To the Graduate Council:

I am submitting herewith a thesis written by Thomas F. Tibbals entitled "On the Use of Wavelets as an Alternative to Fourier Analysis for Transient Dynamic Data". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science.

R. Engels

Remi C. Engels, Major Professor

We have read this thesis
and recommend its acceptance:

T R Kurfess

L Montgomery Smith

Accepted for the Council:

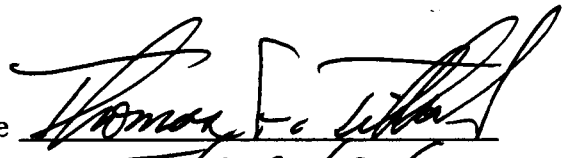
Lew Winkel

Associate Vice Chancellor
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my permission.

Signature 

Date 5/26/94

ON THE USE OF WAVELETS AS AN ALTERNATIVE TO FOURIER ANALYSIS FOR TRANSIENT DYNAMIC DATA

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Thomas F. Tibbals
August 1994

ACKNOWLEDGMENTS

I would like to thank the United States Air Force, the Arnold Engineering Development Center, and Sverdrup Technology, Inc, (AEDC Group) for the opportunity to work in a challenging environment and to attend UTSI as part of my job. This greatly eases the struggle for advanced education while employed, and having tried it before coming to AEDC, I am deeply appreciative. Thanks.

In the course of this thesis research, a number of scholars have assisted me generously. In particular, I would like to thank my committee, Dr. Engels, my major professor, for his continuing encouragement, Dr. Kimble for numerous casual and exciting discussions on wavelets and all the underlying mathematics associated with them which allowed me to grasp the concepts, and Dr. Smith for his critical review of the thesis which significantly improved the presentation of the subject. In addition, I would like to thank Dr. Joseph and Dr. Bomar for excellent instruction in Fourier analysis and digital signal processing, respectively, so even a mechanical type can understand it. Without the support of these individuals and my family, this thesis would not have been possible.

ABSTRACT

Various dynamic data analysis and synthesis tools for one-dimensional time-domain signals are employed to determine the frequency content of a signal for mechanical analysis. When tied to the fundamental frequency of the various components comprising the machinery being evaluated, this information gives an indication of the state or health of the machine. Current techniques for evaluating dynamic data for potential mechanical problems are primarily centered around the Fast Fourier Transform (FFT) and the Short-Time Fourier Transform (STFT).

The use of Fourier analysis for frequency component extraction is restricted to bandlimited stationary signals. Because of this stationary requirement, small transients may not be detected due to a smoothing effect of the FFT, or the FFT spectrum may be smeared due to frequency ramping and abrupt incidents or discontinuities in the signal. Various techniques have been employed to overcome the limitations of the FFT for non-stationary data. These techniques range from windowed FFT (Gabor or Short-Time Fourier Transform), synchronous sampling to remove RPM-ramp effects, Wigner-Ville analysis, and, more recently, wavelet analysis. The primary goals of this thesis are to evaluate and compare various dynamic data analysis and synthesis tools, and to apply them to a few selected signal types.

The research consisted of several phases. First, Fourier analysis is described as it applies to one-dimensional time domain signals. The problems and inadequacies of the FFT analysis are illustrated. With this as a guide, the wavelet alternative to Fourier analysis is described in general, but with particular emphasis on the frequency interpretation of the wavelet analysis. A direct comparison of the wavelet and FFT for stationary data was performed to show that both approaches produce similar results for that regime. The analysis continues with a shift from stationary data to varying degrees of non-stationary data to evaluate the degradation of

the FFT and find limits to its applicability. Finally, the use of wavelets for non-stationary data is illustrated with mechanical engineering examples and compared to the STFT.

The theory, interpretation, and short comings of the FFT analysis were described and limits to the application of the FFT for non-stationary signals were established for the two major non-stationary characteristics of interest to the author, frequency ramps and sudden impulses. It was shown that the FFT, with appropriate bin spacing and oversampling, provides adequate detection and useful information for signals which ramp in frequency, and does so much better than the wavelet approach, even though the time element of the signal is lost in the transformation. In addition, it was shown that the STFT provided superior results for the analysis of a complex signal as compared to the wavelet analysis for the basis functions used. However, it was also shown that the FFT is inadequate to describe a signal containing spikes since the FFT noise floor for a spike cannot be differentiated from the noise floor of a typical real spectrum. The wavelet has the capability to provide the detection of discontinuities in a signal, and therefore is superior to the FFT for such signals, and was shown to be a significant improvement for pulse-echo measurements due to this ability.

TABLE OF CONTENTS

CHAPTER PAGE

INTRODUCTION 1

I. FOURIER ANALYSIS 7
- Theory 7
- Time to Frequency Transformation 14
- Fourier Coefficient Interpretation 15
- Problems/Inadequacies of the Fast Fourier Transform 16

II. TIME-FREQUENCY RESOLUTION 21
- Background 21
- Heisenberg Uncertainty Principle 23
- Short-Time Fourier Transform 25
- Multi-Resolution Analysis 30

III. WAVELET ANALYSIS 33
- Theory 33
- Time to Time-Scale Transformation 46
- Common Wavelet Bases 51

IV. USE OF WAVELETS FOR STATIONARY DATA ANALYSIS . . 56

V. EVALUATION OF THE LIMITS OF FFT APPLICABILITY . . 59
- Frequency Ramp 59
- Sudden Impulses 61

VI. WAVELET APPLICATIONS FOR NON-STATIONARY DATA . . 64
- STFT Comparison 64
- Pulse Echo 65

VII. CONCLUSIONS 69

LIST OF REFERENCES 72

BIBLIOGRAPHY 74

APPENDIXES 77
- APPENDIX A. FIGURES 78
- APPENDIX B. Real-Time Requirements for Transient Spectral Analysis 124
- APPENDIX C. Computer Programs 127

VITA 169

LIST OF FIGURES

FIGURE PAGE

1. SDOF Response Characteristics 79
2. Amplitude-Frequency-Time Space for Fourier Analysis 80
3. Square Wave Approximation as Sum of Sine Waves 81
4. FFT Spectrum for a Two Component Sinusoid 82
5. FFT Spectrum for a Two Component Sinusoid Plus Impulse . . . 83
6. FFT Spectrum for a Ramped Two Component Sinusoid 84
7. FFT Spectrum for a Ramped Two Component Sinusoid with Resonance 85
8. FFT Spectrum of the Rectangular Window 86
9. Interpolation/Smoothing of Impulse via FFT Zero Padding . . . 87
10. STFT Signal Analysis Windows 89
11. Wavelet Signal Analysis Windows 90
12. FFT Spectrum of Rectangularly Windowed Non-Integral Cycle Sinusoids 91
13. Common Windowing Functions and Their Spectra 92
14. STFT Example Signal and FFT Spectrum 94
15. STFT Spectrogram Example 95
16. STFT Spectrogram Example with Improved High Frequency Time Localization 96
17. Wavelet Scaling and Detail Functions 97
18. Haar Wavelet Analysis of a General Digitized Signal 98
19. Error in Reconstruction Following Wavelet Analysis for Signal of Figure 19 100
20. Comparison of Fourier Analysis for Sine Waves and Haar Wavelet Analysis for Square Waves 101
21. Daubechies Wavelets 102
22. DAUB8 Wavelet Applied to Ramped and Impulsed Signals . . . 103
23. Malvar Wavelets 104
24. Real and Imaginary Parts of the Harmonic Wavelet Scaling Function 105
25. Real and Imaginary Parts of the Harmonic Wavelet Detail Function 106
26. Harmonic Wavelet Algorithm via the FFT 107
27. Three Component Sine Wave (Harmonics) and FFT Spectrum . . 108
28. DAUB8 Wavelet Analysis of Figure 27 Signal 109
29. DAUB20 Wavelet Analysis of Figure 27 Signal 110
30. FFT of Level 10 of DAUB20 Wavelet Analysis of Figure 27 Signal 111
31. DAUB20 Wavelet Analysis of Figure 27 Signal with 16 SPC . . . 112
32. FFT of Level 8 of DAUB20 Wavelet Analysis of Figure 27 Signal with 16 SPC 113
33. FFT Analysis of Ramped Sinusoids 114
34. FFT Analysis of Sinusoids Containing Impulses 116

LIST OF FIGURES (Continued)

FIGURE PAGE

35. DAUB4 Wavelet Applied to STFT Example Signal (ref. Figure 14) 118
36. DAUB20 Wavelet Applied to STFT Example Signal (ref. Figure 14) 119
37. Wavelet Analysis of Pulse-Echo Simulated Data 120
38. FFT of Simulated Pulse-Echo Data of Figure 37A 121
39. Haar Wavelet Details Superimposed on Simulated Pulse-Echo Signal 122
40. Wavelet Analysis of Pulse-Echo Simulated Data with Peak NOT Coincident with a 2^j Sample Point 123

INTRODUCTION

The discrete equations of motion for an n-degree-of-freedom structural system are

$$M\ddot{\underline{u}} + C\dot{\underline{u}} + K\underline{u} = \underline{F}$$

where M, C, and K are the mass, damping, and stiffness matrix, respectively. The vector $\underline{F}$ is the forcing vector, and is generally a function of time t.

Substituting the modal transformation

$$\underline{u} = \Phi\underline{q}$$

into the above equation yields a set of modal equations of motion

$$\begin{bmatrix} \ddots & & \\ & m_i & \\ & & \ddots \end{bmatrix} \ddot{\underline{q}} + \Phi^T C \Phi \dot{\underline{q}} + \begin{bmatrix} \ddots & & \\ & k_i & \\ & & \ddots \end{bmatrix} \underline{q} = \underline{P}$$

where

$$\Phi^T M \Phi = \begin{bmatrix} \ddots & & \\ & m_i & \\ & & \ddots \end{bmatrix}, \quad \Phi^T K \Phi = \begin{bmatrix} \ddots & & \\ & k_i & \\ & & \ddots \end{bmatrix}, \quad \textit{and} \quad \Phi^T \underline{F} = \underline{P} \ .$$

The vector $\underline{q}$ is the modal displacement vector. The matrix Φ is called the modal matrix, and its columns are the mode shapes of the structure. The quantities m_i and k_i $(i = 1, 2, \ldots, n)$ are known as the modal masses and stiffnesses, respectively. The vector $\underline{P}$ is the modal force function. The matrix $\Phi^T C \Phi$ is the modal damping matrix which can often be replaced by

$$\Phi^T C \Phi = \begin{bmatrix} \ddots & & \\ & c_i & \\ & & \ddots \end{bmatrix}$$

where the c_i are the modal damping coefficients. Substituting this into the modal equations of motion yields a set of decoupled second order ordinary differential equations with constant coefficients,

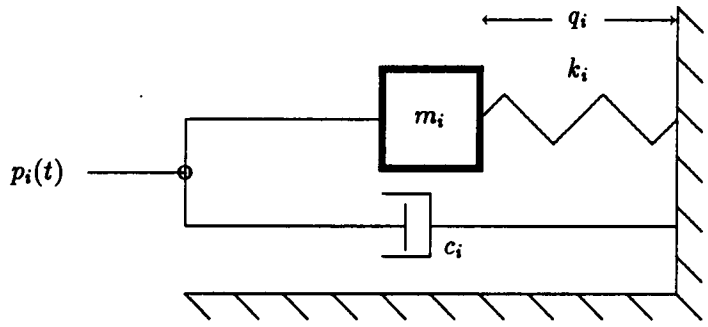
$$m_i \ddot{q}_i + c_i \dot{q}_i + k_i q_i = P_i \ , \qquad i = 1, 2, \ldots, n \ .$$

Dividing through by m_i and letting $\frac{c_i}{m_i} = 2\zeta_i\omega_i$, $\frac{k_i}{m_i} = \omega_i^2$, $\frac{P_i}{m_i} = p_i$, $i = 1, 2, \ldots, n$ finally yields

$$\ddot{q}_i + 2\zeta_i\omega_i\dot{q}_i + \omega_i^2 q_i = p_i \ , \quad i = 1, 2, \ldots, n$$

where ζ_i, ω_i, and p_i are the modal damping factors, natural frequencies, and modal forcing functions, respectively, for the structure.

Each one of the above differential equations represents the equation of motion of a damped linear oscillator as typically depicted in the schematic below.



SIMPLE SDOF MECHANICAL SYSTEM

Therefore, the motion of the structure represented in discrete space by the first equation is equivalent to a set of n decoupled linear oscillators in a modal space. Thus, finding the time history $q(t)$ for a typical damped linear oscillator

$$\ddot{q} + 2\zeta\omega\dot{q} + \omega^2 q = p$$

becomes imperative.

For the underdamped case, this solution is

$$q = e^{-\zeta\omega t}\left(C_1 cos\omega_d t + C_2 sin\omega_d t\right) + q_p$$

where $\omega_n = \sqrt{\frac{k}{m}}$, $\zeta = \frac{c}{2\sqrt{km}}$, $\omega_d = \omega_n\sqrt{1-\zeta^2}$ are the natural frequency, the damping factor, and the damped natural frequency, respectively, of the linear oscillator. q_p is a particular solution depending upon the forcing function $p(t)$. The constants C_1 and C_2 are integration constants determined by the initial conditions. Notice that

the system , once started, will oscillate *continuously* in time with decaying amplitude (see Figure 1[1]). The denominator of ζ is the critical damping factor, i.e., $c_c = 2\sqrt{km}$. If $c = c_c$ and the mass is displaced from equilibrium and released, the mass will return to its equilibrium position without oscillation. With $\zeta = 0$ and again with the mass displaced from equilibrium, no damping results and the mass oscillates perpetually about the equilibrium point with $\omega_d = \omega_n$. All *real* systems, however, have some damping, and the evaluation of the damped response is critical to the design of reliable machinery.

Obviously, frequency (oscillations per time interval) plays an important role in determining how a mechanical system will respond to a vibratory input as well as in determining the cause of the unwanted vibration. Consequently, a frequency analysis is an essential element of mechanical system testing, and can be readily related to the physical characteristics of the system. Therefore, the intent of this thesis is to assess techniques for frequency spectrum estimations available to the mechanical analyst, and to provide an interpretation of the techniques in terms of the basic quantities of time and frequency inherent in the physical model of the mechanical system.

Current techniques for evaluating the frequency content of dynamic data for potential mechanical problems are primarily centered around the discrete Fourier transform which gained popularity due to the work of Cooley and Tukey (1965) who developed a fast algorithm for computing the complex discrete Fourier series (or Discrete Fourier Transform (DFT)), known as the Fast Fourier Transform (FFT). FFT analysis is appropriate for stationary[2] periodic signals, and the DFT can be

[1] Figures appear in the Appendix A.

[2] *stationary* and *non-stationary* are clearly defined in Chapter I.

used to obtain *exact* amplitude values for each center frequency within the fixed frequency spacing of the FFT since it is an *exact* function estimator for **bandlimited** signals where the Fourier series summations are then of finite length, i.e.,

$$x(n) = \sum_{k=0}^{N-1} C_k e^{2\pi i k \frac{n}{N}} = \sum_{k=0}^{N-1} C_k e^{i\omega_k n} \qquad where \;\; \omega_k = \frac{2\pi k}{N} \;\; = k\omega_o = 2\pi k f_o.$$

where $x(n)$ are the given signal sample values, N is the total number of samples used for approximating the signal for the interval of time of interest, C_k are the amplitude coefficients of the approximating sinusoids $e^{2\pi i k \frac{n}{N}}$, and ω_k are the discrete frequencies used in the approximation from DC ($k = 0$) to the maximum frequency based upon the sample rate (number of samples N in time interval T).

The forcing function $p(t)$ is an important parameter for determining the appropriateness of the analysis tool used to determine frequency characteristics of the signal. If $p(t)$ is periodic, then, as will be shown later, the Fourier Series approach is usually appropriate. However, if the signal is entirely random, then the Fourier Integral approach must be considered instead since the *period* is now infinite [ref. 1]. Finally, if the forcing function $p(t)$ is an impulse or has a well defined beginning and ending and the response under investigation also has a well defined beginning and ending, then which analysis tool is appropriate for this transient phenomenon is more difficult to determine. This is the basic question of this thesis.

With the recent progress in computers and digital signal processors, FFT analysis followed by appropriate intra-bin interpolation techniques to improve amplitude resolution at frequencies of interest is an adequate analysis tool for bandlimited broad band stationary signals. However, for non-stationary signals, the FFT may not be appropriate

since, while a signal may be bandlimited[3] (such as due to a ramp in RPM between two distinct finite values), the Fourier transform of non-stationary signals is smeared continuously within the band (see Chapter I, Problems/Inadequacies of the Fast Fourier Transform) and offers no information about the signal. Consequently, there exists a need to assess at what point the FFT/DFT approach breaks down and what form of analysis should be utilized when the FFT/DFT analysis is no longer appropriate. With the vast interest in wavelets, both as a spectral tool in themselves and as a unifying theory for the function approximation area of mathematics, it is time to re-evaluate the analysis and synthesis of time-domain signals for feature extraction pertinent to dynamic (transient, non-stationary) data analysis of mechanical systems.

In the following chapters, Fourier analysis will be presented as an approximation theory with a particular choice of basis functions which makes certain assumptions about the signal to be approximated. It will be shown that these basis functions limit the applicability of Fourier analysis to periodic stationary signals. A discussion on Heisenberg's Uncertainty Principle will set the constraints for all time-frequency analyses, leading up to the requirement for a multi-resolution analysis to adequately determine the characteristics of a non-stationary, non-periodic signal. Wavelet theory will then be presented which parallels that presented for Fourier analysis, and highlights some of the interpretations necessary to preserve the common notion of frequency for non-stationary signal analysis. Common wavelet bases will be dis-

[3] Every *real* signal is actually bandlimited due to the measurement apparatus itself (i.e., limited bandwidth response of signal conditioning equipment). However, what is meant here is that the measurement apparatus *must* have a bandwidth orders of magnitude broader than the transient response of interest since the transient appears to an FFT as having a *full* spectrum. Of course, if such broad sampling was performed, then the FFT bin spacing would broaden such that unsuitable frequency resolution would be obtained. Daubechies [ref. 2] states that a signal is band limited if its Fourier transform has compact support, i.e., the Fourier transform $\hat{x}(\omega)$ is zero outside a finite range of frequencies.

cussed in relation to how they may be used for the class of problems of interest, i.e., machine spectral analysis for ramps and sudden impulses, and pulse-echo data analysis. Finally, conclusions will be stated about the applicability of wavelet analysis as an alternative to Fourier analysis for this class of problems. It should be pointed out that the interest is in fast spectral analysis algorithms for real-time machinery health monitoring purposes, which excludes the use of computationally intensive algorithms or post processing to obtain the signal characteristics necessary for real-time monitoring purposes.

CHAPTER I

FOURIER ANALYSIS

Theory

Fourier analysis is the process of evaluating the frequency content of a signal by approximating the signal as a sum of pure (single frequency) sine waves with discrete frequencies. This is the Fourier series expansion of the signal. On the other hand, Fourier synthesis is the process of generating a complex signal as a sum of pure sinusoids (or tones in acoustics). Fourier analysis was originally developed by the French physicist Jean-Baptiste-Joseph Fourier (1768-1830) for the solution of heat transfer partial differential equations, but the approach is better known for its frequency decomposition of a measured signal.

In addition to the signal being periodic (i.e., $x(t) = x(t+T)\ \forall t$, where T is the period), the signal must meet the Dirichlet conditions for the Fourier series [4] to exist (i.e., the series converges to the continuous signal $x(t)$). The Dirichlet conditions are:

- If the function has discontinuities, their number must be finite in any period.
- The function must contain a finite number of maxima and minima during any period.
- The function must be absolutely integrable in any period, i.e.,

$$\int_0^T |x(t)|dt < \infty$$

where $x(t)$ is a continuous ($\forall t$, at least piecewise) describing function which approximates the actual (real) signal.

[4] Note that if these conditions hold as $T \to \infty$, i.e., the signal is *aperiodic*, then the signal's Fourier Integral Transform exists and the signal can be approximated by a sum of pure sinusoids which vary *continuously* in frequency. In this case the spectrum is continuous instead of discrete as in the Fourier Series representation.

Meeting these conditions implies various limitations to a signal if a Fourier representation is to be valid. First, since $x_i(t)$ must exist for all time, the measurement of a real signal for analysis immediately assumes that the values obtained from the measurement of the signal represent just one piece of an ensemble of many such possible measurements which collectively define a random process $\{x_i(t)\}$ describing the phenomenon under investigation. Here *random* implies that the future time history record from an experiment cannot be predicted within reasonable experimental error from data obtained from an earlier experiment, i.e., the data being analyzed *must* be considered as only one physical realization $(x_i(t))$ of what might have occurred from an infinite number of possibile measurements which make up the ensemble $\{x_i(t)\}$ [ref. 1].

Consequently, to make use of such a small piece of a continuous function describing the phenomenon, the signal must possess some statistical properties. One such property is that for any specific time t_1 various averages exist across all possible time history records $x_i(t)$ for the ensemble $\{x_i(t)\}$, such as the mean value

$$\mu_x(t_1) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i(t_1) \ ,$$

or the mean squared value

$$\psi_x^2(t_1) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i^2(t_1) \ .$$

Based upon these averages, one can define a *stationary* signal as one where these averages do not change in time. Therefore, static measurements of a signal based upon a *single* time record will be valid if the signal is stationary. In this case,

$$\mu_x = \lim_{T\to\infty} \frac{1}{T} \int_0^T x(t)dt$$

$$\psi_x^2 = \lim_{T\to\infty} \frac{1}{T} \int_0^T x^2(t)dt$$

where $x(t)$ is any arbitrary record from ensemble $\{x(t)\}$. This result follows from the *ergodic* theorem which states that "for stationary data, the properties computed from time averages over individual records of the ensemble will be the same from one record to the next and will equal the corresponding properties computed from an ensemble average over the records at any time t if

$$\frac{1}{T}\int_{-T}^{T} |R_{xx}(\tau) - \mu_x^2|\, d\tau \not\Rightarrow 0 \quad \text{as} \quad T \Rightarrow \infty \text{ ." [ref. 1]}$$

Here $R_{xx}(\tau)$ is the average product of the data at times t_1 and $t_1 + \tau$, called the *autocorrelation* at time delay τ, and is given by

$$R_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_0^T x(t)x(t+\tau)dt$$

While *stationary* means that the statistical moments are independent of time, which are defined by

$$\xi_k = \int_{-\infty}^{\infty} x^k p(x)dx \ ,$$

where $p(x)$ is the probability density function of the ensemble $\{x(t)\}$, *ergodic* requires that a time average be identical to a corresponding ensemble average, in which case the signal is **strictly** stationary. In summary, a signal is **stationary** if its properties are statistically invariant over time, and such a signal is also **ergodic** if the time averages equal the statistical averages. For this thesis, stationary signals will be assumed to also be ergodic, and the non-stationary signals will be assumed to have a random event (spike, frequency change, etc.) which violates the stationarity for a short period of time, i.e., a transient.

From this statistical approach, one can see that the space of measure within which Fourier analysis is applicable has been defined. It is often useful to visualize signals as analogous to vectors in a generalized vector space. The Fourier series is

a tool for resolving power[5] signals into such a generalized vector or function space. Recall that a vector in Cartesian space may be described as the projection of the vector upon a triad of mutually orthogonal unit vectors $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ along the x, y, and z axes, respectively, as

$$\mathbf{A} = A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}}$$

The components A_x, A_y, and A_z can also be expressed as

$$A_x = \mathbf{A} \cdot \hat{\mathbf{i}}, \quad A_y = \mathbf{A} \cdot \hat{\mathbf{i}}, \quad A_z = \mathbf{A} \cdot \hat{\mathbf{i}}$$

where the dot product between two vectors is simply

$$\mathbf{A} \cdot \mathbf{B} = ABcos(\Theta_{AB})$$

where Θ_{AB} is the angle between vectors **A** and **B**.

Now consider the representation of a finite-power signal, $x(t)$, defined on the interval $(t_0, t_0 + T)$ in terms of a set of preselected time functions $\Phi_1(t), ..., \Phi_n(t)$. It is convenient to choose these functions with properties analogous to the orthogonal unit vectors of Cartesian space, i.e., orthogonal such that

$$\int_{t_0}^{t_0+T} \Phi_m(t)\Phi_n^*(t)dt = 0 \quad \forall m \neq n$$

and normalized such that

$$\frac{1}{T}\int_{t_0}^{t_0+T} |\Phi_n(t)\Phi_n^*(t)|dt = 1$$

---

[5] Signals are generally classified as *power signals*, *energy signals*, or neither. A **power signal** is one in which the *average* power (P), defined as (normalized to unit resistance) $P = lim_{T\to\infty} \frac{1}{2T}\int_{-T}^{T} |x(t)|^2 dt$ (*watts*), or as $P = lim_{N\to\infty} \frac{1}{2N+1}\sum_{n=-N}^{N} |x(n)|^2$ digitally, is bounded such that $0 < P < \infty$. An **energy signal** is one in which the *total* energy (E), defined as (again, normalized to unit resistance) $E = lim_{T\to\infty} \int_{-T}^{T} |x(t)|^2 dt$ (*joules* = *watts* $*$ *sec*), or digitally as $P = lim_{N\to\infty} \sum_{n=-N}^{N} |x(n)|^2$ is bounded such that $0 < E < \infty$. Note that since $P = \frac{1}{2T}E$, or digitally, $P = \frac{1}{2N+1}E$, an energy signal has zero power while a power signal may have either finite or infinite energy. A periodic signal is a power signal, while most real aperiodic oscillatory signals are energy signals.

where $\Phi_n^*(t)$ is the complex conjugate of $\Phi_n(t)$. Therefore these functions are orthonormal functions on the interval (t_0, t_0+T), where the orthonormality condition was chosen since the signal, being periodic, is bounded in power though unbounded in energy.

Now consider an approximation $\tilde{x}(t)$ to $x(t)$ based upon a series expansion of the form

$$\tilde{x}(t) = \sum_{n=1}^{N} C_n \Phi_n(t)$$

where N is a given finite positive integer and where the constants C_n are to be chosen such that $\tilde{x}(t)$ represents $x(t)$ as closely as possible. A criterion of measure for this *closeness* of approximation must be chosen based upon the signal's characteristics. In this case, we choose to measure the error in a square-integral sense since our concern is with power signals. We define this error as

$$\epsilon_N = \int_{t_0}^{t_0+T} |x(t) - \tilde{x}(t)|^2 dt \ ,$$

and we wish to find the C_n coefficients to minimize this error. Substituting for $\tilde{x}(t)$ yields

$$\epsilon_N = \int_{t_0}^{t_0+T} \left[x(t) - \sum_{n=1}^{N} C_n \Phi_n(t) \right] \left[x^*(t) - \sum_{n=1}^{N} C_n^* \Phi_n^*(t) \right] dt$$

$$\epsilon_N = \int_{t_0}^{t_0+T} |x(t)|^2 dt - \sum_{n=1}^{N} \left[C_n^* \int_{t_0}^{t_0+T} x(t)\Phi_n^*(t)dt + C_n \int_{t_0}^{t_0+T} x^*(t)\Phi_n(t)dt \right] dt + \sum_{n=1}^{N} |C_n|^2 \} \ .$$

Adding and subtracting $\sum_{n=1}^{N} \left| C_n^* \int_{t_0}^{t_0+T} x(t)\Phi_n^*(t) \right|^2 dt$ yields

$$\epsilon_N = \int_{t_0}^{t_0+T} |x(t)|^2 dt - \sum_{n=1}^{N} \left| \int_{t_0}^{t_0+T} x(t)\Phi_n^*(t) \right|^2 dt + \sum_{n=1}^{N} \left| C_n - \int_{t_0}^{t_0+T} x(t)\Phi_n^*(t)dt \right|^2 dt \ .$$

Now since the first two terms above are independent of the C_n coefficients of our approximation, and the last term is non-negative and adds to these two terms, then the only way to minimize the error ϵ_N is to choose the approximation coefficients

C_n such that

$$C_n = \int_{t_0}^{t_0+T} x(t)\Phi_n^*(t)dt \qquad n = 1, 2, ..., N$$

Note that this is an inner product such that $C_n =< x, \Phi_n >$. For special choices of the set of functions $\Phi_1(t), ..., \Phi_N(t)$, referred to as complete sets in the Hilbert function space L^2 of all square-integrable functions, it will be true that

$$\lim_{N\to\infty} \epsilon_{N,min} = \lim_{N\to\infty} \left[\int_{t_0}^{t_0+T} |x(t)|^2 dt - \sum_{n=-M}^{N} |C_n|^2 \right] = 0$$

for any signal that is square-integrable, i.e.,

$$\int_{t_0}^{t_0+T} |x(t)|^2 dt < \infty \ .$$

In the case of zero square-integral error,

$$x(t) = \sum_{n=1}^{\infty} C_n \Phi_n(t)$$

and

$$\frac{1}{T} \int_{t_0}^{t_0+T} |x(t)|^2 dt = \sum_{n=1}^{\infty} |C_n|^2 \ .$$

This is the generalized Parseval theorem which states that the mean power of the periodic signal is the sum of the squares of the approximating coefficients. The exponential functions of the Fourier series form a complete orthogonal set in L^2 such that

$$\Phi_n(t) = e^{in\omega_o t} \ , \qquad n = 0, \pm 1, \pm 2, ..., \infty$$

are axes of L^2 and Fourier analysis projects $x(t)$ onto these axes just as in the case of breaking a vector into its orthogonal components is to project the vector onto the orthogonal axes of Cartesian space. Now $\omega_o = \frac{2\pi}{T}$ where T is the *fundamental* period, that is, T is the **minimum** period for a complete sine wave of the **lowest** frequency of interest. Thus, partial sums of exponential Fourier series minimize the square-integrable error between the series and the signal under investigation [ref 3.].

Since sine waves have a period of 2π, a function $x(t)$ that is square integrable (i.e., piecewise continuous) on the interval $(0, 2\pi)$, i.e.,

$$\frac{1}{2\pi}\int_0^{2\pi} |x(t)|^2 dt < \infty \ ,$$

is an element of the Hilbert function space $L^2(0, 2\pi)$ of all square integrable functions defined on $(0, 2\pi)$, and expansions and contractions of $x(t)$ (i.e., varying the number of cycles on the interval $(0, 2\pi)$) form a basis of the $L^2(0, 2\pi)$ function space. By *basis* is meant a set of linearly-independent functions which can be combined to form another function which also exists in the *same* function space. Then $x(t)$ has a Fourier series representation:

$$x(t) = \sum_{n=-\infty}^{\infty} C_n e^{in\omega_0 t}$$

where, for n=-∞,. . ., -2, -1, 0, 1, 2, . . . ∞,

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} x(t) e^{-in\omega_0 t} dt \ .$$

Notice that the number of basis functions is infinite for this representation.
Per Chui [ref. 4], this series converges in $L^2(0, 2\pi)$ such that

$$\lim_{M,N\to\infty} \frac{1}{2\pi}\int_0^{2\pi} |x(t) - \sum_{n=-M}^{N} C_n e^{in\omega_0 t}|^2 dt = 0 \ .$$

There are three distinct features of $x(t)$ in the Fourier series representation. First, $x(t)$ is decomposed into mutually orthogonal components $C_n e^{in\omega_0 t}$ since the inner product

$$\frac{1}{2\pi}\int_0^{2\pi} C_n e^{in\omega_0 t} C_m e^{-im\omega_0 t} dt = \begin{cases} 0 & m \neq n \\ 1 & m = n \end{cases} \ .$$

Secondly, the bases of this orthonormal representation are all generated from a single function $w(t) = e^{i\omega_0 t}$ by integral dilations, i.e., $w_n(t) = w(nt)$ for all integers n (note that there is no translation parameter, as will be seen later for wavelets, since it makes no sense to translate an infinite extent function - this is handled by

the concept of phase). Finally, the Fourier expansion coefficients are determined via an inner product $c_n =< x, w_n >$. "It is orthogonality that allows us to find each term separately [no cross-terms means no cross-talk or interdependence between frequencies], and it is completeness that allows the sines and cosines to reproduce $x(t)$." [ref. 5]

Time to Frequency Transformation

Any amplitude versus time (or space) signal which meets the conditions discussed above can also be described in the frequency domain by determining the amplitudes and phases of sine waves necessary to be algebraically combined to approximate the original signal. The sine waves then form a basis (i.e., the basic *building blocks*) upon which to build (approximate) the original signal. This is expressed by the general form of the Fourier series

$$x(t) = a_0 + \sum_{n=1}^{\infty}(a_n cosn\omega_0 t + b_n sinn\omega_0 t) = A_0 + \sum_{n=1}^{\infty} A_n cos(n\omega_0 t + \theta_n)$$

where $\omega_0 = 2\pi f_0 = \frac{2\pi}{T}$, $A_0 = a_0$, $A_n = \sqrt{a_n^2 + b_n^2}$, and $\theta_n = tan^{-1}(\frac{b_n}{a_n})$, which is the phase offset. Frequency ω_o determines the fundamental period of the signal $x(t)$ and the coefficients combined with the scaled versions of the fundamental frequency, $n\omega_o$, determine the shape of the signal. Figure 2. [ref. 6] shows the meaning of the above series. The amplitude spectrum of a signal then is merely a plot of the magnitude (A_n) of the sine waves used in the approximation versus the frequency of these pure sine waves. Similarly, the phase spectrum of a signal is merely a plot of the phase (θ_n, i.e., relative location of first peak with respect to time zero) of the sine waves versus the frequency of the pure sine waves.

Fourier Coefficient Interpretation

The a_0 coefficient is the DC (direct current, no oscillation) term and is equal to

the *average* value of $x(t)$ over one period. This is calculated by

$$a_0 = \frac{1}{T}\int_0^T x(t)dt$$

The remaining coefficients, a_n and b_n, are evaluated for n=1,2,3... by

$$a_n = \frac{2}{T}\int_0^T x(t)cosn\omega_0 tdt$$

and

$$b_n = \frac{2}{T}\int_0^T x(t)sinn\omega_0 tdt\ .$$

These coefficients indicate the relative strengths of the sinusoids of frequency $n\omega_0$ as components in the Fourier approximation to $x(t)$. Note that the basis functions (the sine waves) are all derived from a single *mother* sine wave with frequency ω_0 by dilations to frequencies $n\omega_0$ as necessary for the approximation. Also note that the units of these coefficients are the units of $x(t)$ since the time element of integration cancels with the period T. These coefficients will be finite if

$$\frac{1}{T}\int_0^T |x(t)|^2 dt < \infty$$

i.e., finite average power (even though the energy of a periodic signal is infinite). In this case the signal is considered measurable and piecewise continuous.

Recalling that the sine and cosine waves have a phase difference of 90°, it is obvious that any periodic waveform can be approximated in both amplitude and phase by an infinite sum of such fundamental waveforms. However, in practice a finite sum is used where the extent of the sum depends on the accuracy of approximation desired. For example, a square wave theoretically requires an infinite number of sine waves for *exact* representation. However, in practice, the first seven components (odd harmonics 1, 3, 5, 7, 9, 11, and 13) usually suffice (see Figure 3).

Now by utilizing the exponential equivalents of the sine and cosine expressions above, a more compact form of the Fourier series may be obtained as

$$x(t) = \sum_{k=-\infty}^{\infty} C_k e^{ik\omega_0 t}$$

where, for $k = -\infty, \ldots, -2, -1, 0, 1, 2, \ldots \infty$,

$$C_k = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-ik\omega_0 t} dt = a_k - jb_k$$

which is the same as that in the Theory section of this chapter. For a sampled (digital) signal $x_s(n\Delta t)$ as an approximation to $x(t)$ (where Δt = sample interval and $T = N\Delta t$ = analysis time block), we can then approximate c_k as

$$\begin{aligned} C_k &\cong \frac{1}{T} \sum_{n=0}^{N-1} \left(x_s(n\Delta t) e^{-ik\omega_0 n\Delta t} \right) \frac{T}{N} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} x_s(n\Delta t) e^{-ik\omega_0 \frac{n}{N} T} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-ik\omega_0 \frac{n}{N} T} \end{aligned}$$

where $t \rightarrow n\Delta t$ and $\Delta t \rightarrow \frac{T}{N}$, and N is the number of samples (measurement values) of the signal within a given localized time interval of the sampling window.

Problems/Inadequacies of the Fast Fourier Transform

As discussed in the Theory section above, the FFT is valid only for periodic stationary signals. What this means is that by merely applying the FFT, the results assume periodicity and constant frequency content within the sample interval. These assumptions result in the FFT being very localized in frequency, but has no localization in time since the periodic sine wave basis functions have infinite extent. This is not to say that an FFT of a non-stationary signal cannot be performed and the signal reconstructed by an inverse FFT (IFFT), it merely means that the interpretation of the spectrum which results is extremely difficult and suspect physically. For instance, suppose a signal is composed of two distinct frequencies ($x(t) = A(sin\, 2\pi f_1 t + sin\, 2\pi f_2 t)$). The FFT can easily determine the frequency content (see Figure 4) if the sampling meets the Nyquist criterion ($\geq$ two samples per cycle) and the frequency separation is greater than $\frac{1}{N\Delta t}$. However, if an impulse (an event

very localized in time) occurs within the sample interval (as might occur if a stone or a piece of hard ice was ingested by a turbine engine and it hits an instrumented fan blade), then the spectrum merely increases the noise floor to indicate the presence of something other than the basic frequencies of the signal (see Figure 5). This is due to the fact that the sine wave basis functions of the FFT, being of infinite extent, cannot adapt to the transient spike, and no time element remains in the FFT to suggest a spike occurred. This increase in the FFT noise floor is due to the FFT spreading out the spike's energy over the entire spectrum. Recall that it takes an infinite number of sine waves to approximate a square wave and that the Gibbs phenomena results in overshoot at the the points of discontinuity which will *never* disappear [ref. 6 and see Figure 3]. "In the extreme case, the Fourier transform of the delta function $\delta(t-t_o)$ with support at a single point t_o, is $e^{-it_o\omega}$, which certainly covers the whole frequency domain"[ref. 4]. However, since the spike is distributed equally across the whole FFT, the effect is very minor even for a substantial spike. Distributing the spike over more samples (in effect making the transient event longer such that it is less localized in time, like mushy ice or a resilient bug instead of a stone) helps minimally. In this case the FFT is not a suitable tool to approximate the signal and determine its basic characteristics.

A similar problem occurs if the frequency content changes over the sample interval as occurs when the turbine engine is ramping its RPM. Consider a signal composed of the same two frequencies initially, and then allow the frequencies to change linearly within the sample interval ($x(t) = A(sin\, 2\pi f_1 t^2 + sin\, 2\pi f_2 t^2)$, see Figure 6). In this case, the spectrum smears within the bands of frequencies associated with the ramp of RPM, assuming sampling was such as to avoid aliasing (note that the spectrum amplitude goes to zero long before the end of the spectrum). Now if the blade being monitored had passed through a resonant frequency during the

ramp in RPM, this would be undetectable within the smeared spectrum (Figure 7). To adequately handle this case, a time-localized analysis should be performed. The short-time Fourier analysis provides this capability, but requires a radical increase in sample rate (and therefore computation) to incrementally *stop* the ramp much like the concept of slow motion photography. The Short-Time Fourier Transform (STFT) will be described in Chapter II while the *limits of applicability* for the FFT for non-stationary signals will be addressed in Chapter V. For the spike, however, the short-time Fourier transform technique is still not sufficient.

Now the above inadequacies have nothing to do with the sampling or windowing of the signal. The above examples were constructed, based upon the Nyquist criteria, such that the sampled values were zero at the window (sample interval) boundaries to ensure that an integral number of periods were within the sample interval, i.e., a rectangular window was applied to the data synchronized with the fundamental period. Things get even worse if such is not the case, but the intent here is to show the problems under *ideal* conditions. However, any broadening of the main lobes in the frequency domain plots is a consequence of this windowing. The rectangular window has a Fourier transform which is a sinc function (see Figure 8), and, since a multiplication of the window and signal in the time domain results in the convolution of the window spectrum and the signal spectrum in the frequency domain, the window effects become noticeable if the signal frequency is not centered in a FFT frequency bin.

It should also be pointed out that since these signals are synthetic, spikes or other short lived phenomena were synchronized with the sampling to ensure the event was captured. In a real situation, this may not be the case depending upon sampling rates and the phasing of the event relative to the sampling. Therefore, the phenomenon could easily be missed for a real signal and no analysis technique

can bring out information about a signal which is not contained in its samples. Consequently, synthetic signals were used as examples to ensure the point could be made. In addition, while the IFFT will put the spike sample back where it belongs in the time domain, it is not really a *spike* to the IFFT, but merely another sample and the FFT assumes this sample is for a *smooth* transition in the signal with respect to its neighboring samples. This can be illustrated by looking at how the FFT/IFFT interpolates points between the original samples via frequency domain zero padding followed by the IFFT which results in an effective increase in sample rate *as seen by the FFT* (see Figure 9, compare with Figure 5).

These inadequacies result because the basis functions used in the analysis, or approximation, cannot adapt to the transients. Recall that a transient is an event that quickly appears and then disappears, i.e., is compactly supported, or well localized, on the time-line, cannot be statistically predicted, and is therefore **not** periodic within the sample interval. Consequently, these transients are described by functions which do **not** exist in the function space $L^2(0, 2\pi)$, but rather exist in $L^2(\Re)$, and the periodic sinusoids do **not** belong to $L^2(\Re)$. Therefore, it is necessary to develop an analysis approach where the basis functions do not exist for all time such that they can readily adapt to the signal without smearing the fundamental information content or characteristics of the signal. Basically, we need pieces of waves (*wavelets*) we can turn on and off in the vicinity of a transient for a better and more economical approximation. The wavelet approach provides this adaptability, but at the expense of complicating the interpretation of the concept of *frequency*. In essence, *frequency* is defined by Fourier analysis, and is only meaningful for a sinusoid, i.e., a periodic well-defined smooth oscillatory function. However, a mechanical engineer utilizes *frequency* as a count of the stress cycles the component is undergoing which may lead to high cycle fatigue and as a count of overshoot/undershoot cycles

as a structure settles in its response to some excitation. In this context, it does not matter if the cycle is smooth or continuous, only that the stress is varied between minimum and maximum values in a given time interval and that the response settles out after a given number of oscillations. With this broader concept of *frequency* in mind, the interpretation for wavelets discussed later will be made easier.

CHAPTER II

TIME-FREQUENCY RESOLUTION

Background

Before considering wavelets, let us digress and discuss the concepts of time-frequency resolution and localization to make vivid exactly what we are working towards. For a given signal $x(t)$, when one digitally processes the signal to determine its characteristics such as frequency content, it is necessary to examine the signal *locally* in time, e.g., take a *block* of time samples and process it through a N-point FFT, i.e., for period t

$$\widehat{X}(k\omega_o) = \frac{1}{N}\sum_{n=0}^{N-1} x_s(n\Delta t)e^{-ik\omega_0 \frac{n}{N}T}$$

Thus a window is applied in the time domain such that

$$W(t) = g(t - Nt_o)$$

where Nt_o is the window time-width and the window slides along the time-axis. Unless another specific window is applied after the block of data has been obtained, the blocking of data results in this window being rectangular, i.e.,

$$g(t - Nt_o) = \begin{cases} 0 & |t| \geq Nt_o \\ 1 & \text{otherwise} \end{cases} .$$

The sampling rate (F_s) and the number of samples (N) in the block determines the frequency bin spacing of the FFT. According to the Nyquist criteria [ref. 3], the bandwidth of the FFT is $[f : 0, \frac{F_s}{2}]$ which is spread over $\frac{N}{2}$ frequency points such that the frequency bin spacing is

$$\frac{F_s}{2} \div \frac{N}{2} = \frac{F_s}{2} \times \frac{2}{N} = \frac{F_s}{N} .$$

Therefore, regardless of the frequency of interest, the frequency bin width is the same (see Figure 10). Consequently, a low frequency component embedded in a

broadband signal can only be resolved, or localized in frequency, to $\frac{F_s}{N}$ the same as a high frequency component. Therefore, the relative resolution $\frac{\Delta f}{f}$ is not a constant, and while the high frequency resolution may be adequate, the low frequency resolution typically is not. It would be desirable to have the $\frac{\Delta f}{f}$ the **same** for all frequencies which is called a constant-Q analysis (see Figure 11).

Now typically, the data block length N, i.e., the sampling or analyzing window length, are all the same and this window is merely translated to the proper time location (sometimes overlapping the window with the previous one) and the processing is continued. Therefore, the window, or signal envelope, contains varying numbers of oscillations of the approximating sinusoidal components depending upon the frequency of the sinusoids. Consequently, high frequency components have the same time localization as the low frequency components. The lowest frequency may only have <u>one</u> complete cycle in the window, which localizes that component well in time, i.e., one can say that the component was centered at $\frac{N\Delta t}{2}$. The high frequency component may have numerous cycles within the window which means that the component has virtually <u>no</u> time localization, i.e., one cannot say where that component occurred within the time window. A single cycle translating in time gives the optimum time localization. If the window is so narrow such that a whole cycle is not contained in the window, then the spectrum (via the FFT) smears in the low frequencies and the whole spectrum is affected (see Figure 12). Therefore, sample rate combined with window width determines the band of frequencies which can be accurately determined using Fourier analysis, and note that to measure the DC-level appropriately using Fourier coefficients, the window would have to be of infinite width to capture its complete *cycle*.

Now if one considers a transient event that occurs within the data window, as discussed earlier, then it is obvious that neither the low frequency components nor

the high frequency components can adequately respond or adjust to the transient since they cover the whole data block while the transient does not. One might ask, why not reduce the window width such that the window is more localized in time and more apt to respond to the transient? This could be done for a high frequency narrow band signal, but otherwise the low frequency components would be sacrificed at the expense of the high frequency components used to approximate the transient. The Heisenberg Uncertainty Principle places bounds on how time localization and frequency localization trade off. The goal of our analysis is to work within these bounds optimally such that a constant-Q analysis can be performed without unduly sacrificing the low frequency resolution while attempting to detect transients which are high frequency (i.e., short time) events.

Heisenberg Uncertainty Principle

Recall above that the high frequency components had better relative frequency localization than the low frequency components, while just the opposite was true for time localization. Heisenberg's Uncertainty Principle [ref. 7] states that such a trade-off is indeed the case and is to be expected due to the mathematics involved in the localization process. The Heisenberg Uncertainty Principle simply states that the product of the time resolution and the frequency resolution is lower bounded, i.e.,

$$\Delta t \Delta f \geq 4\pi \ .$$

The significance of this equation is that

$$\Delta t \Delta f \geq CONSTANT,$$

since the actual value of the constant is dependent upon the normalization and statistics used in the resolution analysis.

Now due to the way frequency is defined, as

$$f = \frac{1}{T} = \frac{\omega}{2\pi} ,$$

where T = the period for one cycle, then it is obvious that frequency and time are coupled (recall, we determine the frequency components from time samples) and one cannot arbitrarily set one without affecting the other. Also, since they are reciprocally related, then to increase one dictates that the other must decrease. The Heisenberg Uncertainty Principle applies to all such relationships that are describable via a Fourier analysis, e.g., from position and momentum of a particle in quantum mechanics to signal energy relationships.

