Q [illegible]

PROCEEDINGS

SPIE—The International Society for Optical Engineering

Wavelet Applications II

Harold H. Szu
Chair/Editor

17–21 April 1995
Orlando, Florida

SPIE PROCEEDINGS SERIES

Volume 2491
Part One of Two Parts

The Wavelet Processing Workstation: An Interactive MATLAB Based Computational Tool for Wavelet Processing

A. Teolis J. S. Baras[*]
AIMS, Inc., 6110 Executive Blvd., Suite 850, Rockville MD, 20852-3904

ABSTRACT

We present a highly powerful, modular, and interactive software tool for the analysis of time-frequency coherent signals via wavelet transformations. A major design goal of the Wavelet Signal Processing Workstation (WSPW) is to maximize ease of use while minimizing programming complexity. As such, the WSPW makes ample use of graphical mouse driven user interfaces and, in turn, allows powerful signal processing, classification, and identification techniques to be rapidly implemented and tested. Because it has been developed using MATLAB, the WSPW is easily extensible and inherently portable between varying system architectures. Although the emphasis of this paper is on the wavelet representation of signals, the WSPW has proven itself a valuable tool in applications including radar source identification and signal classification.

Keywords: continuous wavelet transform, numerical software, graphical interface

1 INTRODUCTION

1.1 Objective

Our main objective is to provide a high level interactive software tool which makes sophisticated signal processing, classification, and identification capabilities accessible to a substantial base of users. Such a tool will provide potential users with an immediate ability to analyze and classify particular data of interest using sophisticated state of the art signal processing techniques such as the wavelet transform. In this paper we focus on the wavelet representation of signals.

A major design goal of the Wavelet Signal Processing Workstation (WSPW) is to maximize ease of use while minimizing programming complexity. As such, the WSPW makes ample use of graphical mouse driven user interfaces and, in turn, allows powerful signal processing, classification, and identification techniques to be rapidly implemented and tested. Because it has been developed using MATLAB, the WSPW is easily extensible and inherently portable between varying system architectures.

[*]University of Maryland, Department of Electrical Engineering and Institute for Systems Research, College Park, MD 20742

1.2 Non-orthogonal wavelet processing

A primary feature of the WSPW is the ability to perform "continuous" wavelet transformations with user specifiable analyzing wavelets. The wavelet transform is discussed in detail in Section 3.4. Because the analyzing wavelets may be (almost) arbitrary the resulting (discretized) wavelet transforms, in general, lead to redundant wavelet representations, i.e., non-orthogonal wavelet transforms. Except for a modest increase in computationally complexity (O(N log N) instead of O(N)) of the forward (and backward) wavelet transform, the nonorthogonal quality of the transform represents no great drawbacks. On the contrary, the ability to choose almost arbitrary wavelets affords the representation designer a great amount of freedom. For example, one may choose an analyzing wavelet which is bandlimited in frequency and has arbitrarily many vanishing moments in the time domain.

Using a WSPW graphical interface window the user can iteratively design analyzing wavelets in the frequency domain. The user may subsequently perform a wavelet analysis of selected data and then further refine the analyzing wavelet if desired. In this way the user is able to search for an "optimal" wavelet interactively.

1.3 Content

The main intent of this document is to describe the design, functionality, and underlying concepts which have led the development of the WSPW. It is not intended to serve as a users manual for the package.

2 DESIGN CONCEPTS

2.1 Development language

We have selected MATLAB as the primary development language for the WSPW. MATLAB is a very powerful matrix oriented numerical processing language which has been designed in an object oriented and extensible fashion. Essentially, MATLAB is a mature, well supported, and ideally suited language for the WSPW. Moreover, it supports a C-code interface should the need arise for greater control over processing routines. In short, MATLAB's extensiblility, object oriented design, inherent portability between varying system architectures, and its numerical processing abilities, have all led to the decision to develop the WSPW in MATLAB.

2.2 WSPW architecture

At the most basic level of the WSPW architecture is the processing *module*. Each WSPW module is a MATLAB function with a very special structure. In particular, each module consists of four main parts: input, numerical computation, dormant graphics and finally output. Figure 1 depicts the functional form of each WSPW module. Of the four component parts of the processing module, the numerical computation portion is where the bulk of the module's work is performed. It is also that portion which is most native to MATLAB.

