(1,k,1,j);

    for(i=1;i<=q;i++)                         /* Read Data
Values */
        fscanf(irf_in, "%d %lf",&scratch, &h[i]);
    fclose(irf_in);

    ntcov_mat( cov, h, k, q );



    i = 1;                        /* Call phd() for range of
model orders */
    for(p=k1;p<=k2;p+=2){
        for(r=1;r<=k;r++){
            for(c=1;c<=k;c++)
                ccov[r][c] = cov[r][c];
        }
        var[i] = luphd( k, ccov, tmp, p);
        for(r=1;r<=p+1;r++)
            amat[r][i] = tmp[r];
        i++;
    }

    _clearscreen( _GCLEARSCREEN );        /* Output Model
ORder information */
    printf("\n\n\n                                        ");

    printf("MODEL ORDER ESTIMATION \n\n\n");
    printf("        Order                  Variance
Variance ");
    printf("Ratio \n\n");
    printf("         2                     %7.4f     \n",
var[1]);
```

```
    for(i=2;i<=j;i++)
        printf("        %2d                    %7.4f
  %7.4f \n",
                2*i, var[i], var[i-1]/var[i] );

    if(k2 > k1){
        printf("\n\nEnter Model Order: ");
        scanf("%d",&p);
    }
    else{
        p   = k2;
        ans = getch();
    }

    c   = (p+2-k1)/2;
    Var = var[c];                /* Discard Unused Variance
and Coefficents */
    a   = dvector(1,p);
    for(i=1;i<=p;i++)
        a[i] = amat[i+1][c]/amat[1][c];

    free_dvector(var, 1,j);          /* Destroy Vectors and
Matrices */
    free_dmatrix(amat,1,k,1,j);
    free_dmatrix(ccov,1,k,1,k);
    free_dmatrix(cov, 1,k,1,k);



    free_dvector(tmp, 1,k);

    w      = dvector(1,p/2);          /* Create new vectors
*/
    z      = dvector(1,p/2);
    al_amp = dvector(1,p/2);
    om_phs = dvector(1,p/2);

    solvewz( a, p, dt, w, z, al_amp, om_phs );
    solveAp( al_amp, om_phs, h, p, q );

    two_pi = 8.0*atan(1.0);            /* Convert Frequency
and Phase */
    for(i=1;i<=p/2;i++){
        w[i]      /= two_pi;
        om_phs[i] *= 360.0/two_pi;
    }

    _clearscreen( _GCLEARSCREEN );      /*  Output Results
*/

    printf("                                         RESULTS
\n\n");
```

```
    printf("Data File:    ID = %s              Date Recorded
= %s \n\n",
            name, date);
    printf("Processing Model Order = %2d \n", p );
    printf("Length of Data Set = %d   Number of IRF values
= %2d\n",N,q);
    printf("Variance = %f \n\n",Var);
    printf("     Frequency           Damping           A");
    printf("mplitude             Phase \n\n");
    for(i=1;i<=p/2;i++)
        printf("    %10.5f         %10.8f        %10.5f
   %10.6f\n",
                w[i], z[i], al_amp[i], om_phs[i] );

    printf("\n\n\nWrite to Data File? (Y/N): ");
    ans = getch();
    if((ans == 89) || (ans == 121)){
        if((zwA_out = fopen(out_file, "wt")) == NULL){
            printf("\a\nCould  not  open  file  %s!
\n\n",out_file);
            exit(1);
        }

        _strtime(time);
        _strdate(date);
        fprintf(zwA_out,"%s
%s\n",name,time);



        fprintf(zwA_out,"Impulse Response Analysis
      %s\n",date);
        fprintf(zwA_out,"Processing Model Order = %2d \n",
p );
        fprintf(zwA_out,"Length of Data Set = %d   Number
of IRF values = %2d\n",N,q);
        fprintf(zwA_out,"Variance = %f \n\n",Var);
        fprintf(zwA_out,"     Frequency             Damping
      A");
        fprintf(zwA_out,"mplitude            Phase \n\n");
        for(i=1;i<=p/2;i++){
            fprintf(zwA_out,"    %10.5f         %10.8f
 %10.5f",
                    w[i], z[i], al_amp[i] );
            fprintf(zwA_out,"          %10.6f\n", om_phs[i]
);
        }
        fclose(zwA_out);
    }
}
```