Consider the windowed Fourier transform

$$F[x(t)] = T_x^{win}(\omega, t) = \int_{-\infty}^{\infty} x(t) g(t-\tau) e^{-i\omega t} dt$$

or discretized,

$$F[x(t)] = T_{m,n}^{win}(x) = \int_{-\infty}^{\infty} x(t) g(t - mt_o) e^{-in\omega_o t} dt$$

where $g(t - mt_o)$ is the window function and m ranges over the whole time line in steps of t_o (m may be bounded if the signal is time-limited), and n ranges over all frequencies in steps of ω_o (n may be bounded if the signal is band-limited). The time-width of $g(t)$ can be calculated as

$$\Delta W_t = \Delta t = \frac{\int_{-\infty}^{\infty} t^2 |g(t)|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt} ,$$

while the frequency width of $\hat{g}(\omega)$ can be calculated as

$$\Delta \widehat{W}_\omega = \Delta \omega = \frac{\int_{-\infty}^{\infty} \omega^2 |\hat{g}(\omega)|^2 d\omega}{\int_{-\infty}^{\infty} |\hat{g}(\omega)|^2 d\omega} ,$$

where each is based upon the root-mean-square statistically.

For typical windowing functions, Table 1 shows the time and frequency widths,

TABLE 1			
Window Function	Δt	$\Delta\omega$	$\Delta t \Delta\omega$
Gaussian	$\sqrt{2} = 1.414$	$\sqrt{2}$	2
Rectangular	$\frac{\tau}{\sqrt{3}} = .5774\tau$	∞	∞
Hamming	$.5229\tau$	∞	∞
Hanning	$.5196\tau$	∞	∞

where $[0,\ \tau]$ is the window time interval and the windows are (see Figure 13):

GAUSSIAN:

$$g(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(t-\frac{\tau}{2})^2}{2}}$$

RECTANGULAR:

$$g(t) = \begin{cases} 0 & 0 \le t \le \tau \\ 1 & \text{otherwise} \end{cases}$$

HAMMING:

$$g(t) = .08 + .46\left(1 - cos\frac{2\pi t}{\tau}\right)$$

HANNING:

$$g(t) = .5\left(1 - cos\frac{2\pi t}{\tau}\right)$$

Note that the Gaussian window is the smallest time-frequency window, and is the only one for which the equality of the Uncertainty Principle holds. Also note that the window $g(t)$ centers the analysis about time $\frac{\tau}{2}$ for each case, and that the duration of time (the time window width) is τ which is constant for **all** frequencies $n\omega_o$. Again, recall that our desire is to have a window which is optimal for the frequency of interest. Obviously, the window width must change in the areas of interest, and therefore numerous fixed width windows are required. This is called multi-resolution analysis, and is discussed after the next section.

Short-Time Fourier Transform

The Short Time Fourier Transform (STFT) is a modified Fourier processing technique which attempts to localize the analysis in the time-frequency plane, i.e.,

a frequency versus time analysis similar to that of a musical score where the notes (*frequencies*) are played at specific times. To localize something, it is necessary to establish a beginning and an ending to the event. The STFT windows the data in time and then processes all frequencies with this constant window width. Basically, the STFT is the result of any sliding windowed FFT, however, for the STFT, Dennis Gabor in 1946 chose the Gaussian window, probably because it is the only window for which the Heisenberg Uncertainty Principle *equality* holds, and therefore the frequency and time windows are exact scales of each other. In addition, the Fourier Transform of a Gaussian is still a Gaussian and therefore frequency localization also results. While the Gaussian doesn't have a beginning nor an ending shy of $\pm\infty$, Gabor truncated the Gaussian tails to approximate a beginning and an end to the window. The STFT is also called the Gabor Transform when the Gaussian window is used.

Another characteristic of the STFT is that the window is generally narrow in time with high sample rates and/or overlapping data blocks to improve high frequency time localization. In addition, the windowed samples usually overlap to minimize the window effects on amplitude and to illustrate the signal's time dependence within the window resolution. The STFT (or *Gabor Transform*) of signal $x(t)$ is defined as

$$STFT(t,\omega) = \int_{-\infty}^{\infty} x(\zeta - t)g(\zeta)e^{-in\omega\zeta}d\zeta$$

where the Gaussian window $g(\zeta)$, $\{\zeta : [0,\tau]\}$, centered about $\frac{\tau}{2}$ is defined as

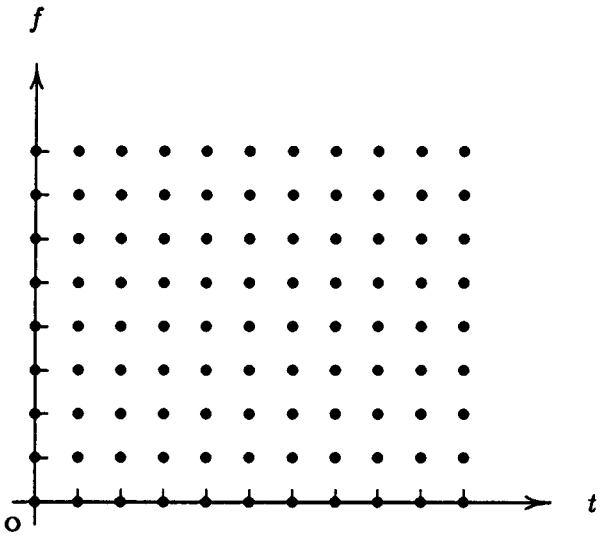
$$g(\zeta) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(\zeta-\frac{\tau}{2})^2}{2\sigma^2}},$$

and σ (standard deviation) is the inflection point of concavity of the curve such that large σ indicates a flat curve and small σ indicates a peaked curve. For comparison purposes with wavelets later, let us define the window $g(\zeta)$ to include the

transformation exponential such that

$$g(\zeta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\zeta - \frac{\tau}{2})^2}{2\sigma^2}} e^{-in\omega\zeta}$$

essentially windows the transform basis functions. Then the STFT is again an inner product $< x, g_{\omega,\tau} >$, and linearity is preserved. Per Chui [ref. 4], as shown in Table 1 (for $\sigma = 1$), the time window width is $\Delta g_\sigma(t) = \sqrt{2}\sigma$ while the frequency (spectral) window width is $\Delta g_\sigma(\omega) = \frac{\sqrt{2}}{\sigma}$, and these window widths are *constant* for the analysis independent of frequency, as shown in the STFT calculation grid below.



STFT CALCULATION GRID

The STFT can be discretized following the same approach as for the Discrete Fourier Transform (see Chapter I, Fourier Coefficient Interpretation). For a sampled (digital) signal $x_s(n\Delta t)$ as an approximation to $x(t)$ (where Δt = sample interval and $T = N\Delta t$ = analysis time block), we can then approximate $c_{k,m}$ for $\zeta = m\Delta t$, $m < N$, as

$$\begin{aligned} c_{k,m} &\cong \sum_{n=0}^{N-1} \left(x_s(m\Delta t - n\Delta t) g(m\Delta t) e^{-ik\omega_0 m\Delta t} \right) \frac{T}{N} \\ &= \frac{T}{N} \sum_{n=0}^{N-1} x_s(m\Delta t - n\Delta t) g(m\Delta t) e^{-ik\omega_0 \frac{m}{N} T} \\ &= \frac{T}{N} \sum_{n=0}^{N-1} x(m-n) g_m e^{-ik\omega_0 \frac{n}{N} T} \end{aligned}$$

where again $t \rightarrow n\Delta t$ and $\Delta t \rightarrow \frac{T}{N}$, N is the number of samples (measurement values) of the signal within a given localized time interval of the sampling window, and M is the number of samples within the Gaussian window $g_m = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(m-\frac{M}{2})^2}{2\sigma^2}} e^{-in\omega m}$. Note that the STFT actually has two windows applied in the time domain. The first window is the blocking of the data itself, just as in the normal DFT. The second window is the Gaussian window which localizes the signal in time by making the signal look as if it has compact support. However, one must remember that this *compact support* is assumed to be **periodic** . In addition, the window must be sufficiently narrow such that the signal is **stationary** within the window so that the Fourier Transform is applicable. Obviously, the STFT will work for a narrowband signal such that an appropriate window can be satisfactorily narrowed without adversely affecting the lower frequencies.

The use of the STFT for gear fault detection via vibrational analysis is described in [ref. 8]. Gear fault detection is an inherently transient analysis since a problem with a gear tooth can only be perceived during the time the tooth is in mesh with other components. The STFT time-frequency output for such an analysis, however, requires some interpretation. Following the example of [ref. 8], consider a signal composed of two sinusoids with stationary properties, another which is amplitude modulated, and two Gaussian shaped impulses as described by

$$x(n) = A_1 \sin\frac{2\pi f_1 n}{N} + \left(1 + A_2 \cos\frac{2\pi f_2 n}{N}\right) A_3 \sin\frac{2\pi f_0 n}{N} + A_4 e^{\frac{B_1^2(n-n_1)^2}{N}} + A_5 e^{\frac{B_2^2(n-n_2)^2}{N}}$$

$$\text{for } 0 \leq n < N-1.$$

The parameters used were: $N = 1024$, $f_o = 10$ Hz, $f_1 = 80$ Hz, $f_2 = 128$ Hz, $A_1 = 1.5$, $A_2 = .75$, $A_3 = 1.0$, $A_4 = 3.0$, $A5 = 3.0$, $B_1 = 64$, $B_2 = 64$, $n_1 = 320$, and $n_2 = 576$ and the sample rate at $2.56F_{max} = 327.68$ Hz. Figure 14 shows the signal and its spectrum. Note that there is no indication of the Gaussian shaped impulses, and the amplitude modulated signal component at 128 Hertz has split into two components centered

around 128 Hertz separated by twice the modulation frequency of $f_o = 10$ Hertz.

Applying the STFT to this signal with a window width of 32 points and no overlap results in the spectrogram of Figure 15. The stationary sinusoids appear as straight horizontal bands in the time-frequency plot, continuous for the whole time interval. Note that the frequency resolution is coarse due to poor frequency localization at the low number of FFT points, while the time resolution adequately shows the amplitude modulation at $f = 128$, but scalloping has smeared the true amplitudes. The Gaussian impulses have lost their time dependence, and only contribute to an amplitude modulation effect for the 10Hz component, since the impulses span more than one FFT window.

Now if the window width is changed to 128 points to provide better frequency localization, then the spectrogram of Figure 16 results. Note that the time resolution has worsened in an attempt to better localize the frequency components. In this case, the Gaussian impulses are visible since the 10Hz peak is sharper due to narrower frequency bin spacing. The low frequency smearing for time slices three and five are the Gaussian impulses at $n = 320$ and $n = 576$, respectively. However, note that the amplitude modulation at 128Hz is no longer discernible, and that it is impossible to determine whether the spikes at 118Hz and 138Hz are real components or a 10Hz amplitude modulation of a 128Hz component. Again, a *constant* window width was utilized for all frequencies. As we shall see with wavelets, the ability to vary the window width with frequency will improve the analysis [ref. 2, pp 3-6]. Also note that by changing window widths in a series of such analyses, we can enhance various signal characteristics with subsequent trade-offs in time-frequency localizations for various components. This series of analyses constitutes a multi-resolution analysis as discussed in the next section.

Multi-Resolution Analysis

Multi-Resolution Analysis is a processing technique that adapts to the frequency range of interest to optimize the resolution in both the time and frequency domains. Generally, while maintaining the constraints of the Heisenberg Uncertainty Principle, this analysis adapts to the frequency such that time resolution becomes arbitrarily good at high frequencies while frequency resolution becomes arbitrarily good at low frequencies. In this thesis, Multi-Resolution Analysis (MRA) is defined simply as the signal processing necessary to obtain arbitrarily good resolution in both the time and frequency domains consistent with the Heisenberg Uncertainty Principle *without regard to a fixed number of time domain samples of the signal*, and sometimes without regard to the orthogonality requirement stated below.

Generally, and mathematically, a multi-resolution analysis consists of an increasing sequence of successive approximation spaces V_j, ranging from coarse to fine, which are closed subspaces within $L^2(\Re)$ and satisfy (following the notation of Daubechies [ref. 2]):

$$\{0\} \rightarrow \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots \rightarrow L^2(\Re) \quad \Rightarrow NESTED$$

$$\overline{\bigcup_{j \in Z} V_j} = L^2(\Re) \quad \Rightarrow CLOSURE$$

$$\bigcap_{j \in Z} V_j = \{0\} \quad \Rightarrow ORTHOGONALITY$$

$$x(t) \in V_j \iff x(at) \in V_{j+1}, \quad a > 1 \quad \Rightarrow MULTI-RESOLUTION.$$

If P_j is an orthogonal projection operator onto V_j, then the closure property ensures that a suitable approximation to $x(t)$ can be ultimately obtained, i.e.,

$$\lim_{j \rightarrow -\infty} P_j x(t) = x(t) \quad \forall x \in V_j.$$

The multi-resolution aspect of these *ladder spaces* is due to the last of the above properties, where all spaces are a *scaled* version of each other such that ultimately

they are scaled versions of a central subspace V_o, i.e.,

$$x(t) \in V_j \iff x(a^j t) \in V_o, \quad a > 1.$$

Here the notion of *scale* is that of cartography in that at a given scale, the signal is replaced by a best approximation that can be *drawn* at that scale. By moving from a coarse scale to a fine scale, one **zooms** in on the details of the signal and hence a more exact representation of the original signal. Obviously, the original sampled signal is all the information that exists and this then becomes the finest resolution in information that is possible, i.e., all further processing can only decompose the signal to those bits of information contained in the signal without creating any new information.

Multi-resolution analysis can be thought of as an approximation sequence based upon successive decimation filters where the finest time resolution is the original (typically oversampled) signal. Subsequent coarser time domain approximations can then be obtained by repeated application of decimate-by-n filtering, where every n samples are removed in the time domain. In the frequency domain, this results in peeling away the higher frequencies with each application of the decimation filter, since the Nyquist criterion must be satisfied. Now to increase low frequency resolution, one simply accumulates more time-domain samples to perform a larger frequency domain transform such that the same N-point FFT results in $\frac{N}{2}$ frequency bins distributed over a much smaller frequency bandwidth due to the filtering and decimation process [Dr.B.W. Bomar, UTSI, private communication]. Consequently, time domain resolution is sacrificed at the expense of frequency domain resolution.

For a multi-component signal, this means that the approximation first satisfies the resolution requirements of the high frequency components. Recall that Nyquist sampling for the higher frequencies means oversampled lower frequency components. However, while this results in the *best* frequency resolution for the highest frequen-

cies, for the lower frequencies this means very poor frequency resolution (though good time resolution). If one decides that a frequency resolution of $q = \frac{\Delta f}{f}$ is appropriate and requires, say, an N-sample FFT, then successive peeling operations via decimation and accumulation will be required to obtain the *best* (Nyquist) frequency resolution for the lowest frequencies, at the expense of time-resolution and a complete loss of the higher frequencies. Consequently, the whole signal is the composition of these approximations or space projections, as described by the closure and nesting properties of the multi-resolution analysis.

Notice that one can also look at multi-resolution analysis as a contraction and expansion (or *dilation*) of the support of the projection/approximation basis functions. This is due to the fact that at high frequency, *best* may be determined to be a N-sampled FFT and therefore the time analysis window is Nt_o, where t_o is the sample time interval. This means that the support of the analysis is much narrower for the higher frequencies than for the lower frequencies accumulated over nNt_o, where a n-decimation was used for the same constant $q = \frac{\Delta f}{f}$. These concepts will be developed further when the wavelet analysis is described.

CHAPTER III

WAVELET ANALYSIS

Theory

It should be pointed out that the bulk of this chapter is merely an expansion of [refs. 2, 4, 10, and 11] to fill in what is considered to be important details of the theory underlying wavelets. Reference 10 is perhaps the best paper for the understanding of wavelets in general and uses the familiar Fourier concepts as a guide, but it is also the least rigorous.

Wavelet theory unifies various signal processing techniques developed independently to overcome the problems of Fourier analysis. For example, multi-resolution signal processing, image compression subband coding, and wavelet series expansions are all considered a single theory mathematically [ref. 10]. Wavelet theory provides a good technique for signal approximation using scaled basis functions, of which Fourier analysis is just one application (where the support is infinite, non-decaying, and the basis functions are sine functions), and allows one to examine the effectiveness of the approximation on deeper mathematical levels systematically. This thesis is primarily concerned with the application of wavelets to non-stationary one-dimensional signals, and therefore will limit its discussion of wavelets to a signal processing point of view. An additional interest is in the use of wavelets to systematically improve the frequency resolution of low frequency components embedded within broadband signals, whether stationary or non-stationary, without unnecessarily burdening the computation of high frequency spectral components. This then will allow real-time low frequency pattern recognition embedded in broadband signals such as required for bearing fault diagnosis based upon engine case accelerometer data [ref. 12].

The Fourier Transform spanned the space $L^2(0, 2\pi)$ and satisfied

$$\int_0^{2\pi} |x(t)|^2 dt < \infty \ .$$

However, for transient signals, there may be no periodicity (at least for the transient component even if embedded in a periodic signal), and therefore we must look for a transform which spans $L^2(\Re)$ and satisfies

$$\int_{-\infty}^{\infty} |x(t)|^2 dt < \infty \ .$$

As stated by Chui [ref. 4, p3],

> Clearly, the two function spaces [for each of these transforms] $L^2(0, 2\pi)$ and $L^2(\Re)$ are quite different. In particular, since (the local average values of) every function in $L^2(\Re)$ must "decay" to zero at $\pm\infty$, the sinusoidal (wave) functions w_n do not belong to $L^2(\Re)$. In fact, if we look for "waves" that generate $L^2(\Re)$, these waves should decay to zero at $\pm\infty$; and for all practical purposes, the decay should be very fast. That is, we look for small waves, or *"wavelets"*, to generate $L^2(\Re)$.

The rapid decay of the *wavelet* provides the **localization** necessary to adapt to local transients in a non-stationary waveform. The fact that we look for a *wave* or oscillating function at all is to somehow preserve the concept of frequency for vibration/harmonic analysis, and we also desire localization in the frequency domain as well.

Now we still want these wavelets to be generated from a single function just as in the Fourier basis functions where

$$w_n(t) = w(nt) = e^{in\omega_o t} \qquad \forall \ integers \ n$$

to ease the analysis. In addition, we still want the functions to be complete in $L^2(\Re)$ and orthogonal so we can be assured that the approximation is appropriate and converges to $x(t)$ in the limit as well as find each coefficient of the approximation separately.

To cover the real line completely with a single function which decays rapidly, i.e., is compactly supported on the real line, it is necessary to shift the function along $\Re$. Consider ψ to be a wavelet basis function, real, and of compact support on $\Re$, then for ψ to span all of $\Re$,

$$\psi_b(x) = \psi(x - b) \qquad x, b \in \Re.$$

This is the translation property of ψ. Now obviously ψ must still be capable of describing different frequencies, since our intent is to perform a frequency analysis of the mechanical phenomena. To accomplish this easily, without the use of sinusoids, we must consider the fundamental definition of frequency as the inverse of the period between zero crossings or the time extent of the compactly supported analysis functions. With this in mind, we consider frequency as embedded in the dilations of the basis function such that

$$\psi_{a,b}(x) \approx \psi(ax - b) \qquad x, a, b \in \Re, \quad a > 0$$

Observe that our development parallels that for Fourier analysis, and seeks a basis function that is obtained from a single function by dilations (ax) to adjust to varying frequencies and translations $(\frac{b}{a})$ to span the real line, i.e.,

$$\psi_{a,b}(x) \approx \psi(ax - b) \approx \psi\left[a\left(x - \frac{b}{a}\right)\right] \qquad x, a, b \in \Re, \quad a > 0,$$

and is different from the Fourier analysis only in the basis functions used, which are **not** periodic and are of compact support.

Now to normalize this basis in $L^2(\Re)$, consider

$$\| \psi(ax - b) \|_2 = \left[\int_{-\infty}^{\infty} |\psi(ax - b)|^2 dx\right]^{\frac{1}{2}} = a^{-\frac{1}{2}} \left[\int_{-\infty}^{\infty} |\psi(x)|^2 dx\right]^{\frac{1}{2}}$$

then

$$\| \psi(ax - b) \|_2 = a^{-\frac{1}{2}} \| \psi(x) \|_2$$

then for $\psi_{a,b}(x)$ to be of unit length,

$$\psi_{a,b}(x) = a^{\frac{1}{2}}\psi(ax - b) \qquad x, a, b \in \Re, \quad a > 0$$

For economy of the analysis, let's restrict ourselves to integral dilations and translations just like we restrict ourselves with the FFT to integral frequencies. Let $\mathcal{Z}$ denote the set of integers, then for $j \in \mathcal{Z}$, $a \rightarrow a_j = a_o^j$, $a_o > 1$, where a_o is the fixed dilation parameter. Now since the width (support) of the analyzing wavelet will be proportional to a_o^j, it follows that we will need to translate in steps proportional to the width of the wavelet basis function to cover the whole real line. Consequently, let $b \rightarrow b_k = kb_o a_o^{-j}$, $k \in \mathcal{Z}$, where b_o is fixed. Then

$$\begin{aligned}\psi_{j,k}(x) &= a_o^{\frac{j}{2}}\psi(a_o^j(x - kb_o a_o^{-j}) \\ &= a_o^{\frac{j}{2}}\psi(a_o^j x - kb_o) \qquad x \in \Re, \quad a > 1, \quad b_o > 0, \quad j, k \in \mathcal{Z}.\end{aligned}$$

Later, b_o will become the sampling rate, however, for now let $b_o = 1$, where it is assumed that the shift k is proportional to the width of ψ.

Notice that each frequency component is not available in this approach, only the individual multiples of a_o times the number of vibrations per interval are described. Consequently, an immediate loss of frequency resolution appears to have occurred in an analysis which we had hoped would improve the lower frequency resolution. This will be addressed later, for now this approach reduces the complexity of the development, and everyone realizes that by taking multiples of something often enough we can get arbitrarily close to the goal even if it is never actually obtained.

It should be pointed out that two distinct notations are found in the literature. Chui [ref. 4] and Y. Meyer [ref. 11] use

$$\psi_{j,k}(x) = a^{\frac{j}{2}}\psi(a^j x - k)$$

and the nested spaces (Multi-Resolution Analysis) as

$$\{0\} \rightarrow \cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \rightarrow L^2(\Re).$$

Whereas Daubechies [ref. 2], A. Grossman, and J. Morlet use the non-standard form

$$\psi_{j,k}(x) = a^{-\frac{j}{2}}\psi(a^{-j}x - k)$$

and the nested spaces (Multi-Resolution Analysis) as

$$\{0\} \rightarrow \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots \rightarrow L^2(\Re).$$

The concepts are identical, and it is merely a choice of how you wish to describe the concept of dilation as either higher j for higher frequency or higher j for wider support on $\Re$ (i.e., coarser resolution). Chui's indexing is appropriate if frequency is the interest, such that higher positive index implies higher frequency and smaller support. This is also appropriate for function approximation where higher j implies higher accuracy (higher number of grid points fit). In this case one goes from coarse scales to finer scales with increasing j. On the other hand, Daubechies' indexing is appropriate (but reversed from normal) when the size of the space is of interest, such that higher negative index implies a smaller space spanned by the basis functions and therefore higher frequency or shorter transients are discernible. Daubechies' approach is more natural for signal processing since the *best* informational resolution is the originally sampled data and we come down from high information content to its lowest primitive as we approximate the signal at coarser and coarser levels with decreasing negative values for j, thereby examining its fundamental frequency components. Daubechies' approach will be followed throughout this thesis.

Since our interest is in the analysis of signals evolving in time, first let us look at what the time description of the wavelet is. Now time is continuous just like the real line. However, one still wants to perform integral dilations and translations utilizing the set of integers $\mathcal{Z}$ which is a subset of $\Re$. Therefore, the wavelets describing a time evolving signal instead of a spatially evolving one would be [in Daubechies'

notation]

$$\psi_{j,k}(t) = a^{-\frac{j}{2}}\psi(a^{-j}t - k\tau) \qquad a > 1, \quad j,k \in \mathcal{Z},$$

and where τ = sample time interval.

In summary, the wavelets are basis functions for the wavelet expansion just as the complex exponentials (sinusoids) are the basis functions for the Fourier expansion. The continuous wavelet expansion of a function (signal) $x(t)$ in $L^2(\Re)$ is the inner product defined as

$$W[x(t)] = (T^{wav}x)(a,\tau) = |a|^{-\frac{1}{2}} \int_{-\infty}^{\infty} x(t)\psi(\frac{t-\tau}{a})dt$$

or discretized [6]

$$W[x(t)] = (T^{wav}_{j,k}x)(a_o,\tau) = |a|^{-\frac{j}{2}} \int_{-\infty}^{\infty} x(t)\psi(a_o^{-j}t - kT)dt \; = < x, \psi_{j,k} >$$

where $a \rightarrow a_o^j$, $\tau \rightarrow ka_o^jT$ and $a_o > 1$ (generally 2), $j,k \in \mathcal{Z}$. The coefficient outside the integral is the normalization factor such that the L^2-norm $\| \psi \|_2 = 1$. The a_o-term is the magnification where j negative and large corresponds to large magnification, and consequently $a_o^jT = \frac{T}{a_o^{|j|}}$ corresponds to small steps and therefore fine details are discernible (i.e., captures high frequency, short transient characteristics).

The major mathematical difference between wavelets and the Fourier basis functions is the requirement that

$$\int_{-\infty}^{\infty} \psi(t)dt \; = \; 0 \; ,$$

which allows the introduction of the dilation (or scale) parameter in order to make the time-frequency window flexible [ref. 4]. However, this also means there is no DC-component in wavelet analyses, but then again, the time-frequency window

[6] Here *discretized* means that only integral shifts in scale and window position are to be considered. However, this is still the **continuous** wavelet transform (CWT) since the integral and $x(t)$ are continuous in time, and this *discretization* is merely for analysis economy.

would have to be infinite to hold the DC-component. What this finally implies is that the final expansion can only reduce the signal to a m-point approximation (for an mth-order wavelet) in time, not a single DC-component.

The wavelet is developed not from just a single mother analyzing function as in the Fourier case, but from a single scaling function which is orthogonal to the analyzing wavelet basis function. In contrast to the STFT which essentially windowed the basis functions (or transforms only) of the expansion, the wavelet truly does window the time domain **and** the Fourier transform directly. This leads naturally to a filtering/decimation process. Consider $\phi(t)$ to be a smoothing function which removes irregularities by averaging the signal locally over some time (or sample) interval. If one performs the inner product of $x(t)$ and $\psi(t)$, then one obtains an approximation of $x(t)$ within the time resolution of $\psi(t)$, i.e., $< x, \psi >$ is a smoothed[7] version of x. The scaling function must also cover the real line and *average* over varying intervals. Consequently, the scaling function has the same form as the wavelet, i.e.,

$$\phi_{j,k}(t) = a^{-\frac{j}{2}}\phi(a^{-j}t - k\tau) \qquad a > 1, \quad j,k \in \mathcal{Z},$$

and where again τ = sample time interval. However, the scaling functions only window, they don't wiggle or oscillate (see Figure 17). The wavelet function extracts the detail from the signal, the difference between the original and smoothed versions, such that the signal approximation is composed of the smoothed version plus the detail removed by the smoothing operation. Therefore, two function spaces are established:

$$\phi \in V \Rightarrow \{0\} \subset \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots \rightarrow L^2(\Re) \quad \Rightarrow \quad SCALE\ SPACE$$

[7] Smoothed in the mathematical sense of less jaggedness (fewer excursions), i.e., a multitude of points is replaced by a single point which is the average of the multiple. Consequently, resolution is **coarser** with increased smoothing.

and

$$\psi \in W \Rightarrow \{0\} \oplus \cdots \oplus W_2 \oplus W_1 \oplus W_0 \oplus W_{-1} \oplus W_{-2} \oplus \cdots \rightarrow L^2(\Re) \quad \Rightarrow \quad DETAIL\ SPACE$$

These spaces are related in that

$$V_{j-1} = V_j \oplus W_j,$$

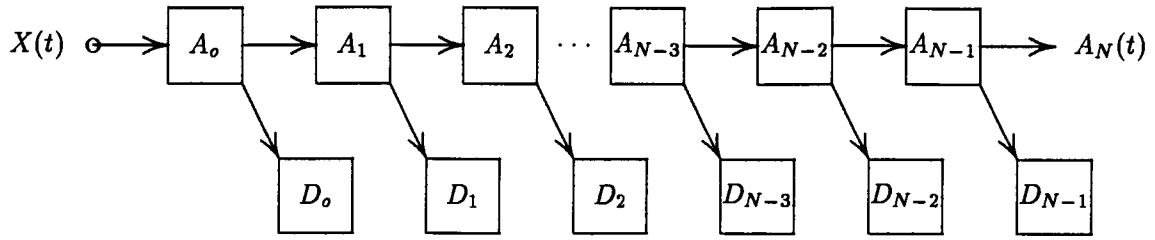
i.e., the next finer level of approximation is generated by the orthogonal sum of the smoothed approximation plus its detail lost due to smoothing at that level. Note that the wavelet spaces W_j inherit the scaling property, i.e.,$x(t) \in V_j \iff x(a^j t) \in V_o$, from V_j, therefore, $x(t) \in W_j \iff x(a^j t) \in W_o$. Also note that W_j is the orthogonal complement of V_j in V_{j-1}, i.e., W_j, the detail, does not exist in V_j, it exists in the next finer level of approximation, V_{j-1}. Consequently, for $j < J$,

$$v_j = V_J \oplus \bigoplus_{k=0}^{J-j-1} W_{J-k} \ , \quad L^2(\Re) = \bigoplus_{k \in Z} W_k \ ,$$

defines a multi-resolution pyramidal decomposition where all of the spaces are orthogonal, i.e.,

$$V_j \perp V_{j'}, \quad W_j \perp W_{j'}, \quad j \neq j' \ \ and \ \ V_j \perp W_j.$$

Note that V_j is decomposed into a coarser approximation plus its lost detail, but **never** do we decompose the detail any further, as shown on the next page. This way the energy of the signal is preserved.



WAVELET PYRAMIDAL DECOMPOSITION

Since the interest in wavelets is for the on-line analysis of a signal which has been digitally sampled, it makes sense to look at wavelets from a filtering (digital signal processing) point of view. Let $a = 2$, then the wavelet will have binary dilations $(2^j n\Delta t)$ and dyadic translations $(\frac{k\Delta t}{2^j})$ where

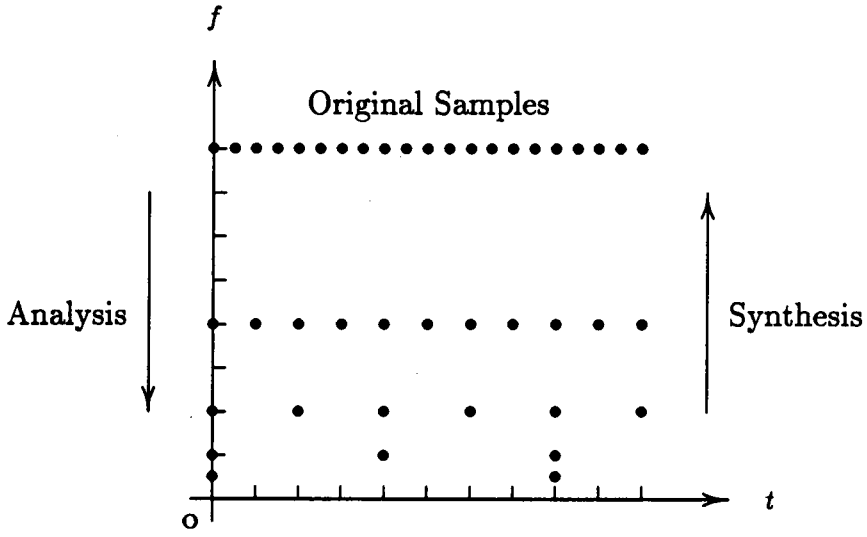
$$\psi_{j,k}(n\Delta t) = 2^{-\frac{j}{2}}\psi(\Delta t(2^{-j}n - k)),$$

or simply,

$$\psi_{j,k}(n) = 2^{-\frac{j}{2}}\psi(2^{-j}n - k).$$

Notice that dilating a signal by a factor of 2 is equivalent to subsampling in the sense that only half the samples are necessary for similar resolution in a given time interval per Nyquist. Also notice that, because of the duality between frequency and time, for a binary dilated signal ($\frac{1}{2}$ the frequency) twice the time interval is required to sample that signal as compared to the original signal, non-dilated version, for a similar level of resolution. Consequently, one can low pass filter, determine the detail lost by subtracting the lowpassed version from the original, and then decimate by a factor of 2 and accumulate more samples (look at longer time intervals) to get any resolution in frequency as desired at the expense of time resolution. This is the MRA described in Chapter II. But now suppose we fixed the number of samples (and therefore the sampling time interval under investigation) and still want to perform an MRA. In addition, we would like to keep the number of coefficients to be determined to be no more than that of the original number of samples (which are the *coefficients* of the highest resolution of the MRA), as opposed to a 50% increase

at each step which results from the above scheme. To accomplish this, the MRA requires the two basis functions described by the wavelet approach. In this case, ϕ performs the lowpass filtering and decimation to coarsen the approximation (smooth the signal), and ψ **simultaneously** performs the highpass filtering at the previous level to preserve the detail (fluctuations) which will be lost going to this coarser level of approximation. The wavelet analysis then processes the data according to the grid as shown on below.

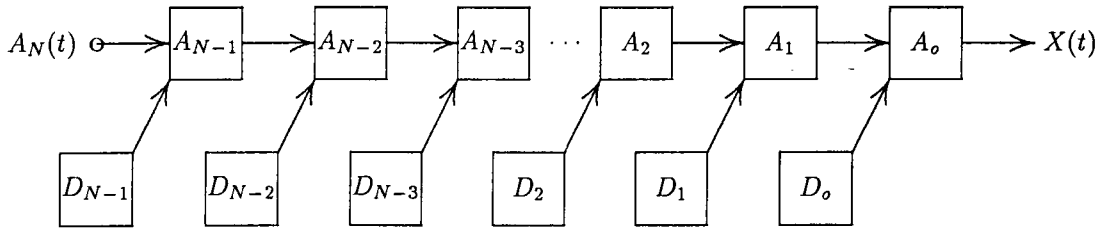


WAVELET CALCULATION GRID

Note again that no DC processing is performed, that the grid is much more sparse than that for the STFT, and finally that not all frequencies are processed, rather, the frequency axis is scaled according to a base 2 logarithm due to the half-band filtering and subsequent decimation by two.

Following the MRA pyramidal algorithm defined above, the original signal becomes smoother with each application of the lowpass filtering followed by decimation until only m-points remain as a linear approximation to the signal for an mth-order wavelet filter, where $m = 2p,\ p \in \mathcal{Z}$, i.e., m is even. As shown below, reconstruction of the signal is possible by zero padding between each point in the coarser approximation, interpolation via the complement of the lowpass filter, and finally adding

back the detail lost at each step via the complement of the highpass (wavelet) filter, **provided the signal energy was preserved.**



WAVELET PYRAMIDAL RECONSTRUCTION

This perfect reconstruction requirement places strong requirements on the filters used for wavelet analysis, which is embodied in the theory of quadrature-mirror filters (QMF) originally developed for subband coding schemes.

Quadrature mirror filters (see schematic on next page) are filters G and H which, for all signals X of finite energy, produce

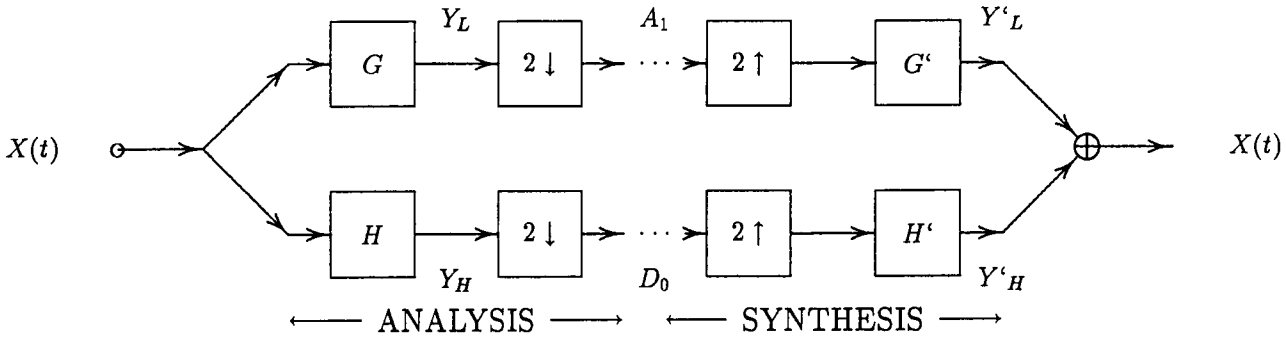
$$\| Y_L \|^2 + \| Y_H \|^2 = \| X \|^2$$

where Y_L is the output from the lowpass filter G and Y_H is the output from the highpass filter H, both of which are operators which map $l^2(\mathcal{Z}) \Rightarrow l^2(2\mathcal{Z})$, where l is the Hilbert space such that $\sum_{n=-\infty}^{\infty} |x(n)|^2 < \infty \quad for\ n \in \mathcal{Z}$. In the case of perfect halfband filters, it is obvious that the coarser approximation Y_L contains only those frequencies below $\frac{\pi}{2}$ of the original signal $x(n)$, while the highpassed signal Y_H contains only those frequencies above $\frac{\pi}{2}$. Consequently, decimation is justified for both the coarser approximation and the detail, holding the number of coefficients ($\{A_1\}$, $\{D_o\}$) constant at the number of original data samples. Note that reconstruction is perfect, and this may be expressed by

$$I = G'G + H'H$$

where G' and H' are the adjoints of the filters G and H, respectively, mapping $l^2(2\mathcal{Z})$ $\Rightarrow$ $l^2(\mathcal{Z})$. In addition, note that if $g(n)$ denotes the impulse response of the lowpass

lowpass filter with frequency response $G(\omega)$, a highpass filter can be obtained by translating $G(\omega)$ by π–radians (i.e., replacing ω by $\omega - \pi$). Therefore, the highpass frequency response $H(\omega) = G(\omega - \pi)$, or $h(n) = e^{j\pi n} g(n) = (-1)^n g(n)$, which is true for finite impulse response (FIR) filters. The following block diagram illustrates the quadrature mirror filtering process.



QUADRATURE MIRROR FILTERING

Daubechies [ref. 2] has shown that such filters can be constructed with all of the desirable characteristics, i.e., compact support in both the frequency and time domains, orthogonality, minimal coefficients (no more than the number of original samples) and perfect reconstruction. In this case the scale functions and wavelet functions, respectively, are

$$\phi_{j,k}(n) = 2^{-j} \sum_k g_k \phi(2^{-j} n - k)$$

and

$$\psi_{j,k}(n) = 2^{-j} \sum_k h_k \psi(2^{-j} n - k),$$

which are known as two scale difference equations or dilation equations, and $h(n) = (-1)^n g(n)$. The construction of wavelets then begins with the scaling function ϕ. Newland [ref. 13, pg 308-321] provides an excellent explanation as to how these filter coefficients are determined based upon:

- **CONSERVATION of AREA (Unique Scaling Function):**

$$\sum_{k=0}^{N-1} g_k = \sqrt{2}$$

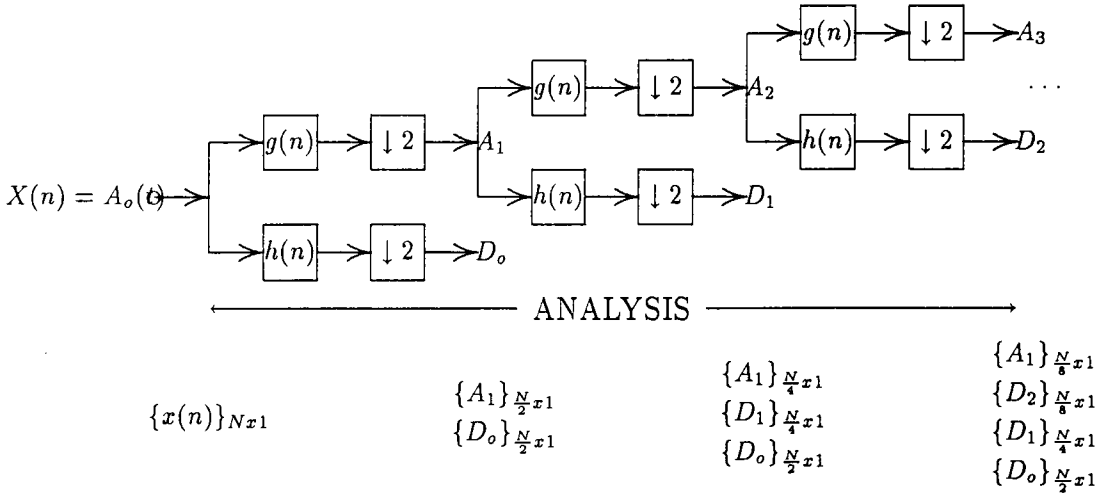
- **ACCURACY:**

$$\sum_{k=0}^{N-1}(-1)^k k^m g_k = 0, \qquad m = 0, 1, 2, \ldots, \frac{N}{2} - 1$$

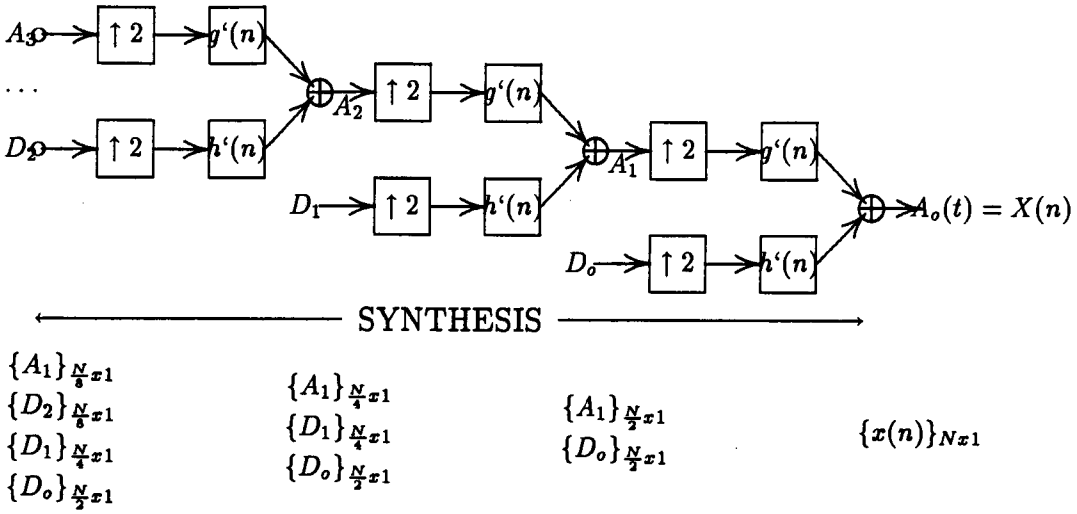
- and **ORTHOGONALITY:**

$$\sum_{k=0}^{N-1} g_k g_{k+2m} = 0, \qquad m \neq 0 \qquad \textit{and} \qquad \sum_{k=0}^{N-1} g_k^2 = 1,$$

where N is the number of coefficients desired. These conditions result in a set of $N+1$ simultaneous equations in terms of g_k, and finally, $h_k = (-1)^k g_k$. Assuming that such filters are available, the next section will illustrate the transformation of time domain data to a time-scale representation resulting from convolutions of the signal with these filters. The approach will follow the pyramidal scheme as coded by Press [ref. 14] and illustrated below.



DISCRETE WAVELET TRANSFORM PYRAMIDAL ALGORITHM



DISCRETE WAVELET INVERSE TRANSFORM PYRAMIDAL ALGORITHM

Time to Time-Scale Transformation

Any finite energy signal can be transformed from the time (or spatial) domain to a time(space)-scale representation via wavelet analysis. However, the choice of the basis functions used in the wavelet analysis isn't as straight forward as in the case of Fourier analysis, and it is this choice which dictates the interpretation of *frequency* necessary to correlate this analysis back to familiar Fourier representations. In this section, an example will be used to illustrate the transformation utilizing the most basic of wavelets, the Haar wavelet. In 1910, A. Haar determined the simplest solution to the question: "Does there exist an orthonormal system $\psi_o(x), \psi_1(x), \cdots, \psi_n(x), \cdots$ of functions defined on $[0,1]$ such that for *any* function $f(x)$, continuous on $[0,1]$, the series $< f, \psi_o > \psi_o + < f, \psi_1 > \psi_1 + \cdots + < f, \psi_n > \psi_n + \cdots$ converges to $f(x)$ **uniformly** on $[0,1]$?"

Referring to Figure 17, Haar's orthonormal basis is

$$\psi(x) = \begin{cases} 1 & 0 \le x \le \frac{1}{2} \\ -1 & \frac{1}{2} \le x \le 1 \\ 0 & \text{otherwise} \end{cases},$$

while the scale function associated with the Haar basis is

$$\phi(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}.$$

Consider a general multi-component sinusoid signal of the following form (see Figure 18):

$$\begin{aligned} x(t) = 1 \; &+ \; 5\,sin(2\pi t/4) \; + \; 4\,cos(2\pi t/9) \; + \; 3\,sin(2\pi t/18) \\ &+ \; 2\,cos(2\pi t/36) \; + \; sin(2\pi t/36) \; + \; 2\,cos(2\pi t/24) \\ &+ \; 3\,sin(2\pi t/12) \; + \; 4\,cos(2\pi t/6) \; + \; 5\,sin(2\pi t/2). \end{aligned}$$

For the Haar case, the quadrature mirror filter operators are

$$g(x(n)) = \frac{1}{\sqrt{2}}(x(n + x(n+1))) \quad \Rightarrow \quad Average$$

$$and \quad h(x(n)) = \frac{1}{\sqrt{2}}(x(n + x(n+1))) \quad \Rightarrow \quad Difference.$$

Therefore the filter coefficients are $\{g(n) : \frac{\sqrt{2}}{2}\}$ and $h(n) = (-1)^n g(n)$, $n = 0, 1$. Note that the sum of the coefficients is $\sqrt{2}$ and, hence, the normalization constant is $\frac{1}{\sqrt{2}}$, following Daubechies' approach (see [ref. 2, pg.195] for other filter coefficients). Applying the Haar wavelet to this signal results in the 3D plots for scale and detail as shown in Figure 18. Note that the analysis reduces the function to a coarser approximation (scaled representation) at each successive convolution of the wavelet with the signal function. Each of these applications is at a different scale, i.e., the basis function is broadening and these are the coefficients for the signal lowpass filtered to that scale.