The additional structure that WSPW adds is implemented in the form of several high level managers which control the overall processing flow. These managers include (i) Workspace Manager, for control

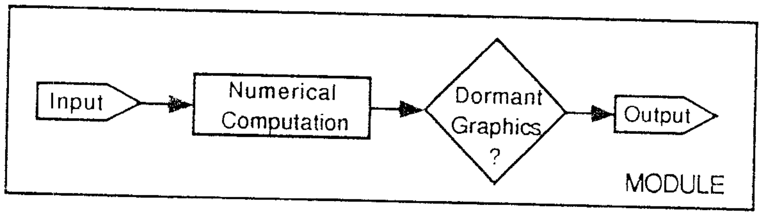


Figure 1: The WSPW module and its four main components

of window placement, and user interface controls; (ii) Display Manager, for control of graphical output; and (iii) Module Manager, for control of module input and output. Figure 2 displays a schematic view of the overall WSPW architecture. Module input/output as well as dormant graphical output is controlled by these high level processing managers. Dormant graphics code allows modules the option of presenting graphical output to the user. Each module may have a section of code dedicated to graphical output whose display and format is controlled via display management routines.

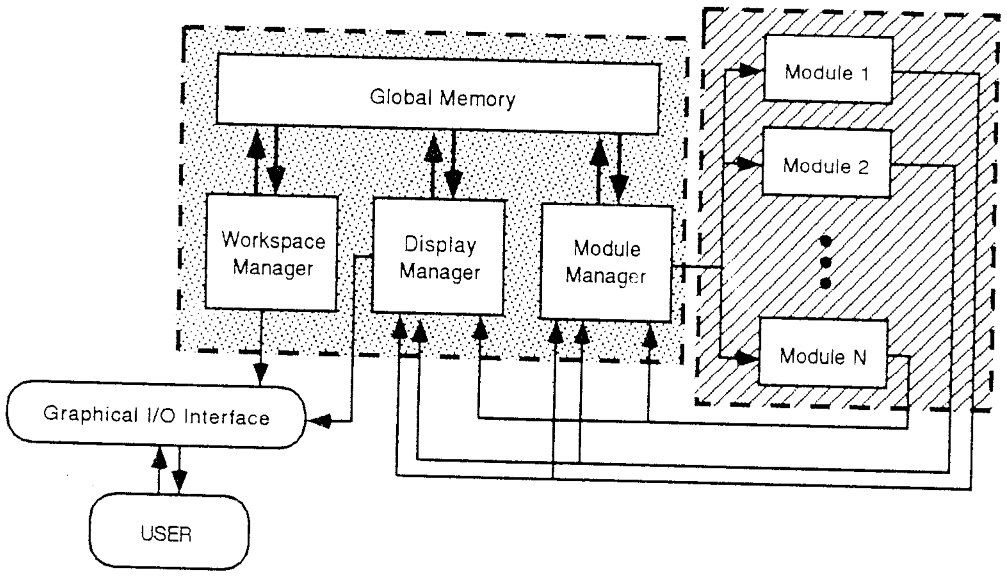


Figure 2: Wavelet Signal Processing Workstation Architecture

The main vehicle for communication among modules is directed by the module manager through the global memory. Modules may ask the module manager to place objects[1] in global memory or retrieve objects placed there by other modules.

Routines for user interface (UI) building and manipulation are included in the graphical I/O (GIO) interface. UI modules are special modules which simply create a user interface. There are GIO routines to help quickly build and edit interfaces. These routines allow the specification of placement of controls

[1]An object is a matrix with a known intepretation, e.g., a *function matrix* is a matrix which has the interpretation that each of its columns is a one dimensional signal.

and display plots in a simulation. Other GIO routines allow modules to query for information from interfaces by name.

WPSW is designed in a completly open architecture. Accordingly, within this architecture users have the freedom to operate at any one of the various levels. At the highest level the user will communicate via the predefined graphical user interfaces and manipulate data via arrangements of predefined modules. At the lowest level a user can write custom modules and user interfaces.

2.3 Synthesizing and loading data

Included with the WSPW are several modules for creating synthesized data for processing demonstrations and algorithm verification. The most useful among these routines is `make_sine` which allows the creation of sums of sinusoidal signals of varying frequency, amplitude and time support. A higher level routine, `signal`, takes one of a predefined set of names as its input and returns associated signals. Of course, users are free to create their own synthesis modules as well.