```
/*  function  ntcov_mat()  -->  Generates  Non-Toeplitz
Covariance Matrix

Written by: Mark ARchibald

Return Value: Void

Arguments:

        cov[1-k][1-k] --> Covariance Matrix
        x[1-n]        --> Data Vector
        k             --> Matrix Dimension
        n             --> Length of Data Vector     */

void ntcov_mat( double **cov, double *x, int k, int n )
{
    int r,c,i;

    for(r=1;r<=k;r++){
        for(c=1;c<=k;c++)
            cov[r][c] = 0.0;
    }

    for(i=k;i<=n;i++){




        for(r=1;r<=k;r++){
            for(c=1;c<=k;c++)
                cov[r][c] += x[i-r+1]*x[i-c+1];
        }
    }

    for(r=1;r<=k;r++){
        for(c=1;c<=k;c++)
            cov[r][c] /= (n-k);
    }
}


/*  function  luphd() -->  Computes the AR coefficents of a
time series using Pisarenko Harmonic Decomposition.

Return Value: Minimum Eigenvalue (Noise Variance)

Arguments:


        k             --> Diminsion of Covariance Matrix
        cov[1-k][1-k] --> Covariance Matrix
```

```
        e_vec[1-(p+1)] --> Eigenvector corresponding to
                           minimum eigenvalue (AR Coeff.)
        p              --> Processing Model Order

Required Functions:

        dludcmp()    LU Decomposition of a Matrix
        dlubksb()    Solves System of Equations
        dotprod()    Computes dot product of two vectors
                          nrutil.c     Vector Creation
                  */

double luphd( int k, double **cov, double *e_vec, int p   )

{
    double  dotprod( double *y, double *x, int n);
    int     i, j, *indx, its = 60;




    double  l1, l2, *x, *y, **ncov, norm;
    double  check, d, tol = 1.0e-12;

    ncov = dmatrix(1,p+1,1,p+1);
    x    = dvector( 1,p+1 );
    y    = dvector( 1,p+1 );
    indx = ivector( 1,p+1 );

    l1 = 0.0;                            /*  Initial guess for
e_val & e_vec  */
    for(i=1;i<=p+1;i++)
        x[i] = 1.0/sqrt(p+1);

    for(i=1;i<=p+1;i++){                /*     Copy applicable
part of cov     */
        for(j=1;j<=p+1;j++)
            ncov[i][j] = cov[i][j];
    }

    dludcmp( ncov, p+1, indx, &d);
    for(i=1;i<=its;i++){
        for(j=1;j<=p+1;j++)
            y[j] = x[j];
        dlubksb( ncov, p+1, indx, y);
        norm  = sqrt(dotprod( y, y, p+1 ));
        check = 0.0;
        for(j=1;j<=p+1;j++){
            y[j] /= norm;
            check += (x[j]-y[j])*(x[j]-y[j]);
        }
        l2 = 1.0/norm;
        if(sqrt(check) < tol){
```

```
            for(j=1;j<=p+1;j++)
                e_vec[j] = y[j];
            free_dmatrix( ncov, 1,p+1,1,p+1 );
            free_dvector( x, 1,p+1 );
            free_dvector(  y, 1,p+1 );
            free_ivector( indx, 1,p+1 );
            return( l2 );
        }
        for(j=1;j<=p+1;j++)
            x[j] = y[j];
        l1 = l2;
    }

    printf("\nEigensolution  failed  to  converge  in  %2d
iterations\n",its);
    free_dmatrix( ncov, 1,p+1,1,p+1 );
    free_dvector( x, 1,p+1 );
    free_dvector(  y, 1,p+1 );



    free_ivector( indx, 1,p+1 );

    printf("\a");
    return( -1.0 );
}


/*  function solveAp() --> Solves for amplitudes and phases
given
                           alpha and omega.