This multi-scale analysis is akin to measuring distance with a wheel and a counter to count revolutions of the wheel. A large wheel is equivalent to a large scale, or wide support of the basis function, in that small details (resolution) are not discernible, i.e., the large wheel passes over small details without being effected by them. A small wheel is equivalent to small scale, or narrow support, such that small details (down into a hole and out again) can be included in the measure. Notice that the basic shape of the *basis function* didn't change, i.e., it is still a wheel!

The detail is a highpass filtered signal, while the scale is a lowpass filtered signal, which can be seen by performing an FFT on the detail components and the scale

components at each step (see Figure 18). Note that the scale components lose an octave with each wavelet convolution and, after the initial detail is determined, successive details result in a bandpass filtering of the signal. Signal reconstruction is obtained by reversing the process. Figure 19 illustrates that the reconstruction process is **perfect** with no detectable error.

The most unique aspect of wavelet analysis is that the time-domain data is transformed not into the familiar 2D amplitude-frequency plane as with Fourier analysis, but rather into a 3D amplitude-scale-time domain. Again, *scale* refers to the broadening of the basis function to fill the time (or space) interval associated with the filter order, i.e., an mth-order wavelet filter requires m data points for the convolution process regardless of the level of the analysis. Consequently, the basis function is more compact for the first level of wavelet analysis and broadens in time (or space) with each successive level of the analysis due to the decimation process which lengthens the time (or space) interval between successive data points. The inverse of this scale, or *period-of-existence*, of the basis function takes on a similarity to *frequency* in a mechanical sense if the basis function has a single zero-crossing as in the Haar and 4-tap Daubechies wavelets. This is due to the fact that a material fiber would undergo a tensile/compressive cycle at a rate of once per scale of the basis function, similar to that of a sine wave in Fourier analysis.

This can be better shown if one considers a sine wave signal composed of two harmonics plus the fundamental for Fourier analysis sampled at eight (8) times per cycle for the highest harmonic, and the same except a square wave for Haar Wavelet analysis as shown in Figure 20. The Fourier analysis applied to the sine waves illustrates each of the harmonics directly in the Fourier (amplitude vs. frequency) plane as coefficients of the sine waves making up the signal represented as spikes of the appropriate amplitude at the appropriate frequency. The wavelet analysis

applied to the square waves illustrates these harmonics in the 3D time-scale plane as **detail** coefficients, which are constant valued functions of the appropriate amplitude at the appropriate scale of the basis function extending for all time (or space) of the sample block. These functions are constant-valued in time since the square wave signal is periodic, just as the Fourier coefficients are constant in time (or space) since Fourier analysis assumes stationary signals and all temporal (or spatial) dependence is lost following the FFT.

Recall that the wavelet detail coefficients are the amplitudes of the fluctuating ***continuous*** wavelet basis functions, just as the FFT coefficients are the amplitudes of the ***continuous*** sine wave basis functions, to be combined to synthesize the respective signals. Similarly, the wavelet scale coefficients are the decimated signal samples, i.e., a smoothed representation of the signal at a coarser scale. For the Haar wavelet applied to a square wave oversampled at eight (8) samples per cycle, the wavelet amplitudes are all zero except the third (3rd) level of decomposition (for 512 samples, this is level 6 of the 9 possible, i.e., $2^9 = 512$, level nine (9) being the original samples), which is constant for all values of time (or space) for the periodic square wave. Consequently, the Haar wavelet does for square waves what Fourier analysis does for sine waves, i.e., determines the exact scale or frequency match, respectively, for a signal composed of the respective basis functions. The Haar wavelets, of coarse, can only provide such perfect scale matches if the data is properly aligned to harmonics and sampled appropriately, but this too is no different than the FFT where pure spikes will occur only if the data has integral frequencies and is sampled with zero phase. Otherwise smearing occurs in the FFT spectrum, and the wavelet analog is to have numerous non-zero, and typically oscillatory, details at scales **below** the highest harmonic in the signal. It should be noted that at the third level of decomposition, the wavelet analysis has decimated the original signal by a factor of eight (8), the

exact factor of the samples per cycle, leaving only **one sample per cycle.** This holds for other sampling factors as well. In other words, the log_2 of the number of samples per cycle is the decomposition level which will be non-zero if the signal is a series[8] of the basis function of the wavelet. At this level, the reciprocal of the interval between the remaining samples is the *frequency* of the square wave. This is exactly what was meant by τ as defined on pages 36 and 38, where the basis function now translates by the sampling period to generate the signal. In the case of wavelets, the **serial** rate is only a single point per cycle, which is required to define where the series of Haar wavelets is to be placed to construct the square wave. Nyquist is still the limiting **sample** rate for a signal, however, since it takes two points to define the interval τ and to perform the decomposition, where the Haar wavelet filter must be convolved with the signal and the minimum number of sample points then is the size of the filter, i.e., two, the smallest wavelet filter. It should also be pointed out that the choice of the wavelet basis function is equivalent to making an assumption about the composition of the signal. The use of a Haar wavelet for a square wave is making the obvious assumption that a square wave is made up of the Haar basis, just as in the case of using the FFT makes the assumption that the signal is composed of sine functions. Each basis applied to the wrong signal will generate broad *spectra* which tend to yield little or no information about the signal composition. The potential of wavelets is that we now have a broad selection of basis functions to apply to a problem to detect characteristics which are smeared utilizing only the FFT, or to perform pattern recognition analysis where we generate a new basis dependent upon the pattern of interest. The next section will describe other wavelet bases.

[8] Here *series* means a series in time (or space) of wavelet basis functions laid end to end.

Common Wavelet Bases

Recall from the previous sections of this chapter that the wavelet expansion merely identified the requirements for a function to be a wavelet. Nowhere, with the exception of the Haar wavelet, was it ever clearly stated what these basis functions were, looked like, or how they should be used. Unlike Fourier analysis, where only one set of basis functions is used, the sine functions, wavelet analysis must include a choice of the basis function itself. A variety of basis functions have been constructed and appear in the literature utilizing (and sometimes relaxing) the requirements defined above. Most notably are those of Daubechies, Chui, Malvar, Meyer, and Newland, each with slightly different interpretation of the applicability of the requirements for the analysis of interest resulting in different forms of wavelets. Daubechies, Malvar, and Newland's (harmonic) wavelets will be discussed here since these appear to have the most applicability to the signal processing which is the subject of this thesis.

As described in Meyer [ref. 11], two different approaches to transient analysis can be proposed (originally by Ville, 1947). One can either cut the signals into time slices and then pass these time slices through a system of filter banks to analyze them, or one can first filter different frequency bands and then cut these bands into time slices to study their energy fluctuations. The first approach results in signal packets and the wavelets of Malvar, while the second results in wavelet packets, or Daubechies' wavelets. Both approaches have merit, and in fact both are widely used today by analysis engineers without realizing that they are doing a variation of wavelet analysis. But while the analysis techniques have been around for some time, the theoretical foundations for such analyses and the resulting unification of these various techniques is due to wavelet theory.

Daubechies wavelets are compactly supported (with varying numbers of non-vanishing coefficients) and orthonormal resulting in an asymmetric form with varying degrees of smoothness (see Figure 21). These wavelets are implemented utilizing the pyramidal algorithm as quadrature mirror digital filters. The two term Daubechies wavelet (Daub2) is the Haar wavelet discussed above. However, these wavelets, except for perhaps the higher order ones (larger number of coefficients), do not resemble a natural (sinusoidal) response for a vibrating element. Consequently, these wavelets complicate the interpretation of the spectral analysis when using such basis functions for typical signals from mechanical elements. Generally, these wavelets generate a full spectrum (i.e., non-constant values for all scales), offering little engineering information (see Figure 22). These wavelets may provide information in those cases where rapid high level pulses occur embedded in more natural oscillations when combined with traditional Fourier analysis, or where harmonic filtering (via the scale functions) is required for more critical Fourier analysis. In addition, these wavelets appear to be very good at detecting the edges of discontinuities within a signal. These areas need further investigation, which is ongoing at various locations.

Another form of wavelet is the Malvar wavelet [ref. 11, pg 78-79]. The Malvar wavelets are wavelets that slice the time interval rather than filter the signal into frequency intervals. The Malvar wavelets slice the signal into time intervals and convolve these slices with a window which results in each piece being smoothed at the beginning and ending (attack and decay, respectively) to avoid abrupt discontinuities. The interval between the attack and decay can be of varying lengths, called the stationary period, and is windowed with a constant function (see Figure 23). This amounts to a new window for windowed Fourier analysis, but where the window is applied to the basis functions of the approximation rather than the signal

itself. The Malvar wavelet is defined as:

$$\psi_{j,k}(t) = \sqrt{\frac{2}{l_j}} w_j(t) \cos \frac{\pi}{l_j} \left(k + \frac{1}{2}\right)(t - a_j)$$

where $k = 0, 1, 2, \ldots$ and $j \in \mathcal{Z}$ and

$$0 \leq w_j(t) \leq 1 \quad \forall t \in \Re$$

$$w_j(t) = 1 \quad if \quad a_j + \alpha_j \leq t \leq a_{j+1} - \alpha_{j+1}$$

$$w_j(t) = 0 \quad if \quad t < a_j - \alpha_j \quad or \quad t \geq a_{j+1} + \alpha_{j+1}$$

$$w_j^2(a_j + \tau) + w_j^2(a_j - \tau) = 1 \quad if \quad |\tau| \leq \alpha_j$$

$$w_{j-1}(a_j + \tau) = w_j(a_j - \tau) \quad if \quad |\tau| \leq \alpha_j$$

For a division of the time line into dyadic intervals, $a_j = 2^j$, $\alpha_j = \frac{1}{3}2^j$, and $l_j = 2^j = \alpha_j + \alpha_{j+1}$, and for $w(t) = 0$ outside $[\frac{2}{3}, \frac{8}{3}]$, $w(2-t) = w(2-t)$ and $w^2(t) + w^2(2-t) = 1$ on $[\frac{2}{3}, \frac{4}{3}]$, then

$$\psi_{j,k}(t) = 2^{\frac{-j+1}{2}} w(2^{-j}t) \cos \left[\pi \left(k + \frac{1}{2}\right) 2^{-j} t\right]$$

and

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}} \psi(2^{-j}t - k) \quad j, k \in \mathcal{Z}.$$

These functions form a basis of $L^2(\Re)$ and are also orthonormal. These wavelets preserve the common notion of frequency, and may well have application to vibration analysis for situations that are only quasi-transient. However, at this time, the discrete Malvar wavelets have not been put into a form which is fast and can be readily used as a filter for digital signal processing. The filters are actually the transfer function of $w_j(t)$, and due to the logic involved, are not yet conducive to real-time processing. Another problem with these wavelets is that "although they are given by a very simple formula, they are not obtained by translation, change of scale, and modulation (or modulation by $e^{i\omega t}$) of a fixed function $g(t)$" [ref. 11]. The

use of these wavelets involves the convolution process just as in the calculation of the discrete Fourier transform (DFT, without using the FFT algorithm), which is very expensive to calculate. Further research is needed to apply these wavelets to a general vibration problem.

Finally, Newland's harmonic wavelets [ref. 13] are orthogonal wavelets that, unlike Daubechies' wavelets, which cannot be expressed in functional form, are derivable from a simple analytical formula. The harmonic wavelet is compact in the frequency domain, but spans the entire real axis or time domain. However, they decay in the time domain as a function of $\frac{1}{t}$, in contrast to wavelets derived from dilation equations (such as Daubechies') which are identically zero outside the range of their coefficients (i.e., for a N-coefficient Daubechies wavelet, the wavelet is zero beyond $N-1$ intervals). The main advantage of the harmonic wavelet is that it is significantly faster than wavelets derived from dilation equations, and since they are described by a complex function, two real wavelets can be generated simultaneously just as in the case of simultaneously obtaining two spectrums with the FFT for real data. In addition, the harmonic wavelets are implemented using the FFT/IFFT and the harmonic wavelet coefficients for each scale can be generated in parallel rather than serially for the dilation equation wavelets. Finally, the harmonic wavelet, because it is based on the FFT, is essentially already widely supported in the DSP community and therefore can be efficiently implemented in existing hardware. The continuous harmonic scale function and wavelet are shown in Figures 24 and 25, respectively, and are defined as

$$\phi_k(t) = \frac{e^{i2\pi(t-k)} - 1}{i2\pi(t-k)} ,$$

$$\psi_{j,k}(t) = \frac{e^{i4\pi(2^j t-k)} - e^{i2\pi(2^j-k)}}{i2\pi(2^j t-k)} \qquad for\ j = 0\cdots\infty,\ -\infty < k < \infty .$$

Note that these functions are complex, and that the scaling function is a single

parameter equation since it spans the whole time axis while the wavelet dilates. The discrete harmonic wavelet coefficients are defined as

$$\psi_{2^j+k} = \sum_{s=0}^{2^j-1} F_{2^j+s} e^{\frac{i2\pi sk}{2^j}} \qquad k = 0 \cdots 2^j - 1 \ ,$$

and the complex conjugate as

$$\widetilde{\psi_{2^j+k}} = \sum_{s=0}^{2^j-1} F_{N-(2^j+s)} e^{\frac{i2\pi sk}{2^j}} \qquad k = 0 \cdots 2^j - 1 \ ,$$

where F_q are the FFT frequency coefficients for the original time or spatial data set. Note that these equations amount to taking the inverse FFT of the signal's original FFT coefficients grouped by octaves. This scheme is illustrated in Figure 26. Because of the smoothness and shape of the harmonic wavelet, it is expected to have application in vibration analysis of impulses and in pulse echo signal analysis. Further research in these areas is required.

CHAPTER IV

USE OF WAVELETS FOR STATIONARY DATA ANALYSIS

As described in Chapter III, the interpretation of wavelet analysis using the Haar wavelet applied to a stationary square wave is identical to that of Fourier analysis applied to sine waves. However, the similarity generally does not hold for other more general signals, and the interpretation of what wavelet analysis is providing for a stationary signal becomes more difficult. In this section, an attempt will be made to describe what information can be obtained from wavelet analysis and how that information must be interpreted with respect to familiar Fourier analysis for stationary signals.

Consider a signal composed of three sinusoids with frequencies in multiples of one, two, and four, i.e., $x(t) = A(sin\,2\pi f_0 t + sin\,2\pi f_1 t + sin\,2\pi f_2 t)$, where $f_1 = 2f_0$ and $f_2 = 4f_0$. The results of applying the FFT to this signal are shown in Figure 27, while application of the Daubechies eight-term wavelet (Daub8) to this signal results in Figure 28. In this plot the original signal (wavelet level 10 since the number of samples = $1024 = 2^{10}$) is shown at scale index two and the first wavelet iteration (level nine) is shown at scale index three. Therefore, subtracting two from the scale index in these 3D wavelet plots will yield the wavelet iteration number, which, when subtracted from 10 (the highest wavelet level for these data) yields the wavelet scale. Now, the signal has three distinct frequencies, f_o=10Hz, f_1=20Hz, and f_2=40Hz. The sampling rate was $2.56f_{max}$ or 102.4Hz. The first wavelet iteration filters the signal and subsamples by a factor of two, which means at level nine the sample rate of the scale is effectively 51.2Hz, and so on. The wavelet plots shown are the **details** lost from each level due to the filtering and subsequent downsampling at each wavelet iteration, therefore, the higher frequencies are peeled away with each wavelet iteration. Since the signal has three distinct frequencies in binary steps,

then these frequencies can be peeled away singly and therefore only the first three wavelet levels (9,8, and 7) are non-zero (except for some noise). The noisy (non-constant) traces are due to trying to approximate the signal with a basis function which does not match, the same occurs with the FFT where it smears between primary sine components in real signals.

The approximation resulting from the wavelet expansion may be expressed as

$$X(t) = A_1 D(2^9 t) + A_2 D(2^8 t) + A_3 D(2^7 t) + ERROR$$

where A_i are the magnitudes of the details at each level and $D(2^j t)$ are the wavelet basis functions for Daub8. With Daub8, the wavelet levels are coarser and small ripples exist in levels 6 and 5 (see Figure 28), while for Daub20 (Figure 29) all levels other than the primaries are purely constant without ripples and identically zero. An FFT performed on these levels indicates (see Figure 30) a single major frequency and a sidelobe of approximately one-fifth of the major lobe. This minor lobe is due to the error cited above, and becomes less pronounced if the sampling is an even multiple of two. Using the same signal but sampled at 16 samples per cycle for the maximum frequency results in Figure 31. Note that the original signal is not shown in this 3D plot to get better magnitude resolution. The three levels that are high are the three signal components, which start at the fourth wavelet iteration since the over-sampling was 2^4. The noisy trace at the third wavelet iteration (level 7) is again the error due to the basis/signal mismatch. A spectrum of this level yields a single spike at the right edge of the FFT (see Figure 32).

The wavelet expansion of a stationary signal yields results that can be adequately interpreted as the components of the signal, and therefore is comparable to the FFT. However, the wavelet only works on harmonics (signal components spaced in frequency by 2^j times the fundamental) and the error cited above would span many more levels if tighter frequency spacing existed. The FFT is adequate for

the stationary case and should not be replaced with a misapplication of wavelets. The wavelet may have application to stationary signals where the higher harmonics of essentially a pure harmonic signal are buried in noise. This may aid in modal analysis of noisy signals, but further research in this area must be performed.

CHAPTER V

EVALUATION OF THE LIMITS OF FFT APPLICABILITY

It was mentioned in Chapter I that the FFT tends to break down under certain circumstances, such as non-stationarity and sudden impulses or discontinuities. In this chapter, the FFT will be evaluated to put these comments into perspective, and to attempt to find the limits of applicability of the FFT for such circumstances.

Currently, testing in which the author is involved does not provide data which is non-stationary since the test is slowed appropriately so as to avoid transient responses in the data, or if a transient event is captured, seldom is the data made public. Consequently, simulated data will be used for this chapter due to the lack of real data and such that the degree of non-stationarity may be controlled. Further, no random noise will be applied which could mask the true limits of applicability of the FFT.

Frequency Ramp

To evaluate the applicability of the FFT for a non-stationary signal that ramps in frequency, a simulated signal was generated of the form

$$x(t) = A_1 sin(c2\pi f_1 t) + A_2 sin(c2\pi f_2 t),$$

where $c = (1 + rt)$, r is the linear input ramp, and c ranges linearly from 1.0 to the maximum shown in Table 2. In essence, this signal is a typical chirp which is usually characterized as being of the form $Asin(\alpha t^2)$.

The sampling of this signal was at eight times the maximum fundamental frequency (f_1 = 10 Hz, f_2 = 40 Hz, therefore F_s = 320 Hz) to provide room for the ramp and still meet the Nyquist criteria. The interval of the ramp coincides with the interval of the data block. The ramp rate was controlled to ensure that the

signal was properly sampled over the whole interval, and the results obtained then have nothing to do with aliasing. It was assumed that the data acquisition system was started coincident with the initiation of the ramp, such that the whole interval consisted of the ramp with no stationary period. The test parameters are shown in Table 2, and the signal and FFT spectra are shown in Figure 33.

TABLE 2 - Frequency Ramp Test Cases

A_1	A_2	Ramp	c_{max}	f_1 Range	f_2 Range	Figure
1.0	0.5	.0005	1.012788	None	None	33A
1.0	0.5	.005	1.127875	10 → 11Hz	40 → 50Hz	33B
1.0	0.5	.05	2.278750	10 → 35Hz	40 → 140Hz	33C
1.0	0.0	.01	2.023000	10 → 31Hz	N/A	33D
1.0	0.0	.015	2.534500	10 → $\geq F_s/2$	N/A	33E

Notice that each spectrum based upon the FFT was smeared between the base frequency and the maximum frequency attained due to the ramp, unless the ramped signal has the maximum frequency within one bin of the base signal frequencies for the FFT size used, and therefore no smearing is expected. Note that the waist of the smearing occurs at the midpoint of the smear, and has the appearance of a beating of the base frequency with a signal which is one half the width of the smeared area. Seldom is a signal composed of a continuous band of frequencies as shown in these spectra, especially for mechanical components, and therefore, a ramp is detectable with an FFT so long as the base frequencies are separated sufficiently to prevent one ramping into and smearing the other. In such a case, it would be better to use an analysis technique which preserved the time element for visualization of the ramp. This also indicates that the FFT cannot be applied satisfactorily to a ramp signal if the total ramp in frequencies occurs in less than two adjacent FFT bins. Consequently, appropriate bin spacing must be kept in mind and weighed against the amount of oversampling and the expected ramp frequency endpoint for FFT analysis of ramp signals to yield useful information.

Figure 22 is the same signal as Figure 33C with Daub8 applied to it. To the author, the FFT was superior to the Daub8 wavelet analysis in this case. While the wavelet analysis preserved the time element of the signal, the interpretation is difficult and the presence of the ramp is not as easily identified. However, this again is due to the mismatch of the basis functions to the simulated signal, and this may improve with a *real* signal, unless real signals truly are sinusoidal!

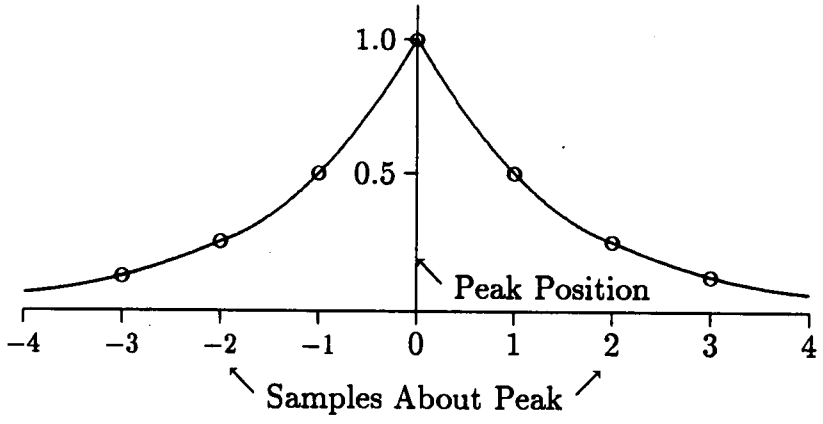
Sudden Impulses

To evaluate the applicability of the FFT for a non-stationary signal that contains impulses, a simulated signal was generated of the form

$$x(t) = A_1 sin(2\pi f_1 t) + A_2 sin(2\pi f_2 t) + A_3\hat{\delta}(t-\tau_1) + A_4\hat{\delta}(t-\tau_2),$$

where the delta functions are **not** the strict mathematical delta functions, and are merely notational for a spike of variable support over multiple data samples instead of just a single sample. This was accomplished by linearly building up to the magnitude of the spike symmetrically over an input number of samples by the following algorithm applied to the signal value for that sample, as illustrated below:

SEVEN POINT SPIKE VALUES for *POSITIVE* SIGNAL
to be *ADDED* to SIMULATED SIGNAL SAMPLES
(*NORMALIZED* AMPLITUDE EXAMPLE)



```
k = NSpikePts - 1;              /* Input k MUST be ODD */
if( k < 0 )
        k = 0;
l = k/2;
m = -l;
if( i == SpikeLoc1 + m )        /* Simulated Noise Spike */
{
        if( x[i] >= 0.0 )
                x[i] = x[i] + SpikeMag/(double)(abs(m) + 1);
        else
                x[i] = x[i] - SpikeMag/(double)(abs(m) + 1);
        m++;
        if( m > l )
                m = -l;         /* Resets m for NEXT Spike */
}
```

Variable Support Spike Generator Algorithm

This algorithm allows an extreme discontinuity if the impulse support spans a zero crossing of the signal (see Figure 34A). While this may not be the case for a real signal, it does provide a worst case for this analysis. The sampling of this signal was at 2.56 times the maximum fundamental frequency which would be typical for a real signal to meet the Nyquist criteria. The effect of the spike on the FFT is a function of the number of samples in the data block, the magnitude of the spike, and the support of the spike. Recall, the FFT is a convolution which spreads the characteristics at one sample over all of the samples of the ensemble. This is illustrated by comparing Figure 34A, which has 1024 points in its data block, to that of Figure 34B which is the same signal with spikes of the same magnitude and relative support, but with 256 sample points in its data block. Notice that the level of the noise floor is significantly lower for the larger block size even though the signals are functionally identical, and Figure 34A is a case of extreme discontinuity due to sampling closer to a zero crosing. Consequently, the number of data samples in the FFT block was reduced to 256 from the usual 1024 for this thesis. The spike amplitude was 16 times the amplitude of the fundamental component of the signal such that the effects of the spike could be seen. The test parameters are shown in

Table 3, and the signal and FFT spectra are shown in Figure 34.

TABLE 3 - Sudden Impulse Test Cases						
A_1	A_2	A_3	A_4	N	*NSpikePts*	Figure
1.0	0.5	16.0	16.0	1024	500	34A
1.0	0.5	16.0	16.0	256	125	34B
1.0	0.5	16.0	16.0	256	9	34C
1.0	0.5	16.0	16.0	256	3	34D
1.0	0.5	16.0	16.0	256	1	34E

It is obvious that the sharper the spike (narrower support) the more pronounced is the smearing of the noise floor. However, this is not the case for the sharpest spike of all, a single sample of support. This is most likely due to the whole band being influenced and therefore the energy is more evenly distributed for the true impulse (single sample spike) than for the wider supported spikes, which appear to have a band limited spectrum. Notice that each spectrum based upon the FFT, does not really detect the spike, it merely smears the noise floor and still zeros in on the stationary characteristics of the signal. Consequently, the FFT can not be used for a signal containing spikes since the noise floor for a spike cannot be differentiated from the noise floor of a typical real spectrum. The wavelet has already been shown to provide detection of discontinuities in a signal, and therefore is superior to the FFT for such signals. However, the wavelet would not do as good a job on the stationary components of the signal as the FFT since the FFT basis matches the signal (or is assumed so for vibratory/harmonic analysis) and the wavelet basis generally would not. Consequently, this is a case where the two analysis techniques complement each other and should be used together for such analyses. Figure 22 is the same signal as Figure 34C with Daub8 applied to it. Notice how distinct the presence of the spikes are.

CHAPTER VI

WAVELET APPLICATIONS FOR NON-STATIONARY DATA

STFT Comparison

Consider the same signal used for the STFT analysis discussion of Chapter II (see Figure 14). Application of the Daubechies four-term wavelet (Daub4) to this signal results in Figure 35. Again, the original signal (wavelet level 10 since the number of samples = 1024 = 2^{10}) is shown at scale index two and the first wavelet (level nine) is shown at scale index three. Therefore, subtracting two from the scale index in these 3D wavelet plots will yield the wavelet iteration number, which, when subtracted from 10 (the highest wavelet level for these data) yields the wavelet scale. Now the signal has three distinct frequencies, f_0=10Hz, f_1=80Hz, and f_2=128Hz. The sampling rate was 2.56*f_{max} or 327.68Hz. The first wavelet filters the signal and subsamples by a factor of two, which means at level nine the sample rate of the wavelet scale is effectively 163.84Hz. This is essentially twice f_1, and therefore the 80 Hz signal is shown as a sinusoid at scale index four, which is level eight (this was verified using a single 80 Hz component sampled at $\frac{128}{80} \cdot 2.56$ or 4.096 samples per cycle). The trace at scale index three (level nine) should be the f_o-modulated 128Hz cosine wave. Recall, the wavelet shows the detail lost by the filtering at the previous level. The interpretation of this trace is not fully understood by the author and it is suspected that this trace is noisy due to the wavelet basis/signal mismatch as discussed in Chapter IV. The Gaussian impulses are observed as spikes at sample time indices 320 and 576 which broaden with successive wavelet iterations. Note that the temporal characteristics were preserved with the wavelet transform, unlike the FFT, and the spikes occur at the **peaks** of the impulse even though the impulses were rather broad (64 samples each). This is due to the slope change from one side

of the peak to the other and therefore an *edge* is detected by the wavelet at this point. The trace at scale index six (level six) is an artifact of sampling the 10Hz signal at a factor of 32.768 ($\frac{128}{10} \cdot 2.56$) samples per cycle, which is an exact multiple of the sampling rate (this, too, was verified using a single 10Hz component sampled at 32.768 samples per cycle). This artifact appears to be due to a beating between the wavelet zeros and the signal zeros at this level, but this could not be verified by the author. This artifact persists regardless of the order of the wavelet, from Haar to Daub20. The only significant changes going to other wavelet orders were the coarsening of the traces with the Haar wavelet and the smoothing of the traces with the Daub20 wavelet, and the more pronounced spikes at the Gaussian impulse positions due to less noise with the Daub20. The Daub20 wavelet results are shown in Figure 36. Note that neither Figure 35 nor Figure 36 contain any information about the amplitude modulation aspect of the signal. This is a deficiency of the wavelet analysis, at least as it is presently understood and interpreted by the author.

The STFT provided results which were considered superior to that of the wavelet analysis in interpretation aspects, but the wavelet analysis is much faster, has less impact on the hardware requirements for sampling, and provides suitable information once the interpretation is worked out. In addition, the wavelet could be used in multiresolution schemes to properly filter the signal for FFT analysis or to complement the FFT by searching for impulses or other discrete non-stationary events. Further research into the interpretation of wavelet data must be provided, and perhaps the new harmonic wavelet will shed some light in this area.

Pulse Echo

Pulse echo is a technique for measuring distances in materials nonintrusively through the use of sound waves. In brief, a sound impulse is propagated into the material through an appropriate acoustic coupling and a sensor detects the echo *or*

reflected wave caused by the abrupt change in the index of refraction at the opposite edge of the material (this can be the interface between two different materials or an air/material interface). The thickness d of the material is then found as

$$d = c\frac{t}{2}$$

where c is the characteristic speed of sound for the material, and t is the round trip time from initiation of the pulse through the material to the discontinuity in the index of refraction and back through the material to the sensor. The sensors are typically piezoelectric crystals of appropriate natural frequency which act as the transmitter when electrically pulsed and act as the receiver when stimulated by the returning sound wave echo (hence, the name Pulse-Echo). The signals resemble an impulse with ringing, and were simulated by single sided sinc functions with a reduced amplitude for the echo to simulate material transmittance properties. The functions used were of the form

$$x[t] = u_{-1}(t-\tau_1)A_1\frac{sin(t-\tau_1)}{(t-\tau_1)} + u_{-1}(t-\tau_2)A_2\frac{sin(t-\tau_2)}{(t-\tau_2)} .$$

where τ_1 was near the beginning of the ensemble, τ_2 was at the midpoint of the ensemble, and A_2 was $.5A_1$. The first sinc function was spread over the whole interval once started, while the second sinc function started such as to peak at the exact midpoint and build upon the first impulse, hence the unit step functions $u_{-1}(t-\tau)$. The signal was sampled at 16 samples per cycle for the 1Hz sinewave component of the sinc functions, and 1024 point ensembles were used. Random noise was added to the signal, and various noise floor amplitudes were evaluated. The closeness in time between the simulated pulses could be controlled to simulate varying material thickness. The test parameters are summarized in Table 4, and the wavelet 3D detail plots are shown in Figure 37.

TABLE 4 - Pulse-Echo Test Cases			
A_1	A_{NOISE}	N_{START}	Figure
1.0	0.0	128.0	37A
1.0	$0.5A_1$	128.0	37B
1.0	$0.9A_1$	128.0	37C
1.0	0.0	169.0	40A
1.0	0.0	385.0	40B

In pulse-echo measurements, assuming the material speed of sound characteristics are well known, the most important parameter is an accurate measurement of the time interval between the main pulse and its echo. Any point on the pulses can be used as a reference point, since the same sensor is used for transmitting and receiving it will have the same response function, but it is necessary to be consistent. Since real signals generally have a significant noise floor, it is usually most accurate to use the peaks in the pulses as the reference points. However, most pulse-echo instruments use a threshold test and capitalize on the rapid decay of the response because detecting the true peak location is difficult electronically. The use of wavelets may change this situation due to their ability to detect edges and rapid slope changes in a signal.

Various Daubechies wavelets were applied to the signal with the intent of locating the peaks as consistent reference points for determining the time interval. The Haar (Daub2) wavelet provided the best results, and was used for this analysis. The FFT applied to the signal of Figure 37A merely shows a pulse near DC and offers no timing information (see Figure 38). It does illustrate that the signal is appropriately sampled, however, since the spectrum is essentially zero well before the Nyquist foldover frequency. The Figure 39 illustrates the ability of the wavelet to detect the location of the peaks at wavelet levels four and three. Level four is significant since at this level the signal is sampled at essentially the Nyquist criteria as mentioned in the discussion *Time to Time-Scale Transformation* of Chapter III.

Even in the presence of significant noise (Figure 37C), the Haar wavelet was able to detect the peaks in either of these two levels. While the fourth level has a second peak which becomes more pronounced with increasing noise, the third level clearly distinguishes which peak in level four is the appropriate peak of the pulse for timing purposes. However, the ability of wavelets to determine the peak locations is tied to whether or not the peak was sampled and if the sample index n is a factor of 2^j. This is illustrated in Figure 40A where the starting point was at $n = 169$, and a maximum error of 13% (for level 3, level four is 7%) is incurred due to the subsampling resulting in the peak being missed slightly. This is equivalent to the scalloping effect of the FFT when the signal frequency does not correspond to the FFT bin center frequency but rather is offset from the center of the bin. Figure 40B takes this situation to the extreme by narrowing the distances between the pulses and having n a non-integral power of two. This is probably the minimum spacing that would be considered a typical pulse-echo signal (for thinner materials, higher frequencies (tighter pulses) are typically used to space out the pulses for better time resolution of the peaks). The peaks are still well defined in level four, while level three has rounded and smeared the peaks.

In summary, the wavelet analysis provided a consistent set of timing points for improving the resolution of the pulse-echo measurement technique, especially when the signal is buried in significant noise. The limitations of the approach appears to be the pulse peak location relative to an integral power of two sample points within the ensemble. This could be resolved by performing a sliding wavelet analysis such that the peak would become more pronounced as it fell on an integral power of two sample, and then timing between two different wavelets at the same level. Work is being performed to make the sliding wavelet computationally efficient, and to calculate a particular level without having to calculate all previous levels serially.

CHAPTER VII

CONCLUSIONS

The primary goal of this thesis was to evaluate various dynamic data analysis and synthesis tools to illustrate the differences and similarities in their mathematical background, provide a physical interpretation of these transform methods, and show application of these tools to some selected signal types. The techniques evaluated were the FFT, STFT, and various wavelet analyses as they might be applied to a real-time machine monitoring system. The mathematical background for these techniques was provided with emphasis on their application to one-dimensional non-stationary transient signals and the interpretation of the analysis results based upon approximation of signals using basis functions analogous to basis vectors. Simulated examples were provided to aid one's understanding of the analysis and to establish firmly the reasons for looking for an alternative to the typical Fourier approach.

The theory, interpretation, and shortfalls of the FFT analysis were described and limits to the application of the FFT for non-stationary signals were established for the two major non-stationary characteristics of interest to the author, frequency ramps and sudden impulses. It was shown that the FFT, with appropriate bin spacing and oversampling, provides adequate detection and useful information for signals which ramp in frequency, and does so much better than the wavelet approach, even though the time element of the signal is lost in the transformation. It was also shown that the FFT is inadequate to describe a signal containing spikes since the FFT noise floor for a spike cannot be differentiated from a typical real spectrum's noise floor. The wavelet has the capability to provide the detection of discontinuities in a signal, and therefore is superior to the FFT for such signals, and was shown to be an improvement for pulse-echo measurements due to this ability. However, the wavelet distorts the interpretation of the concept of frequency and therefore is

not acceptable for the stationary components of the signal which are still of interest for vibratory/harmonic analysis in the presence of spikes. Consequently, the case of spikes embedded in an otherwise stationary signal is where the two analysis techniques complement each other and should be used together.

The major attempt to fix the FFT for transient signals, the STFT, was described and provided an introduction into the modern multiresolution techniques. Again, once the interpretation is worked out, the STFT is generally adequate for determining the non-stationary characteristics of a signal. The STFT provided results which were considered superior to that of the wavelet analysis in interpretation aspects, and correctly determined the information about the amplitude modulation aspect of the signal which the wavelet analysis did not detect. However, the wavelet analysis is much faster, has less impact on the hardware requirements for sampling, and provides suitable information for other non-stationary characteristics once the interpretation is worked out, and again may be useful as a complement to the FFT to obtain results similar to the STFT with less stringent sampling and processing requirements.

The use of wavelets as an alternative to Fourier analysis for dynamic/transient data is still in its infancy. It is clear that several advantages can be obtained through the use of wavelets for non-stationary data, but the interpretation of the results still requires much work. New wavelets are appearing in the literature which may turn out to be suitable compromises for vibratory/harmonic analysis which embody the strong points of the FFT with the ability of the wavelet to detect transient phenomena. It is the generality and sound mathematical foundation that wavelets offers the signal analysts for pattern detection that is unique. Just as in Fourier analysis, where the sine wave basis functions match the sinusoids (solutions to the differential equations) describing the phenomena, so too will wavelet analysis

provide valuable information providing the basis functions match the phenomena. The strength of wavelets is that there now exists a mathematical formalism to aid in building new basis functions to match patterns in a signal for detection in a real-time monitoring system.

The Fourier approach has become such a natural way of viewing signal decomposition, that the newer wavelet or other techniques are constantly evaluated based upon the strengths and weaknesses of the Fourier analysis. In an attempt to objectively determine the best tool for analyzing non-stationary signals of interest to the author and work out their interpretation, the requirements for real-time transient spectral analysis for machine health monitoring functions which exhibit these types of signals are defined in Appendix B. It is hoped that this will aid the development of models to properly weigh the merits of new analysis techniques without bias of the popularity of Fourier analysis.

Finally, it is the opinion of this author that wavelet analysis will play a major supportive role in signal analysis, complementing the FFT for non-stationary signal decomposition. The wavelet approach could be used in multiresolution schemes to properly filter the signal for FFT analysis or to complement the FFT by searching for impulses or other discrete non-stationary events. It may also be possible to develop new bases which are combinations of signal characteristics to aid in the detection of patterns in machine condition monitoring data which currently require multiple analyses and detailed experience to detect, thereby nullifying the possibility of real-time condition monitoring and control. It is not the intent of the author to replace Fourier analysis, but rather to supplement it in those areas where it is not appropriate. Further research into the interpretation of wavelet data must be provided, and perhaps the new harmonic wavelet will shed some light in this area.

REFERENCES

REFERENCES

1. Bendat, J.S. and Piersol, A.G. **Engineering Applications of Correlation and Spectral Analysis**, John Wiley & Sons, New York, NY, 1980.

2. Daubechies, I. **Ten Lectures on Wavelets**, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

3. Proakis, J.G. and Manolakis, D.G. **Introduction to Digital Signal Processing**, Macmillan Publishing Company, New York,NY, 1988.

4. Chui, C.K. **An Introduction to Wavelets**, Academic Press, Inc, San Diego, CA, 1992.

5. Strang G. **Introduction to Applied Mathematics**, Wellesley-Cambridge Press, Wellesley, MA, 1986.

6. Ramirez, R.W. **The FFT, Fundamentals and Concepts**, Pren-tice-Hall, Inc., Englewood Cliffs, NJ 1985.

7. Resnick, R. **Basic Concepts in Relativity and Early Quantum Theory**, John Wiley & Sons, Inc., New York, NY 1972

8. Wang, W.J. and McFadden, P.D., **Early detection of gear failure by vibration analysis - I. Calculation of the time-frequency distribution**, *Mechanical Systems and Signal Processing*, Vol. 7, No. 3, May 1993, pp 193-203.

9. Ziemer, R.E., Tranter, W.H., Fannin, D.R., **Signals and Systems: Continuous and Discrete**, Macmillan Publishing Company, New York, NY 1989.

10. Rioul, O. and Vetterli, M. **Wavelets and Signal Processing**, *IEEE Signal Processing Magazine*, October 1991, pp 14-38.

11. Meyer, Y. (Translated by Ryan, R.D.) **Wavelets: Algorithms and Applications**, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1993.

12. Hite, S.W., III, **An Algorithm for Determination of Bearing Health through Automated Vibration Monitoring**, *Arnold Engineering Development Center Technical Report AEDC-TR-93-19 (AD-A274 591)*, December, 1993.

13. Newland, D.E., **An Introduction to Random Vibrations, Spectral, & Wavelet Analysis**, Longman Scientific & Technical, John Wiley & Sons, New York, NY, 1993.

14. Press, W.H. **Wavelet Transforms**, *Harvard-Smithsonian Center for Astrophysics*, PREPRINT, Cambridge, MA, 1991.

BIBLIOGRAPHY

BIBLIOGRAPHY

Bendat, J.S. and Piersol, A.G. Engineering Applications of Correlation and Spectral Analysis, John Wiley & Sons, New York, NY, 1980.

Childers, D.G., Skinner, D.P., and Kemerait, R.C. **The Cepstrum: A Guide to Processing,** *Proceedings of the IEEE*, vol. 65, no. 10, October 1977, pp. 1428-1443.

Chui, C.K. An Introduction to Wavelets, Academic Press, Inc, San Diego, CA, 1992.

Cody, M.A. **The Fast Wavelet Transform,** *Dr. Dobb's Journal*, April 1992, pp 16-28.

Coifman, R.R., Meyer, Y., and Wickerhauser, V. **Wavelet Analysis and Signal Processing,** Wavelets and Their Applications, Mary Beth Ruskai, et al., editors, Jones and Bartlett Publishers, Boston, MA, 1992.

Courrech, J. **Condition Monitoring of Machinery,** Shock and Vibration Handbook, edited by Harris, C.M., McGraw-Hill Book Company, New York, NY, 1988, pp. 16-1 thru 16-18.

Curtis, A.J. **Concepts in Vibration Data Analysis,** Shock and Vibration Handbook, edited by Harris, C.M., McGraw-Hill Book Company, New York, NY, 1988, pp. 22-1 thru 22-28.

Daubechies, I. Ten Lectures on Wavelets, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

Daubechies, I. **The Wavelet Transform, Time-Frequency Localization and Signal Analysis,** *IEEE Transactions on Information Theory*, vol. 36, no. 5, September 1990, pp. 961-1005.

Janse, C.P. and Kaizer, J.M. **Time-Frequency Distributions of Loudspeakers: The Application of the Wigner Distribution,** *Journal of Audio Engineering Society*, vol 31, no 4, April 1983, pp 198-223.

Joen, J.J. and Shin, Y.S. **Pseudo Wigner-Ville Distribution, Computer Program and its Applications to Time-Frequency Domain Problems,** *Naval Postgraduate School Report NPS-ME-93-002*, March 1, 1993.

Li, C.J. and Ma, J. **Bearing Localized Defect Detection Through Wavelet Decomposition of Vibrations,** Economic Implications of Mechanical Failure Prevention, edited by Pusey, H.C. & Pusey, S.C., Vibration Institute, Willowbrook, IL, 1992, pp. 53-62.

Mallat, S.G. **A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,** *IEEE Transactions on Pattern Analysis and Machine Intel- ligence*, vol. 11, no. 7, July 1989, pp. 674-693.

Najafi, B. and Hakim, H. **A Comparative Study of Non-Parametric Spectral Estimators for Application in Machine Vibration Analysis,** *Machine Systems and Signal Processing*, vol. 6, no. 6, Academic Press, London, 1992, pp. 551-574.

Permann, D. and Hamilton, I. **Wavelet Analysis of Time Series for the Duffing Oscillator: The Detection of Order Within Chaos,** *Physical Review Letters*, vol. 69, no. 18, November 1992, pp 2607-2610.

Proakis, J.G. & Manolakis, D.G. Introduction to Digital Signal Processing, Macmillan Publishing Company, New York, NY, 1988.

Ramirez, R.W. The FFT, Fundamentals and Concepts, Prentice-Hall, Inc., Englewood Cliffs, NJ 1985.

Resnikoff, H.L. **Wavelets and Adaptive Signal Processing,** *Optical Engineering*, vol. 31, no. 6, June 1992, pp. 1229-1234.

Resnikoff, H.L. and Burrus, C.S. **Relationships Between the Fourier Transform and the Wavelet Transform,** *SPIE vol 1348 Advanced Signal-Processing Algorithms, Architectures, & Implementations*, July 1990, pp. 291-300.

Rioul, O. and Vetterli, M. **Wavelets and Signal Processing,** *IEEE Signal Processing Magazine*, October 1991, pp 14-38.

Strang, G. 1989, **Wavelet Transforms versus Fourier Transforms,** *Bulletin of the American Mathematical Society*, vol. 28, no. 2, April 1993, pp. 228-305.

Strang, G. 1989, **Wavelets and Dilation Equations: A Brief Introduction,** *SIAM Review*, vol. 31, December, 1989, pp. 614-627.

Tucker, Michael **Adaptive Wavelet Signal Processing,** U.S. Army Strategic Defense Command, Huntsville, AL 35807-3801, *DTIC Report No. AD-B166-456*, July 1992.