Typically, users will have a database of signals on which they would like to perform some processing. File formats from database to database are likely to have a great variety. Despite this situation there is a large amount of generalization which we may suppose about a signal database:

(i) file names usually carry some meaning, i.e., they are constructed in a manner indicative in some way of the data which is contained in the file,

(ii) files usually contain a header portion which describes various attributes associated with the data, e.g., indexing information, time of day data was collected, processing history, etc., and

(iii) file structures are usually well defined according to some prescribed though perhaps greatly flexible arrangement of data in the file.

Given these observations we have provided some structure to the signal database via database modules. Ultimately, however, it is difficult to avoid the need for users to provide some sort of low level I/O routines specific to their own databases. Perhaps, the cleanest and easiest solution to the file format problem is to convert all database files to the standard MATLAB file binary format; however, in this case all data must be stored as double precision floating point numbers. Such a storage scheme may require much more memory than is necessary for many types of data.

3 MULTI-RATE SIGNAL PROCESSING

Multi-rate signal processing refers to the joint processing of discrete signals arising from the sampling of analog signals at different sampling rates. Our interest in multi-rate signal processing is spawned by the implementation of the "continuous" wavelet transform where signals at different sampling rates are naturally generated. Accordingly, techniques to handle multi-rate signals and systems are a prerequisite to the development of the WSPW. In this regard, the main tool which we have developed is that of the "sampled signal structure". The sampled signal structure is a concept which easily facilitates the implementation of multi-rate signal processing.

3.1 Sample signal structure

Simply stated, a sampled signal structure is a pair of MATLAB variables, e.g., [y yd], which describes a vector (sample set) y and an associated discrete lattice yd. The underlying notion here is that the vector y comes from an analog signal sampled on the lattice described by yd. We assume a finite sample vector y that is a $1 \times N$ row vector and the sample lattice is described by a 1×2 vector yd. The first element yd(1) of yd indicates the assumed uniform sample period Δ while the second element yd(2) is T where $[-T, T]$ is the symmetric interval on which the signal is defined. More precisely, the discrete vector y arises from the sampling of a function $y \in L^2[-T,T]$ (the finite energy signals on $[-T,T]$) as

$$\mathtt{y(n)} \;= y((n - \frac{N}{2} - 1)\Delta), \quad n = 1, 2, \ldots, N,$$

where $T = \frac{N\Delta}{2}$. Figure 3 shows the sampling lattice Γ associated with a sample structure $\{\Delta, T\}$. The sampling lattice is uniformly distributed over the interval $[-T, T]$ as

$$\Gamma = \Gamma(\Delta, T) \triangleq \left\{(n - \frac{N}{2} - 1)\Delta\right\}_{n=1}^{N},$$

where $N = 2T/\Delta$ (required to be an integer).

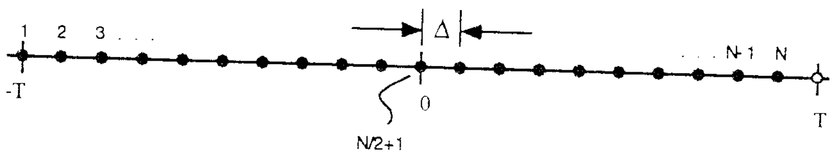


Figure 3: Discrete lattice specified by a sample structure

We shall develop all of our processing in terms of sampled signals. The pair $y_d = \{\Delta, T\}$ is called the *sampling structure* associated with the function y. In MATLAB we shall adhere to the convention that if a signal is named f then its associated sampling structure is named fd. In other words, the suffix "d" is appended to a signal's name to yield the name of the signal's sampling structure. We note that not every sample rate, interval pair $\{\Delta, T\}$ in $\mathbb{R} \times \mathbb{R}$ is admissible. Namely, the condition

$$\frac{2T}{\Delta} \in \mathbb{Z}$$

must hold. This condition has the consequence of forcing 0 to be a member of the sampling lattice Γ.

3.2 Algebraic manipulation

Given two sampled signals with possibly differing sampling structures a basic requirement is to perform elementary algebraic operations like addition, subtraction and multiplication. Our basic approach to performing operations on signals with mismatched sample structures involves simply a resampling process. For example, signals to be combined in some algebraic manner are each resampled on some common sampling lattice. Now with identical sampling structures, the resulting resampled signal vectors may be directly combined via the desired algebraic operation.

Although the resampling strategy is sound in principle there are a host of caveats which must be addressed in a successful implementation. In particular, the resampling process must satisfy certain properties if the result of the algebraic operation is to be accurate and meaningful. The main problem is to insure that the new sampling rate is consistent with the frequency content, i.e., it satisfies the Nyquist rate, of the combined signal. For example, in the case of multiplication the proper sampling rate for the product signal is the sum of individual sampling rates.

