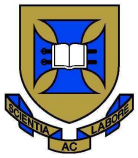
TFSA 6.2

Time-Frequency Signal Analysis Toolbox

Developed by Professor Boualem Boashash
Signal Processing Research Concentration, UQ Centre for Clinical Research,
The University of Queensland,
Royal Brisbane and Women's Hospital,
Herston, QLD 4029,
Australia

May, 2013



THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Acknowledgments

Developed within the Signal Processing Research Laboratory with the assistance of Peter Boles, Andrew Reilly, Gordon Frazer, Ben Hatton, Firas Zureiqat, Branko Ristic, Mark Roessgen, Geoff Roberts, Jonathon Ralston, D. Robert Iskander, Braham Barkat, Victor Sucic, Nick Ryan, Mark Keir and John O' Toole.

Contents

Licence agreement

Support policy

It is the intent of the Signal Processing Research Laboratory to continue to update TFSA to reflect new ideas and algorithms, and to correct bugs which may be discovered. Bug report are welcome, and should contain sufficient information to reliably reproduce the aberrant behaviour. Our address for matters concerning the TFSA package is:

TFSA
c/o Prof. B. Boashash
Signal Processing Research Concentration, UQ Centre for Clinical Research,
The University of Queensland,
Building 71/918,
Royal Brisbane and Women's Hospital,
Herston, QLD 4029,
Australia .

For fast technical support, contact Dr. J. O' Toole at:
`j.otoole@ieee.org`.

Part I

TFSA 6.2: User's Guide

Chapter 1

Introduction

1.1 Welcome

Welcome to the TFSA 6.2 toolbox. TFSA 6.2 is a time-frequency signal analysis toolbox which provides the user with a variety of tools for analysing non-stationary signals. The "TFSA 6.2: User's Guide" is designed to give an overview of the features and capabilities of TFSA 6.2 for new users as well as provide sufficient information for its use in many applications. Further detailed technical information regarding the functionality of the software is given in Part 5.6, where the "TFSA 6.2: Reference Guide" is found. If you encounter any problems running this version of TFSA on your system, refer to Chapter 3 "Getting Started" where information regarding the correct installation procedure for TFSA 6.2 is detailed. TFSA version 6.2 has been developed for MATLAB version 6.1 upwards.

1.2 New features of TFSA 6.2

If you have used earlier versions of TFSA, you will find many improvements in this version.

- A new graphic user interface (GUI) has been designed to minimise user effort and typing. Based on the powerful MATLAB[1] graphical tools, this new interface facilitates ease of use of the TFSA 6.2 tools and utilities.
- New time-frequency signal analysis tools have been added. A full list of the tools implemented in TFSA 6.2 is given in chapter 2 "TFSA 6.2: An Overview".
- All code in the TFSA toolbox has been fully optimised to operate under a MATLAB platform.
- An on-line HTML help system is included in this version for user convenience. Help buttons concerning most tools and utilities have been included throughout the package.

[1] MATLAB is a trademark of MathWorks Inc.

Chapter 2

TFSA 6.2: An Overview

TFSA 6.2 is a time-frequency signal analysis toolbox which provides the user with a variety of tools for analysing non-stationary signals. Although the original TFSA was written for use on PC's with MS-DOS, the latest version, TFSA 6.2, has been written to operate on both Linux and MS-Windows operating systems. An important feature of TFSA 6.2 is that it has been implemented by integrating efficient C code with MATLAB script language.

TFSA 6.2 is composed of a major C computational core and a MATLAB script interface:

1. **C Functions**
   The main analysis routines have been coded in C for flexibility, portability and performance. Note that the graphical display functions are only provided in MATLAB, because of its platform-independent interface.

2. MATLAB **functions**
   The MATLAB MEX file interface facility is utilised to call the main C code routines from within MATLAB. This is incorporated with existing MATLAB M files and a graphical user interface (GUI) to give a powerful, user friendly package. Furthermore, the TFSA 6.2 MATLAB interface has been carefully designed to ensure maximum productivity and seamless integration with existing MATLAB functionality. The MATLAB TFSA 6.2 code is fully portable, and can be run on any computer with MATLAB version 6.1 upwards.