APPENDIXES

APPENDIX A

FIGURES

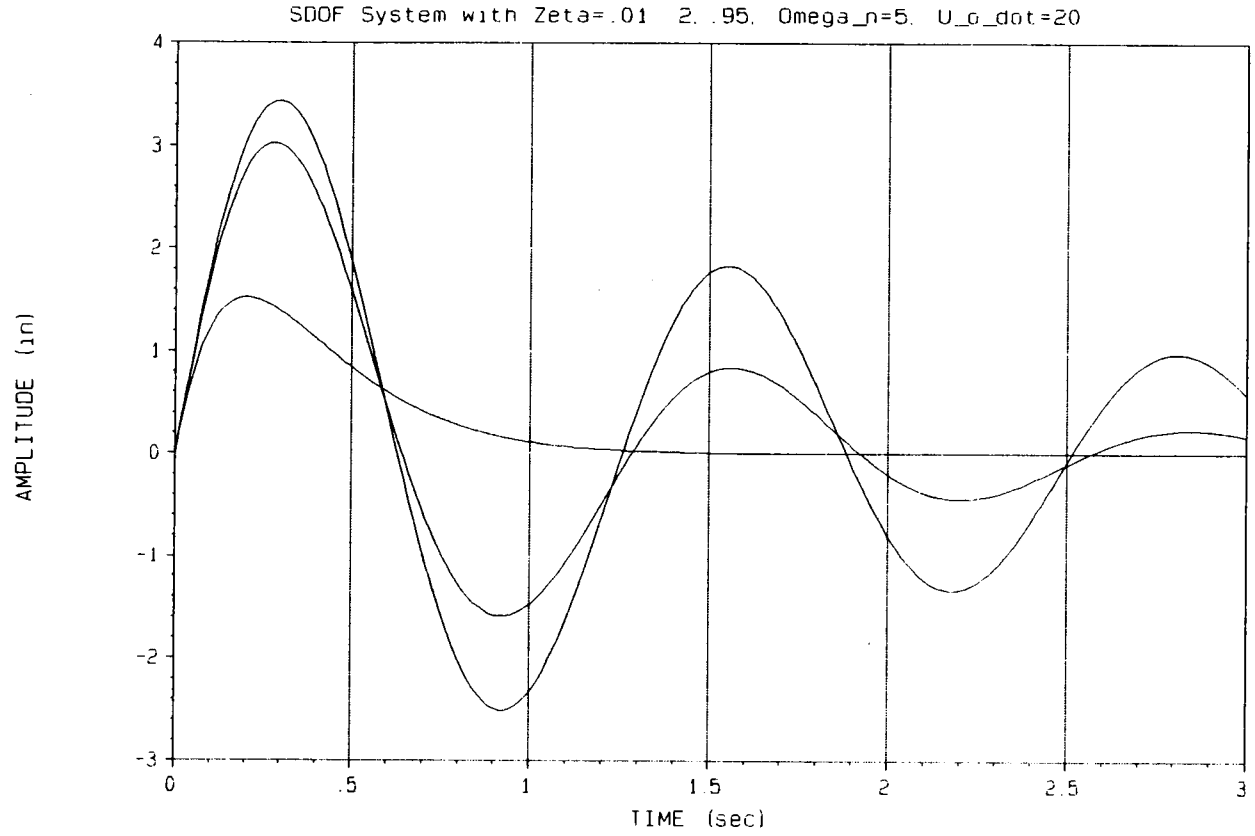


Figure 1. SDOF Response Characteristics

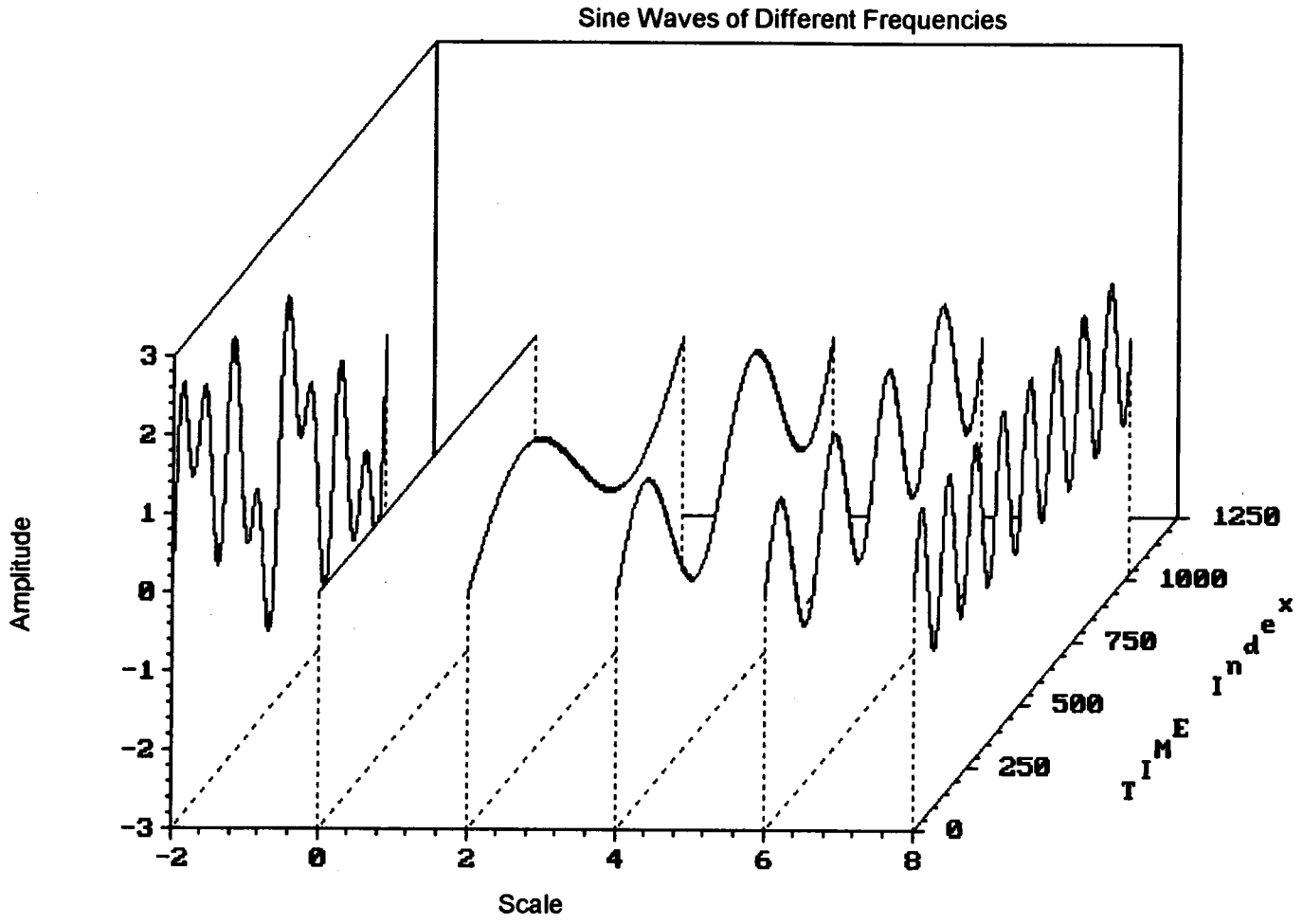


Figure 2. Amplitude-Frequency-Time Space for Fourier Analysis

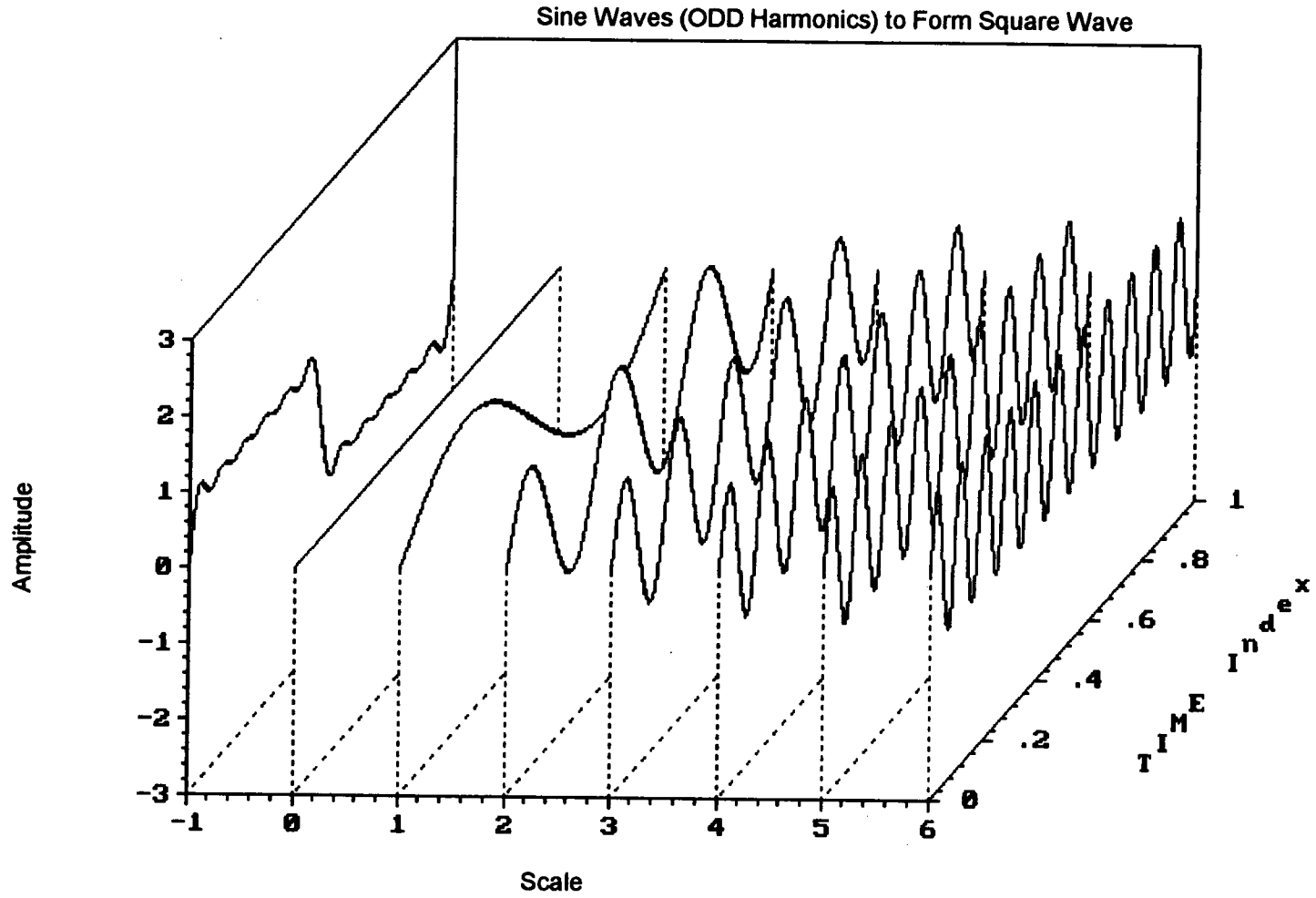
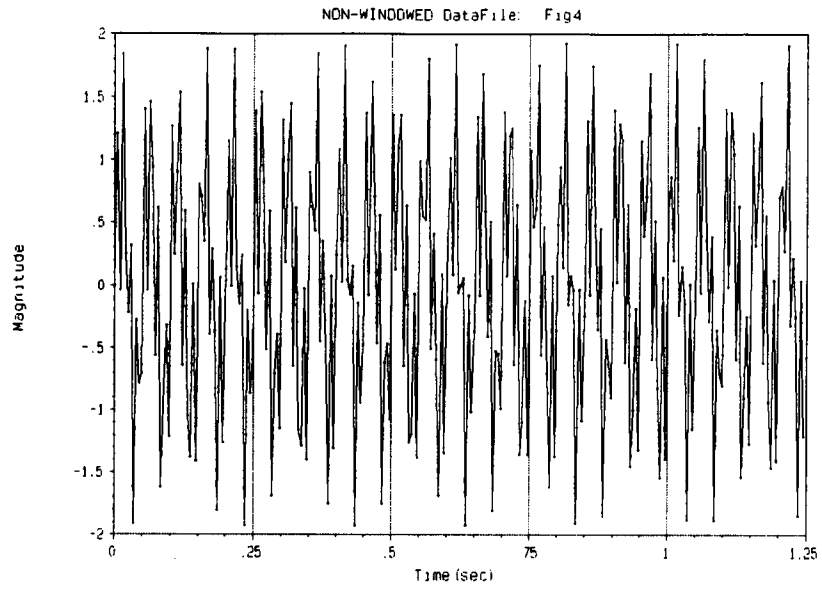
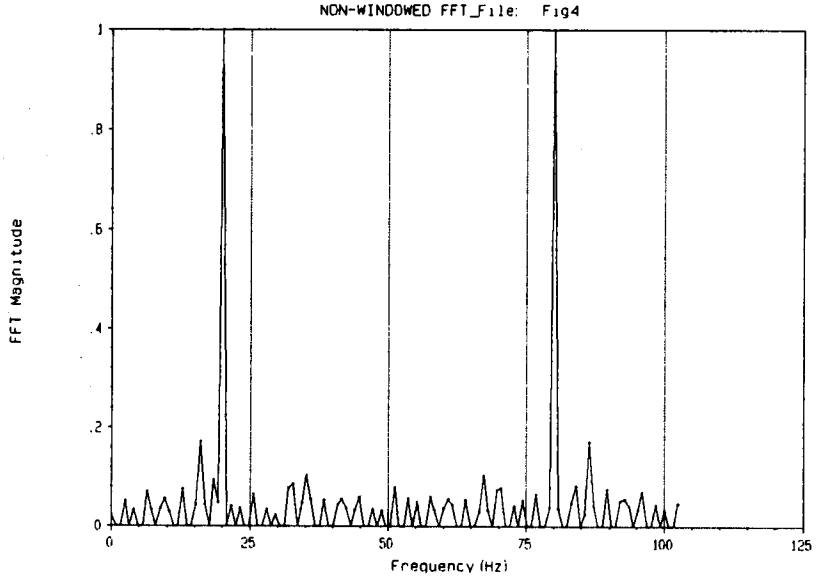


Figure 3. Square Wave Approximation as Sum of Sine Waves

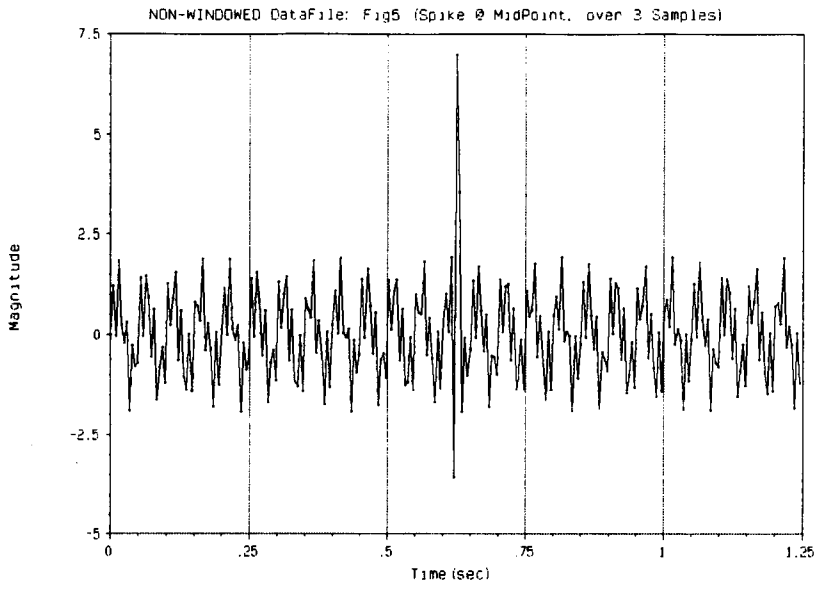


a.

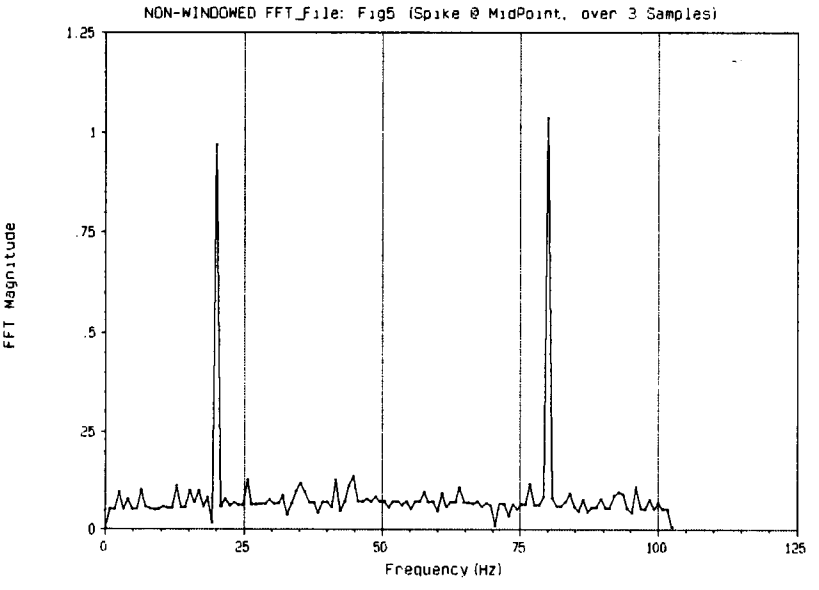


b.

Figure 4. FFT Spectrum for a Two Component Sinusiod

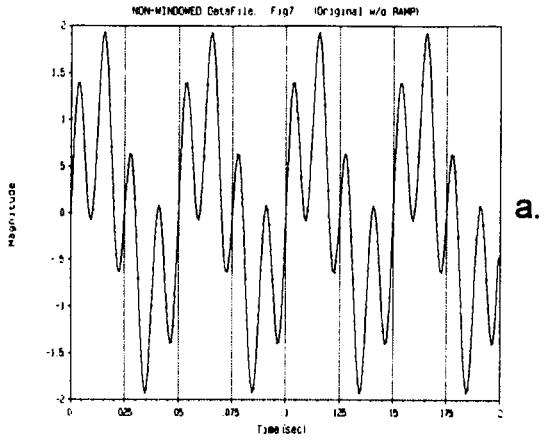


a.

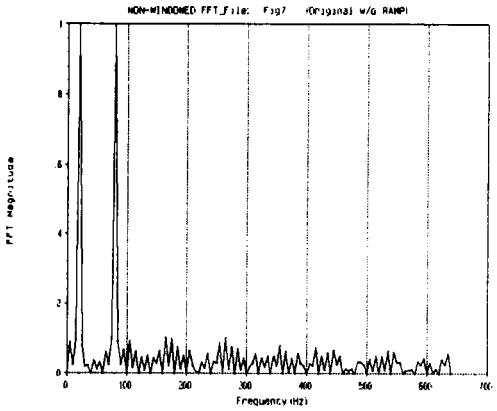


b.

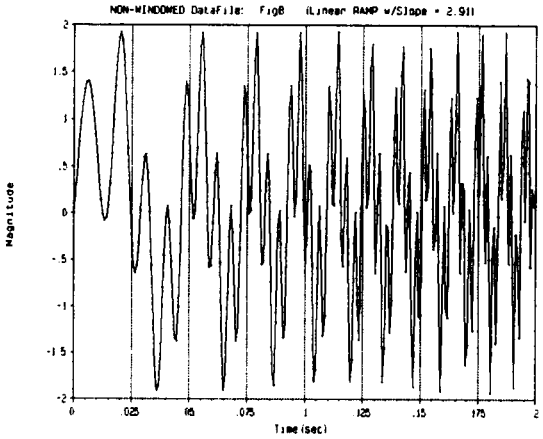
Figure 5. FFT Spectrum for a Two Component Sinusoid Plus Impulse



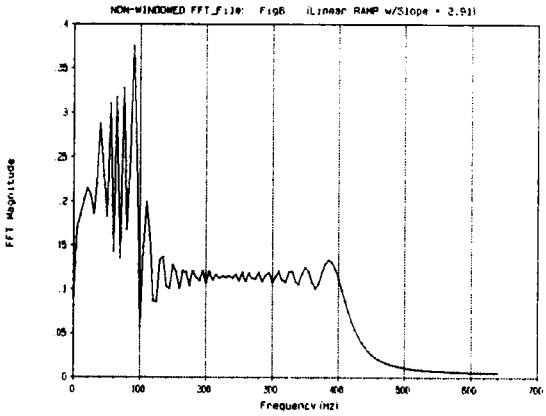
a.



b.

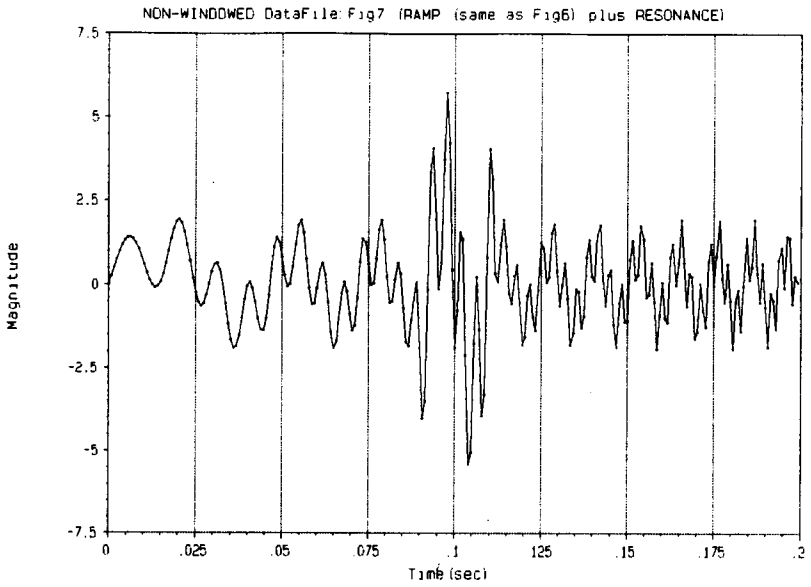


c.

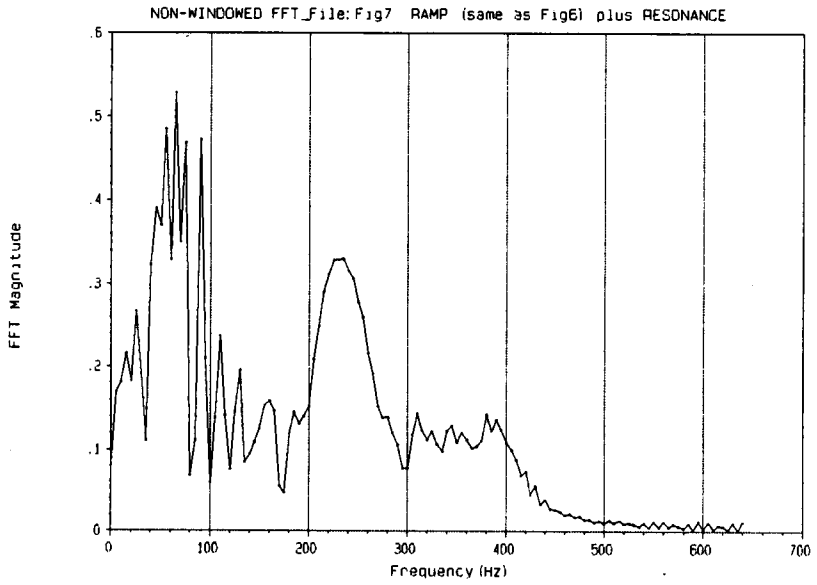


d.

Figure 6. FFT Spectrum for a Ramped Two Component Sinusoid



a.



b.

Figure 7. FFT Spectrum for a Ramped Two Component Sinusoid with Resonance

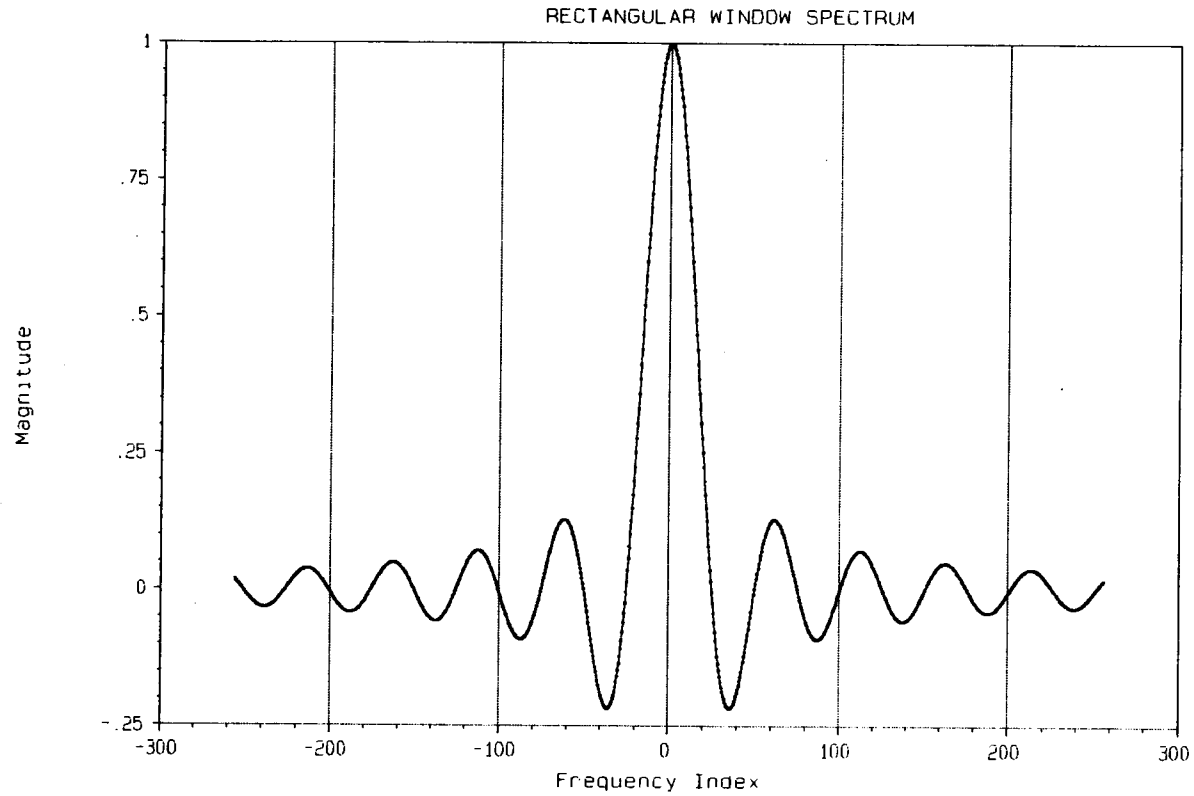


Figure 8. FFT Spectrum of the Rectangular Window

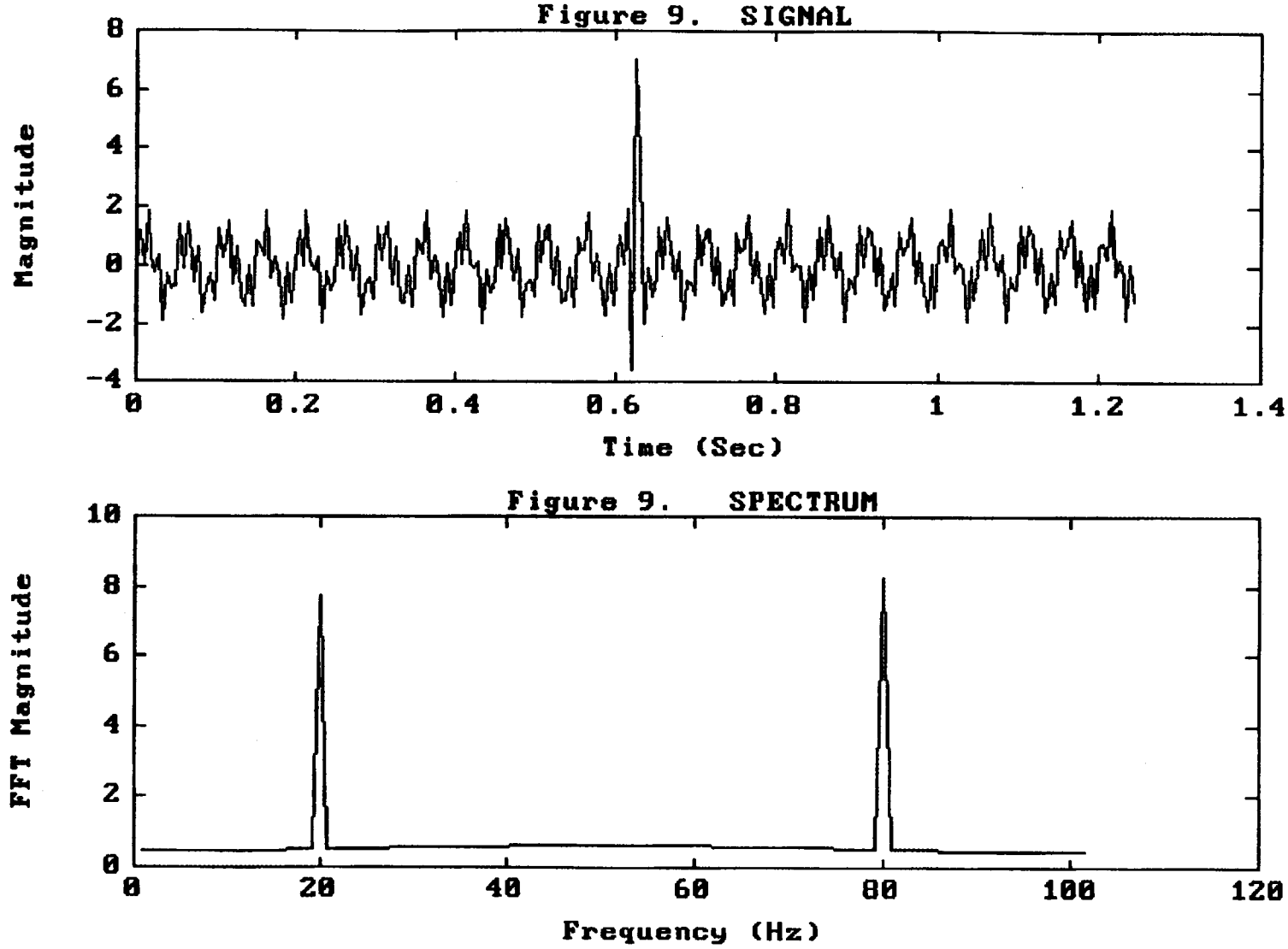


Figure 9. Interpolation/Smoothing of Inpulse via FFT Zero Padding

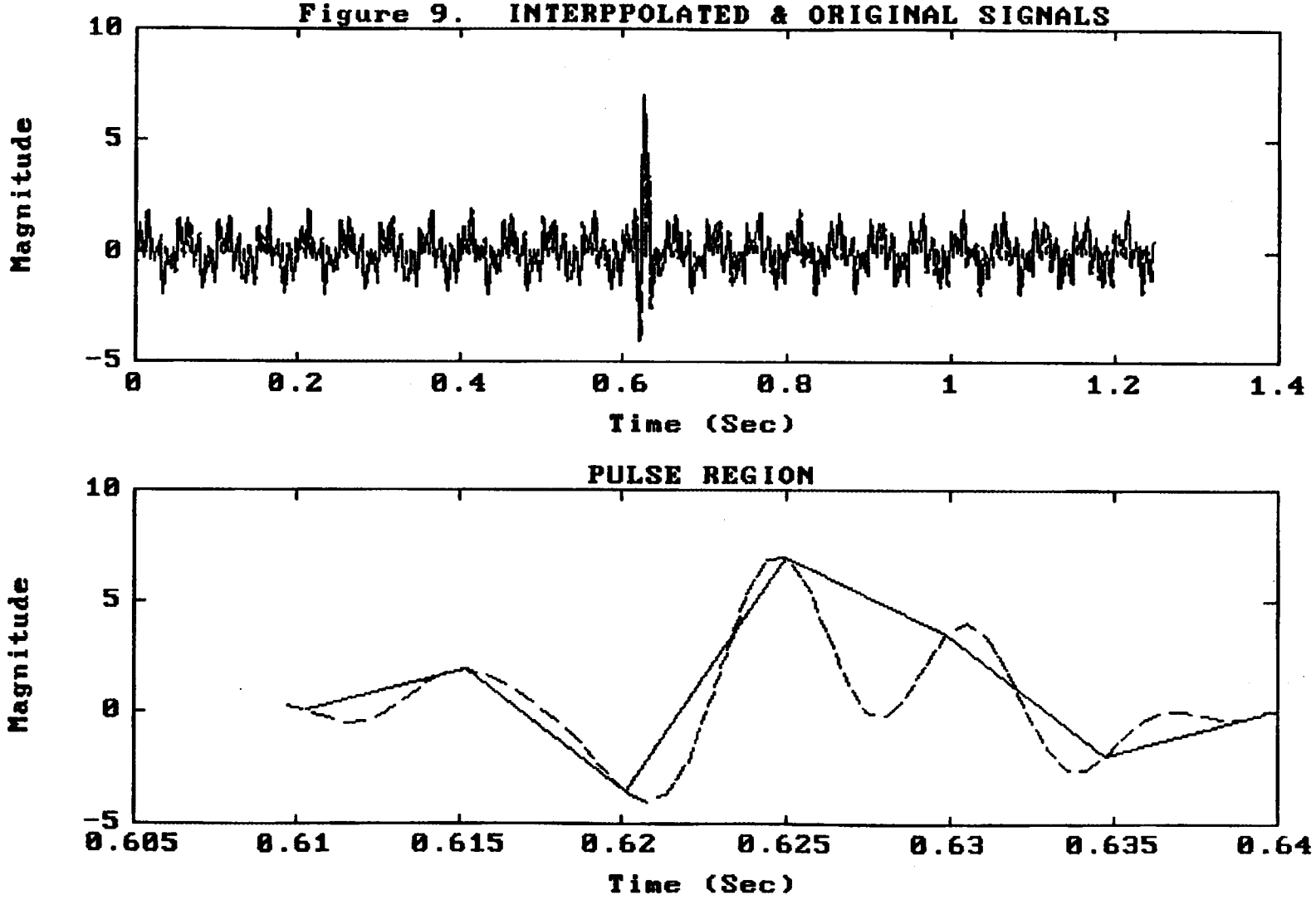


Figure 9 (Continued). Interpolation/Smoothing of Impulse via FFT Zero Padding

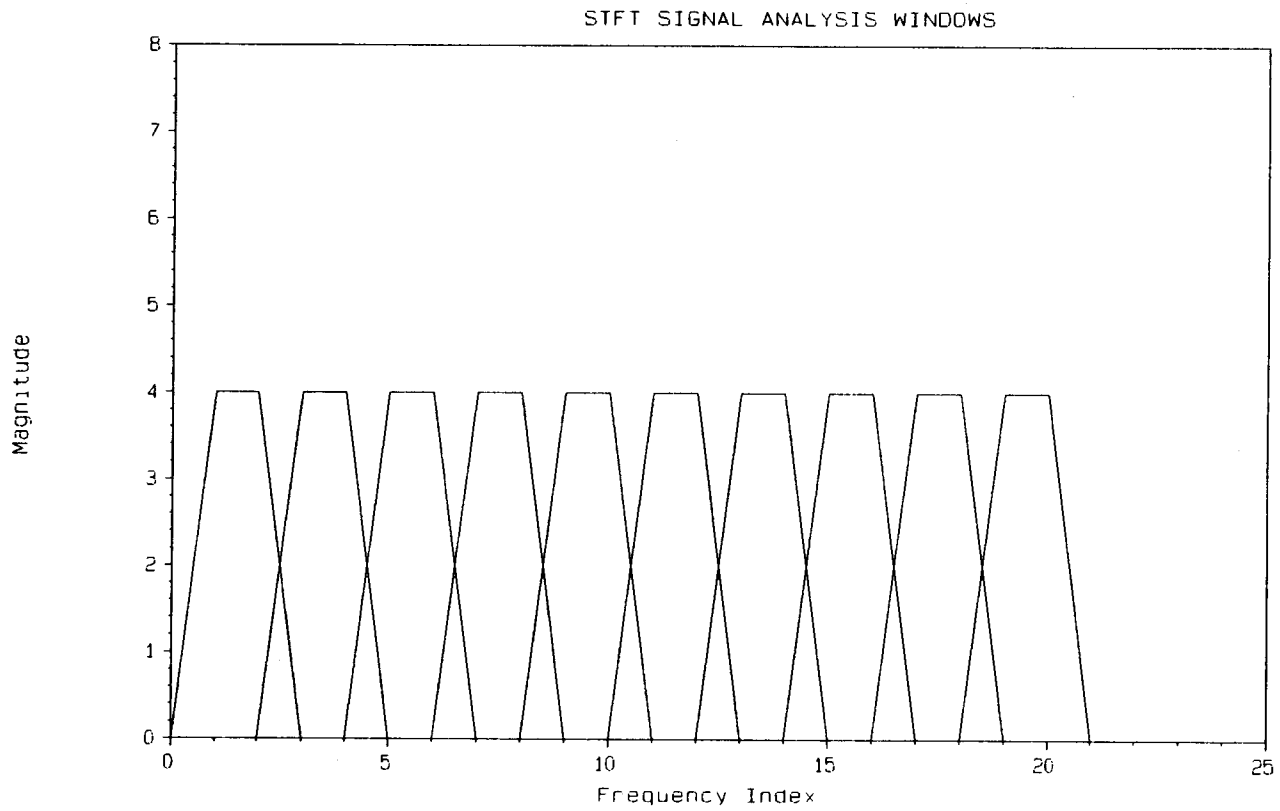


Figure 10. STFT Signal Analysis Windows

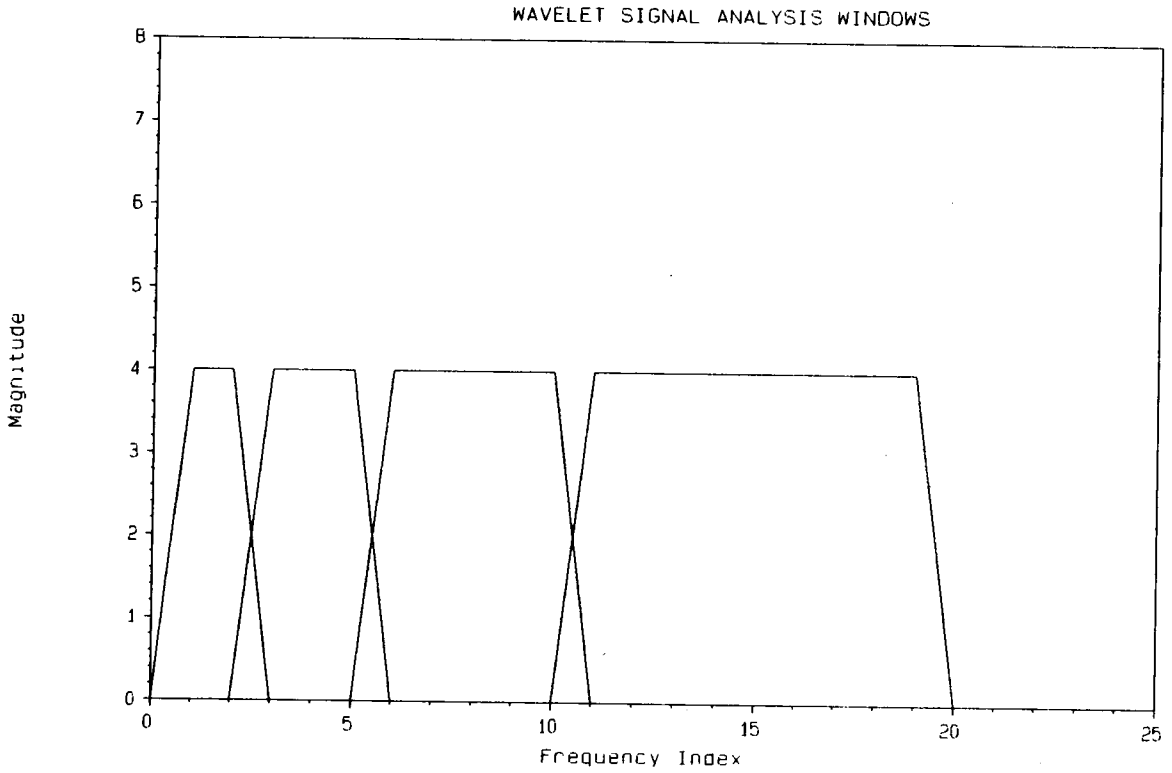
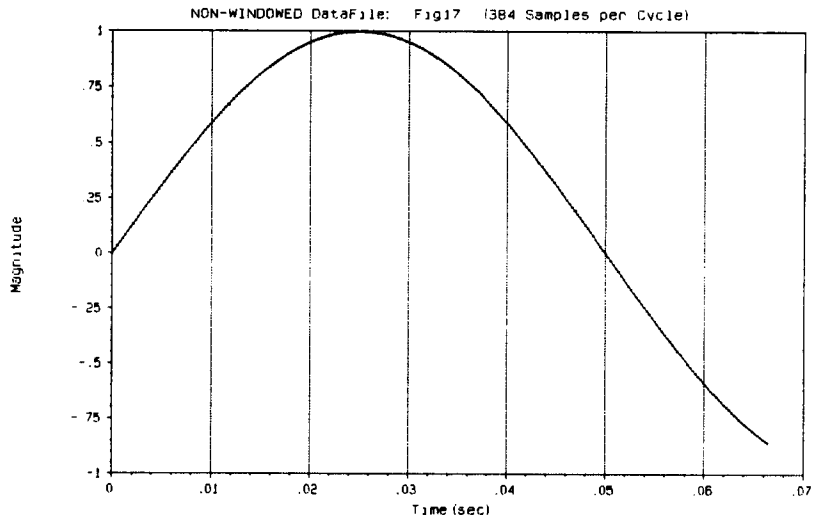
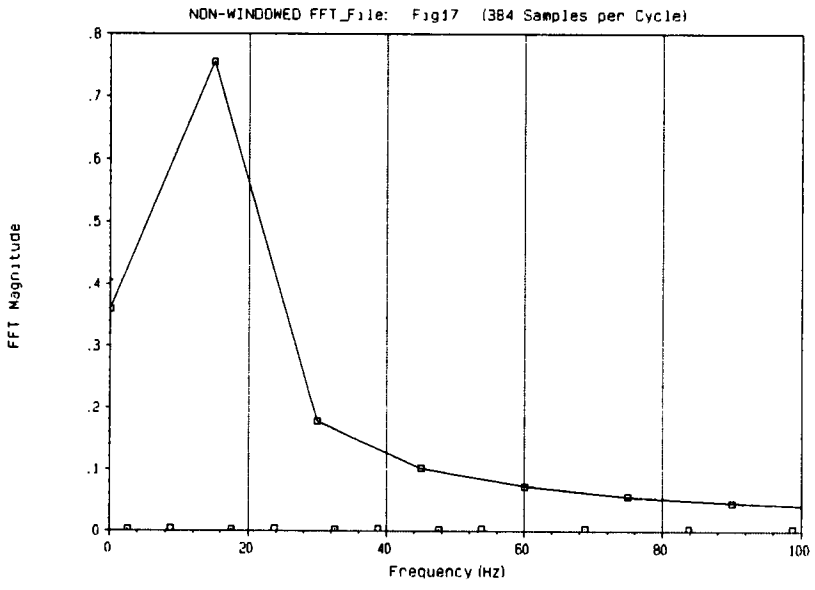


Figure 11. Wavelet Signal Analysis Windows

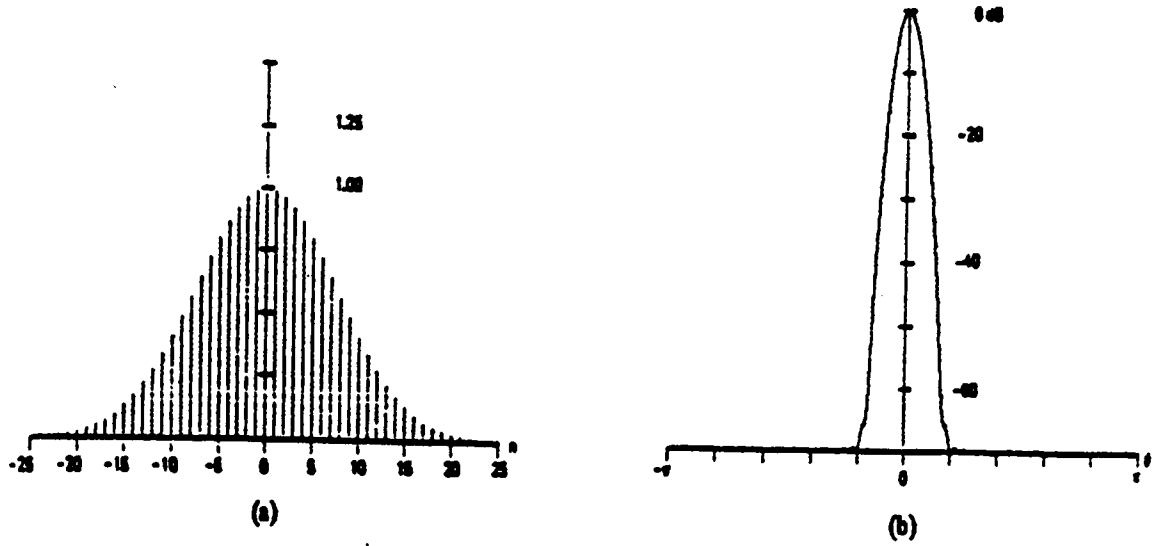


a.

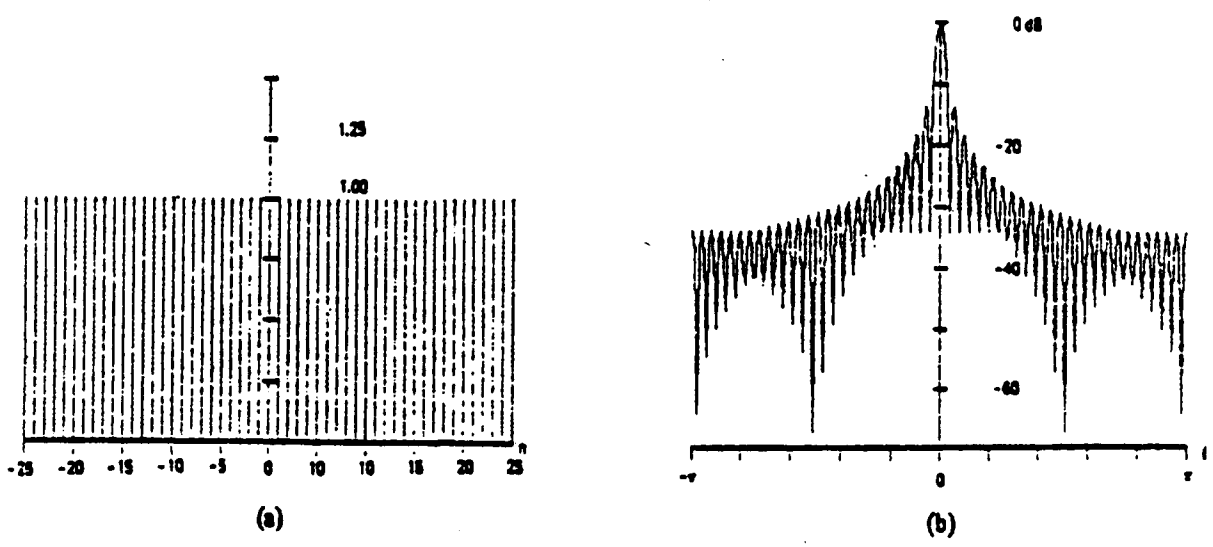


b.

Figure 12. FFT Spectrum of Rectangularly Windowed Non-Integral Cycle Sinusiods

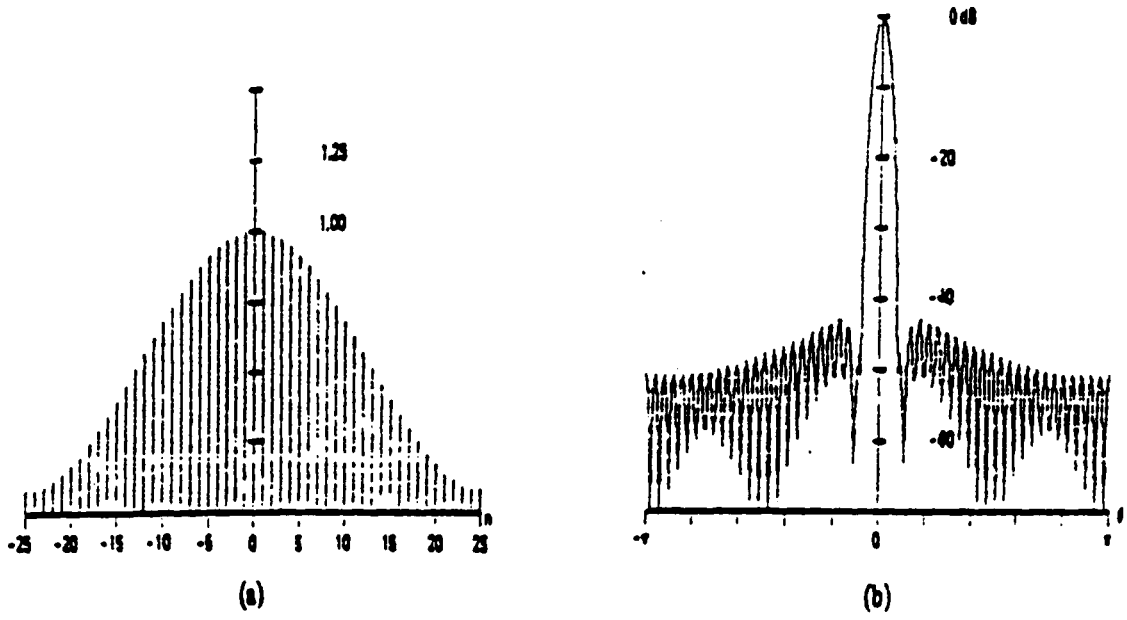


(a) Gaussian window. (b) Log-magnitude of transform ($\alpha = 3.5$).