In the remainder of this section we shall develop some mathematical tools which describe our approach to the problem. We limit the discussion to the case of one dimensional signals. As mentioned earlier, a basic premise in this work is that all vectors are assumed to come from the sampling of an underlying analog signal.

Let f and g be two analog bandlimited signals with bandlimits of Ω_1 and Ω_2 respectively. The required Nyquist sampling rates for each signal is respectively $\Delta_1^{-1} = 2\Omega_1$ and $\Delta_2^{-1} = 2\Omega_2$. Let L_Ω denote a sampling operator (at a rate 2Ω) as

$$\mathtt{f} = L_\Omega f \overset{\triangle}{=} \{f(t_n)\}_{n=1}^{N} \tag{1}$$

where $t_n = (n - \frac{N}{2} - 1)\Delta$, $N = \frac{2T}{\Delta}$ and $\Delta^{-1} = 2\Omega$. As is well known via the classical sampling theorem, meeting the Nyquist sampling rate insures that a bandlimited signal is recoverable from its samples through sinc interpolation. Thus, in principle, the operator L_Ω has an inverse L_Ω^{-1}. Further, let $R_{\Omega,\Omega'} \overset{\triangle}{=} L_{\Omega'} L_\Omega^{-1}$ be the resampling operator (resamples an Ω' sampled signal at the rate governed by Ω). With $\odot$ the algebraic operation $(+\text{-}/{}^*)$ of interest, the general form of the operation on the two sampled mismatched signal vectors $L_{\Omega_1} f$ and $L_{\Omega_2} f$ is

$$L_\Omega(f \odot g) = (R_{\Omega,\Omega_1}(L_{\Omega_1} f)) \odot (R_{\Omega,\Omega_2}(L_{\Omega_2} f)).$$

Here the proper value of Ω is determined by the algebraic operation. We shall end this section here since a full mathematical analysis, though necessary, is beyond the scope of this paper and we shall relate such an analysis elsewhere.

3.3 "Continuous" Fourier transform

$L^2(\mathbb{R})$ is the space of complex-valued finite energy signals defined on the real line $\mathbb{R}$. The *norm* of an element $f \in L^2(\mathbb{R})$ is

$$\|f\|_2 \overset{\triangle}{=} \int |f(t)|^2 dt < \infty,$$

where integration is over $\mathbb{R}$, and the *inner product* of $f, g \in L^2(\mathbb{R})$ is

$$\langle f, g \rangle = \int f(t)\overline{g}(t)dt.$$

Recall also that the *convolution* of $f, g \in L^2(\mathbb{R})$ is defined by $f * g(t) \overset{\triangle}{=} \int f(u)g(t-u)du$. The *Fourier transform* of a signal $f \in L^2(\mathbb{R})$ is

$$\widehat{f}(\gamma) = \int f(t)\, e^{-j2\pi t\gamma} dt.$$

for $\gamma \in \widehat{\mathbb{R}} (\overset{\triangle}{=} \mathbb{R})$, where convergence of the integral to $\hat{f}$ is in the L^2-sense.

Suppose f is an Ω bandlimited signal. Given a sampled version $\mathtt{f}$ of f it is often desirable to obtain an approximation to the Fourier transform $\hat{f}$ from knowledge of $\mathtt{f}$. Such an approximation may be made using the fast Fourier transform (FFT) with care given to the sampling structure of the signal to be transformed. To be more precise let $\mathtt{f} = L_\Omega f$ as in Equation (1) with the sample structure $\mathtt{fd} = \left\{\frac{1}{2\Omega}, T\right\}$. The discrete Fourier transform $\hat{\mathtt{f}}$ of $\mathtt{f}$ is

$$\hat{\mathtt{f}} \overset{\triangle}{=} \left\{\hat{\mathtt{f}}_k\right\} \overset{\triangle}{=} \left\{\sum_{n=1}^{N} \mathtt{f}_n e^{-j2\pi nk/N}\right\}.$$