                           Return Value: Void

                           Arguments:

                           A[1-p/2]  ->  Array  of  alpha
values - Ret. amplitudes
                           w[1-p/2]  ->  Array  of  omega
values - Ret. phases
                           h[1-q]   -> Array of time series
values
                           p        -> Autoregressive Model
order
                           q             -> Number of data
points

                           Required Functions:

                           dludcmp.c  LU  dcomposition  of
matrix
                           dlubksb.c LU backsubstition
```

```
sol'n of eqn system  */

void solveAp( double *A, double *w, double *h, int p, int q
)
{
    double **CS, *GB, **SCCS, e, *d, pi;
    int    i,j,k, *indx;

    CS   = dmatrix(0,q,1,p );
    SCCS = dmatrix( 1,p,1,p );
    GB   = dvector( 1,p );
    indx = ivector( 1,p );

    pi = 4.0*atan( 1.0 );

    for(i=0;i<=q;i++){                              /*   Populate
[C|S] Matrix   */
        for(j=1;j<=p/2;j++){
            e = exp(-i*A[j]);



            CS[i][j]    =  e*cos(i*w[j]);
            CS[i][j+p/2] =  e*sin(i*w[j]);
        }
    }

    for(i=1;i<=p;i++){                          /*   Premultipy by
transpose */
        for(j=1;j<=p;j++){
            SCCS[i][j] = 0.0;
            for(k=0;k<=q;k++)
                SCCS[i][j] += CS[k][i]*CS[k][j];
        }
        GB[i] = 0.0;
        for(k=1;k<=q;k++)
            GB[i] += CS[k][i]*h[k];
    }

    dludcmp( SCCS, p, indx, d );        /*   Solve for G's
and B's   */
    dlubksb( SCCS, p, indx, GB );

    for(i=1;i<=p/2;i++){                            /*   Solve for
Amp and phase  */
        if( GB[i+p/2] == 0.0 )
            w[i] = pi/2.0;
        else
            w[i] = atan( -GB[i]/GB[i+p/2] );
        A[i] = GB[i]*sin(w[i]) + GB[i+p/2]*cos(w[i]);
    }
```

```
    free_dvector( GB,1,p );
    free_ivector( indx,1,p );
    free_dmatrix( CS,0,q,1,p );
    free_dmatrix( SCCS,1,p,1,p );

}


/*   function   solvewz() --> Solves for Natural frequency
and
                                damping ratio given the
autoregressive
                                coefficents.

                                Return value:  void

                                Arguments:

                                a[1-p]       -->    Array  of
autoregressive coefficents



                                p           -->  Autoregressive
Model Order
                                w[1-p/2] -->  Array of circular
frequencies
                                z[1-p/2] -->  Array of damping
ratios
                                dt        -->  Sample period
                                A[1-p/2]  -->  Array  of  alpha
values
                                B[1-p/2]  -->  Array  of  omega
values

                                Required Functions:

                                dzroots.c      Finds  roots  of
polynomial
                                  dlaguer.c  Polishes roots of
polynomial
                                dsort4.c     Sorts four arrays
via one index
                                  dindexx.c  Indexes an array
                                nrutil.c     Utility functions

            */

#include <math.h>
#include <nrutil.h>
#include <cvector.h>

#define TRUE 1
```

```
#define FALSE 0

void solvewz( double *a, int p, double dt, double *w,
double *z,
                double *A, double *B )
{
    fcomplex  *ar, *roots;
    int        i;

    roots = cvector( 1,p );
    ar    = cvector( 0,p );

    ar[p].r = 1.0;
    ar[p].i = 0.0;
    for(i=1;i<=p;i++){
        ar[p-i].r = a[i];
        ar[p-i].i = 0.0;
    }

    dzroots( ar,p,roots,TRUE );




    for(i=1;i<=p/2;i++){
        A[i] = -log( roots[2*i-1].i*roots[2*i-1].i +
                     roots[2*i-1].r*roots[2*i-1].r)/2.0 ;
        B[i] = atan( fabs(roots[2*i-1].i/roots[2*i-1].r) );

        z[i] = sqrt( A[i]*A[i]/(B[i]*B[i] + A[i]*A[i]) );
        w[i] = A[i]/(z[i]*dt);
    }

    if(p>2)
        dsort4( p/2,w,z,A,B );

    free_cvector( roots,1,p);
    free_cvector( ar,0,p);
}

#include <math.h>

#define TINY 1.0e-20;

void dludcmp(double **a, int n,int *indx, double *d)
{
        int i,imax,j,k;
        double big,dum,sum,temp;
        double *vv;