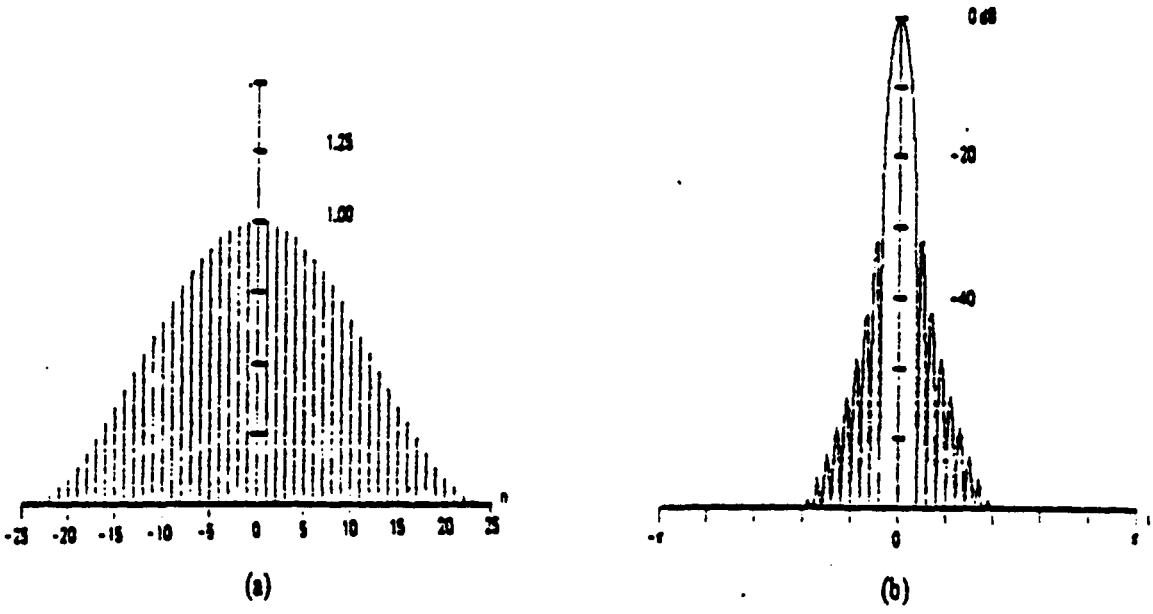


(a) Rectangle window. (b) Log-magnitude of transform.

Figure 13. Common Windowing Functions and Their Spectra

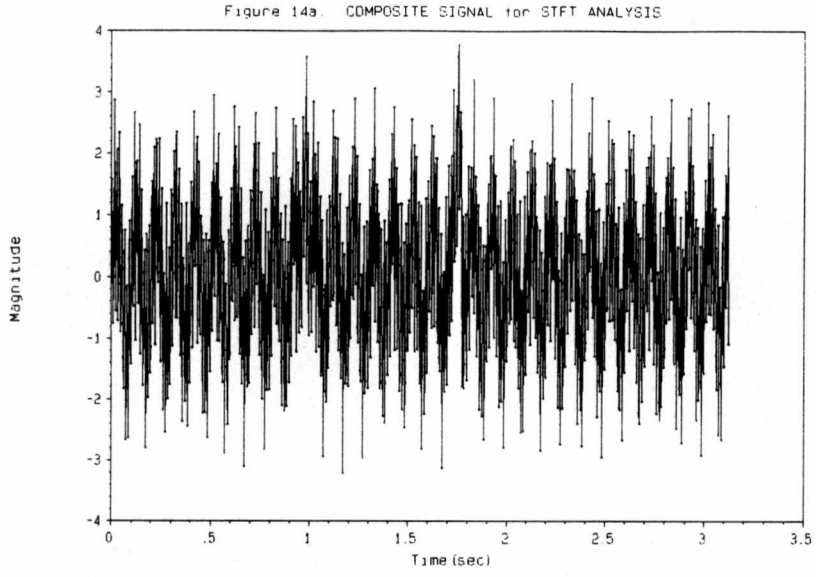


(a) Hamming window. (b) Log-magnitude of Fourier transform.

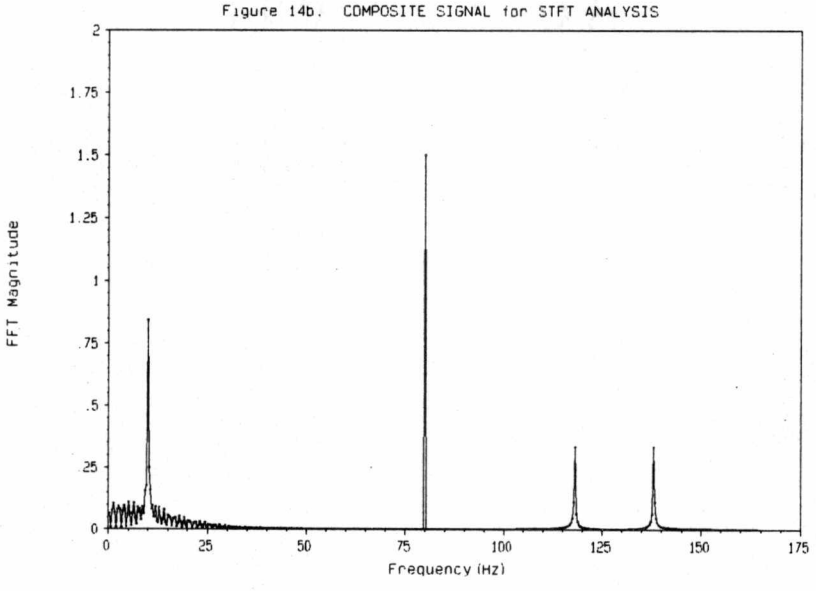


(a) $\cos^2(n\pi/N)$ window. (b) Log-magnitude of transform.

Figure 13 (Continued). Common Windowing Functions and Their Spectra

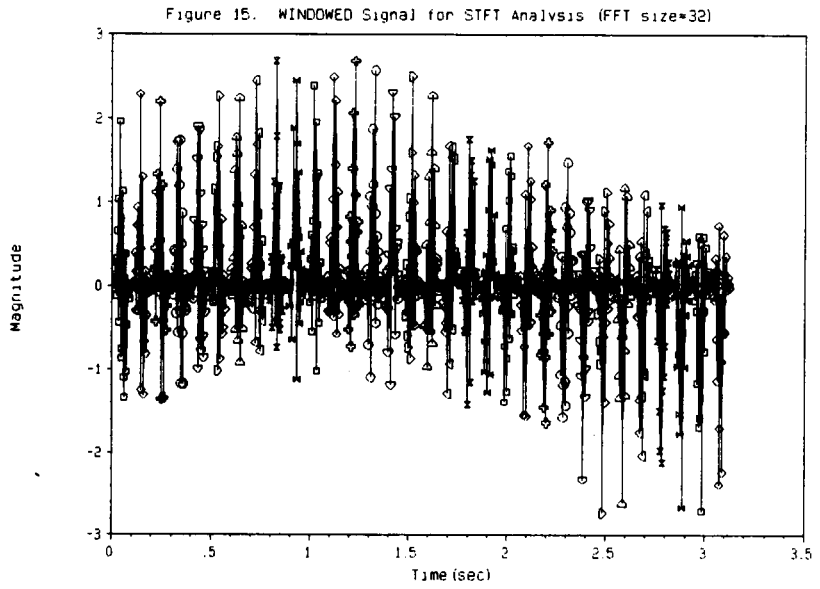


a.

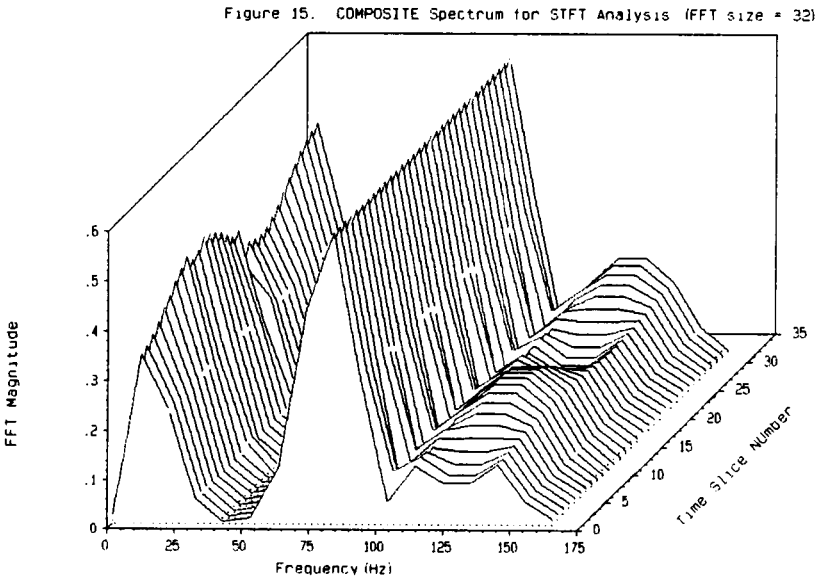


b.

Figure 14. STFT Example Signal and FFT Spectrum

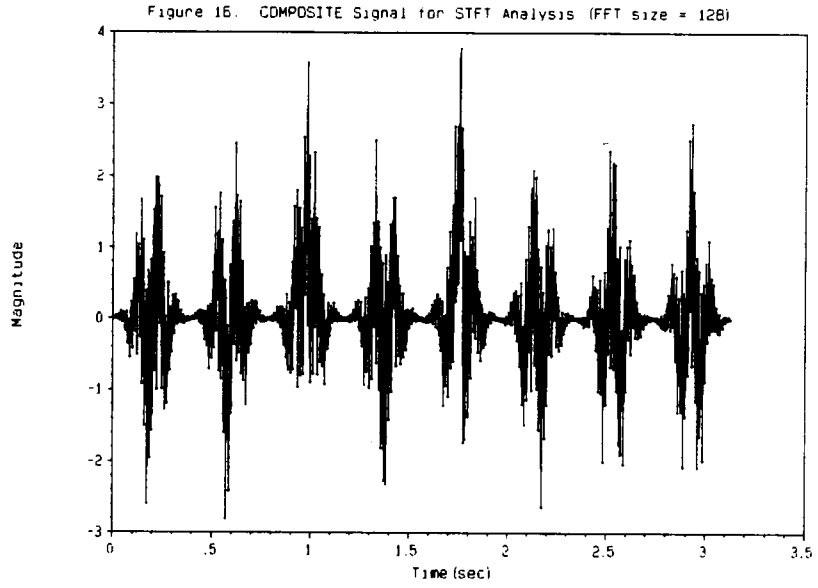


a.

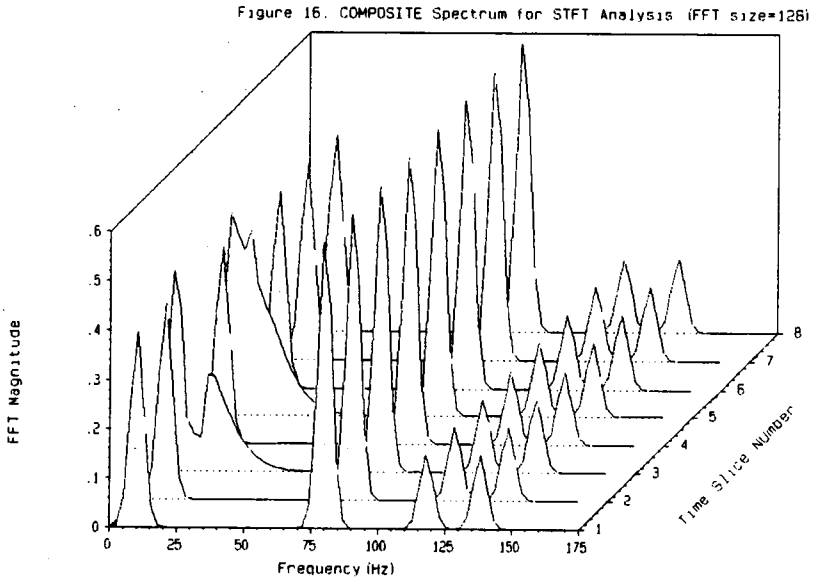


b.

Figure 15. STFT Spectrogram Example



a.



b.

Figure 16. STFT Spectrogram Example with Improved High Frequency Time Location

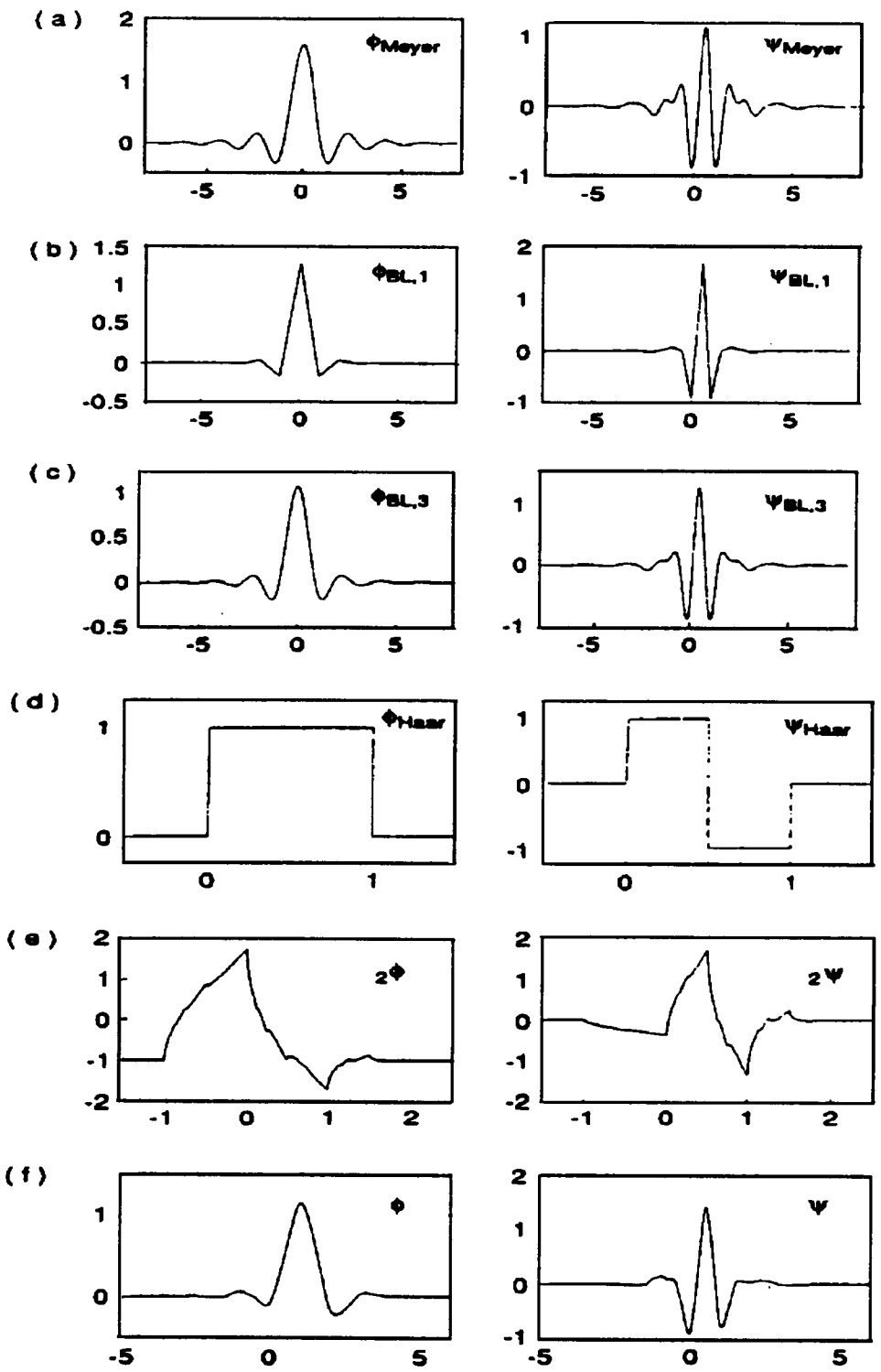


Figure 17. Wavelet Scaling and Detail Functions

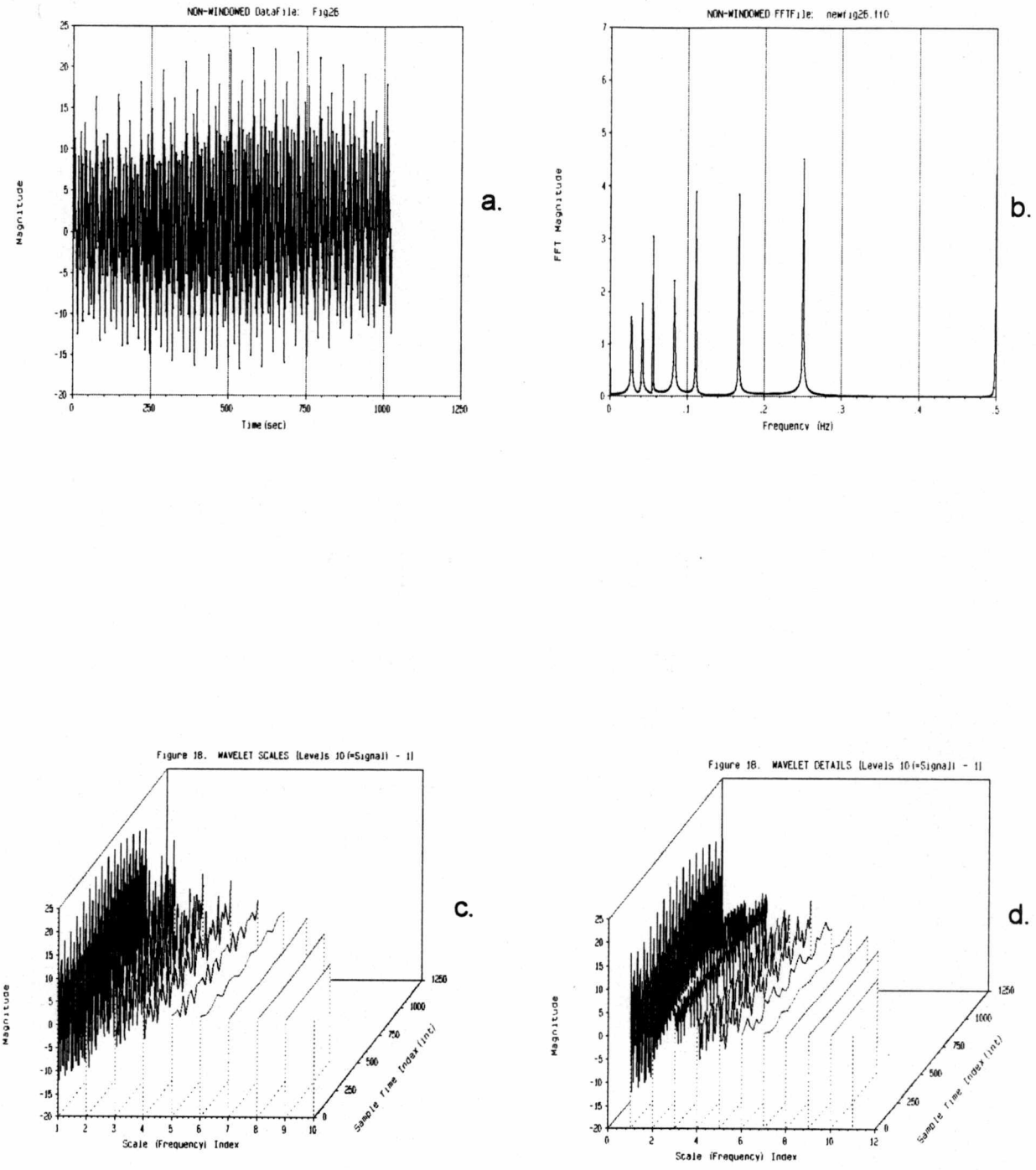
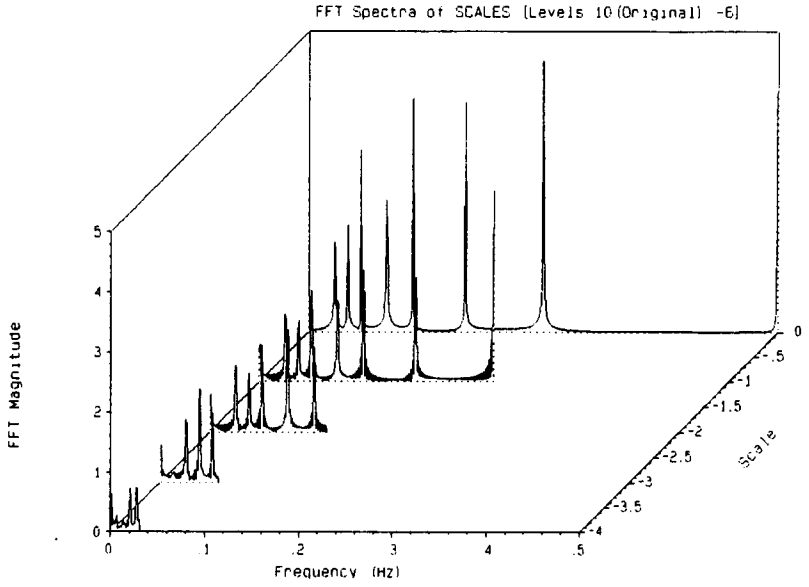
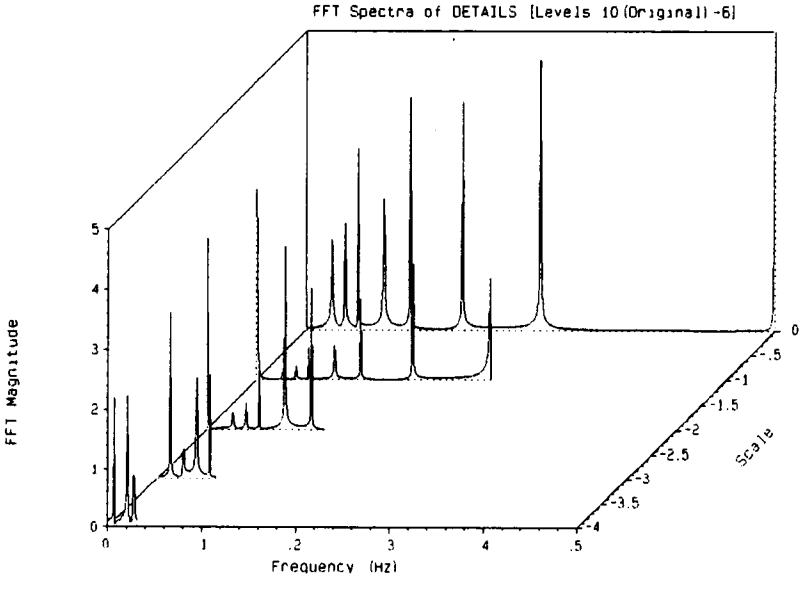


Figure 18. Haar Wavelet Analysis of a General Digitized Signal



e.



f.

Figure 18 (Continued). Haar Wavelet Analysis of a General Digitized Signal

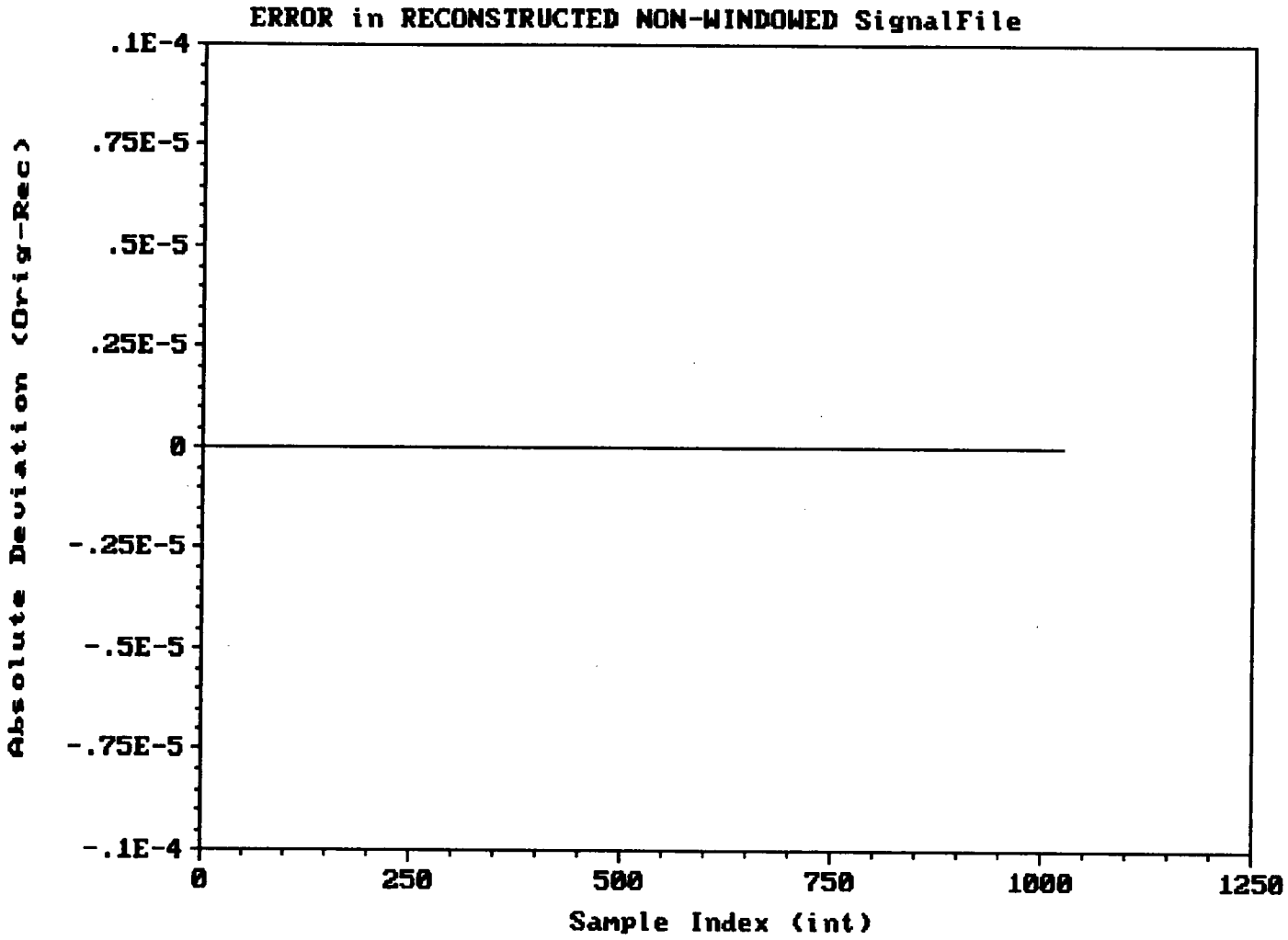


Figure 19. Error in Reconstruction Following Wavelet Analysis for Signal of Figure 19

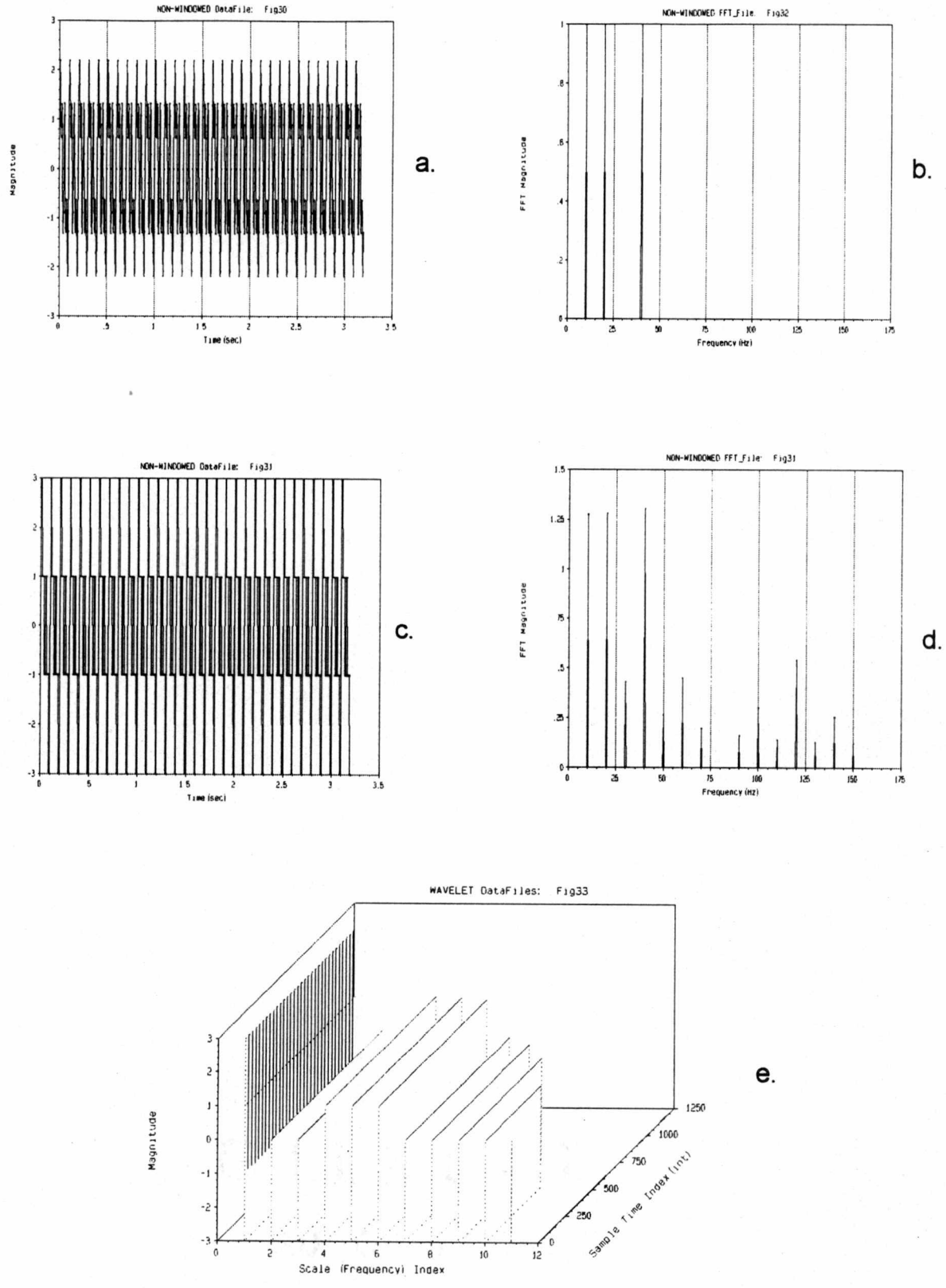


Figure 20. Comparison of Fourier Analysis for Sine Waves and Haar Wavelet Analysis for Square Waves

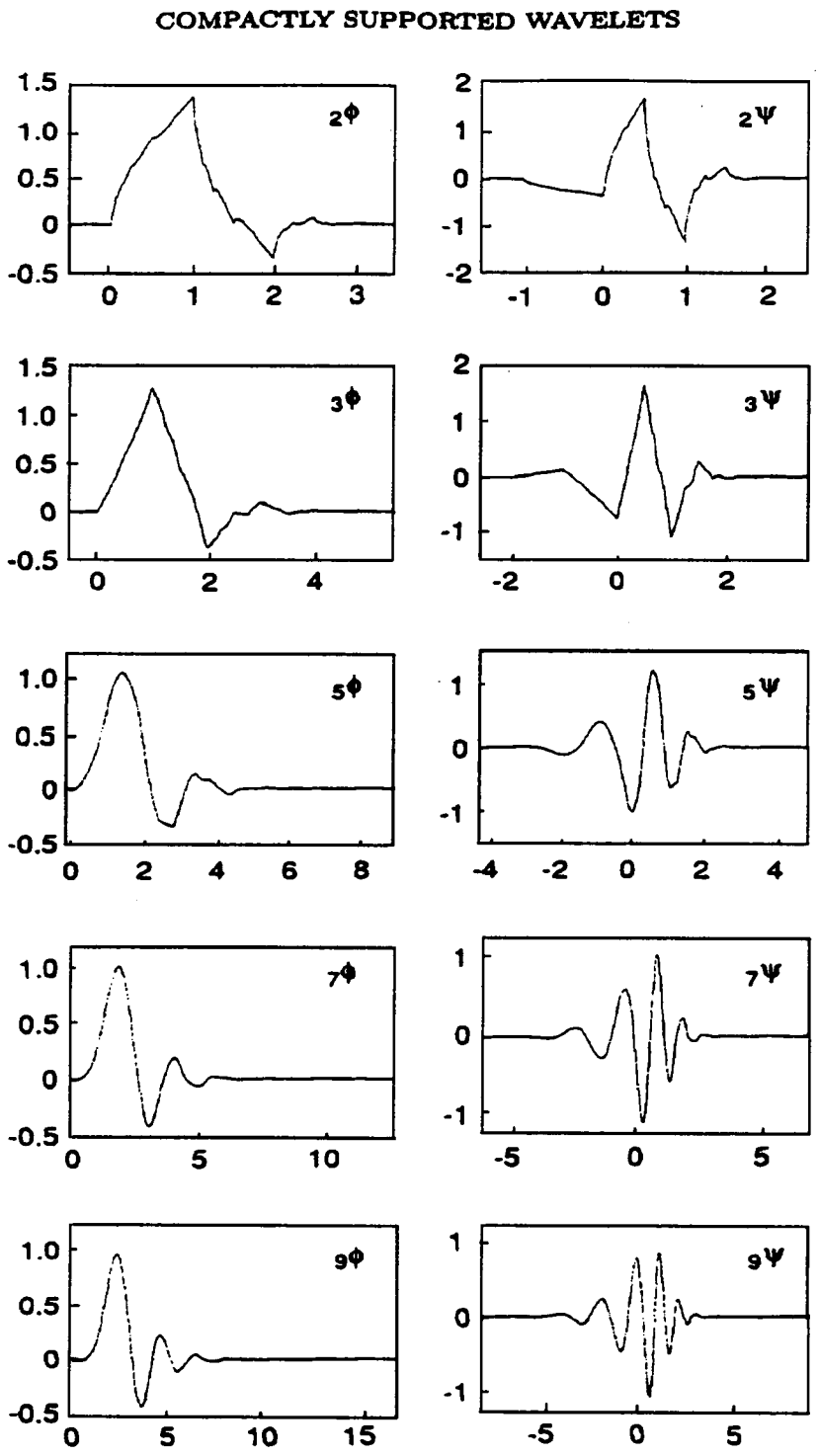
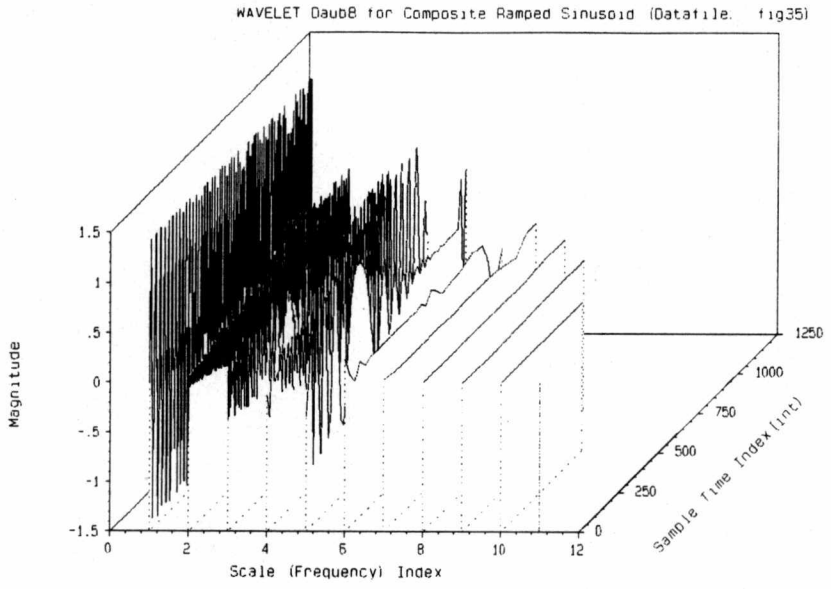
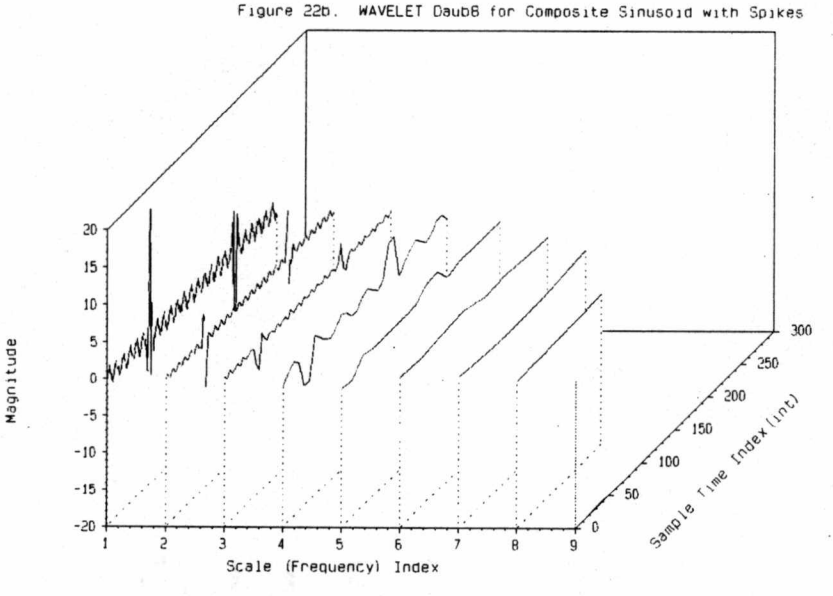


Figure 21. Daubechies Wavelets



a.

Figure 22b. WAVELET Daub8 for Composite Sinusoid with Spikes



b.