The sampled version $L_T\hat{f}$ of the continuous Fourier transform $\hat{f}$ of the signal f may be approximated from the sampled version $L_\Omega f$ as[2]

$$\begin{aligned} L_T\hat{f} \overset{\triangle}{=} \left\{\hat{f}\left(\frac{k}{2T}\right)\right\} &\overset{\triangle}{=} \left\{\int f(t)\, e^{-j2\pi t \frac{k}{2T}}\right\} \\ &\approx \left\{\frac{1}{2\Omega}\sum f\left(\frac{n}{2\Omega}\right) e^{-j2\pi \frac{n}{2\Omega}\frac{k}{2T}}\right\} \\ &= \left\{\frac{1}{2\Omega}\sum (L_\Omega f)_n\, e^{-j\frac{2\pi}{N}nk}\right\} \\ &= \frac{1}{2\Omega}\cdot \hat{\mathtt{f}} \end{aligned}$$

since $2\Omega \cdot 2T = N$. Included in Table 1 are the relationships between the sample structures in the time and frequency domains.

signal	*sample vector*	*sample structure*
f	$\mathtt{f} = L_\Omega f = \{f(\frac{n}{2\Omega})\}_{n=-N/2}^{N/2-1}$	$\mathtt{fd} = \left\{\frac{1}{2\Omega}, T\right\}$
$\hat{f}$	$\hat{\mathtt{f}} = 2\Omega L_T\hat{f} = \left\{2\Omega\hat{f}(\frac{k}{2\Omega})\right\}_{k=-N/2}^{N/2-1}$	$\hat{\mathtt{fd}} = \left\{\frac{1}{2T}, \Omega\right\}$
D_sf	$D_s\mathtt{f} = L_{s\Omega}D_sf = \left\{s^{\frac{1}{2}}f(\frac{n}{2\Omega})\right\}_{n=-N/2}^{N/2-1}$	$D_s\mathtt{fd} = \left\{\frac{1}{2s\Omega}, \frac{T}{s}\right\}$

Table 1: Operational relations among sample structures

Figure 4 shows the computation of the continuous Fourier transform for the "textbook" example function $f(t) = te^{-ct}1_{[0,\infty]}$, where 1_X is the characteristic function of the set X. The closed form solution for the Fourier transform of this signal is easily computed to be $\hat{f}(\gamma) = \frac{1}{(c+j2\pi\gamma)^2}$. In the figure the value of $c = 4$. The figure shows the time domain signal f in the top plot ($U = 1_{[0,\infty]}$ is the step function), the result of the computation using the FFT (the WSPW module `sfft`) as just described (real and imaginary parts), and the closed form solution (real and imaginary parts) in the bottom plot.

[2]Of course, in the actual implementation the DFT is computed using the fast Fourier transform (FFT). Besides multiplication by the factor of 2Ω, the use of the FFT requires an additional unwrapping step to arrive at a proper sample vector/sample structure pair.

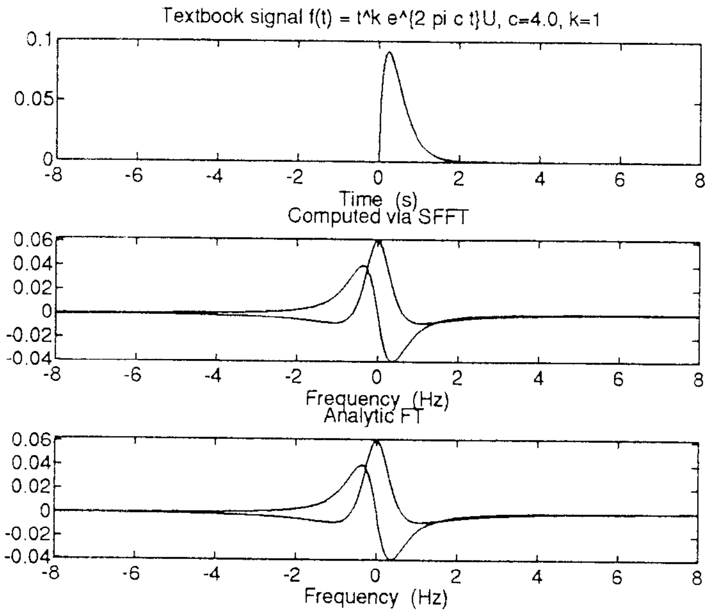


Figure 4: Textbook signal and its Fourier Transform computed analytically and via sfft

3.4 "Continuous" wavelet transform

In this section we introduce some basic notation and describe the continuous wavelet transform. There are three important unitary operators which are associated with the wavelet transform. They are the dilation, translation, and involution operators which are defined below. Of these three operators it is the dilation operator which is responsible for the multi-rate nature of the wavelet transform. This is because a non-trivial dilation will alter the sample structure of a signal, viz., Table 1.