        vv=dvector(1,n);
        /*   *d=1.0;  */
```

```
        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
                        if  ((temp=fabs(a[i][j]))  >  big)
big=temp;
                if (big == 0.0) nrerror("Singular matrix in
routine LUDCMP");
                vv[i]=1.0/big;
        }
        for (j=1;j<=n;j++) {
                for (i=1;i<j;i++) {
                        sum=a[i][j];
                        for  (k=1;k<i;k++)  sum  -=
a[i][k]*a[k][j];
                        a[i][j]=sum;
                }
                big=0.0;
                for (i=j;i<=n;i++) {
                        sum=a[i][j];
                        for (k=1;k<j;k++)
                                sum -= a[i][k]*a[k][j];
                        a[i][j]=sum;



                        if ( (dum=vv[i]*fabs(sum)) >= big)
{
                                big=dum;
                                imax=i;
                        }
                }
                if (j != imax) {
                        for (k=1;k<=n;k++) {
                                dum=a[imax][k];
                                a[imax][k]=a[j][k];
                                a[j][k]=dum;
                        }
                        /*   *d = -(*d);   */
                        vv[imax]=vv[j];
                }
                indx[j]=imax;
                if (a[j][j] == 0.0) a[j][j]=TINY;
                if (j != n) {
                        dum=1.0/(a[j][j]);
                        for  (i=j+1;i<=n;i++)  a[i][j]  *=
dum;
                }
        }
        free_dvector(vv,1,n);
}

#undef TINY
```

```
void dlubksb(double **a, int n, int *indx, double *b)
{
        int i,ii=0,ip,j;
        double sum;

        for (i=1;i<=n;i++) {
                ip=indx[i];
                sum=b[ip];
                b[ip]=b[i];
                if (ii)
                        for  (j=ii;j<=i-1;j++)  sum  -=
a[i][j]*b[j];
                else if (sum) ii=i;
                b[i]=sum;
        }
        for (i=n;i>=1;i--) {
                sum=b[i];
                for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
                b[i]=sum/a[i][i];
        }
}




/*    function dotprod()   --> Computes dot product of two
vectors

                              Return Value: The Dot Product

                              Arguments:

                              y[1-n] --> First Vector
                              x[1-n] --> Second Vector (May
be same vector)
                              n      --> Length of Vector
                */

double dotprod( double *y, double *x, int n)
{
    double norm = 0.0;
    int    i;

    for(i=1;i<=n;i++)
        norm += y[i]*x[i];

    return( norm );
}

#define EPS 2.0e-12
#define MAXM 100
```

```
void  dzroots(fcomplex  *a,int  m,  fcomplex  *roots,  int
polish)

        /*  !!! Must Use Double Fcomplex Definition !!!  */

{
        int jj,j,i;
        fcomplex x,b,c,ad[MAXM];
        void dlaguer();

        for (j=0;j<=m;j++) ad[j]=a[j];
        for (j=m;j>=1;j--) {
                x=Complex(0.0,0.0);
                dlaguer(ad,j,&x,EPS,0);
                if  (fabs(x.i)  <=  (2.0*EPS*fabs(x.r)))
x.i=0.0;
                roots[j]=x;
                b=ad[j];
                for (jj=j-1;jj>=0;jj--) {
                        c=ad[jj];
                        ad[jj]=b;
                        b=Cadd(Cmul(x,b),c);
                }




        }
        if (polish)
                for (j=1;j<=m;j++)
                        dlaguer(a,m,&roots[j],EPS,1);
        for (j=2;j<=m;j++) {
                x=roots[j];
                for (i=j-1;i>=1;i--) {
                        if (roots[i].r <= x.r) break;
                        roots[i+1]=roots[i];
                }
                roots[i+1]=x;
        }
}