Figure 22. DAUB8 Wavelet Applied to Ramped and Impulsed Signals

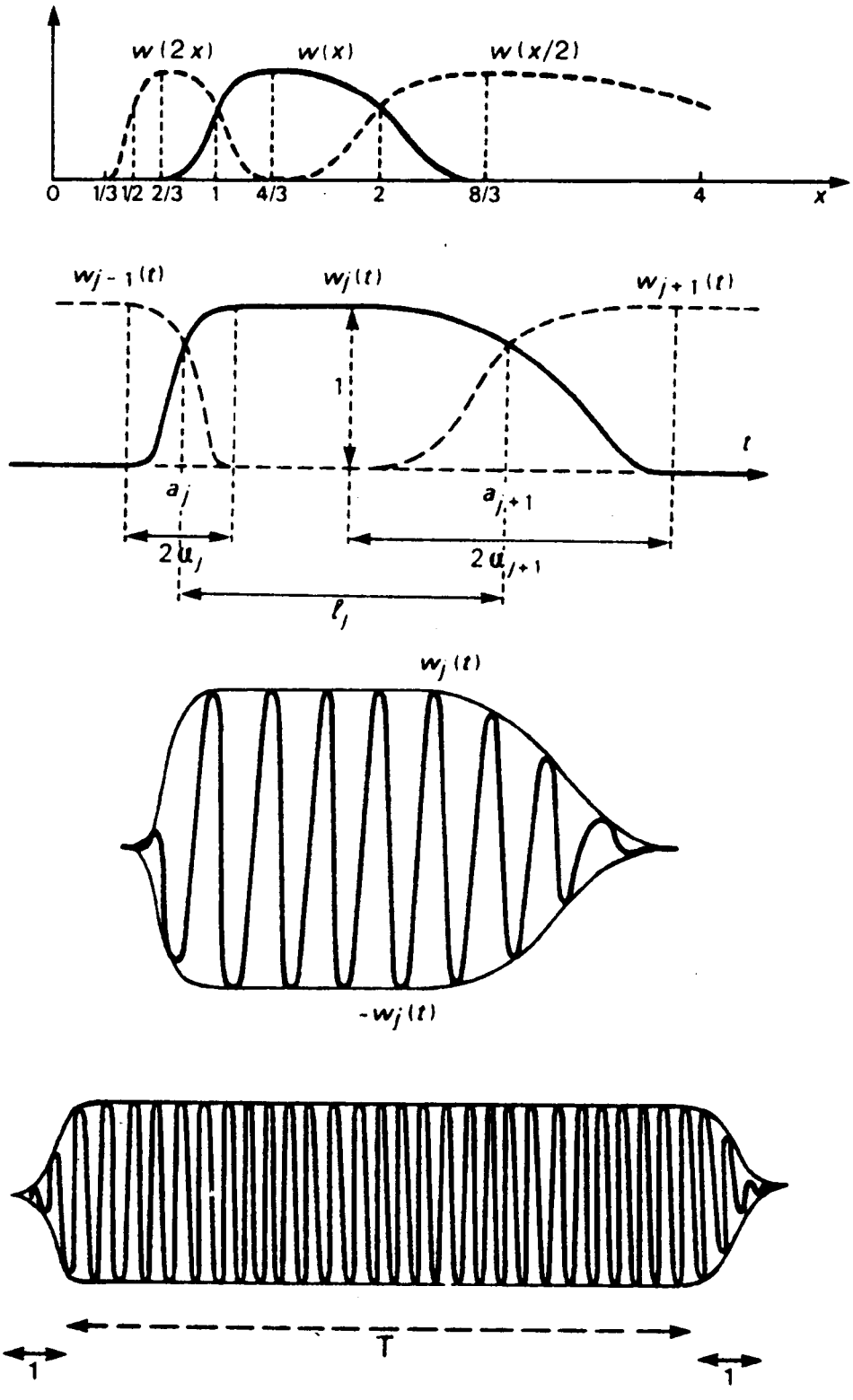


Figure 23. Malvar Wavelets

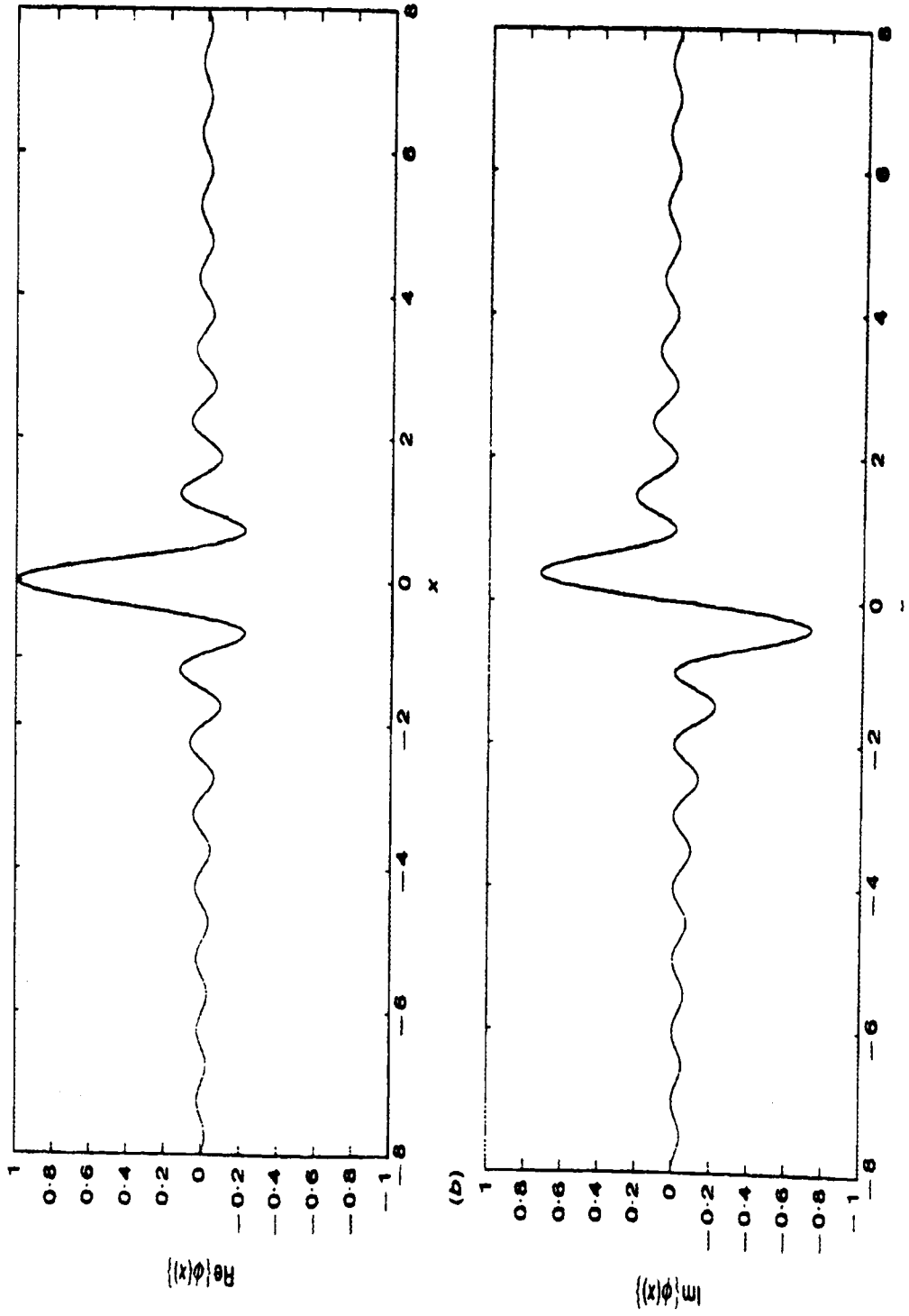


Figure 24. Real and Imaginary Parts of the Harmonic Wavelet Scaling Function

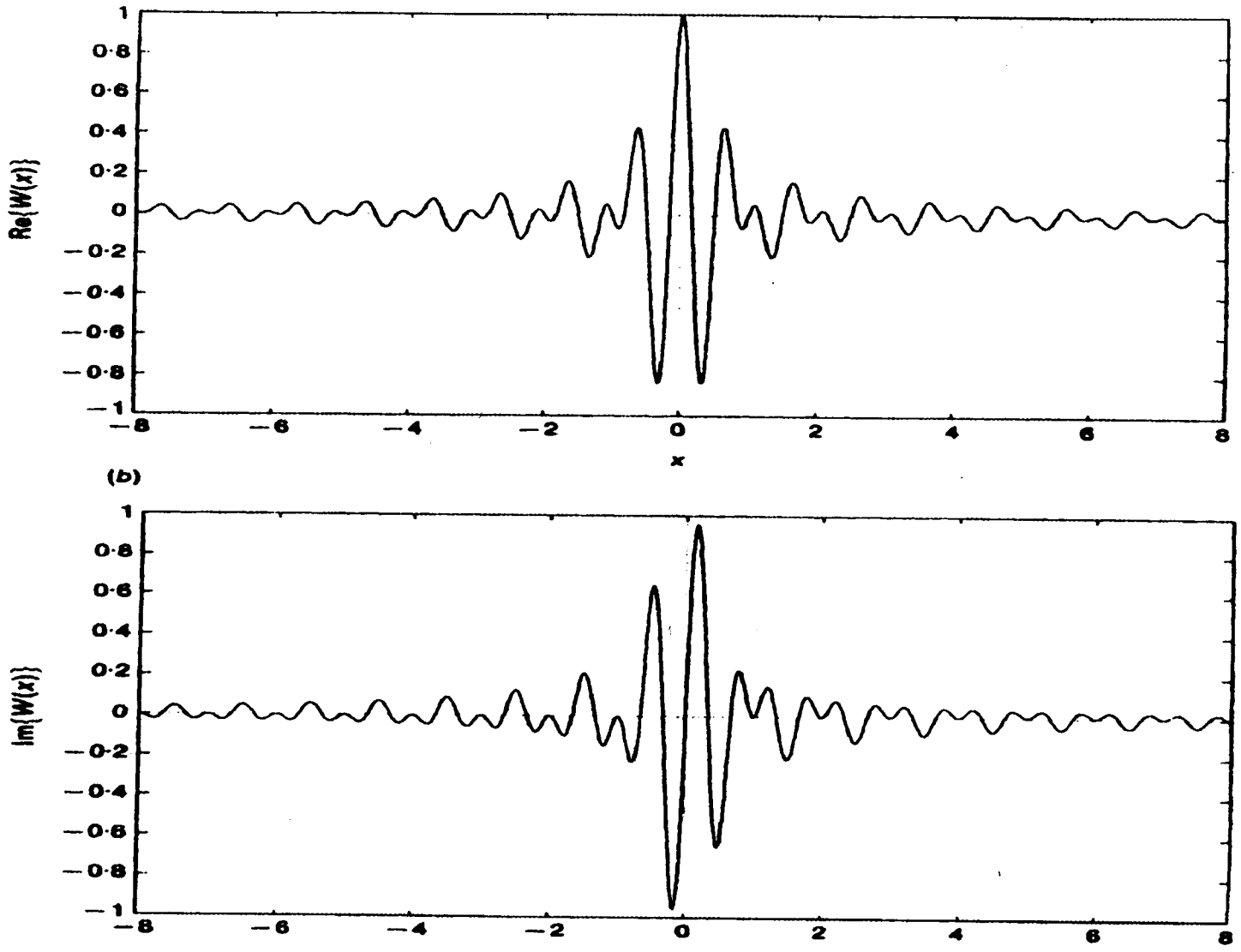


Figure 25. Real and Imaginary Parts of the Harmonic Wavelet Detail Function

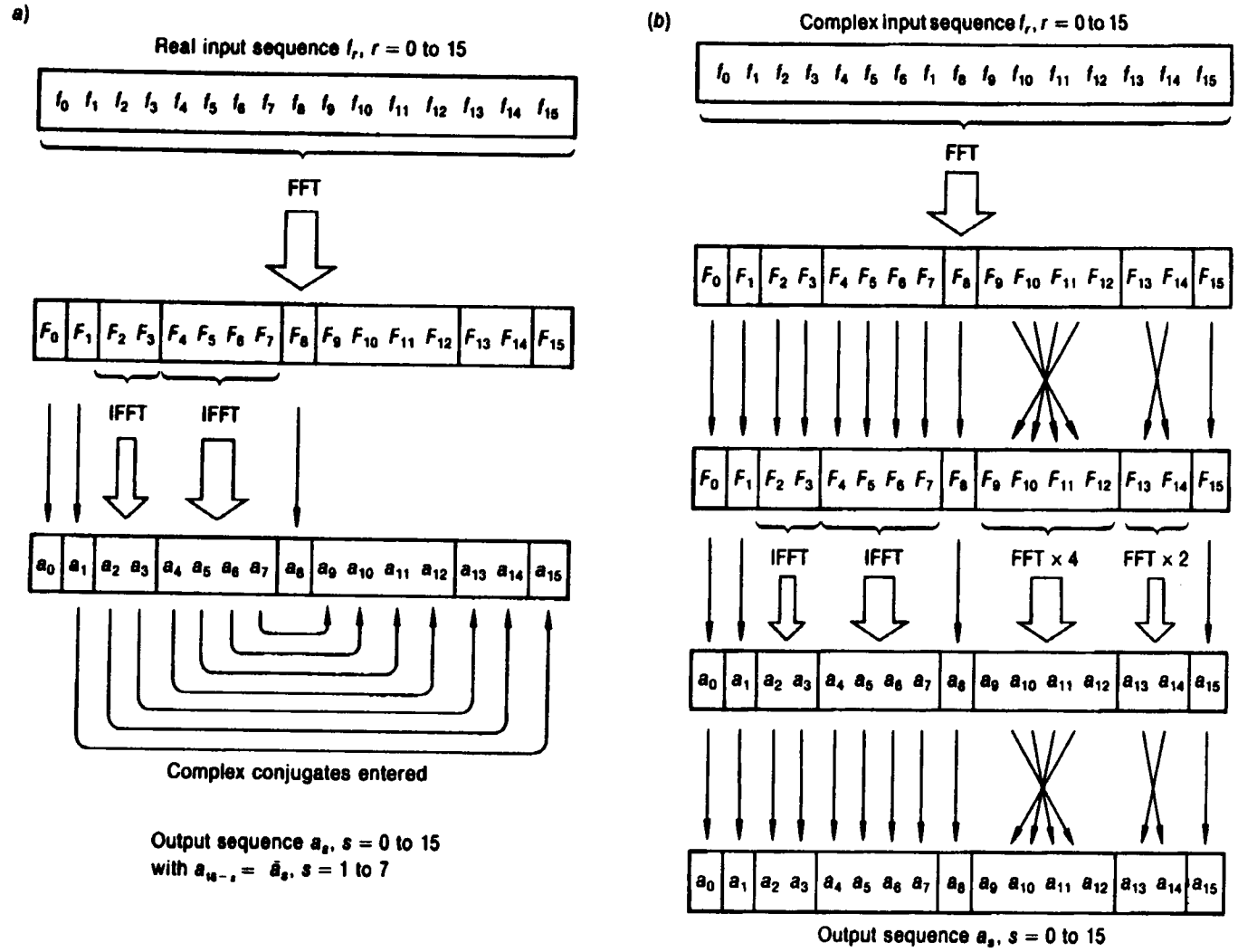
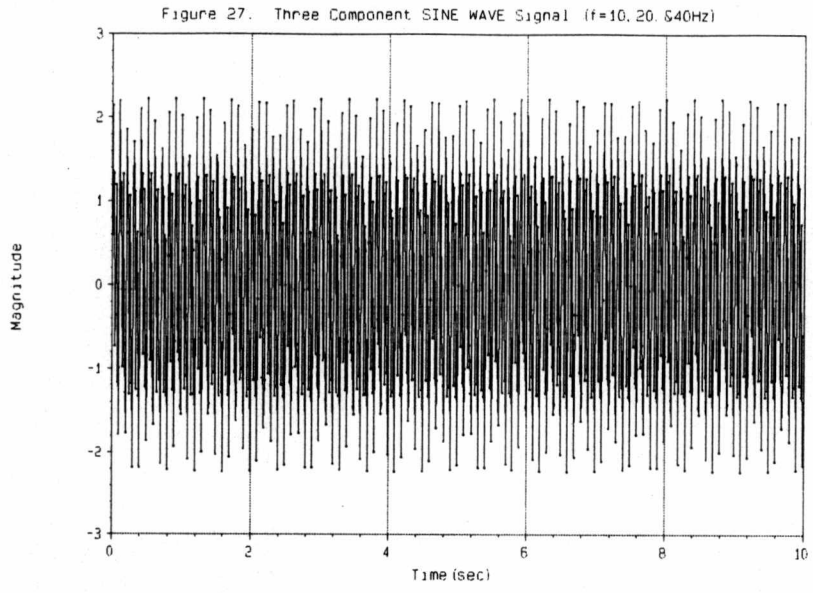


Figure 26. Harmonic Wavelet Algorithm via the FFT



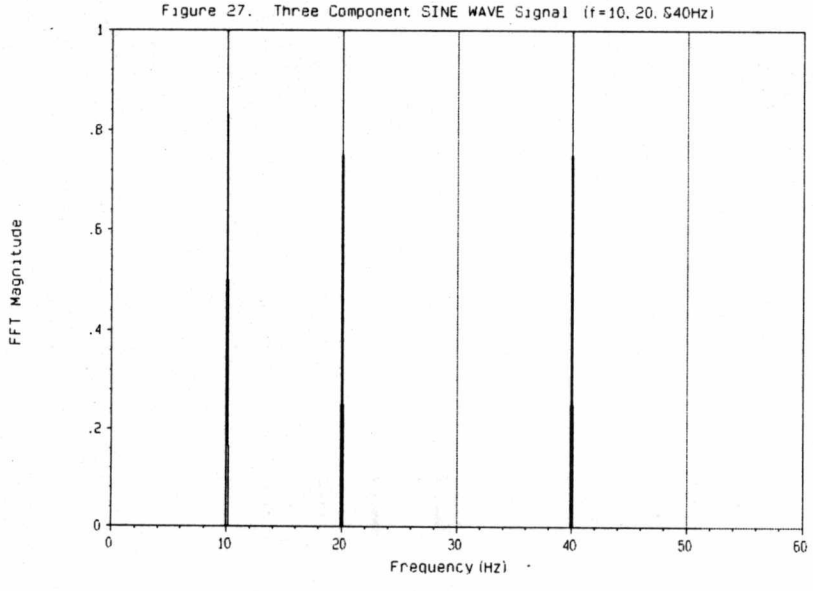


Figure 27. Three Component Sine Wave (Harmonics) and FFT Spectrum

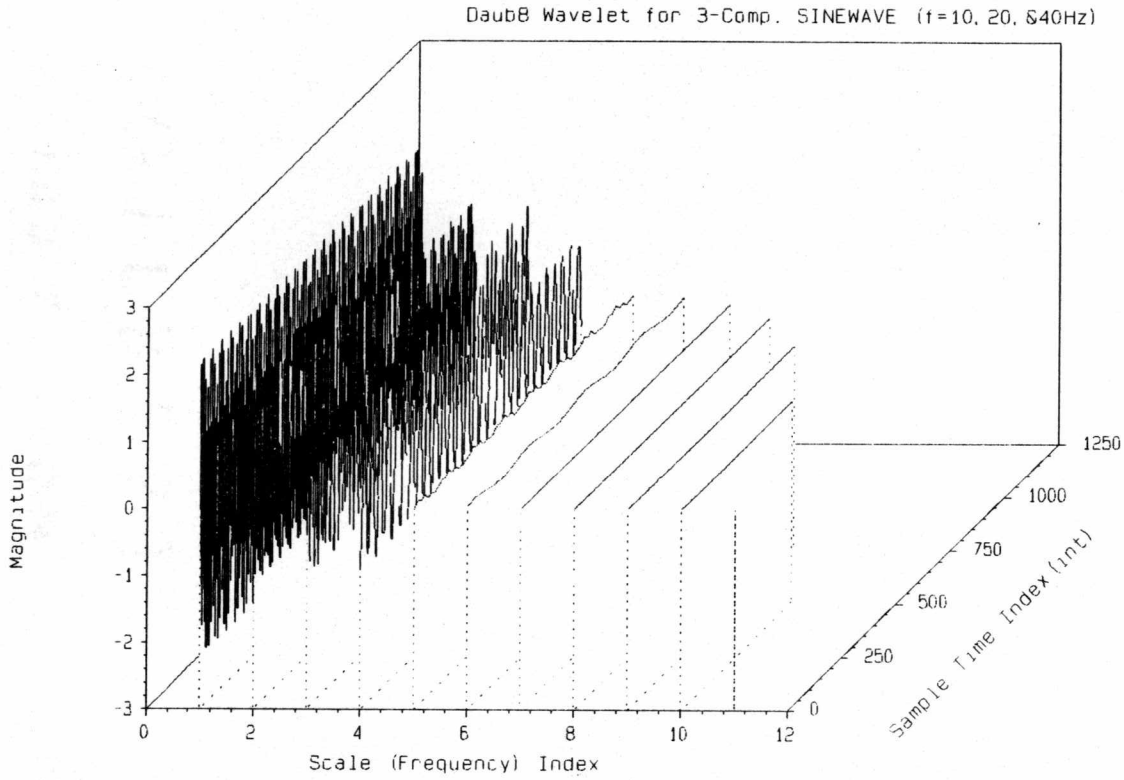


Figure 28. DAUB8 Wavelet Analysis of Figure 27 Signal

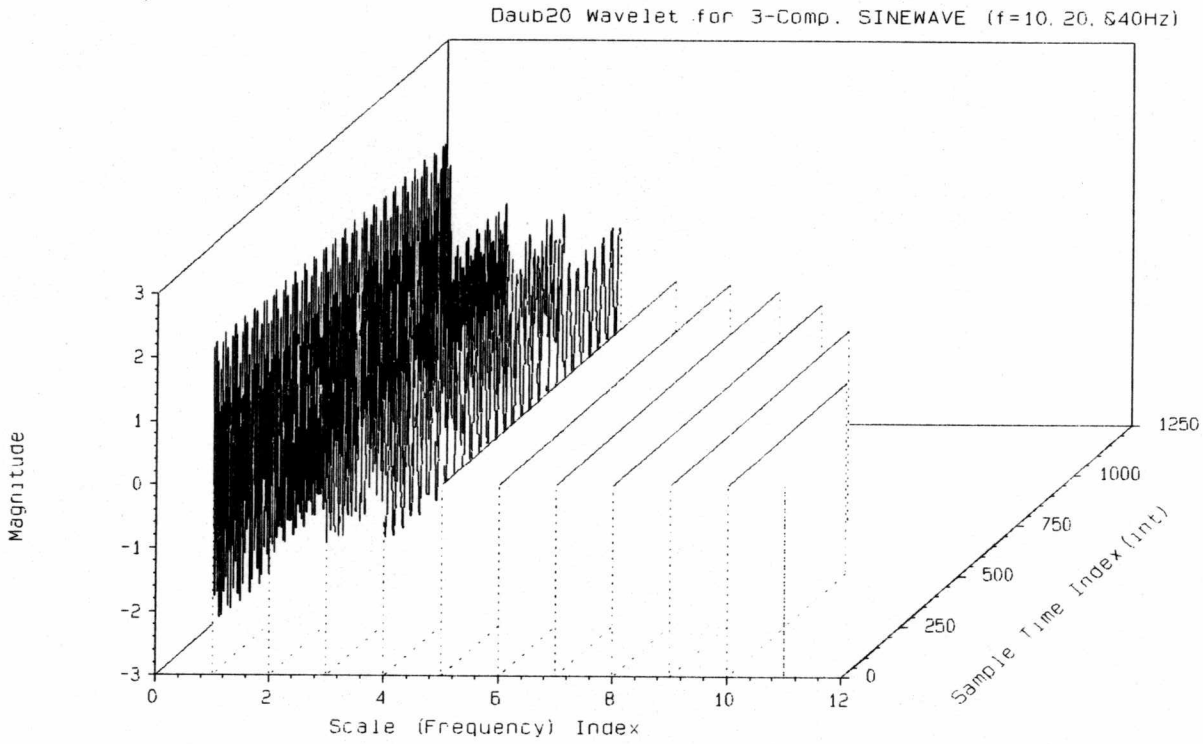


Figure 29. DAUB20 Wavelet Analysis of Figure 27 Signal

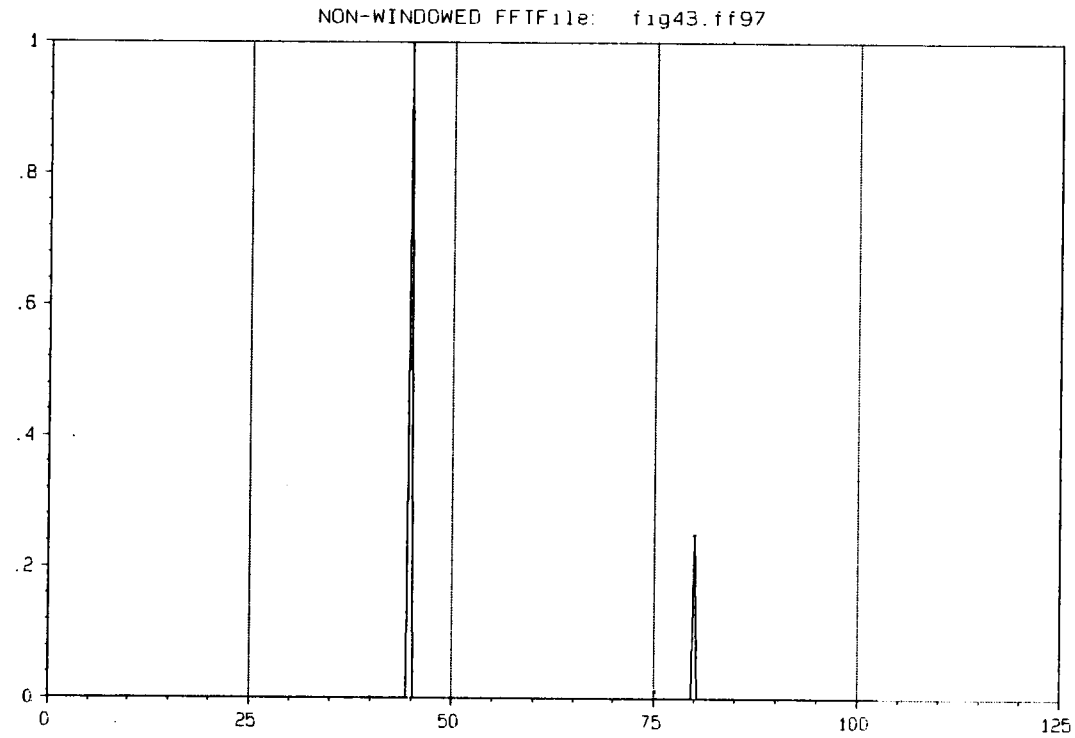


Figure 30. FFT of Level 10 DAUB20 Wavelet Analysis of Figure 27 Signal

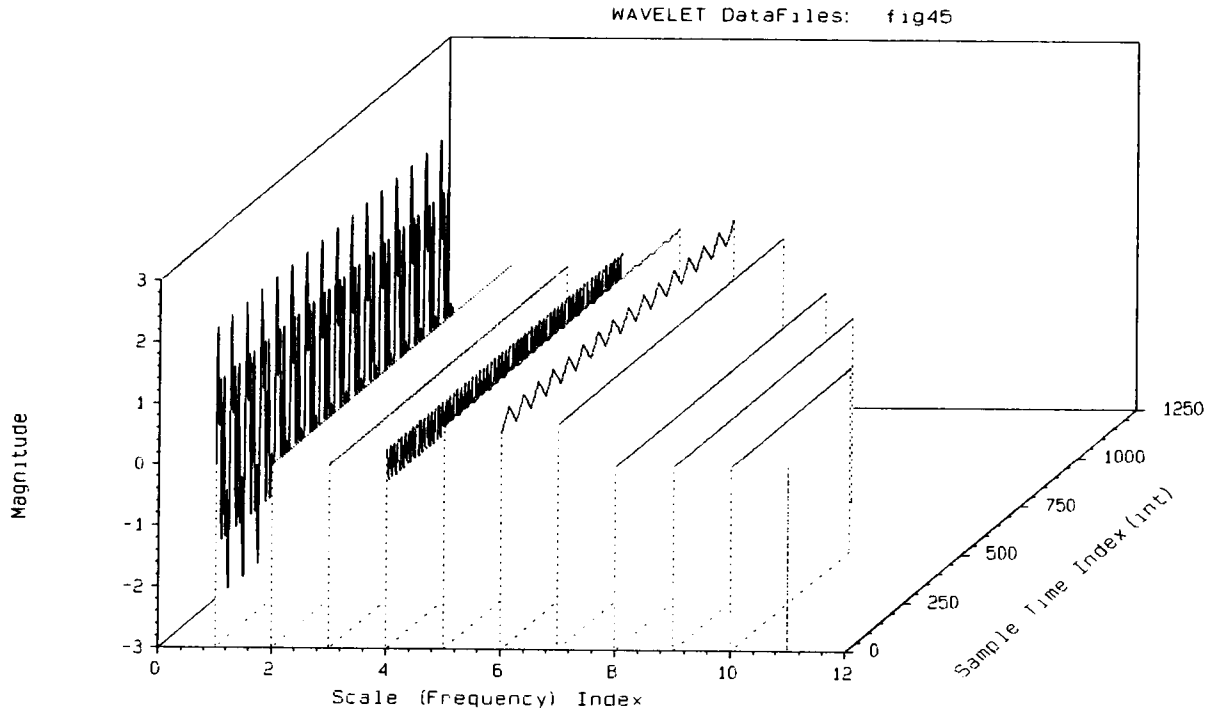


Figure 31. DAUB20 Wavelet Analysis of Figure 27 Signal with 16 SPC

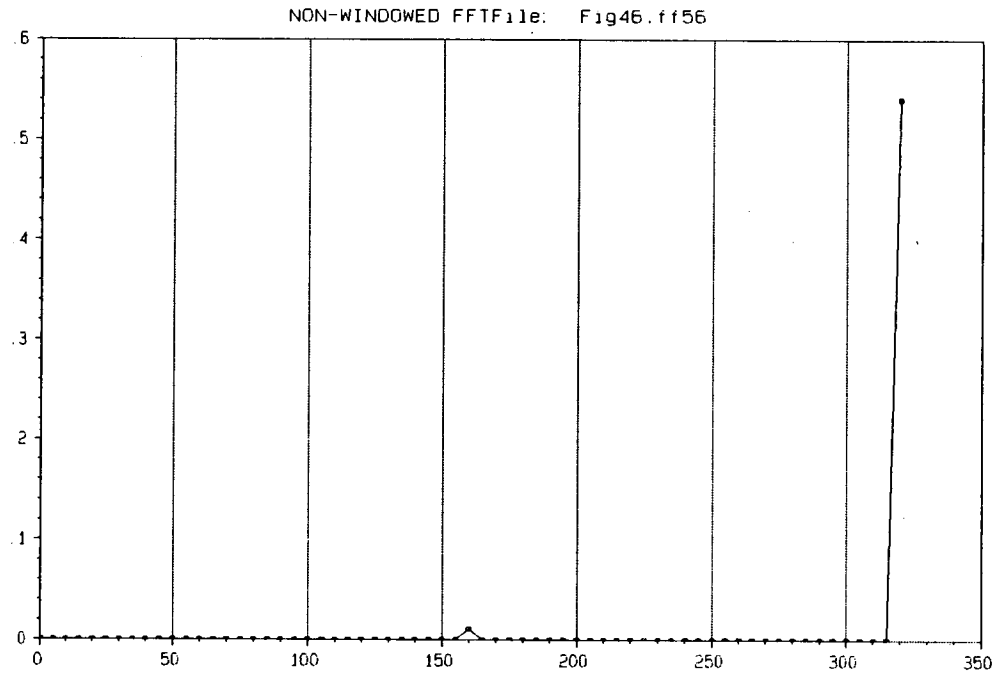


Figure 32. FFT of Level 8 of DAUB20 Wavelet Analysis of Figure 27 Signal with 16 SPC

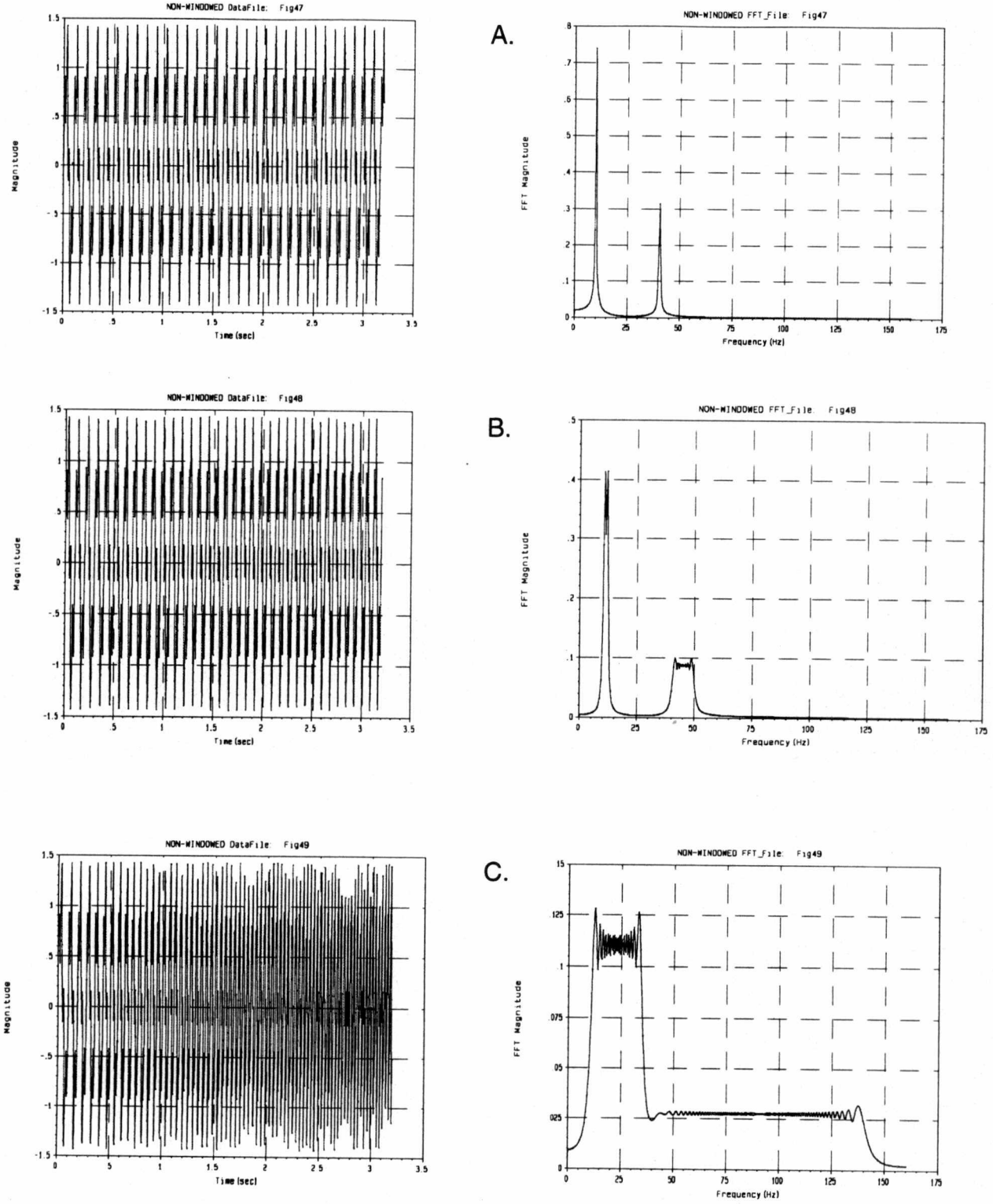
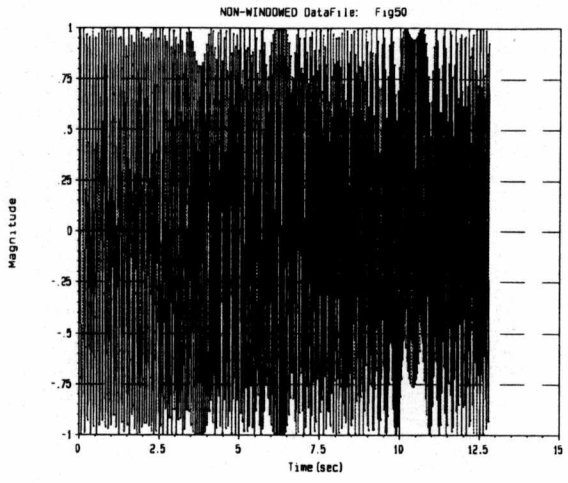
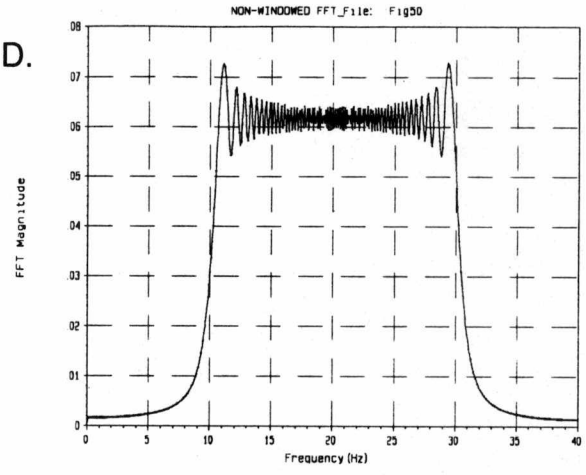
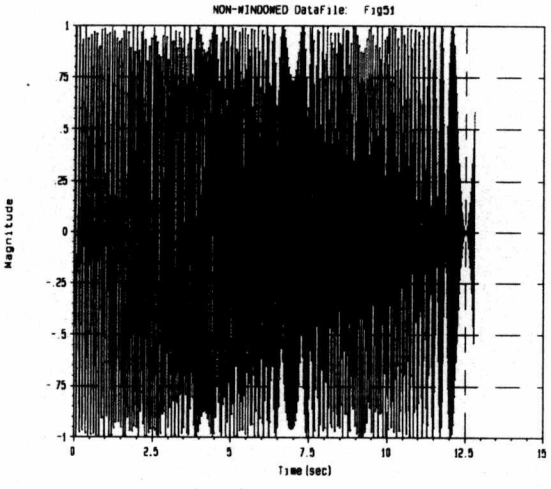


Figure 33. FFT Analysis of Ramped Sinusoids



D.





E.

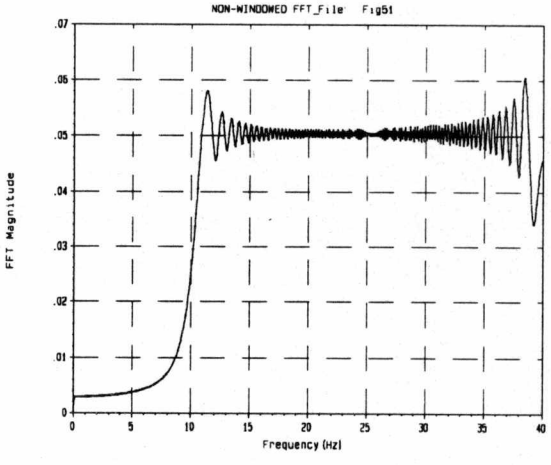


Figure 33 (Continued). FFT Analysis of Ramped Sinusoids

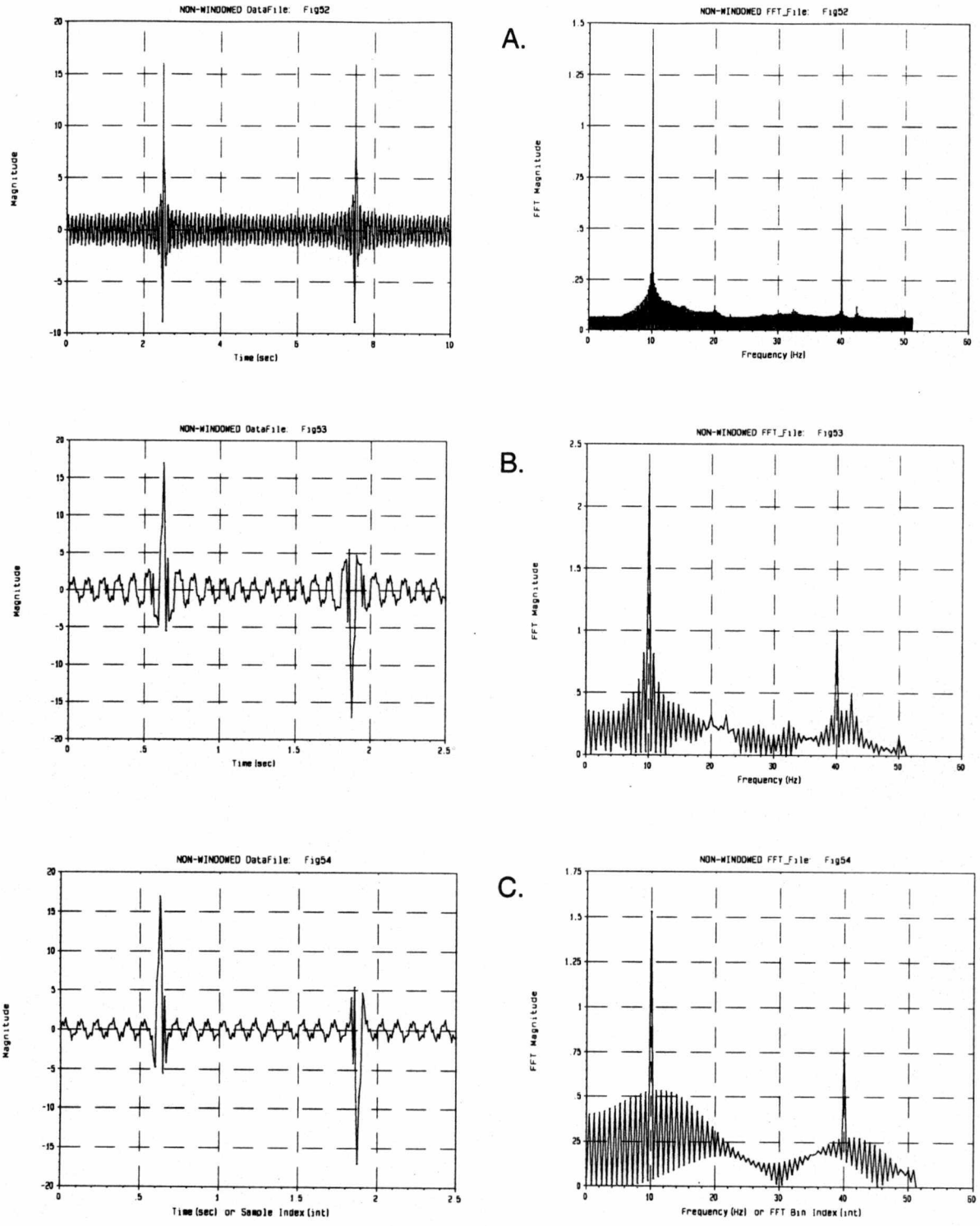
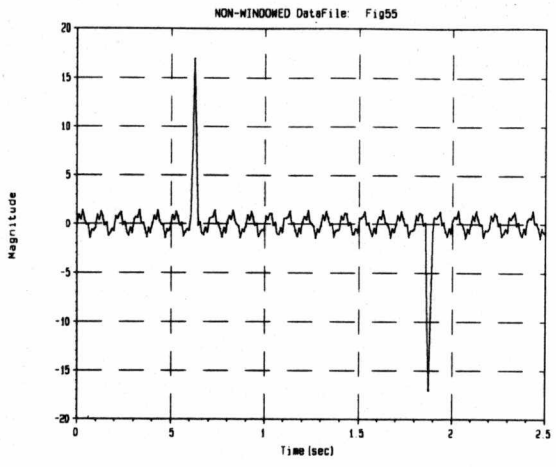
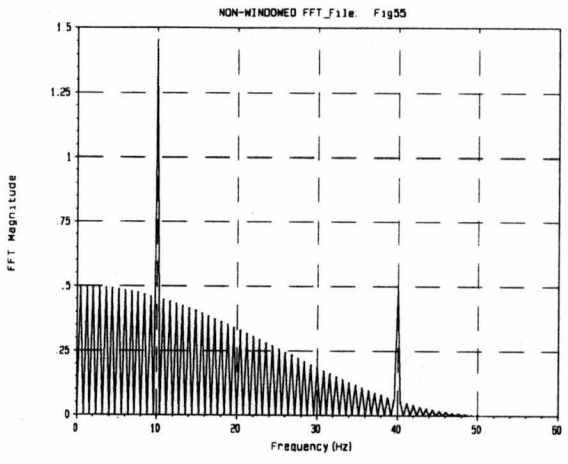
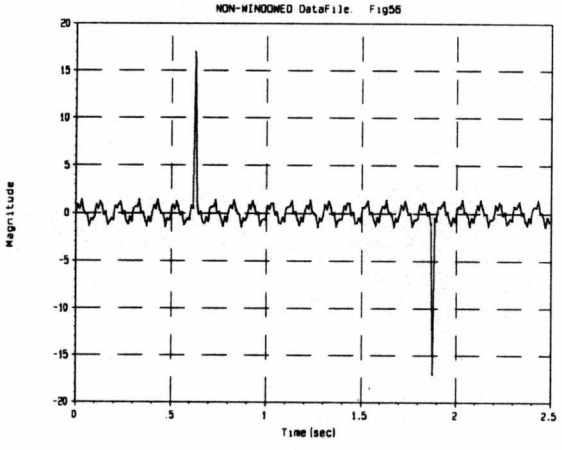


Figure 34. FFT Analysis of Sinusoids Containing Impulses



D.





E.

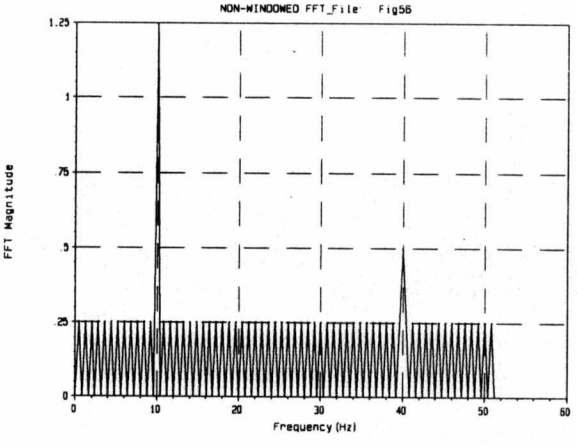


Figure 34 (Continued). FFT Analysis of Sinusoids Containing Impulses

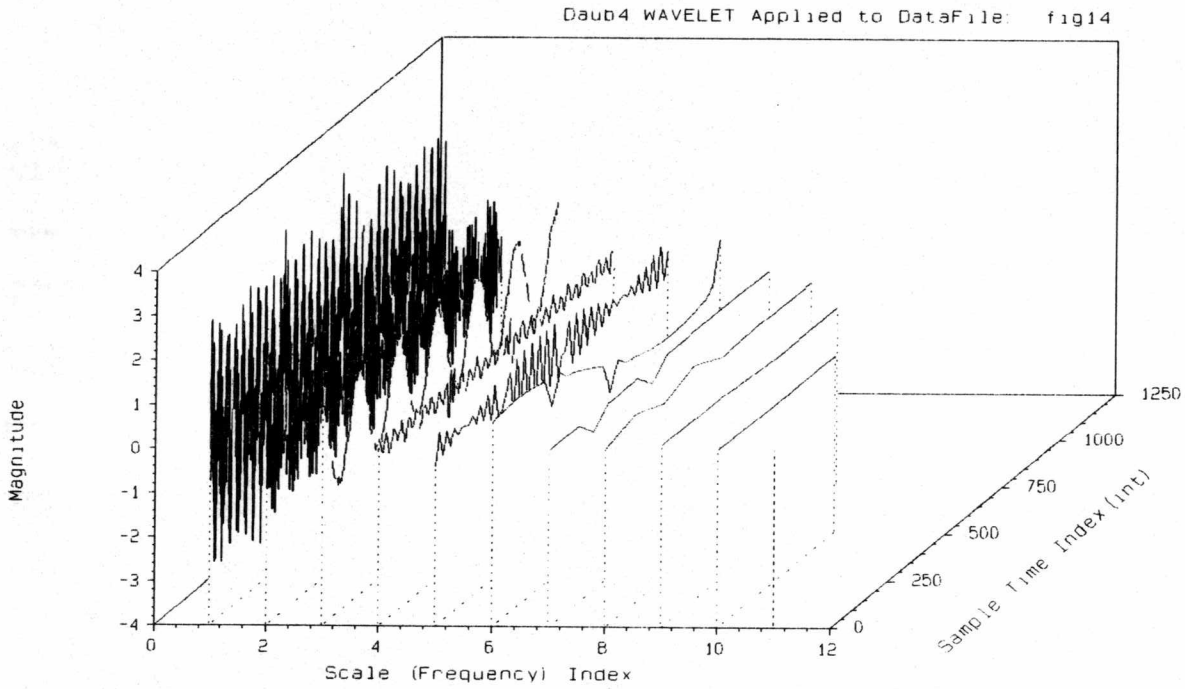


Figure 35. DAUB4 Wavelet Applied to STFT Example Signal (ref. Figure 14)

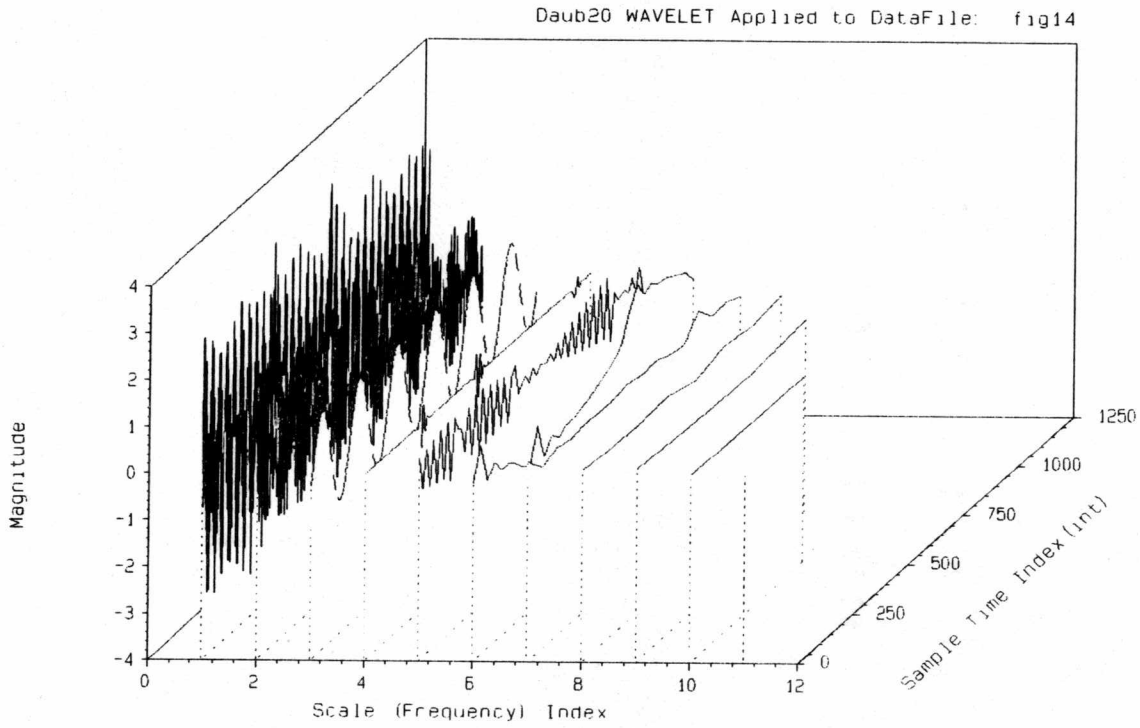


Figure 36. DAUB20 Wavelet Applied to STFT Example Signal (ref. Figure 14)