Dilation For $s > 0$, the L^2-*dilation* operator D_s is defined by $D_sg(t) \triangleq s^{\frac{1}{2}}g(st)$ for $g \in L^2(\mathbb{R})$. As such, $(D_sg)\hat{}(\gamma) = s^{-\frac{1}{2}}\hat{g}(s^{-1}\gamma) = D_{s^{-1}}\hat{g}(\gamma)$.

Translation For $u \in \mathbb{R}$, the *translation* operator τ_u is defined by $\tau_ug(t) \triangleq g(t-u)$ for $g \in L^2(\mathbb{R})$. As such, $(\tau_ug)\hat{}(\gamma) = e^{-j2\pi u\gamma}\hat{g}(\gamma)$.

Involution For $g \in L^2(\mathbb{R})$, $\tilde{g}$ is the *involution* of g defined as $\tilde{g}(t) \triangleq \overline{g}(-t)$, where $\overline{g}$ is the complex conjugate of g.

For a fixed $g \in L^2(\mathbb{R})$, the continuous *wavelet transform* W_gf of $f \in L^2(\mathbb{R})$ is the function,

$$W_gf(t,s) \triangleq \langle f, \tau_tD_s\tilde{g}\rangle = (f * D_sg)(t), \tag{2}$$

defined on the *time-scale plane* $t \in \mathbb{R}, s > 0$. The inner product $\langle f, \tau_tD_s\tilde{g}\rangle$ yields a quantitive value which describes how much f is like $\tau_tD_s\tilde{g}$.

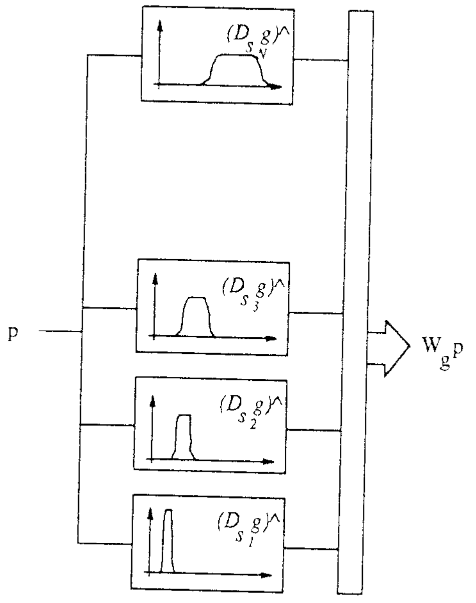


Figure 5: The Wavelet Transform as a filter bank

Because the wavelet transform may be written in terms of a convolution (2) it has a natural intepretation as a bank of linear filters. Each filter in the bank is determined by a particular value of the scale variable s. Thus, the wavelet transform of a signal f at a given scale $s = s_0$ is the output of a linear filter with impulse response $D_{s_0}g$. In general, we may discretize the scale axis to yield a countable set of scale values $\{s_m\}$ which in turn specifies the linear filter bank characterized by the set of impulse responses $\{D_{s_m}\tilde{g}\}$. Typically, we choose $s_k = a_0^k$ for some constant $a_0 > 0$. This situation is depicted schematically in Figure 5. Note that as the filter index m increases there are two primary effects on the filter frequency responses: (i) bandwidths increase, and (ii) the interval of frequency supports are translated towards higher frequencies. Thus, filters with larger indices m respond to higher frequencies.

4 PROCESSING EXAMPLE

Due to space limitations, it is difficult to demonstrate all of the functionality of the WSPW; however, we have chosen an example which illustrates some of the key features of the WSPW.

The example signal is a synthetic signal designed as a time progression of three windowed, yet overlapping, complex exponentials. This signal is called "opacket" for *overlapping packet*. Each successive windowed exponential has a higher frequency than its predecessor. The overlapping packet sequence is given by

$$\sum_{k=1}^{N} A(t - t_k)\exp(j2\pi\Omega_k t),$$

where $N = 3$, A is a window function, $(t_1, t_2, t_3) = (-35, -15, -25)\,\mu s$ and $(\Omega_1, \Omega_2, \Omega_3) = (6, 11, 20)$ MHz.