#undef EPS
#undef MAXM

#include <math.h>

#define EPSS 6.e-16
#define MAXIT 100

void  dlaguer(fcomplex *a,  int  m,fcomplex  *x,  double  eps,
int polish)
```

```
     /*  !!! Must Use Double Fcomplex Definition !!! */

{
        int j,iter;
        double err,dxold,cdx,abx;
        fcomplex sq,h,gp,gm,g2,g,b,d,dx,f,x1;
        void nrerror();

        dxold=Cabs(*x);
        for (iter=1;iter<=MAXIT;iter++) {
                b=a[m];
                err=Cabs(b);
                d=f=Complex(0.0,0.0);
                abx=Cabs(*x);
                for (j=m-1;j>=0;j--) {
                        f=Cadd(Cmul(*x,f),d);
                        d=Cadd(Cmul(*x,d),b);
                        b=Cadd(Cmul(*x,b),a[j]);
                        err=Cabs(b)+abx*err;
                }
                err *= EPSS;
                if (Cabs(b) <= err) return;
                g=Cdiv(d,b);
                g2=Cmul(g,g);
                h=Csub(g2,RCmul(2.0,Cdiv(f,b)));



                sq=Csqrt(RCmul((float)
(m-1),Csub(RCmul((float) m,h),g2)));
                gp=Cadd(g,sq);
                gm=Csub(g,sq);
                if (Cabs(gp) < Cabs(gm))gp=gm;
                dx=Cdiv(Complex((float) m,0.0),gp);
                x1=Csub(*x,dx);
                if (x->r == x1.r && x->i == x1.i) return;
                *x=x1;
                cdx=Cabs(dx);
                if (iter > 6 && cdx >= dxold) return;
                dxold=cdx;
                if (!polish)
                        if (cdx <= eps*Cabs(*x)) return;
        }
        nrerror("Too many iterations in routine LAGUER");
}

#undef EPSS
#undef MAXIT

#include <malloc.h>
#include <stdio.h>
```

```
void nrerror(error_text)
char error_text[];
{
        fprintf(stderr,"Numerical  Recipes  run-time
error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}


float *vector(nl,nh)
int nl,nh;
{
        float *v;

        v=(float  *)malloc((unsigned)
(nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{




        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");

        return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double  *)malloc((unsigned)
(nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");

        return v-nl;
}


float **matrix(nrl,nrh,ncl,nch)
```

```
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float **) malloc((unsigned)
(nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in
matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned)
(nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in
matrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned)
(nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in
dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned)
(nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in
dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i,**m;

        m=(int **)malloc((unsigned)
```

```
(nrh-nrl+1)*sizeof(int*));
        if  (!m)  nrerror("allocation  failure  1  in
imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int  *)malloc((unsigned)
(nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in
imatrix()");
                m[i] -= ncl;
        }
        return m;
}


float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

        m=(float  **)  malloc((unsigned)
(oldrh-oldrl+1)*sizeof(float*));
        if  (!m)  nrerror("allocation  failure  in
submatrix()");



        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++)
m[j]=a[i]+oldcl-newcl;

        return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
```

```
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}



void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));