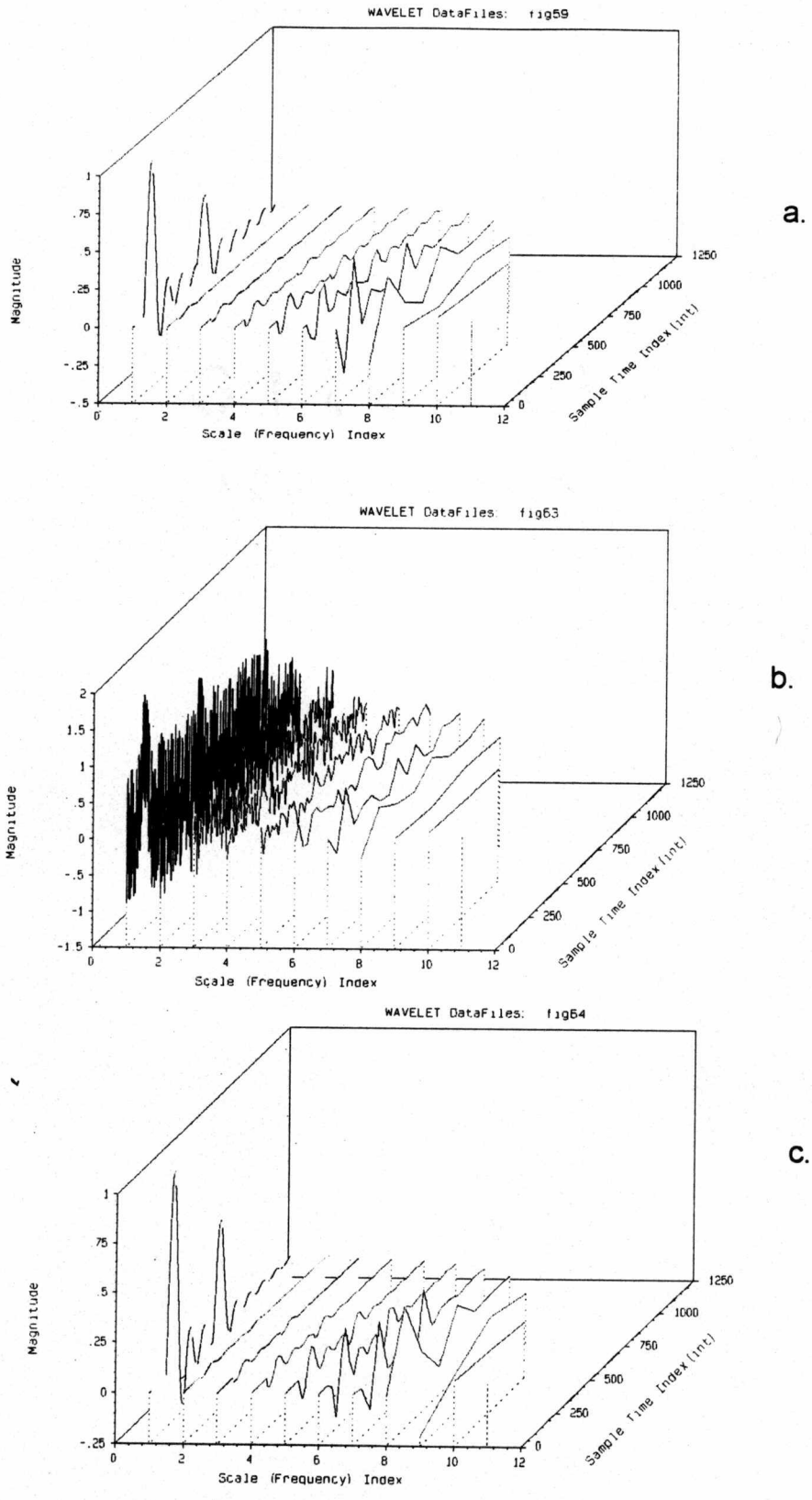


Figure 37. Wavelet Analysis of Pulse-Echo Simulated Data

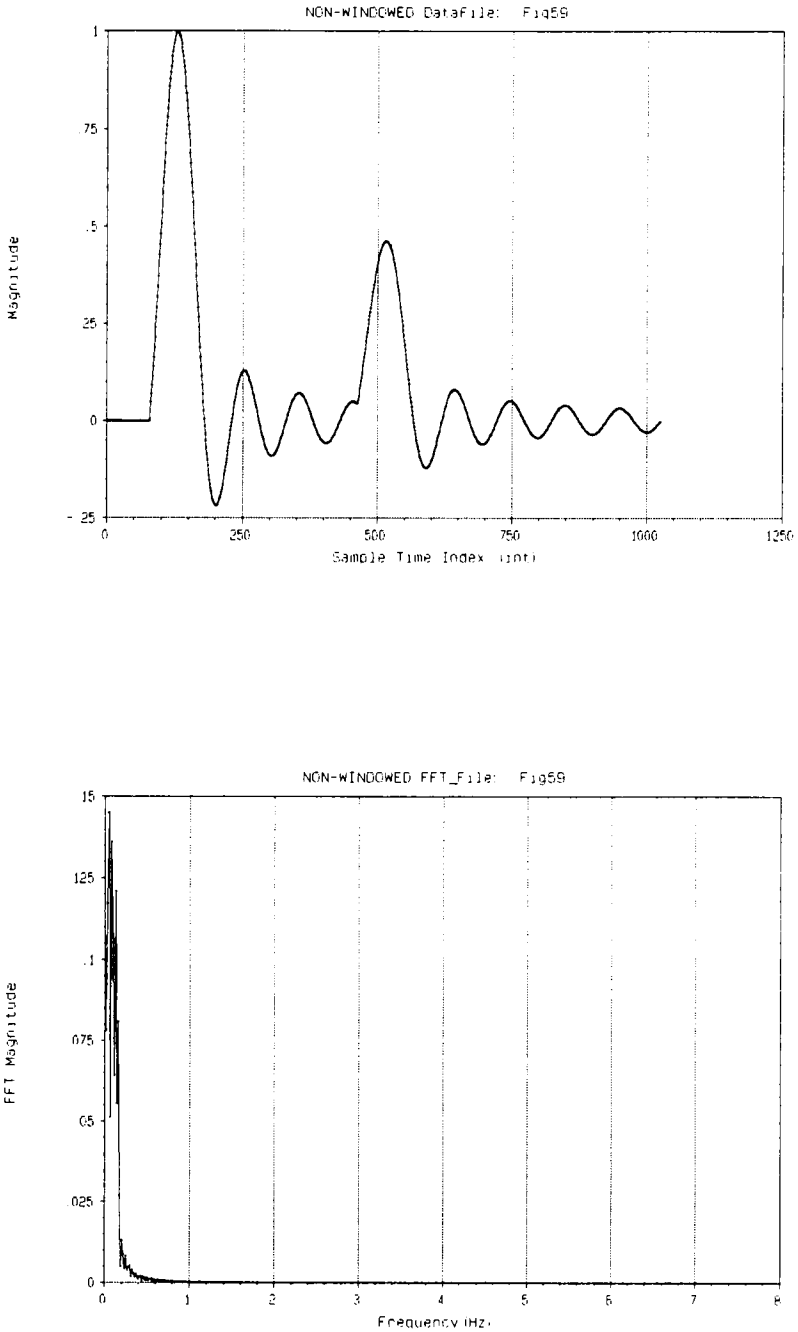
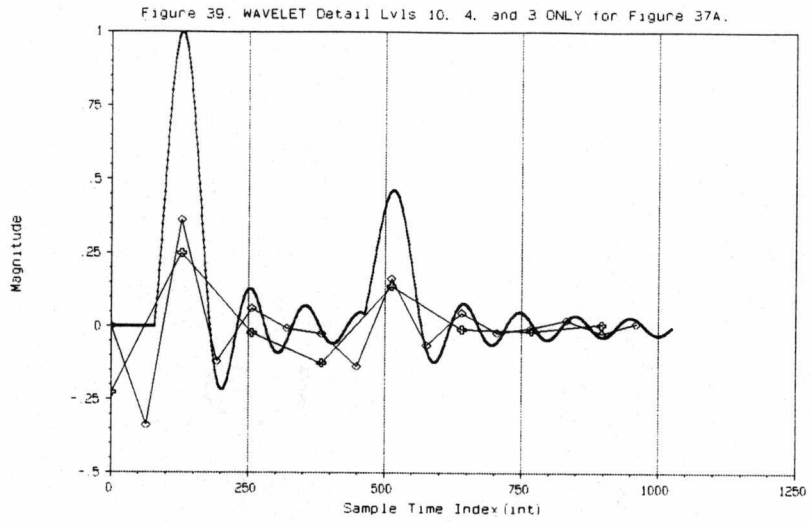
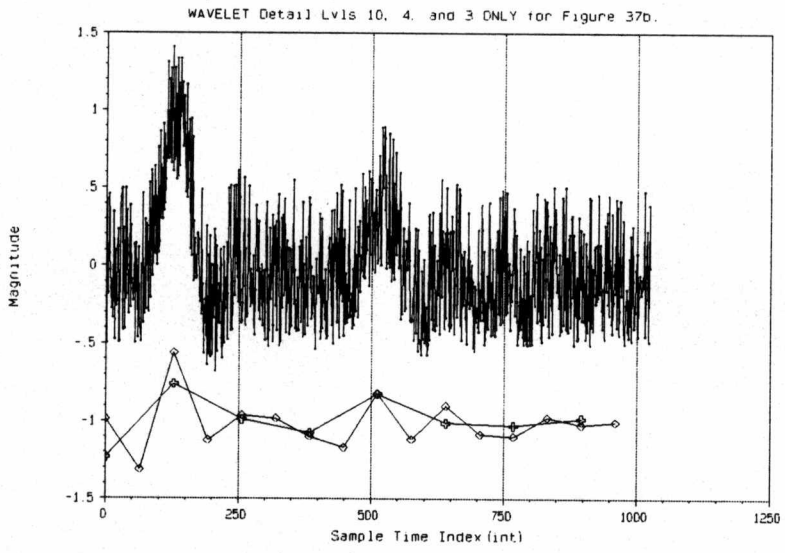


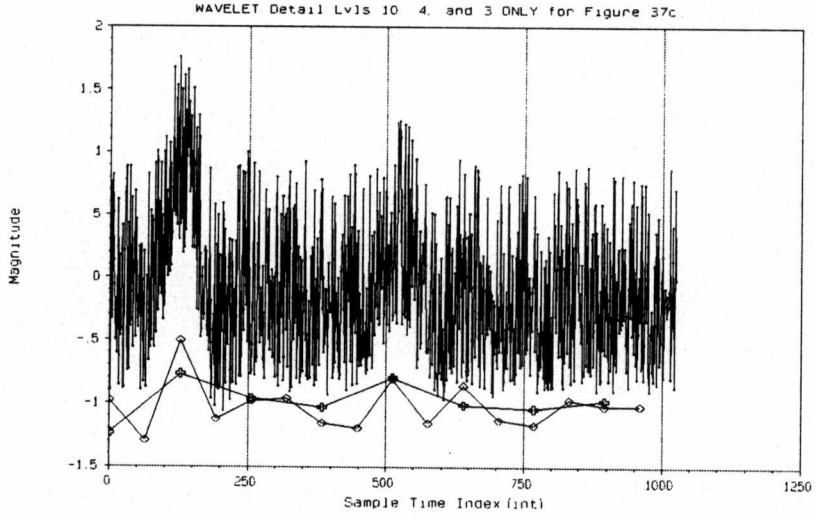
Figure 38. FFT of Simulated Pulse-Echo Data of Figure 37A



a.



b.



c.

Figure 39. Haar Wavelet Details Superimposed onSimulated Pulse-Echo Signal

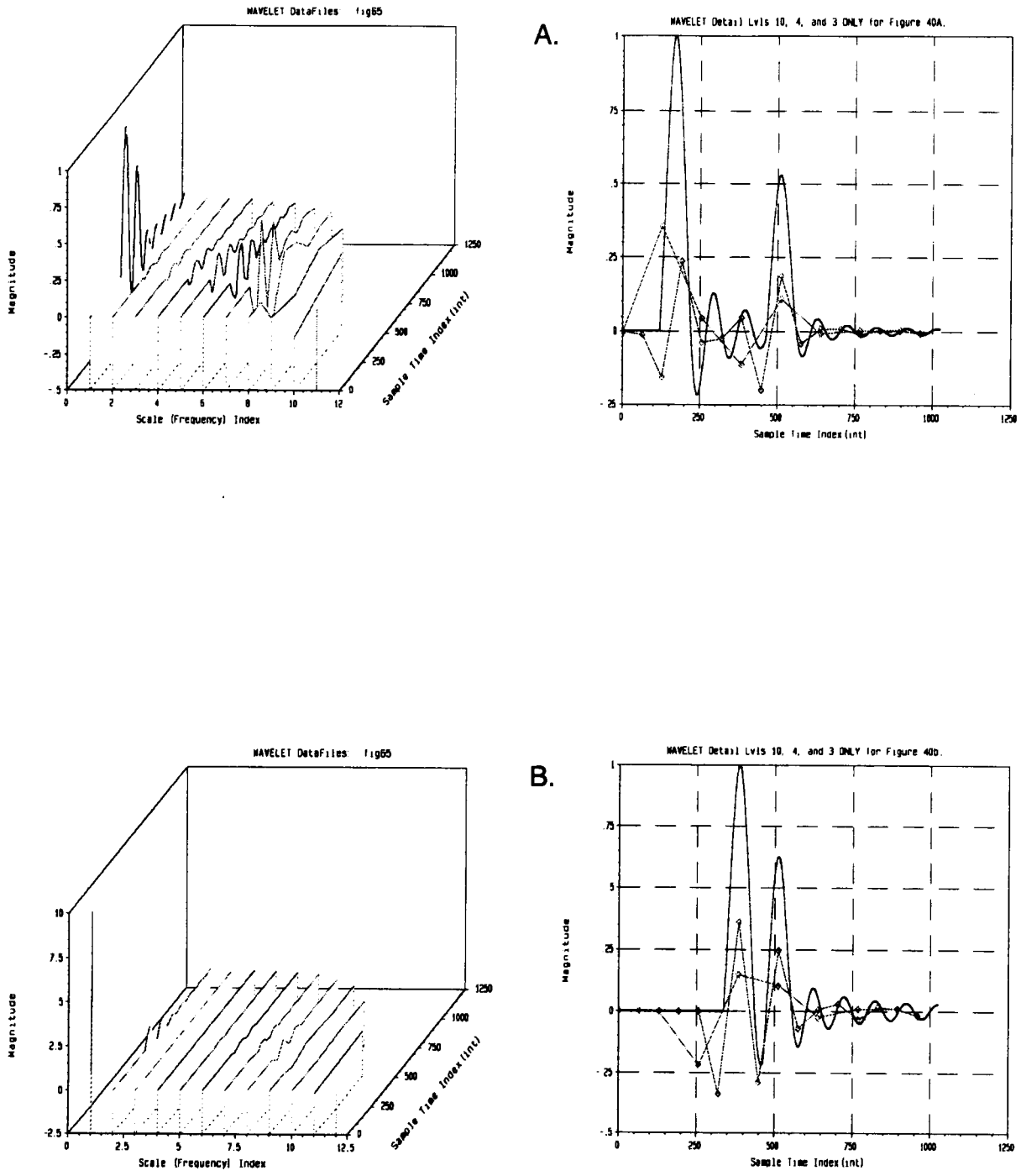


Figure 40. Wavelet Analysis of Pulse-Echo Simulated Data with Peak NOT Coincident with a 2^i Sample Point

APPENDIX B

REAL-TIME REQUIREMENTS for TRANSIENT SPECTRAL ANALYSIS

REAL-TIME REQUIREMENTS for TRANSIENT SPECTRAL ANALYSIS

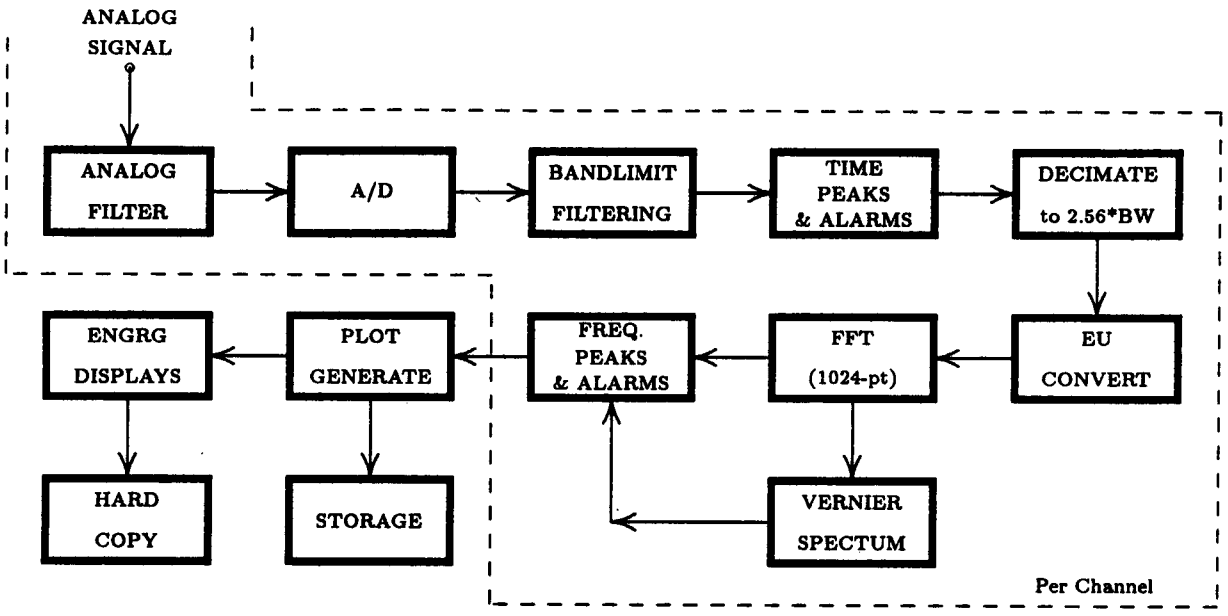
As mentioned in Chapter III, further research is required to adequately evaluate the use of wavelets for real-time transient signal analysis. To this end, this section describes the requirements for such analysis to aid in establishing a criterion for comparing various signal processing approaches. The requirements defined here are closely modeled after a system in use for aeromechanical evaluations of turbine engines which processes data in the time and Fourier domains. The requirements are as follows:

- 1. Broad system bandwidth without amplitude rolloff.
- 2. Perform FFT in $\leq$50% of the time it takes to acquire next data block.
- 3. Respond to an amplitude warning level excursion in either the time or frequency domains within .1 second.
- 4. Provide spectral information continuously regardless of transients and RPM ramping.
- 5. Provide frequency resolution (i.e., resolve two closely spaced frequency components) of better than or equal to .04% in the lower 5% of the bandwidth of the system while simultaneously providing better than or equal to .25% frequency resolution for the remaining bandwidth of the system.
- 6. Provide peak amplitudes in the time domain to within 2% assuming a sine wave at the maximum system bandwidth.
- 7. Provide peak amplitudes in the frequency domain to within 2% regardless of where the spectral component lies within the FFT bin spacing.

A *transient* event will be defined as an event which takes place within three (3) data samples at maximum bandwidth, and contributes at least 10% additional amplitude to the normal steady state signal or definable ramp in the signal. The

use of three samples ensures at least one sample occurs on the transient excursion.

To aid in modeling such a processing system, the following simple block diagram is provided:



SIMPLE SIGNAL PROCESSING BLOCK DIAGRAM

COMPUTER PROGRAMS

The following computer programs were written to accomplish this thesis:

- **SIG-GEN** → This program generates and plots a data file and its FFT for use by other analysis programs. This program was written in C completely by the author and can generate various signals from sine waves and square waves with frequency ramping and random or placed spikes with varying support all from an interactive user interface. The output files can be in a ¡x¿¡y¿ format or ¡y¿ only format to be compatible with other programs and spreadsheets. SIG-GEN generates a SPLOT (copyrighted by William G. Hood) plot file and shells to SPLOT for the graphics.
- **TTWAVLET** → This program generates and plots various spectral and wavelet transformations for an input datafile (from SIG-GEN or otherwise). This program was written in C completely by the author, however, it utilizes the wavelet routine developed by Dr. Kimble at UTSI (Dr. Kimble's code is a C version of the FORTRAN code published by Press [ref. 13]) and the FFT routines of Quinn-Curtis which were purchased for my work at AEDC. TTWAVLET also generates a SPLOT (copyrighted by William G. Hood) plot file and shells to SPLOT for the graphics.
- **DWAVEPLT** → This program performs a wavelet analysis on an input datafile (from SIG-GEN or otherwise) utilizing DAUBWAVE (Steve Gollmer, Purdue University) under a shell and generates a SPLOT (copyrighted by William G. Hood) 3D plot file and shells to SPLOT for the graphics. This program is command line driven, and is invoked using the following format:

  DWAVEPLT <Infile> <Wavelet Order> <Levels to Analyze> <#-Samples>.

In DWAVEPLT, the input file defining the signal is assumed to have an extension of *.sig*, the wavelet files generated will then follow the DAUBWAVE format and will attach other extensions to this basic file name.

The program listings for the above programs follow. A hard disk of the programs is available from the author upon request, and may be contacted at (615)454-6605 (AEDC). The file input for DAUBWAVE is a single column of y-values only. DAUBWAVE and SPLOT are available on numerous bulletin board services (BBSs). The SPLOT program is ShareWare and must be registered, but is very inexpensive. The Quinn-Curtis routines are part of a commercial package, and may be purchased by calling Quinn-Curtis at (617)449-6155. Other FFT routines exist which could be substituted for the Quinn-Curtis routines, but these routines are very good, are reasonably priced, and come with numerous other routines for plotting and analyzing engineering data.

APPENDIX C

COMPUTER PROGRAMS

SIG-GEN.c

```
/*              This Program's Name is  SIG-GEN.c
   This program generates data files for simulated signals for SPLOT
   and MATLAB.
*/

#include <math.h>
#include <string.h>
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include "..\\quinkurt\\realtype.h"
#include "..\\quinkurt\\fft.h"

#define SIGNAL double
#define NCOEFS_MAX 8
#define NMAX 1025

int NSAMPLES;
double NSAMPLES_PER_CYCLE;

/* ------- Macro to get a random integer within a specified range -------- */
#define getrandom( min, max ) ((rand() % (int)(((max)+1) - (min))) + (min))

#define PI 3.141592654

/*****************************************************************************/
FILE *file_mgr( char *fn, char *mode )
{
   FILE *fp;

   switch( mode[0] )
   {
      case 'w':
      case 'W':      /* OPEN File for WRITING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
         else
         {
            printf( "\nUnable to open FILE :  %s!", fn );
            exit( 9 );
         }
      }
      case 'r':
      case 'R':      /* OPEN File for READING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
         else
         {
            printf( "\nUnable to open FILE :  %s!", fn );
```

```
            exit( 9 );
         }
      }
      default:
      {
         printf( "\nBAD FILE MANAGER PARAMETERS, EXITING!!" );
         exit( 99 );
      }
   }
   return( fp );
}

/*****************************************************************************/
void store_data( SIGNAL *data, SIGNAL *x, char *file_name, int ndata,
                                    char filetype )
{
   static FILE *fp_out;
   int i, j;

   fp_out = file_mgr( file_name, "w" );    /* OPEN Data File for WRITING */
 /* printf( "Writing File: %s    *fp_out = %04x\n", file_name, fp_out ); */
 /* fprintf( fp_out, "%d\n", ndata ); */
   if( filetype == 's' || filetype == 'S' )
      for( i = 0; i < ndata; i++ )          /*  SAVE THE DATA!!  */
         fprintf( fp_out, "%13.6f\n", (float)data[i] );
   else
      for( i = 0; i < ndata; i++ )          /*  SAVE THE DATA!!  */
         fprintf( fp_out, "%13.6f %13.6f\n", (float)x[i], (float)data[i] );
   fclose( fp_out );
}

/*****************************************************************************/
void plot_wave( char *fn, int windowed, char filetype, char fft_select )
{
   FILE *fpo;
   char path[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
   char fname[_MAX_FNAME], ext[_MAX_EXT], buf[_MAX_PATH + 10];
   char sig_file[_MAX_FNAME+_MAX_EXT];
   int nsignal_coeff, i, m, n = 0, nn, ndetail_coefs;

   strcpy( path, fn );
   printf( "\n%s\n", path );
   _splitpath( path, NULL, NULL, fname, ext );
   strcpy( ext, "plt" );
   _makepath( path, NULL, NULL, fname, ext );
   printf( "%s\n", path );
   fpo = file_mgr( path, "w" );
   fprintf( fpo, "DASH ON\n" );
   fprintf( fpo, "xsamples %d\n", NSAMPLES );

 /* Plot Original Signal */
   strcpy( sig_file, fname );
```

```
  strcat( sig_file, ".sig" );
  printf( "%s\n", sig_file );
  fprintf( fpo, "open %s\n", sig_file );
/* fprintf( fpo, "skip 1\n" ); */
  if( filetype == 's' || filetype == 'S' )
     fprintf( fpo, "ready %d\n", NSAMPLES );
  else
     fprintf( fpo, "read %d\n", NSAMPLES );
  fprintf( fpo, "close\n" );
  fprintf( fpo, "xgrid\n" );
  fprintf( fpo, "ygrid\n" );
  fprintf( fpo, "symbols 1\n" );
  if( windowed )
     fprintf( fpo, "title WINDOWED DataFile:  %s\n", fn );
  else
     fprintf( fpo, "title NON-WINDOWED DataFile:  %s\n", fn );
  fprintf( fpo, "xlabel Time(sec) or Sample Index(int)\n" );
  fprintf( fpo, "ylabel Magnitude\n" );
  fprintf( fpo, "plot\n" );
  if( fft_select != 0 )
     fprintf( fpo, "noplot 1\n" );

/* Use SPLOT for FFT */
  if( fft_select == 0 )
  {
     fprintf( fpo, "fft 1\n" );
     fprintf( fpo, "plotfft2\n" );  /* was  "plotfft2 .5\n"   */
  }
  else
  {
   /* Plot FFT */
     strcpy( sig_file, fname );
     strcat( sig_file, ".FFT" );
     printf( "%s\n", sig_file );
     fprintf( fpo, "open %s\n", sig_file );
   /* fprintf( fpo, "skip 1\n" ); */
     fprintf( fpo, "read %d\n", NSAMPLES/2+1 );
     fprintf( fpo, "close\n" );
  }

  if( fft_select != 0 )
  {
     fprintf( fpo, "symbols 2\n" );
     fprintf( fpo, "xlabel Frequency(Hz) or FFT Bin Index(int)\n" );
     fprintf( fpo, "ylabel FFT Magnitude\n" );
     if( windowed )
        fprintf( fpo, "title WINDOWED FFT_File:  %s\n", fn );
     else
        fprintf( fpo, "title NON-WINDOWED FFT_File:  %s\n", fn );
     fprintf( fpo, "plot\n" );
  }
```

```
 /* fprintf( fpo, "wait\n" ); */

   fclose( fpo );
   sprintf( buf, "SPLOT %s", path );
   system( buf );
}

/* ------------------------------------------------------------------ */
void main()
{
   SIGNAL *sig_real, *sig_imag, *sig_fft_mag, *sig_fft_freq, *x;
   SIGNAL *sig_sine[3];
   double dc_offset, nspc, ns, di, amin, amax, atmp;
   double chirp_factor[3] = {0.0, 0.0, 0.0}, ramp = 0.0;
   double spike_mag1 = 0.0, spike_mag2 = 0.0;
   double f[3], ampl[3], theta[3] = {0.0, 0.0, 0.0}, fmax, f_ratio[3];
   int i = 0, j = 0, k, l, m, nn, min_level, level, window, row;
   int nsc, spike_loc1, spike_loc2, nspike_pts;
   char cont, winsig, win_type, chirp_type, spike_type, filetype='p';
   char random_signal, fft_select;
   char fn_out[33], fn_out1[33], fn_out2[33], bufr[5], sqrwave;
            /* Allow room for 'drive:\dir\subdir\filename.ext' */

   sig_real = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_real ) )
      exit( 9 );
   sig_imag = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_imag ) )
      exit( 9 );
   x = calloc( NMAX, sizeof( SIGNAL ) ); /* x-array for plots */
   if( BadMem( x ) )
      exit( 9 );
   sig_fft_mag = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_fft_mag ) )
      exit( 9 );
   sig_fft_freq = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_fft_freq ) )
      exit( 9 );
   sig_sine[0] = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_sine[0] ) )
      exit( 9 );
   sig_sine[1] = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_sine[1] ) )
      exit( 9 );
   sig_sine[2] = calloc( NMAX, sizeof( SIGNAL ) );
   if( BadMem( sig_sine[2] ) )
      exit( 9 );

 /* _clearscreen( _GCLEARSCREEN );
   for( i = 0; i < NMAX; i++ )
      printf( "%4d  sig_sine[0]=%7.4lf   sig_sine[1]=%7.4lf   sig_sine[2]=%7.4lf\n",
               i, sig_sine[0][i], sig_sine[1][i], sig_sine[2][i] );
```

```
*/
 _clearscreen( _GCLEARSCREEN );
 _settextposition( 3, 5 );
 printf( "Enter name of OUTPUT file(s) EXCLUDING extension (6 char MAX): ");
 scanf( "%s", fn_out);
 _settextposition( 5, 5 );
 printf( "Enter Output FILETYPE (Single or xy-Pairs, S/P): ");
 filetype = getche();
 _settextposition( 7, 5 );
 printf( "Input a DC OFFSET: ");
 scanf( "%lf", &dc_offset );

 _settextposition( 9, 5 );
 printf( "Do you want a RANDOM SIGNAL? (Y/N): ");
 random_signal = getche();
 row = 9;
 if( random_signal == 'N' || random_signal == 'n' )
 {
    _settextposition( row+=2, 5 );
    printf( "Input CHIRP TYPE (E=Exponential, L=Linear, N=NO Chirp): ");
    chirp_type = getche();
    if( chirp_type != 'n' && chirp_type != 'N' )
    {
       _settextposition( row+=2, 5 );
       printf( "Input Frequency RAMP (delta_f/delta_t x.xx): " );
       scanf( "%lf", &ramp );
    }
 }
 _settextposition( row+=2, 5 );
 printf( "Input NUMBER of SAMPLES and NUMBER of SAMPLES/CYCLE (n x.xx): " );
 scanf( "%d %lf", &NSAMPLES, &NSAMPLES_PER_CYCLE );
 if( random_signal == 'N' || random_signal == 'n' )
 {
    _settextposition( row+=2, 5 );
    printf( "Input NUMBER of Sine Components?  (1-3): " );
    scanf( "%d", &nsc );
    fmax = 0.0;
    for( i = 0; i < nsc; i++ )
    {
       _settextposition( ++row, 5 );
       printf( "Input FREQ. (Hz), AMPL., and PHASE (deg) for Sine Component #%d: ", i+1 );
       scanf( "%lf %lf %lf", &f[i], &ampl[i], &theta[i] );
       if( f[i] > fmax )
          fmax = f[i];
       theta[i] = theta[i] * PI / 180.0;
    }
    _settextposition( row+=2, 5 );
    printf( "Do you want SQUARE WAVE equivalent)?  (Y/N): " );
    sqrwave = getche();
 }
 else
    nsc = 1;
```

```
_settextposition( row+=2, 5 );
printf( "RANDOM (2 each) or DEFINED Spikes (1 or 2)?  (R/D, N=None): " );
spike_type = getche();
if( spike_type == 'D' || spike_type == 'd' )
{
   _settextposition( ++row, 5 );
   printf( "Input FIRST Simulated SPIKE LOCATION and MAGNITUDE (0-%d x.xx): "
                                    , NSAMPLES );
   scanf( "%d %lf", &spike_loc1, &spike_mag1 );
   if( spike_mag1 != 0.0 && (spike_type == 'D' || spike_type == 'd' ) )
   {
      _settextposition( ++row, 5 );
      printf( "Input SECOND Simulated SPIKE LOCATION and MAGNITUDE (%d-%d x.xx): "
                              , spike_loc1+1, NSAMPLES );
      scanf( "%d %lf", &spike_loc2, &spike_mag2 );
   }
   else
   {
      spike_loc2 = 0;
      spike_mag2 = 0.0;
   }
}

if( spike_type != 'N' && spike_type != 'n' )
{
   _settextposition( ++row, 5 );
   printf( "Input the number of POINTS in the SIMULATED SPIKE (ODD Integer): " );
   scanf( "%d", &nspike_pts );
   k = nspike_pts - 1;
   if( k < 0 )
      k = 0;
   l = k/2;
   m = -l;
}

_settextposition( row+=2, 5 );
printf( "Do you want signal WINDOWED? (Y/N): ");
winsig = getche();
if( winsig == 'Y' || winsig == 'y' )
{
   _settextposition( ++row, 5 );
   printf( "Select Window Function: ");
   printf( "\n          (R) Rectangular" );
   printf( "\n          (P) Parzen" );
   printf( "\n          (H) Hanning" );
   printf( "\n          (W) Welch" );
   printf( "\n          (Z) Hamming" );
   printf( "\n          (B) Exact Blackman" );
   _settextposition( row, 40 );
   win_type = getche();
   _settextposition( 25, 1 );
   if( win_type == 'R' || win_type == 'r' )
```

```
        window = 0;                /* Rectangular */
      if( win_type == 'P' || win_type == 'p' )
        window = 1;                /* Parzen (Triangular) */
      if( win_type == 'H' || win_type == 'h' )
        window = 2;                /* Hanning */
      if( win_type == 'W' || win_type == 'w' )
        window = 3;                /* Welch */
      if( win_type == 'Z' || win_type == 'z' )
        window = 4;                /* Hamming */
      if( win_type == 'B' || win_type == 'b' )
        window = 5;                /* Exact Blackman */
    }
    _settextposition( row+=6, 5 );
    printf( "Do you want FFT via SPLOT or QUINN-CURTIS? (S/Q): ");
    fft_select = getche();

    printf( "\n\n" );

/* ------------------------- Generate Signal ------------------------- */

    if( random_signal == 'N' || random_signal == 'n' )
      nspc = NSAMPLES_PER_CYCLE * fmax;
    else
      nspc = 1.0;
    ns = (double)NSAMPLES;

  /* RANDOM Spikes */
    if( spike_type == 'R' || spike_type == 'r' )
    {
      spike_loc1 = getrandom( 1, NSAMPLES );
      spike_mag1 = 1.0;
      spike_loc2 = getrandom( spike_loc1, NSAMPLES );
      spike_mag2 = 1.0;
    }
    if( spike_type != 'N' && spike_type != 'n' )
    {
      printf( "\nSpike Locations @ i1 = %d & i2 = %d\n",
                          spike_loc1, spike_loc2 );
      printf( "    Spike_Mag1 = %f & Spike_Mag2 = %f       ...press any key\n",
                          spike_mag1, spike_mag2 );
      getch();
    }
    i = 0;
    while( i < NSAMPLES )
    {
      sig_sine[j][i] = 0.0;
      di = (double)i;
      x[i] = di/nspc;
      for( j = 0; j < nsc; j++ )
      {
        if( random_signal == 'N' || random_signal == 'n' )
        {
```

```
            f_ratio[j] = f[j] / fmax;

           /* CHIRP only after 20% of Steady-State */
           /* if( i < (int)(0.2*(float)NSAMPLES) )
            {
               if( chirp_type == 'e' || chirp_type == 'E' ||
                           chirp_type == 'n' || chirp_type == 'N')
                  chirp_factor[j] = 0.0;
               else
                  chirp_factor[j] = 1.0;
            }
            else
           */ {
              /* Go back to Steady State for last 20% */
              /* if( i >= (int)(0.2*(float)NSAMPLES) && i <= (int)(0.8*(float)NSAMPLES) ) */
                  if( chirp_type == 'e' || chirp_type == 'E' ||
                              chirp_type == 'n' || chirp_type == 'N')
/* See Commenting @ end */        chirp_factor[j] = f_ratio[j]/f[j]*ramp*(di);/* - 0.2*(float)NSAMPLES);*/
                  else
/* See Commenting @ end */        chirp_factor[j] = 1.0 + f_ratio[j]/f[j]*ramp*(di);/* - 0.2*(float)NSAMPLES);*/
            }
            if( chirp_type == 'n' || chirp_type == 'N')
               chirp_factor[j] = 0.0;
            if( chirp_type == 'e' || chirp_type == 'E' ||
                        chirp_type == 'n' || chirp_type == 'N')
 /* EXP CHIRP */        sig_sine[j][i] = dc_offset/(double)nsc
                     + ampl[j]*sin(exp(chirp_factor[j])*(2.0*PI*f[j]*x[i] + theta[j]));
            else
 /* LINEAR CHIRP */     sig_sine[j][i] = dc_offset/(double)nsc
                     + ampl[j]*sin(chirp_factor[j]*(2.0*PI*f[j]*x[i] + theta[j]));
         }
         else
            sig_sine[j][i] = (SIGNAL)getrandom( -10000, 10000 )/10000.0;
      }

      if( sqrwave == 'Y' || sqrwave == 'y' )
      {
         sig_real[i] = 0.0;
         for( j = 0; j < nsc; j++ )
         {
            if( sig_sine[j][i] > 0.0 )
               sig_sine[j][i] = ceil( sig_sine[j][i] );
            if( sig_sine[j][i] < 0.0 )
               sig_sine[j][i] = floor( sig_sine[j][i] );
            sig_real[i] = sig_real[i] + sig_sine[j][i];
         }
      }
      else
      {
         sig_real[i] = 0.0;
         for( j = 0; j < nsc; j++ )
            sig_real[i] = sig_real[i] + sig_sine[j][i];
```

```
      }

      if( i == spike_loc1+m && spike_mag1 != 0.0 )
      {
         if( sig_real[i] >= 0.0 )      /* Simulated Noise Spike */
            sig_real[i] = sig_real[i] + spike_mag1/(double)(abs(m)+1);
         else                          /* Simulated Noise Spike */
            sig_real[i] = sig_real[i] - spike_mag1/(double)(abs(m)+1);
         m++;
         if( m > 1 )
            m = -1;
      }
      if( i == spike_loc2+m && spike_mag2 != 0.0 )
      {
         if( sig_real[i] >= 0.0 )      /* Simulated Noise Spike */
            sig_real[i] = sig_real[i] + spike_mag2/(double)(abs(m)+1);
         else                          /* Simulated Noise Spike */
            sig_real[i] = sig_real[i] - spike_mag2/(double)(abs(m)+1);
         m++;
         if( m > 1 )
            m = -1;
      }

      printf( "%4d \t %12.5f \t %12.5f \t\t Chirp_Factor[0]= %f\n",
                     i, x[i], sig_real[i], chirp_factor[0] );
      sig_imag[i] = 0.0;
      i++;
   }

   if( sqrwave == 'Y' || sqrwave == 'y' )
  /* FIXES the fact that a true square wave contains no zero points */
   {
      for( i = 0; i < NSAMPLES; i++ )
      {
         if( (sig_real[i] - dc_offset) == 0 )
            sig_real[i] = sig_real[i+1];
      }
   }

/* ------------------------------------------------------------------- */

   if( winsig == 'Y' || winsig == 'y' )
      WindowFFTData( sig_real, sig_imag, (int)NSAMPLES, window );

  /* Store Data Samples */
  /* store_data( SIGNAL *data, float *x, char *file_name, int ndata,
                     char filetype = 's' = SINGLE ==> Y-only
                                   = 'p' = PAIRS  ==> X & Y */
   strcpy( fn_out1, fn_out );
   strcat( fn_out1, ".sig" );
   printf( "Storing... FILE: %s\n", fn_out1 );
   store_data( sig_real, x, fn_out1, NSAMPLES, filetype );
```

```
 if( fft_select == 'Q' || fft_select == 'q' )
 {
  /* Generate Fourier Transform (FFT) */

    printf( "Doing FFT\n" );
    FFTCalc( sig_real, sig_imag, (int)NSAMPLES );
  /* for( i = 0; i < NSAMPLES/2+1; i++ )
    {
      printf( "Sig_Real[%4d] = %10f    Sig_Imag[%4d] = %10f\n",
                        i, sig_real[i], i, sig_imag[i] );
      getch();
    } */


  /* Store Fourier (FFT) Coefficients */

    strcpy( fn_out1, fn_out );
    strcat( fn_out1, ".fft" );
    printf( "FFT Mag & Freq CONVERSION\n" );
    for( i = 0; i <= NSAMPLES/2; i++ )
    {
      sig_fft_mag[i] = FFTMagnitude( sig_real, sig_imag, (int)NSAMPLES, i );
      sig_fft_freq[i] = FFTFrequency( (int)NSAMPLES, nspc, i );
    }
    store_data( sig_fft_mag, sig_fft_freq, fn_out1, NSAMPLES/2+1, 'p' );
 }

/* Plot Signal and FFT (via SPLOT) */

 if( (winsig == 'Y' || winsig == 'y')
              && (fft_select == 'S' || fft_select == 's') )
    plot_wave( fn_out, 1, filetype, 0 );
 else
    if( (winsig == 'N' || winsig == 'n')
              && (fft_select == 'S' || fft_select == 's') )
      plot_wave( fn_out, 0, filetype, 0 );

 if( (winsig == 'Y' || winsig == 'y')
              && (fft_select == 'Q' || fft_select == 'q') )
    plot_wave( fn_out, 1, filetype, 1 );
 else
    if( (winsig == 'N' || winsig == 'n')
              && (fft_select == 'Q' || fft_select == 'q') )
      plot_wave( fn_out, 0, filetype, 1 );

 fcloseall();
 if( filetype == 's' || filetype == 'S' )
 {
  /* strcpy( fn_out1, fn_out );
    strcat( fn_out1, ".x" );
    for( i = 0; i < NSAMPLES; i++ )
```

```
        x[i] = (SIGNAL)i;
      printf( "Storing...  FILE: %s\n", fn_out1 );
      store_data( x, x, fn_out1, NSAMPLES, filetype );
     */ printf("\n\n\t     ONLY Y Data stored in file:  %s\n", fn_out1 );
    }
    else
      printf("\n\n\t       X-Y Data stored in file:  %s\n", fn_out1 );
    exit( 0 );
}
```