Figure 6 displays a WSPW I/O interface with the signal "opacket" (real and imaginary parts) and

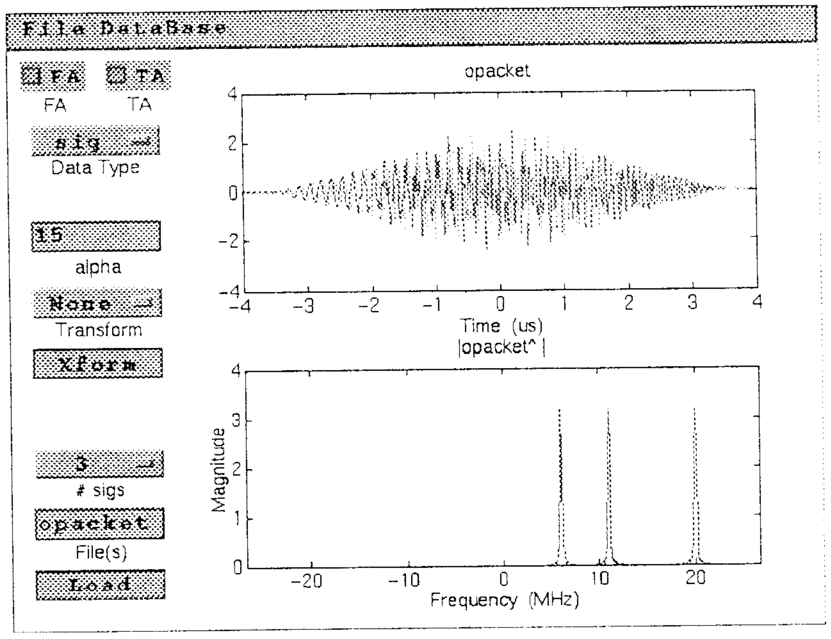


Figure 6: User I/O interface for WSPW showing the signal "opacket" and its Fourier transform

its magnitude Fourier transform. UI controls are distributed along the left hand side of the interface window. The "opacket" signal was loaded into the window by typing its name into the UI control labeled "File(s)" and then depressing the "Load" button. This action is coded in a corresponding loading process module. Plots of the signal and its Fourier transform are displayed by the loading module's dormant graphical output. Note that the Fourier transform window clearly indicates the presence of the frequencies 6, 11 and 20 MHz; however, there is no indication as to the times at which each frequency occurs. In other words, this is an illustration of the well known fact that the Fourier transform has good localization in frequency but none in time.

Figure 7 shows the "continuous" wavelet transform of the signal "opacket" as computed by the WSPW after selecting "cwt" in the "Xform" menu of Figure 6 and then depressing the "Transform" button. Note that the wavelet transform jointly resolves the signal "opacket" in time and frequency. This time-frequency localization property of the wavelet transform is dependent on the analyzing wavelet. If the analyzing wavelet is not well localized in time and frequency then the resulting wavelet transform will also fail to be so. Thus, it is important to design the "best" analyzing wavelet for a given application.

Towards the goal of providing a tool for wavelet optimization, the parametric bandlimited wavelet interface allows the user to vary analyzing wavelets so as to achieve desired affects. The parametric bandlimited wavelet interface is displayed in Figure 8. This is a specially designed tool for the specification of bandlimited analyzing wavelets and corresponding filter banks which are parameterized by a

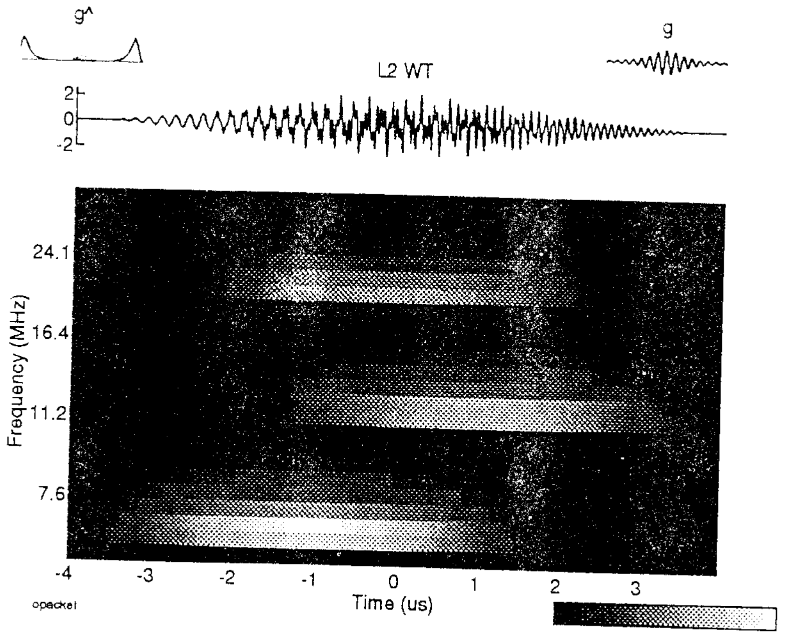


Figure 7: Wavelet transform of the signal "opacket"

few specific quantities. The form of the wavelet g which is depicted in the interface is