}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}



void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```

```
float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned)
(nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in
convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;

        return m;
}


void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}



#include <math.h>


fcomplex Cadd(a,b)
fcomplex a,b;
{       fcomplex c;
        c.r=a.r+b.r;
        c.i=a.i+b.i;
        return c;
}

fcomplex Csub(a,b)
fcomplex a,b;
{       fcomplex c;
        c.r=a.r-b.r;
        c.i=a.i-b.i;
        return c;
}

fcomplex Cmul(a,b)
fcomplex a,b;
```

```
{       fcomplex c;
        c.r=a.r*b.r-a.i*b.i;
        c.i=a.i*b.r+a.r*b.i;
        return c;
}

fcomplex Complex(re,im)
double re,im;
{       fcomplex c;
        c.r=re;
        c.i=im;
        return c;
}

fcomplex Conjg(z)
fcomplex z;
{       fcomplex c;
        c.r=z.r;
        c.i = -z.i;
        return c;
}

fcomplex Cdiv(a,b)
fcomplex a,b;
{       fcomplex c;
        double r,den;
        if (fabs(b.r) >= fabs(b.i)) {
                r=b.i/b.r;
                den=b.r+r*b.i;
                c.r=(a.r+r*a.i)/den;
                c.i=(a.i-r*a.r)/den;
        } else {
                r=b.r/b.i;
                den=b.i+r*b.r;
                c.r=(a.r*r+a.i)/den;
                c.i=(a.i*r-a.r)/den;
        }
        return c;
}

double Cabs(z)
fcomplex z;
{
        double x,y,ans,temp;
        x=fabs(z.r);
        y=fabs(z.i);
        if (x == 0.0)
                ans=y;
        else if (y == 0.0)
```

```
                ans=x;
        else if (x > y) {
                temp=y/x;
                ans=x*sqrt(1.0+temp*temp);
        } else {
                temp=x/y;
                ans=y*sqrt(1.0+temp*temp);
        }
        return ans;
}

fcomplex Csqrt(z)
fcomplex z;
{       fcomplex c;
        double x,y,w,r;
        if ((z.r == 0.0) && (z.i == 0.0)) {
                c.r=0.0;
                c.i=0.0;
                return c;
        } else {
                x=fabs(z.r);
                y=fabs(z.i);
                if (x >= y) {
                        r=y/x;

w=sqrt(x)*sqrt(0.5*(1.0+sqrt(1.0+r*r)));
                } else {
                        r=x/y;

w=sqrt(y)*sqrt(0.5*(r+sqrt(1.0+r*r)));




                }
                if (z.r >= 0.0) {
                        c.r=w;
                        c.i=z.i/(2.0*w);
                } else {
                        c.i=(z.i >= 0) ? w : -w;
                        c.r=z.i/(2.0*c.i);
                }
                return c;
        }
}

fcomplex RCmul(x,a)
double x;
fcomplex a;
{       fcomplex c;
        c.r=x*a.r;
        c.i=x*a.i;
        return c;
```

```
}

void  dsort4(int  n,double  *ra,  double  *rb,  double  *rc,
double *rd)
{
        int j,*iwksp,*ivector();
        double *wksp,*dvector();
        void dindexx(),free_dvector(),free_ivector();

        iwksp=ivector(1,n);
        wksp=dvector(1,n);
        dindexx(n,ra,iwksp);
        for (j=1;j<=n;j++) wksp[j]=ra[j];
        for (j=1;j<=n;j++) ra[j]=wksp[iwksp[j]];
        for (j=1;j<=n;j++) wksp[j]=rb[j];
        for (j=1;j<=n;j++) rb[j]=wksp[iwksp[j]];
        for (j=1;j<=n;j++) wksp[j]=rc[j];
        for (j=1;j<=n;j++) rc[j]=wksp[iwksp[j]];
        free_dvector(wksp,1,n);
        free_ivector(iwksp,1,n);
}


fcomplex *cvector(int nl, int nh)
{
    fcomplex *v;

    v    =    (    f    c    o    m    p    l    e    x
*)malloc((unsigned)(nh-nl+1)*sizeof(fcomplex) );
    if (!v) nrerror("allocation failure in cvector()");




    return v-nl;
}

void free_cvector(fcomplex *v, int nl, int nh)
{
    free((char*) (v+nl));
}

void dindexx(int n, double *arrin, int *indx)
{
        int l,j,ir,indxt,i;
        float q;

        for (j=1;j<=n;j++) indx[j]=j;
        l=(n >> 1) + 1;
        ir=n;
        for (;;) {
                if (l > 1)
```

```
                        q=arrin[(indxt=indx[--l])];
                else {
                        q=arrin[(indxt=indx[ir])];
                        indx[ir]=indx[1];
                        if (--ir == 1) {
                                indx[1]=indxt;
                                return;
                        }
                }
                i=l;
                j=l << 1;
                while (j <= ir) {
                        if  (j  <  ir  &&  arrin[indx[j]]  <
arrin[indx[j+1]]) j++;
                        if (q < arrin[indx[j]]) {
                                indx[i]=indx[j];
                                j += (i=j);
                        }
                        else j=ir+1;
                }
                indx[i]=indxt;
        }
}
```

# VITAE

Charles Mark Archibald was born in Birmingham, Alabama on March 3, 1960. After growing up in Alabama, Mark served for several years in the U. S. Navy. Upon completion of his tour of duty, Mark began work toward a bachelors degree at the University of Alabama. In 1987 he transferred to the University of Alabama in Huntsville, from which he graduated, with honors, in 1988. In the fall of that year, Mark began his graduate studies at Virginia Polytechnic Institute and State University. After completion of his Master of Science degree in Mechanical Engineering, Mark will continue studies toward a doctoral degree.