TTWAVLET.c

```
/*              This Program's Name is   TTWAVLET.c
   As an example of the use of wt1, here is a program which computes
   the wavelet transform of a 1-D input sequence representing a time
   signal made up of sine waves for comparison between Fourier and
   wavelet analysis.
   (Based upon programs by Dr. Kimble, UTSI, Oct 1992 - Mar 1993.)
*/

#include <math.h>
#include <string.h>
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include "..\\quinkurt\\realtype.h"
#include "..\\quinkurt\\fft.h"

/* #define DEBUG 1 */

#undef CEPSTRUM

#define NON_WRAP 1      /* i.e., No Edge Wrapping in Wavelet Transform */
#define SIGNAL double
#define NCOEFS_MAX 8
#define NMAX 1025
#define PI 3.141592654

int NSAMPLES, NWAVCOEFS, PWR_2;

/*****************************************************************************/
FILE *file_mgr( char *fn, char *mode )
{
   FILE *fp;

   switch( mode[0] )
   {
      case 'w':
      case 'W':     /* OPEN File for WRITING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
         else
         {
            printf( "\nUnable to open FILE :  %s!", fn );
            exit( 9 );
         }
      }
      case 'r':
      case 'R':     /* OPEN File for READING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
```

```
        else
        {
           printf( "\nUnable to open FILE :  %s!", fn );
           exit( 9 );
        }
     }
     default:
     {
        printf( "\nBAD FILE MANAGER PARAMETERS, EXITING!!" );
        exit( 99 );
     }
  }
  return( fp );
}

/*****************************************************************************/
void get_data( SIGNAL *data, SIGNAL *x, char *file_name, int ndata,
               char filetype ) /* = 's' = SINGLE ==> Y-only
                               = 'p' = PAIRS  ==> X & Y */
{
  FILE *fp_in;
  int i, j;
  float x_tmp, y_tmp;

  fp_in = file_mgr( file_name, "r" );    /* OPEN Data File for READING */
  if( filetype == 's' || filetype == 'S' )
  {
     for( i = 0; i < ndata; i++ )          /*  GET THE DATA!!  */
     {
        fscanf( fp_in, "%e\n", &y_tmp );
        data[i] = (double)y_tmp;
        x[i] = (double)i;
     }
  }
  else
  {
     for( i = 0; i < ndata; i++ )          /*  GET THE DATA!!  */
     {
        fscanf( fp_in, "%e %e\n", &x_tmp, &y_tmp );
        x[i] = (double)x_tmp;
        data[i] = (double)y_tmp;
     }
  }
  fclose( fp_in );
}

/*****************************************************************************/
void store_data( SIGNAL *data, SIGNAL *x, char *file_name, int ndata,
                 int nlevls, int start_file, int leave_open,
                 char filetype ) /* = 's' = SINGLE ==> Y-only
                                 = 'p' = PAIRS  ==> X & Y */
{
```

```
    static FILE *fp_out;
    int i, j;

    if( start_file == 1 )
       fp_out = file_mgr( file_name, "w" );/* OPEN Data File for WRITING */
   /* printf( "Writing File: %s    *fp_out = %04x\n", file_name, fp_out ); */
   /* fprintf( fp_out, "%d %d\n", ndata, nlevls ); */
    if( filetype == 's' || filetype == 'S' )
       for( i = 0; i < ndata; i++ )          /*  SAVE THE DATA!!  */
          fprintf( fp_out, "%13.6e\n", (float)data[i] );
    else
       for( i = 0; i < ndata; i++ )          /*  SAVE THE DATA!!  */
          fprintf( fp_out, "%13.6e %13.6e\n", (float)x[i], (float)data[i] );
    if( leave_open == 0 )
    {
       fclose( fp_out );
     /* printf( "File CLOSED\n" ); */
    }
  /* else
       printf( "File Left OPEN\n" );
    getch(); */
}

/*****************************************************************************/
/*****************************************************************************/
void plot_wave( char *fn, int nlevels, int windowed, char filetype )
{
    FILE *fpo;
    char path[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
    char fname[_MAX_FNAME], ext[_MAX_EXT], buf[_MAX_PATH + 10];
    char sig_file[_MAX_FNAME+_MAX_EXT], fft_sig_file[_MAX_FNAME+_MAX_EXT];
    char wav_file[_MAX_FNAME+_MAX_EXT], fft_wav_file[_MAX_FNAME+_MAX_EXT];
    char cep_sig_file[_MAX_FNAME+_MAX_EXT];
    int nsignal_coeff, i, m, n = 0, nn, ndetail_coefs;

    strcpy( path, fn );
    printf( "\n%s\n", path );
    _splitpath( path, NULL, NULL, fname, ext );
    strcpy( ext, "plt" );
    _makepath( path, NULL, NULL, fname, ext );
    printf( "%s\n", path );
    fpo = file_mgr( path, "w" );
    strcpy( sig_file, fname );
    if( filetype == 's' || filetype == 'S' )
       strcat( sig_file, ".mod" );
    else
       strcat( sig_file, ".sig" );
    printf( "%s\n", sig_file );

  /* Plot Original Signal */
    fprintf( fpo, "open %s\n", sig_file );
  /* fprintf( fpo, "skip 1\n" ); */
```

```
    fprintf( fpo, "xsamples %d\n", NSAMPLES );
    fprintf( fpo, "read %d\n", NSAMPLES );
    fprintf( fpo, "close\n" );
    fprintf( fpo, "xgrid\n" );
    fprintf( fpo, "ygrid\n" );
    if( windowed )
       fprintf( fpo, "title WINDOWED DataFile:  %s\n", fn );
    else
       fprintf( fpo, "title NON-WINDOWED DataFile:  %s\n", fn );
    fprintf( fpo, "plot\n" );
    fprintf( fpo, "noplot 1\n" );

   /* Plot FFT of Original Signal */
    strcpy( fft_sig_file, fname );
    strcat( fft_sig_file, ".fft" );
    printf( "%s\n", fft_sig_file );
    fprintf( fpo, "open %s\n", fft_sig_file );
   /* fprintf( fpo, "skip 1\n" ); */
    fprintf( fpo, "read %d\n", NSAMPLES/2 );
    fprintf( fpo, "close\n" );
    fprintf( fpo, "xgrid\n" );
    fprintf( fpo, "ygrid\n" );
    fprintf( fpo, "scatter 2\n" );
    fprintf( fpo, "spikes 2\n" );
    if( windowed )
       fprintf( fpo, "title WINDOWED FFTDataFile:  %s\n", fft_sig_file );
    else
       fprintf( fpo, "title NON-WINDOWED FFTDataFile:  %s\n", fft_sig_file );
    fprintf( fpo, "plot\n" );
    fprintf( fpo, "noplot\n" );
    fprintf( fpo, "noplot 2\n" );

   /* Plot IFFT of Original Signal */
    strcpy( fft_sig_file, fname );
    strcat( fft_sig_file, ".ift" );
    printf( "%s\n", fft_sig_file );
    fprintf( fpo, "open %s\n", fft_sig_file );
   /* fprintf( fpo, "skip 1\n" ); */
    fprintf( fpo, "read %d\n", NSAMPLES );
    fprintf( fpo, "close\n" );
    fprintf( fpo, "xgrid\n" );
    fprintf( fpo, "ygrid\n" );
    if( windowed )
       fprintf( fpo, "title WINDOWED IFFTDataFile:  %s\n", fft_sig_file );
    else
       fprintf( fpo, "title NON-WINDOWED IFFTDataFile:  %s\n", fft_sig_file );
    fprintf( fpo, "plot\n" );
    fprintf( fpo, "noplot\n" );
    m=3;
    fprintf( fpo, "noplot 1-%d\n", m );

#if defined( CEPSTRUM )
```

```
  /* Plot CEPSTRUM of Original Signal */
    strcpy( cep_sig_file, fname );
    strcat( cep_sig_file, ".cep" );
    printf( "%s\n", cep_sig_file );
    fprintf( fpo, "open %s\n", cep_sig_file );
  /* fprintf( fpo, "skip 1\n" ); */
    fprintf( fpo, "read %d\n", NSAMPLES/4 );
    fprintf( fpo, "close\n" );
    fprintf( fpo, "xgrid\n" );
    fprintf( fpo, "ygrid\n" );
    if( windowed )
       fprintf( fpo, "title WINDOWED CEPDataFile:  %s\n", cep_sig_file );
    else
       fprintf( fpo, "title NON-WINDOWED CEPDataFile:  %s\n", cep_sig_file );
    fprintf( fpo, "xlabel TIME\n" );
    fprintf( fpo, "plot\n" );
    fprintf( fpo, "noplot\n" );
    m = m + 1;
    fprintf( fpo, "noplot 1-%d\n", m );
#endif

  /* Plot Current Level of Wavelets */
    fprintf( fpo, "symbols\n" );
    strcpy( wav_file, fname );
    strcat( wav_file, ".wav" );
    printf( "%s\n", wav_file );
    fprintf( fpo, "open %s\n", wav_file );
  /* fprintf( fpo, "skip 1\n" ); */
    fprintf( fpo, "xgrid\n" );
    fprintf( fpo, "ygrid\n" );
    for( i = PWR_2-nlevels; i < PWR_2; i++ )
    {
       nn = (int)pow( 2.0, (double)i );
       if( n == 0 )
       {
          fprintf( fpo, "read %d\n", nn );
          fprintf( fpo, "title 1st %d WAVELET Coefficients - Level %d SMOOTHED Signal\n", nn, i );
          fprintf( fpo, "plot\n" );
          n++;
          fprintf( fpo, "noplot %d\n", n+m );
          fprintf( fpo, "read %d\n", nn/2+1 );
          fprintf( fpo, "title FFT of Level %d WAVELET SCALE Coefficients\n", i );
        /* fprintf( fpo, "show\n" );
          fprintf( fpo, "wait\n" );
        */ fprintf( fpo, "scatter %d\n", n+m+1 );
          fprintf( fpo, "spikes %d\n", n+m+1 );
          fprintf( fpo, "plot\n" );
          fprintf( fpo, "line %d\n", n+m+1 );
          fprintf( fpo, "clrdata %d\n", m+2 );
        /* fprintf( fpo, "show\n" );
          fprintf( fpo, "wait\n" );
        */ ndetail_coefs = nn;
```

```
        fprintf( fpo, "read %d\n", nn );
        fprintf( fpo, "title Next %d WAVELET Coefficients - Level %d DETAIL\n", nn, i );
        fprintf( fpo, "plot\n" );
        fprintf( fpo, "noplot %d\n", n+m+1 );
        fprintf( fpo, "read %d\n", nn/2+1 );
        fprintf( fpo, "title FFT of Level %d WAVELET DETAIL Coefficients\n", i );
      /* fprintf( fpo, "show\n" );
        fprintf( fpo, "wait\n" );
      */ fprintf( fpo, "scatter %d\n", n+m+2 );
        fprintf( fpo, "spikes %d\n", n+m+2 );
        fprintf( fpo, "plot\n" );
        fprintf( fpo, "line %d\n", n+m+2 );
        fprintf( fpo, "clrdata %d\n", m+2+1 );
      /* fprintf( fpo, "show\n" );
        fprintf( fpo, "wait\n" );
      */ n++;
    }
    else
    {
        fprintf( fpo, "read %d\n", nn );
        fprintf( fpo, "title Next %d WAVELET Coefficients - Level %d DETAIL\n", nn, i );
        fprintf( fpo, "noplot %d\n", n+m );
        fprintf( fpo, "plot\n" );
        n++;
    }
  }
  fprintf( fpo, "close\n" );

/* Plot ALL Wavelet Coefficients as Composite Plot */
  fprintf( fpo, "title WAVELET Coefficients\n" );
  fprintf( fpo, "noplot\n" );
  fprintf( fpo, "noplot 1-%d\n", m );
  fprintf( fpo, "plot\n" );

/* Plot Reconstructed Signal (INVERSE Wavelet Transform) */
  strcpy( sig_file, fname );
  strcat( sig_file, ".inv" );
  printf( "%s\n", sig_file );
  fprintf( fpo, "clrdata 2-%d\n", n+m+1 );
  fprintf( fpo, "open %s\n", sig_file );
/* fprintf( fpo, "skip 1\n" ); */
  i = PWR_2 - nlevels + 1;
  nn = (int)pow( 2.0, (double)i );
  fprintf( fpo, "xsamples %d\n", nn );
  fprintf( fpo, "read %d\n", nn );
  fprintf( fpo, "close\n" );
  fprintf( fpo, "noplot\n" );
  fprintf( fpo, "line 1\n" );
  fprintf( fpo, "line 2\n" );
  fprintf( fpo, "symbols\n" );
  if( windowed )
     fprintf( fpo, "title RECONSTRUCTED WINDOWED SignalFile:  %s\n", fn );
```

```
    else
        fprintf( fpo, "title RECONSTRUCTED NON-WINDOWED SignalFile:  %s\n", fn );
   /* fprintf( fpo, "show\n" );
    fprintf( fpo, "wait\n" );
   */ fprintf( fpo, "plot\n" );
    fclose( fpo );
    sprintf( buf, "SPLOT %s", path );
    system( buf );
}

/*****************************************************************************/
/*
    1-D Wavelet Transform filter.  Adapted from a program provided
    by William H. Press and adapted by Ken Kimble, UTSI, 1992.

    This routine implements the pyramid algorithm, replacing a[0:n-1]
    by its wavelet transform (isign=1), or the inverse operation (isign=-1).
    The procedure wtstep, whose actual name must be supplied in calling
    wt1, is the "atomic" wavelet filter.  Examples of wtstep are daub4
    and pwt (preceeded by pwtset).
*/

/*-------------------------------------------------------------------------*/
wt1( SIGNAL *a, int n, int nn, int levls, int isign, int (*wtstep)() )
{
    int i;
    double smax, dmax;

    if( n < levls )
        return 0;
    if( isign >= 0 )
    {
      /* nn = n; */
      /* while( nn >= levls ) */
        {
            (*wtstep)(a, nn, isign);
#ifdef DEBUG
            printf( "wt1: nn = %d\n", nn );
            for( i = 0; i < n; i++)
                printf( "a[%4d] = %10f\n", i, a[i] );
          /* break; */
#endif
            smax = 0.0; dmax = 0.0;
            for( i = 0; i < nn; i++ )
            {
                if( fabs(a[i]) > smax )
                    smax = fabs(a[i]);
                if( fabs(a[i+nn]) > dmax )
                    dmax = fabs(a[i+nn]);
            }
            fprintf( stdout, "Smax=%le  Dmax=%le  Level %ld\n",
                    smax, dmax, -1+(long)(log10((double)nn)/log10(2.0)+.5) );
```

```
            printf( "\n                    ... press any key\n" );
            getch();
            _clearscreen( _GCLEARSCREEN );
          /* nn /= 2; */
        }
    }
    else
    {
     /* nn = levls; */
     /* while( nn <= n ) */
      {
        (*wtstep)(a, nn, isign);
       /* nn *= 2; */
      }
    }
    return 0;
}

/*-------------------------------------------------------------------
   Here is haar2, implementing the 2-coefficient Haar wavelet:
  -------------------------------------------------------------------*/
haar2( SIGNAL *a, int n, int isign )
{
    SIGNAL C[] = { .5, .5 };
    SIGNAL wksp[NMAX];
    int i, j, k, nh, nh1, nhm1;

    if( n < NWAVCOEFS )
        return 0;
    if( n > NMAX )
        return -1;

    nh = n / 2;
    nh1 = nh + 1;
    nhm1 = nh - 1;
    if( isign >= 0 )    /* WAVELET Transform */
    {
        i = 0;
#if !defined( NON_WRAP )
        for( j = 0; j < n - NWAVCOEFS - 1; j += 2 )
#else
        for( j = 0; j < n-1; j += 2 )
#endif
        {
         /* COEFFICIENTS of SMOOTHED Approximation to Original Signal */
           wksp[i] = a[j]*C[0] + a[j+1]*C[1];

         /* DETAIL of Signal at Level z, where 2^z = nh */
           wksp[i+nh] = a[j]* C[1] - a[j+1]*C[0];
           ++i;
        }
#if !defined( NON_WRAP )
```

```
                /* Edge Wrapping */
        wksp[i] = a[n-1]*C[0] + a[n]*C[1];
        wksp[i+nh] = a[0]*C[1] - a[1]*C[0];
#endif
    }
    else        /* WAVELET Inverse Transform */
    {
            /* Edge Reconstruction from Wrapping */
#if !defined( NON_WRAP )
        wksp[0] = a[n-1]*C[1] + a[0]*C[0];
        wksp[1] = a[n-1]*C[0] + a[0]*C[1];
#endif
        j = 0;
        for( i = 0; i < nh; ++i )
        {
          wksp[j]   = 2.0*a[i+nh]*C[1] + 2.0*a[i]*C[0];
          wksp[j+1] = -2.0*a[i+nh]*C[0] + 2.0*a[i]*C[1];
          j += 2;
        }
    }
    for( i = 0; i < n; ++i )
      a[i] = wksp[i];
    return 0;
}

/*-----------------------------------------------------------------
   Here is daub4, implementing the 4-coefficient Daubechies wavelet:
  -----------------------------------------------------------------*/
daub4( SIGNAL *a, int n, int isign )
{
  /*
   See Daubechies, "Ten Lectures on Wavelets", page 195, for N = 2,
   where the normalization constant is sqrt(2) = C0+C1+C2+C3.
  */
   SIGNAL C[] = { .4829629131445341,  .8365163037378077,
          .2241438680420134, -.1294095225512603 };
   SIGNAL wksp[NMAX];
   int i, j, k, nh, nh1, nhm1;

   if( n < NWAVCOEFS )
      return 0;
   if( n > NMAX )
      return -1;

   nh = n / 2;
   nh1 = nh + 1;
   nhm1 = nh - 1;
   if( isign >= 0 )   /* WAVELET Transform */
   {
      i = 0;
#if !defined( NON_WRAP )
      for( j = 0; j < n - NWAVCOEFS; j += 2 )
```

```
#else
      for( j = 0; j < n; j += 2 )
#endif
      {
       /* COEFFICIENTS of SMOOTHED Approximation to Original Signal */
        wksp[i] = a[j]*C[0] + a[j+1]*C[1] + a[j+2]*C[2] + a[j+3]*C[3];
        wksp[i] = wksp[i] / sqrt( 2.0 );

       /* DETAIL of Signal at Level z, where 2^z = nh */
        wksp[i+nh] = a[j]*C[3] - a[j+1]*C[2] + a[j+2]* C[1] - a[j+3]*C[0];
        wksp[i+nh] = wksp[i+nh] / sqrt( 2.0 );
        ++i;
      }
#if !defined( NON_WRAP )     /* Edge Wrapping */

      wksp[i]    = a[n-2]*C[0] + a[n-1]*C[1] + a[0]*C[2] + a[1]*C[3];
      wksp[i]    = wksp[i] / sqrt( 2.0 );
      wksp[i+nh] = a[n-2]*C[3] - a[n-1]*C[2] + a[0]*C[1] - a[1]*C[0];
      wksp[i+nh] = wksp[i+nh] / sqrt( 2.0 );
#endif
   }
   else         /* WAVELET Inverse Transform */
   {
#if !defined( NON_WRAP )     /* Edge Reconstruction from Wrapping */

      wksp[0] = a[nhm1]*C[2] + a[n-1]*C[1] + a[0]*C[0] + a[nh]*C[3];
      wksp[0] = wksp[0] * sqrt( 2.0 );
      wksp[1] = a[nhm1]*C[3] - a[n-1]*C[0] + a[0]*C[1] - a[nh]*C[2];
      wksp[1] = wksp[1] * sqrt( 2.0 );
#endif
      j = 2;
      for( i = 0; i < nh-1; ++i )
      {
         wksp[j]   = a[i]*C[2] + a[i+nh]*C[1] + a[i+1]*C[0] + a[i+nh1]*C[3];
         wksp[j]   = wksp[j] * sqrt( 2.0 );
         wksp[j+1] = a[i]*C[3] - a[i+nh]*C[0] + a[i+1]*C[1] - a[i+nh1]*C[2];
         wksp[j+1] = wksp[j+1] * sqrt( 2.0 );
         j += 2;
      }
   }
   for( i = 0; i < n; ++i )
      a[i] = wksp[i];
   return 0;
}

/*-------------------------------------------------------------------
   Here is daub8, implementing the 8-coefficient Daubechies wavelet:
  -------------------------------------------------------------------*/
daub8( SIGNAL *a, int n, int isign )
{
  /*
   See Daubechies, "Ten Lectures on Wavelets", page 195, for N = 4,
```

```
   where the normalization constant is sqrt(2) = C0+C1+C2+C3+C4+C5+C6+C7.
  */
   SIGNAL C[] = { .2303778133088964,  .7148465705529154,
          .6308807679398587, -.0279837694168599,
          -.1870348117190931,  .0308413818355607,
          .0328830116668852, -.0105974017850690 };
   SIGNAL wksp[NMAX];
   int i, j, k, nh, nh1, nh2, nh3, nhm1, nhm2, nhm3;

   if( n < NWAVCOEFS )
     return 0;
   if( n > NMAX )
     return -1;

   nh = n / 2;
   nh1 = nh + 1;
   nh2 = nh + 2;
   nh3 = nh + 3;
   nhm1 = nh - 1;
   nhm2 = nh - 2;
   nhm3 = nh - 3;
   if( isign >= 0 )   /* WAVELET Transform */
   {
     i = 0;
#if !defined( NON_WRAP )
     for( j = 0; j < n - NWAVCOEFS; j += 2 )
#else
     for( j = 0; j < n; j += 2 )
#endif
     {
       /* COEFFICIENTS of SMOOTHED Approximation to Original Signal */
       wksp[i] = a[j]*C[0] + a[j+1]*C[1] + a[j+2]*C[2] + a[j+3]*C[3]
            + a[j+4]*C[4] + a[j+5]*C[5] + a[j+6]*C[6] + a[j+7]*C[7];
       wksp[i] = wksp[i] / sqrt( 2.0 );

       /* DETAIL of Signal at Level z, where 2^z = nh */
       wksp[i+nh] = a[j]*C[7] - a[j+1]*C[6] + a[j+2]*C[5] - a[j+3]*C[4]
            + a[j+4]*C[3] - a[j+5]*C[2] + a[j+6]*C[1] - a[j+7]*C[0];
       wksp[i+nh] = wksp[i+nh] / sqrt( 2.0 );
       ++i;
     }
#if !defined( NON_WRAP )     /* Edge Wrapping */

     wksp[i]    = ( a[n-6]*C[0] + a[n-5]*C[1] + a[n-4]*C[2] + a[n-3]*C[3]
          + a[n-2]*C[4] + a[n-1]*C[5] + a[0]*C[6] + a[1]*C[7] )
          / sqrt( 2.0 );
     wksp[i+1]  = ( a[n-4]*C[0] + a[n-3]*C[1] + a[n-2]*C[2] + a[n-1]*C[3]
          + a[0]*C[4] + a[1]*C[5] + a[2]*C[6] + a[3]*C[7] )
          / sqrt( 2.0 );
     wksp[i+2]  = ( a[n-2]*C[0] + a[n-1]*C[1] + a[0]*C[2] + a[1]*C[3]
          + a[2]*C[4] + a[3]*C[5] + a[4]*C[6] + a[5]*C[7] )
          / sqrt( 2.0 );
```

```
      wksp[i+nh]   = ( a[n-6]*C[7] - a[n-5]*C[6] + a[n-4]*C[5] - a[n-3]*C[4]
                + a[n-2]*C[3] - a[n-1]*C[2] + a[0]*C[1] - a[1]*C[0] )
                / sqrt( 2.0 );
      wksp[i+1+nh] = ( a[n-4]*C[7] - a[n-3]*C[6] + a[n-2]*C[5] - a[n-1]*C[4]
                + a[0]*C[3] - a[1]*C[2] + a[2]*C[1] - a[3]*C[0] )
                / sqrt( 2.0 );
      wksp[i+2+nh] = ( a[n-2]*C[7] - a[n-1]*C[6] + a[0]*C[5] - a[1]*C[4]
                + a[2]*C[3] - a[3]*C[2] + a[4]*C[1] - a[5]*C[0] )
                / sqrt( 2.0 );
#endif
    }
    else        /* WAVELET Inverse Transform */
    {
#if !defined( NON_WRAP )     /* Edge Reconstruction from Wrapping */

      wksp[0] = ( a[nhm3]*C[6] + a[n-3]*C[1] + a[nhm2]*C[4] + a[n-2]*C[3]
            + a[nhm1]*C[2] + a[n-1]*C[5] + a[0]*C[0] + a[nh]*C[7] )
            * sqrt( 2.0 );
      wksp[2] = ( a[nhm2]*C[6] + a[n-2]*C[1] + a[nhm1]*C[4] + a[n-1]*C[3]
            + a[0]*C[2] + a[nh]*C[5] + a[1]*C[0] + a[nh1]*C[7] )
            * sqrt( 2.0 );
      wksp[4] = ( a[nhm1]*C[6] + a[n-1]*C[1] + a[0]*C[4] + a[nh]*C[3]
            + a[1]*C[2] + a[nh1]*C[5] + a[2]*C[0] + a[nh2]*C[7] )
            * sqrt( 2.0 );
      wksp[1] = ( a[nhm3]*C[7] - a[n-3]*C[0] + a[nhm2]*C[5] - a[n-2]*C[2]
            + a[nhm1]*C[3] - a[n-1]*C[4] + a[0]*C[1] - a[nh]*C[6] )
            * sqrt( 2.0 );
      wksp[3] = ( a[nhm2]*C[7] - a[n-2]*C[0] + a[nhm1]*C[5] - a[n-1]*C[2]
            + a[0]*C[3] - a[nh]*C[4] + a[1]*C[1] - a[nh1]*C[6] )
            * sqrt( 2.0 );
      wksp[5] = ( a[nhm1]*C[7] - a[n-1]*C[0] + a[0]*C[5] - a[nh]*C[2]
            + a[1]*C[3] - a[nh1]*C[4] + a[2]*C[1] - a[nh2]*C[6] )
            * sqrt( 2.0 );
#endif
      j = 6;
      for( i = 0; i < nh-1; ++i )
      {
        wksp[j]   = a[i]*C[6] + a[i+nh]*C[1] + a[i+1]*C[4] + a[i+nh1]*C[3]
              + a[i+2]*C[2] + a[i+nh2]*C[5] + a[i+3]*C[0] + a[i+nh3]*C[7];
        wksp[j]   = wksp[j] * sqrt( 2.0 );
        wksp[j+1] = a[i]*C[7] - a[i+nh]*C[0] + a[i+1]*C[5] - a[i+nh1]*C[2]
              + a[i+2]*C[3] - a[i+nh2]*C[4] + a[i+3]*C[1] - a[i+nh3]*C[6];
        wksp[j+1] = wksp[j+1] * sqrt( 2.0 );
        j += 2;
      }
    }
    for( i = 0; i < n; ++i )
      a[i] = wksp[i];
    return 0;
}

/*****************************************************************************/
```

```
int power_of_two( int n )
{
   int pwr2 = 0, m;

   m = n;
   if( n <= 0 )
      exit( 99 );
   while( m >= 2 )
   {
      pwr2++;
    /* printf( "m = %d \t pwr2 = %d\n", m, pwr2 ); */
      m/=2;
   }
   printf( "\npwr2 = %d \t CHECK: 2^pwr2 = %lf\n",
                           pwr2, pow( 2.0, (double)pwr2 ) );
   return( pwr2 );
}

/* ------------------------------------------------------------------ */
void main()
{
   SIGNAL *sig_real, *sig_imag, *sig_fft_mag;
   SIGNAL *a, *c, *q, *cep_real, *cep_imag;
   SIGNAL amin, amax, atmp;
   SIGNAL *b, *d, z, *p;
   int i = 0, j = 0, k, l, m, nn, min_level, level, window;
   char cont, winsig, wavelet_type, win_type, filetype;
   char path[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
   char fname[_MAX_FNAME], ext[_MAX_EXT], fname1[_MAX_FNAME], ext1[_MAX_EXT];
   char fn_in[_MAX_PATH], fn_out[_MAX_PATH];
   char fn_out1[_MAX_PATH], fn_out2[_MAX_PATH], bufr[5];

   _clearscreen( _GCLEARSCREEN );
   _settextposition( 3, 5 );
   printf( "Enter INPUT file INCLUDING .ext and #REAL Samples (f n): ");
   scanf( "%s %d", fn_in, &NSAMPLES );

   PWR_2 = power_of_two( NSAMPLES );

   strcpy( path, fn_in );
   printf( "\n%s\n", path );
   _splitpath( path, drive, dir, fname, ext );
   strcpy( fn_out, fname );
   printf( "fn_out = %s\n", fn_out );


   _settextposition( 10, 5 );
   printf( "Enter Input FILETYPE (Single or xy-Pairs, S/P): ");
   filetype = getche();
   if( filetype == 's' || filetype == 'S' )
   {
      strcpy( fname1, fname );
```

```
    strcpy( ext1, "mod" );
    _makepath( fn_out1, drive, dir, fname1, ext1 );
  }
  _settextposition( 12, 5 );
  printf( "Do you want DAUB4, DAUB8, or HAAR WAVELET? (4/8/H): ");
  wavelet_type = getche();

  if( wavelet_type == '4' )
    NWAVCOEFS = 4;
  if( wavelet_type == '8' )
    NWAVCOEFS = 8;
  if( wavelet_type == 'H' || wavelet_type == 'h' )
    NWAVCOEFS = 2;
  printf( "\n   NWAVCOEFS = %d\n", NWAVCOEFS );

/* Allocate MEMORY */
  sig_real = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( sig_real ) )
    exit( 9 );
  sig_imag = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( sig_imag ) )
    exit( 9 );
  sig_fft_mag = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( sig_imag ) )
    exit( 9 );
  cep_real = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( cep_real ) )
    exit( 9 );
  cep_imag = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( cep_imag ) )
    exit( 9 );
  a = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( a ) )
    exit( 9 );
  c = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( c ) )
    exit( 9 );
  q = calloc( 2*(NSAMPLES+NWAVCOEFS), sizeof( SIGNAL ) );
  if( BadMem( q ) )
    exit( 9 );
  b = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( b ) )
    exit( 9 );
  d = calloc( NSAMPLES+NWAVCOEFS, sizeof( SIGNAL ) );
  if( BadMem( d ) )
    exit( 9 );
  p = calloc( 2*(NSAMPLES+NWAVCOEFS), sizeof( SIGNAL ) );
  if( BadMem( p ) )
    exit( 9 );

  _settextposition( 15, 5 );
  printf( "Do you want signal WINDOWED? (Y/N): ");
```

```
 winsig = getche();
 if( winsig == 'Y' || winsig == 'y' )
 {
    _settextposition( 16, 5 );
    printf( "Select Window Function: ");
    printf( "\n          (R) Rectangular" );
    printf( "\n          (P) Parzen" );
    printf( "\n          (H) Hanning" );
    printf( "\n          (W) Welch" );
    printf( "\n          (Z) Hamming" );
    printf( "\n          (B) Exact Blackman" );
    _settextposition( 16, 30 );
    win_type = getche();
    _settextposition( 25, 1 );
    if( win_type == 'R' || win_type == 'r' )
       window = 0;              /* Rectangular */
    if( win_type == 'P' || win_type == 'p' )
       window = 1;              /* Parzen (Triangular) */
    if( win_type == 'H' || win_type == 'h' )
       window = 2;              /* Hanning */
    if( win_type == 'W' || win_type == 'w' )
       window = 3;              /* Welch */
    if( win_type == 'Z' || win_type == 'z' )
       window = 4;              /* Hamming */
    if( win_type == 'B' || win_type == 'b' )
       window = 5;              /* Exact Blackman */
 }
 printf( "\n\n" );

/* Read Data */
/* get_data( SIGNAL *data, SIGNAL *x, char *file_name, int ndata,
                  char filetype = 's' = SINGLE ==> Y-only
                                = 'p' = PAIRS  ==> X & Y */
 printf( "\nREADING... FILE: %s\n", fn_in );
 get_data( sig_real, b, fn_in, NSAMPLES, filetype );
 if( filetype == 'S' || filetype == 's' )
    store_data( sig_real, b, fn_out1, NSAMPLES, 0, 1, 0, 'p' );
 for(i = 0; i < NSAMPLES+NWAVCOEFS; i++ )
    printf( "b[%d] = %13.6lf    sig_real[%d] = %13.6lf\n",
                           i, b[i], i, sig_real[i] );
 printf( "\n                          ... press any key\n" );
 getch();

 for(i = 0; i < NSAMPLES+NWAVCOEFS; i++ )
    sig_imag[i] = 0.0;
 if( winsig == 'Y' || winsig == 'y' )
    WindowFFTData( sig_real, sig_real, (int)(NSAMPLES+NWAVCOEFS), window );

 for(i = 0; i < NSAMPLES+NWAVCOEFS; i++ )
    a[i] = sig_real[i];
 amax = fabs( a[0] );
 amin = fabs( a[0] );
```

```
  for( k = 0; k < NSAMPLES+NWAVCOEFS; k++ )
  {
     if( fabs( a[k] ) > amax )
        amax = fabs( a[k] );
     if( fabs( a[k] ) < amin )
        amin = fabs( a[k] );
  }

 /* for( k = 0; k < NSAMPLES+NWAVCOEFS; ++k )
  {
     atmp = fabs( a[k] ) / amax;
     if( atmp != 0.0 )
        atmp = B / (B - log( atmp ));
     putchar( (int)(atmp*255.0) );
  }
 */

  /* Generate Fourier Transform (FFT) */

  printf( "Doing FFT\n" );
  FFTCalc( sig_real, sig_imag, (int)NSAMPLES );
 /* for(i = 0; i < NSAMPLES/2+1; i++ )
  {
     printf( "Sig_Real[%4d] = %10f    Sig_Imag[%4d] = %10f\n",
                        i, sig_real[i], i, sig_imag[i] );
     getch();
  } */


  /* Store Fourier (FFT) Coefficients */

  strcpy( fn_out1, fn_out );
  strcat( fn_out1, ".fft" );
  printf( "FFT Mag & Freq CONVERSION\n" );
  for( i = 0; i <= NSAMPLES/2; i++ )
  {
     sig_fft_mag[i] = FFTMagnitude( sig_real, sig_imag, (int)NSAMPLES, i );
  }
  store_data( sig_fft_mag, b, fn_out1, NSAMPLES/2+1, 0, 1, 0, 'p' );

#if defined( CEPSTRUM )
  /* Generate Cepstrum Transform (CT) */

  printf( "Doing CT\n" );
  for(i = 0; i < NSAMPLES/2; i++ ) /* Throws away the values at the
                        Nyquist Frequency (last FFT value),
                        which should be zero anyway */
  {
     if( sig_real[i] != 0.0 )
        cep_real[i] = log10( pow( sig_real[i], 2.0 ) );
     else
        cep_real[i] = 0.0;
```

```
      if( sig_imag[i] != 0.0 )
         cep_imag[i] = log10( pow( sig_imag[i], 2.0 ) );
      else
         cep_imag[i] = 0.0;
      printf( "Cep_Real[%4d] = %10f    Cep_Imag[%4d] = %10f\n",
                        i, cep_real[i], i, cep_imag[i] );
      getch();
    }
    FFTCalc( cep_real, cep_imag, (int)(NSAMPLES/2) );
    for(i = 0; i < NSAMPLES/4+1; i++ )
      printf( "Cep_Real[%4d] = %10f    Cep_Imag[%4d] = %10f\n",
                        i, cep_real[i], i, cep_imag[i] );
    getch();

  /* Store Cepstrum (FFT) Coefficients */

    strcpy( fn_out1, fn_out );
    strcat( fn_out1, ".cep" );
  /* for(i = 0; i < NSAMPLES/4+1; i++ )
      cep_real[i] = cep_real[i]/(SIGNAL)(NSAMPLES/2);
  */ store_data( cep_real, b, fn_out1, NSAMPLES/4+1, 0, 1, 0, 'p' );
#endif

  /* Generate INVERSE Fourier Transform (IFFT) */

    printf( "Doing IFFT\n" );
    FFTInvCalc( sig_real, sig_imag, (int)NSAMPLES );
  /* for(i = 0; i < NSAMPLES; i++ )
      printf( "Sig_Real[%4d] = %10f    Sig_Imag[%4d] = %10f\n",
                        i, sig_real[i], i, sig_imag[i] );
    getch();
  */

  /* Store INVERSE Fourier (IFFT) Coefficients */

    strcpy( fn_out1, fn_out );
    strcat( fn_out1, ".ift" );
    store_data( sig_real, b, fn_out1, NSAMPLES, 0, 1, 0, 'p' );

  /* Generate Wavelet Transform */

    if( wavelet_type == '4' )
      min_level = 4;
    if( wavelet_type == '8' )
      min_level = 8;
    if( wavelet_type == 'H' || wavelet_type == 'h' )
      min_level = 2;

    nn = NSAMPLES;
    level = 0;
    z = (SIGNAL)0.5;
    while( nn >= min_level )
```

```
{
   printf( "Doing FWT\n" );
   if( wavelet_type == '4' )
      wt1( a, NSAMPLES, nn, min_level, 1, daub4 );
   if( wavelet_type == '8' )
      wt1( a, NSAMPLES, nn, min_level, 1, daub8 );
   if( wavelet_type == 'H' || wavelet_type == 'h' )
      wt1( a, NSAMPLES, nn, min_level, 1, haar2 );

/* Store Wavelet Coefficients */

   strcpy( fn_out2, fn_out );
   strcat( fn_out2, ".wav" );

   for( k = 0; k < nn; k += 2 )
      b[k/2] = (b[k] + b[k+1]) / 2.0;
   for( k = 0; k < nn; k += 2 )
      b[k/2+nn/2] = b[k/2];

/* Generate Fourier Transform (FFT) of WAVELET COEFFICIENTS */

  /* For SCALE Coefficients: */

   printf( "Doing FFT on SCALE Coefficients\n" );
   for( k = 0; k < nn/2; k++ )
   {
      sig_real[k] = a[k];
      sig_imag[k] = 0.0;
      d[k] = (SIGNAL)k;
   }
   FFTCalc( sig_real, sig_imag, (int)(nn/2) );
  /* for(i = 0; i < nn/4+1; i++ )
      printf( "Sig_Real[%4.0f] = %10f    Sig_Imag[%4.0f] = %10f\n",
                  d[i], sig_real[i], d[i], sig_imag[i] );
   getch(); */
   printf( "SCALE  FFT Mag & Freq CONVERSION\n" );
   for( i = 0; i <= nn/4; i++ )
   {
      sig_fft_mag[i] = FFTMagnitude( sig_real, sig_imag, (int)(nn/2), i );
   }

  /* Merge Fourier (FFT) Coefficients of WAVELET SCALE in amongst
        Wavelet Coefficient Vector per Press' Format          */
   printf( "MERGING Scale FFT Data w/ Wavelet Vector\n" );
   for( k = 0; k < nn/2; k++ )  /* Original SCALE Coefficients */
   {
      q[k] = a[k];
      p[k] = b[k];
   }
   for( k = 0; k < nn/4+1; k++ ) /* INSERT Scale FFT Coefficients */
   {
      q[k+nn/2] = sig_fft_mag[k];
```

```
      p[k+nn/2] = (SIGNAL)k;
    }
    for( k = nn/2; k < nn; k++ ) /* CONTINUE w/ DETAIL Coefficients */
    {
      q[k+nn/4+1] = a[k];
      p[k+nn/4+1] = b[k];
    }

  /* For DETAIL Coefficients: */

    printf( "Doing FFT on DETAIL Coefficients\n" );
    for( k = nn/2; k < nn; k++ )
    {
      sig_real[k-nn/2] = a[k];
      sig_imag[k-nn/2] = 0.0;
      d[k] = (SIGNAL)(k - nn/2);
    }
    FFTCalc( sig_real, sig_imag, (int)(nn/2) );
  /* for(i = 0; i < nn/4+1; i++ )
      printf( "Sig_Real[%4.0f] = %10f    Sig_Imag[%4.0f] = %10f\n",
                d[i], sig_real[i], d[i], sig_imag[i] );
    getch(); */
    printf( "DETAIL  FFT Mag & Freq CONVERSION\n" );
    for( i = 0; i <= nn/4; i++ )
    {
      sig_fft_mag[i] = FFTMagnitude( sig_real, sig_imag, (int)(nn/2), i );
    }

  /* Merge Fourier (FFT) Coefficients of WAVELET DETAIL in amongst
       Wavelet Coefficient Vector per Press' Format          */
    printf( "MERGING Scale FFT Data w/ Wavelet Vector\n" );
    for( k = 0; k < nn/4+1; k++ ) /* INSERT DETAIL FFT Coefficients */
    {
      q[k+nn+nn/4+1] = sig_fft_mag[k];
      p[k+nn+nn/4+1] = (SIGNAL)k;
    }
    for( k = nn; k < NSAMPLES; k++ ) /* CONTINUE w/ DETAIL Coefficients */
    {
      q[k+nn/2+2] = a[k];
      p[k+nn/2+2] = b[k];
    }
    store_data( q, p, fn_out2, NSAMPLES+nn/2+2, level, 1, 0, 'p' );

/* Generate INVERSE Wavelet Transform */

    printf( "Doing IFWT\n" );
    for( k = 0; k < NSAMPLES+NWAVCOEFS; k++ )
    {
      c[k] = a[k];
      d[k] = b[k];
    }
    if( wavelet_type == '4' )
```

```
            wt1( c, NSAMPLES, nn, min_level, -1, daub4 );
        if( wavelet_type == '8' )
            wt1( c, NSAMPLES, nn, min_level, -1, daub8 );
        if( wavelet_type == 'H' || wavelet_type == 'h' )
            wt1( c, NSAMPLES, nn, min_level, -1, haar2 );

    /* Store INVERSE Wavelet Coefficients */

        strcpy( fn_out2, fn_out );
        strcat( fn_out2, ".inv" );

        k = 0;
        for( j = 0; j < nn; j += 2 )
        {
            d[j] = b[k] - z;
            d[j+1] = b[k] + z;
            k++;
        }
        store_data( c, d, fn_out2, NSAMPLES, level, 1 ,0, 'p' );

    /* Plot Signal and Wavelets (via SPLOT) */

        if( winsig == 'Y' || winsig == 'y' )
            plot_wave( fn_out, level+1, 1, filetype );
        else
            plot_wave( fn_out, level+1, 0, filetype );

        _settextposition( 15, 10 );
        if( nn > min_level )
        {
            printf( "Continue with another level of expansion? (Y/N): \n\n" );
            cont = getch();
            if( cont == 'N' || cont == 'n' )
                break;
        }
        nn /= 2;
        level++;
        z = z * 2.0;
    }
    fcloseall();
    exit( 0 );
}
```

DWAVEPLT.c

```
/*              This Program's Name is   DWAVEPLT.c
   This program uses SPLOT to plot the files generated by DAUBWAVE.c
   (from Purdue) utilizing the same filename extensions.
*/

#include <math.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <malloc.h>

#define MAXLEVELS 13    /* Maximum of 4096 data points (levels 0-12 incl.) */

/*****************************************************************************/
FILE *file_mgr( char *fn, char *mode )
{
   FILE *fp;

   switch( mode[0] )
   {
      case 'w':
      case 'W':      /* OPEN File for WRITING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
         else
         {
            printf( "\nUnable to open FILE :  %s!", fn );
            exit( 9 );
         }
      }
      case 'r':
      case 'R':      /* OPEN File for READING */
      {
         if ( ( fp = fopen( fn, mode ) ) != NULL )
            break;
         else
         {
            printf( "\nUnable to open FILE :  %s!", fn );
            exit( 9 );
         }
      }
      default:
      {
         printf( "\nBAD FILE MANAGER PARAMETERS, EXITING!!" );
         exit( 99 );
      }
   }
   return( fp );
}
```

```
/*****************************************************************************/
void generate_SPLOT_plt_file( char *fn, char *fn_ext[5],
                              int nlevels, int NSAMPLES )
{
   FILE *fpo;
   char path[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
   char fname[_MAX_FNAME], ext[_MAX_EXT], buf[_MAX_PATH + 10];
   char sig_file[_MAX_FNAME+_MAX_EXT];
   int i, n;

   strcpy( path, fn );
   printf( "\n%s\n", path );
   _splitpath( path, NULL, NULL, fname, ext );
   strcpy( ext, "plt" );
   _makepath( path, NULL, NULL, fname, ext );
   printf( "%s\n", path );
   fpo = file_mgr( path, "w" );
   fprintf( fpo, "DASH ON\n" );
/*   fprintf( fpo, "xsamples %d\n", NSAMPLES );   */

  /* Plot DATA */

   fprintf( fpo, "xgrid\n" );
   fprintf( fpo, "ygrid\n" );
   fprintf( fpo, "zgrid\n" );
   fprintf( fpo, "symbols\n" );
   fprintf( fpo, "title WAVELET DataFiles:  %s\n", fn );
   fprintf( fpo, "xlabel Sample Time Index(int)\n" );
   fprintf( fpo, "ylabel Magnitude\n" );
   fprintf( fpo, "zlabel Scale (Frequency) Index\n" );

   fprintf( fpo, "xsamples %d\n", nlevels );
   fprintf( fpo, "xstart 1\n" );
   fprintf( fpo, "xend %d\n", nlevels );
   fprintf( fpo, "y(x)=x\n" );
   fprintf( fpo, "usez 1\n" );
   fprintf( fpo, "xsamples %d\n", NSAMPLES );
   fprintf( fpo, "xstart 0\n" );
   fprintf( fpo, "xend %d\n", NSAMPLES );
   fprintf( fpo, "rotate\n" );

   strcpy( sig_file, fname );
   strcat( sig_file, ".sig" );
   printf( "%s\n", sig_file );
   fprintf( fpo, "open %s\n", sig_file );
  /* fprintf( fpo, "skip 1\n" ); */
   fprintf( fpo, "ready %d\n", NSAMPLES );    /* PlotSet #2 is ORIGINAL */
   fprintf( fpo, "close\n" );

   n = NSAMPLES / 2;
   for( i = nlevels; i > 0; i-- ) /* All DETAILS, Level 0=COURSEST SCALE */
```

```
    {
      strcpy( sig_file, fname );
      strcat( sig_file, fn_ext[i] );
      printf( "%s\n", sig_file );

      fprintf( fpo, "open %s\n", sig_file );
     /* fprintf( fpo, "skip 1\n" ); */
      fprintf( fpo, "read %d\n", n );
      fprintf( fpo, "close\n" );
      n = n/2;
    }

   fprintf( fpo, "plot3d\n" );
  /* fprintf( fpo, "wait\n" ); */

   fclose( fpo );
   sprintf( buf, "SPLOT %s", path );
  /* system( buf ); */
}

/*****************************************************************************/
/* HyperHex - This routine takes an integer between 0 and 36 and converts
       it into a character that goes from 0-9,A-Z.
*/
char HyperHex( int value )
{
  if( (value > 35) || (value < 0) )
    {
    fprintf( stderr, "Value too big to specify the file extension properly.\n" );
    exit( 1 );
    }
  if( value < 10 )
    {
    return( '0' + value );
    }
  else
    {
    return( 'A' + value - 10 );
    }
}

/*****************************************************************************/
char **FileEXT( int wavelet, int maxlevel )
{
   char **ext;   /* .abc for each Wavelet Decomposition Level */
   int i;

  /* construct the output name */

   ext = (char **)calloc( MAXLEVELS, sizeof( char * ) );
   for( i = 0; i < MAXLEVELS; i++ )
      ext[i] = (char *)calloc( 5, sizeof( char ) );
```

```
    for( i = maxlevel; i >= 0; i-- )
    {
       ext[i][0] = '.';
       ext[i][1] = HyperHex( maxlevel );
      /* use the wavelet type for the second byte */
       ext[i][2] = HyperHex( wavelet );
      /* use the level of the decomposition for the third byte */
       ext[i][3] = HyperHex( i );
       ext[i][4] = '\0';
       printf( "ext[%d] = %s\n", i, ext[i] );
    }
    printf( "                    ... press any key\n" );
    getch();
    return( ext );
}

/*****************************************************************************/
void fix_wavelet_files( char *fn, char *fn_ext[5], int nlevels, int NSAMPLES )
{
    FILE *fpo;
    char wav_file[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
    char fname[_MAX_FNAME], ext[_MAX_EXT], buf[_MAX_PATH + 10];
    double *ydata, *xdata;
    float y_tmp;
    int i, j, k, l;

    printf( "Fixing WAVELET Files ...\n" );
    ydata = (double *)calloc( NSAMPLES, sizeof( double ) );
    xdata = (double *)calloc( NSAMPLES, sizeof( double ) );
    strcpy( wav_file, fn );
    printf( "\n%s\n", wav_file );
    _splitpath( wav_file, drive, dir, fname, ext );
    for( i = 0; i < NSAMPLES; i++ )
       xdata[i] = (double)i;
    k = NSAMPLES/2;
    l=2;
    for( i = nlevels; i >= 0; i-- )
    {
       strcpy( ext, fn_ext[i] );
       _makepath( wav_file, drive, dir, fname, ext );
       printf( "i = %d   %s\n", i, wav_file );
       fpo = file_mgr( wav_file, "r" );
       for( j = 0; j < k; j++ )
       {
          fscanf( fpo, "%e", &y_tmp );
          ydata[j] = (double)y_tmp;
         /* printf( "ydata[%d][%d] = %lf\n", i, j, ydata[j] ); */
       }
       fclose( fpo );
       fpo = file_mgr( wav_file, "w" );
       for( j = 0; j < k; j++ )
          fprintf( fpo, "%13.6e %13.6e\n", (float)xdata[j*l], (float)ydata[j] );
```

```
        fclose( fpo );
        k = k/2;
        l = l*2;
    }
}

/*****************************************************************************/
void main( int argc, char *argv[] )
{
    FILE *fpo;
    char path[_MAX_PATH], drive[_MAX_DRIVE], dir[_MAX_DIR];
    char fname[_MAX_FNAME], ext[_MAX_EXT], buf[_MAX_PATH + 10];
    char bufr[81], **fn_ext;
    int NSAMPLES;

    _clearscreen( _GCLEARSCREEN );
    strcpy( path, argv[1] );
    printf( "\n%s\n", path );
    _splitpath( path, drive, dir, fname, ext );
    strcpy( ext, "sig" );
    _makepath( path, drive, dir, fname, ext );
    printf( "%s\n", path );

    if( argc < 2 )
    {
        fprintf( stderr, "    Useage:\n\n" );
        fprintf( stderr, "\tDWAVEPLT <Input file>  <#Wavelet Order>  <#Levels>  <#DataSamples>\n" );
        fprintf( stderr, "\t\t(NOTES:  1.  Input file EXCLUDES .ext)\n" );
        fprintf( stderr, "\t\t(       2.  Wavelet Order 2=HAAR, rest=DAUBECHIES.)\n" );
        exit( 1 );
    }
    else
    {
        sprintf( bufr, "DaubWave %s -o %s -d%d -n3 -s%d\n",
                path, argv[1], atoi(argv[2]), atoi(argv[3]) );
        NSAMPLES = atoi( argv[4] );
        printf( "\nbufr = %s  ---  NSAMPLES = %d\n", bufr, NSAMPLES );
        printf( "                    ... press any key\n" );
        getch();
        system( bufr );
        fn_ext = FileEXT( atoi( argv[2] ), atoi( argv[3] ) );
        fix_wavelet_files(fname, fn_ext, atoi( argv[3] ), NSAMPLES);
        generate_SPLOT_plt_file( fname, fn_ext, atoi( argv[3] ), NSAMPLES );
    }
    exit( 0 );
}
```

VITA

Thomas Floyd Tibbals was born August 10, 1950 in the small northern Michigan town of Alpena on the beautiful shores of Lake Huron. He attended parochial elementary and high schools, and was more interested in all things mechanical rather than school, which offered only drafting to meet his interests. He graduated from Catholic Central High School in June 1968, and was not interested in a four year college, nor was he college material at that point. His interest in jet engine mechanics was squashed when the program was not offered at the local community college. The following November he entered the United States Air Force where he received training in aircraft hydraulic and pneumatic systems and later was placed in charge of all in-shop repair. Following a tour of duty in Thailand in support of the Viet Nam War, he was released from active duty as an E-5 in time to attend Alpena Community College in August 1972, with a strong ambition to become an engineer due to the exposure to complex aircraft systems.

Due to economic conditions, he moved to Cleveland, Ohio where he obtained a job as a hydraulic and pneumatic component assembler in order to establish residency for entrance into Cuyahoga Community College in September 1973. He received an Associate of Arts degree and entered Cleveland State University in September 1975 while continuing to work for the same small firm as a hydraulic and pneumatic component designer. In June 1977 he graduated Cum Laude with an Engineering Science/Physics degree, and in August 1977 was awarded the Bachelor of Science degree in Mathematics.

In May 1978 he entered the United States Air Force Officer Training School, and was commissioned in August 1978. He was stationed at Edwards AFB, California as a flight test engineer involved in evaluating hydraulic systems, icing characteristics, and arresting hook dynamics. In August 1982, he was released from active

duty and accepted the position of Chief Engineer and General Manager of a small hydraulic and pneumatic controls design firm in Cleveland, Ohio. In December 1986, he accepted his current position as a Senior Engineer with Sverdrup Technology, Inc. (AEDC Group) in the Propulsion Testing Technology section at Arnold Engineering Development Center near Tullahoma, Tennessee. He is a Major in the USAF Reserves.

The above details are presented to illustrate that one can still become an engineer even if his high school grades are not appropriate, and perhaps the better engineers are lost in our school system because we cater to those who would make it anyway, the academically gifted, even if their engineering insight is lacking, while leaving the *shop* students unchallenged. We must stop regarding shop courses as for those who have failed academically, and challenge these individuals to excell in their interests. Einstein stated that imagination was more important that knowledge, and in many cases, a skilled laborer will have the imagination to make something work when the engineer's knowledge has failed to solve the problem. Our schools do not recognize the imagination needed to create an item from raw materials or fix an item to work when the engineer's design is not adequate. Our politicians need not wonder why America is losing its manufacturing capability, it is due to the lack of respect for the skilled laborer and the enormous emphasis on professional careers. I wonder what good an engineer is if there is no machinist or welder or other craft or technician to build and keep running what he designed! For all those skilled laborers who encouraged me and assisted me with my bicycle, go-kart, and car modifications as a young tinkerer, you will always have my respect, THANKS!