The current release of TFSA (TFSA 6.2) supports the following functions:

1. Generation of test signals and noise:
   - Linear FM
   - Quadratic FM
   - Cubic FM
   - Stepped FM
   - Sinusoidal FM
   - Hyperbolic FM
   - Gaussian and uniform noise
2. Generation of time-frequency distributions:

- Wigner-Ville distribution
- Smoothed Wigner-Ville distribution
- Spectrogram
- Rihaczek-Margenau-Hill distribution
- Windowed-Rihaczek-Margenau-Hill distribution
- Choi-Williams distribution
- B distribution
- Modified-B Distributions
- Extended Modified-B Distributions
- Compact Support Kernel based Distributions
- Extended Compact Support Kernel based Distributions
- Born-Jordan distribution
- Zhao-Atlas-Marks distribution
- Cross Wigner-Ville distribution
- Polynomial Wigner-Ville distribution (order 6 kernel)
- Polynomial Wigner-Ville distribution (order 4 kernel)
- Ambiguity Function

3. Generation of time-scale signal representations:
   - Daubechies 4, 12 and 20 coefficient wavelet transforms and scalogram
4. Instantaneous frequency estimation algorithms:
   - Finite phase difference:
     - First order (FFD)
     - Second order (CFD)
     - Fourth order
     - Sixth order
   - Weighted phase difference
   - Zero crossing
   - Adaptive LMS
   - Adaptive RLS
   - Least squares polynomial coefficients
   - Peak of the spectrogram
   - Peak of the Wigner-Ville distribution
   - Peak of the polynomial Wigner-Ville distribution
5. Signal Synthesis from the following time-frequency distributions:
   - Short-Time Fourier Transform
   - Spectrogram

- Wigner-Ville distribution

6. Other DSP tools:
   - Analytic signal calculation
   - Power spectrum
   - Demo Signals (signal1, whale1, bat1, eeg1)
7. Data visualisation routines

2.1 References

Many of the algorithms used in TFSA 6.2 are fully described in the book "Time-Frequency Signal Analysis and Processing: A Comprehensive Reference", B. Boashash (Ed), Elsevier 2003. For additional information see the book chapter "Time-Frequency Signal Analysis", by B. Boashash in *Advances in Spectral Estimation and Array Processing*, editor S. Haykin, and the papers "Estimating and Interpreting the Instantaneous Frequency of a Signal – Part 1: Fundamentals and Part 2: Algorithms and Applications", in *Proceedings of the IEEE*, No. 4, Vol. 80, April 1992.

2.2 System requirements

A Linux or MS-Windows based computer system is required, running MATLAB version 6.1 or higher. Licencing information for MATLAB is available from The MathWorks, Inc.

Chapter 3

Getting Started

TFSA 6.2 has been written to run on a platform supporting MATLAB. To run TFSA 6.2 you need to have MATLAB installed. In order to run the TFSA 6.2 GUI, a minimum system requirement of 32 megabytes RAM on PCs is needed to avoid memory problems.

3.1 Installation

The distribution for Linux and MS-Windows is distributed pre-compiled. To install the distribution extract the zipped file and follow the instructions in the `README` text file.

3.2 Running TFSA 6.2

To run TFSA 6.2 on MS-Windows, all you need to do is click the MATLAB icon on the screen and then type **tfsa6** at the MATLAB prompt.

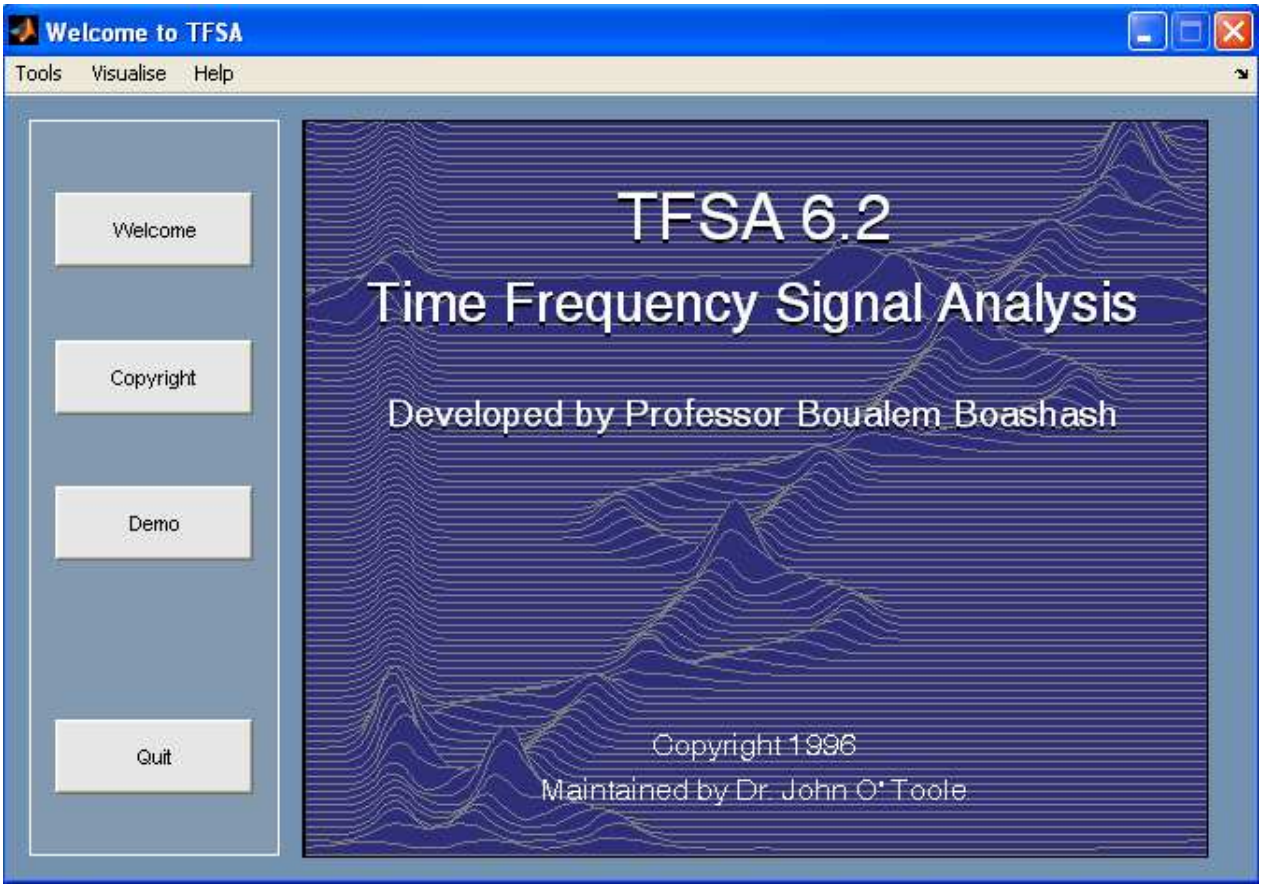
If you are running TFSA 6.2 on UNIX workstations, invoke MATLAB from the shell by typing **matlab** <**ENTER**>[1], wait for the MATLAB prompt and then type **tfsa6**.