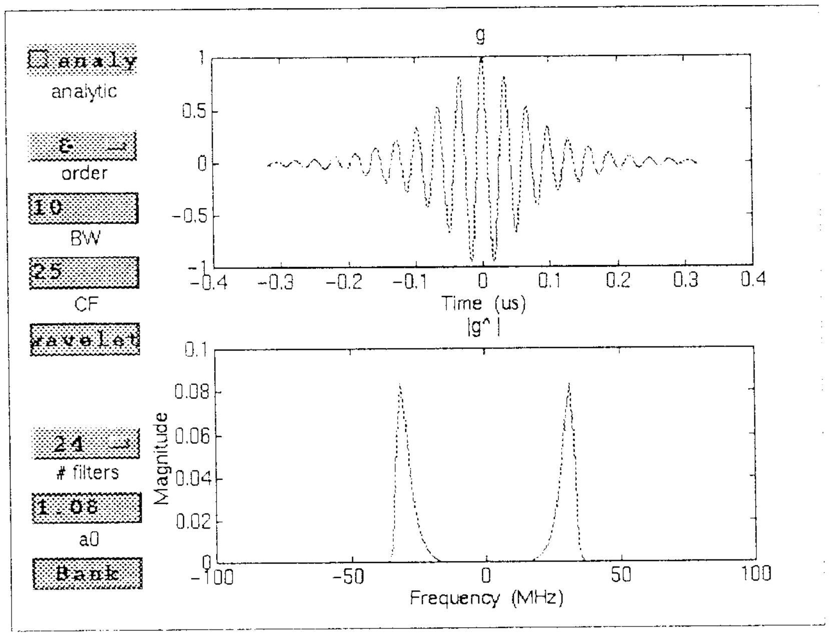
$$\hat{g}_0(\gamma) = \begin{cases} 0, & |\gamma| \leq \gamma_1, \\ \sin^2[\frac{\pi}{2\gamma_r}(\gamma - \gamma_1)], & |\gamma| \in [\gamma_1, \gamma_2), \\ 1, & |\gamma| \in [\gamma_2, \gamma_3), \\ \cos^2[\frac{\pi}{2\gamma_f}(\gamma - \gamma_3)], & |\gamma| \in [\gamma_3, \gamma_4), \\ 0, & \text{otherwise}. \end{cases}$$

$$\hat{g}(\gamma) \stackrel{\triangle}{=} \hat{g}_N(\gamma) = \gamma^N \cdot \hat{g}_0(\gamma)$$

where N is the "order" parameter and the γ's specify the frequency breakpoints. Note that N has a direct effect on the effective bandwidth of the filter $\hat{g}$ such that larger values of N yield smaller bandwidths. Through the order parameter the user can create filters with better frequency localization without sacrificing too much the time resolution. The γ's can be alternatively translated to equivalent analyzing wavelet parameters of (i) bandwidth, (ii) order and (iii) center frequency, while filter bank parameters are specified as (i) number of filters, and (ii) dilation constant a_0.

5 CONCLUSION

We have presented the design and functionality of an interactive tool, the Wavelet Signal Processing Workstation (WSPW), for signal processing with non-orthogonal wavelet transformations. We have developed numerical computational methods to deal with the problems associated with multi-rate signal processing and described the implementation of the "continuous" versions of the Fourier and wavelet transforms using these numerical methods. Representing signals using a non-orthogonal family



analytic ☐
order 6
BW 10
CF 25
# filters 24
a0 1.06

Figure 8: Parametric bandlimited wavelet interface showing the analyzing wavelet used for the signal "opacket"

of wavelets allows the representation designer a great amount of freedom and emphasizes the need for optimization tools. We illustrated one such tool for the parametric specification of bandlimited wavelet filter banks with a numerical example.

6 ACKNOWLEDGMENTS

This work was performed for the Naval Research Laboratory (Code 5740).