If you have any problems with starting TFSA 6.2 on your system, refer back to section 3.1 of this guide for information regarding the correct setup and installation procedure.

[1] UNIX is a case sensitive system.

3.3 Welcome screen

After invoking TFSA 6.2, the welcome screen will appear to the user (see figure below).



It contains four buttons. These are:

- **Welcome**: Pressing this button will give general information regarding TFSA 6.2.
- **Copyright**: The licence agreement screen appears by clicking this button.
- **Demo**: Pressing this button gives a demonstration of TFSA 6.2.
- **Quit**: Pressing this button will abort the TFSA session.

3.3.1 Menu bar

The TFSA 6.2 menu bar is composed of three major menu items: **Tools**, **Visualise**, and **TFSA Help**.

Any menu item can be opened by placing the pointer on the corresponding item and then clicking the left hand side button of the mouse.

3.3.2 Accessing menus sub-items

You can access any menu sub-item by clicking with the left hand-side button of the mouse on the corresponding menu item and whilst holding this button down, dragging the mouse across the sub-menu bar until encountering the required sub-item, then releasing the mouse button. Another way of accessing the menu sub-items is by clicking once on each menu item or sub-item to enable the required action.

3.4 Data accesses

All references to "signal" or "array" names refer to variables in the MATLAB workspace. Values are read from and written to this space, and can be accessed normally from the MATLAB command line. Thus two signals can be added together using the following procedure:

1. Generate **time1** from the GUI **Signal Generation** pop-up.
2. Generate **time2** from the GUI **Signal Generation** pop-up.
3. In the MATLAB command-line window, type

```
time3 = time1 + time2
```

4. **time3**, the sum of the two generated signals, can now be accessed from the GUI.

Important: Since all results are written to the MATLAB workspace, no results are saved to disk unless specifically saved using the `save` command in the MATLAB command-line window. For example, to save all the variables you have been working with in a session, enter the command `save mydata`. All variables will be saved to the file `mydata.mat`. These variables can be restored at a later date with the command `load mydata`. Variables can also be saved in ASCII format; refer to your MATLAB documentation for more detail.

3.5 Compatibility with other MATLAB functions

Remember that the GUI is just an interface to the TFSA functions[2]. All normal MATLAB features remain in place. For example, titles, labels, figure properties and variables may all be changed using the MATLAB command-line interface. Figures can be printed using the MATLAB `print` command, as normal. To change the "current" figure, just click the left mouse button inside the figure window of interest.

3.6 MATLAB command line interface

TFSA 6.2 functions can be accessed from MATLAB in the same manner as the standard MATLAB functions. For a list of functions in TFSA 6.2 type

[2]Note that the following functions are not available from within the GUI: `quadknl, psde.`

```
help tfsa6
```

Help on any of the individual functions can be obtained by typing

```
help <functionname>
```

Further help may be obtained in some cases by typing

```
type <functionname>
```

Detailed information on each function can be found in chapter 6.

Chapter 4

TFSA 6.2 Basics

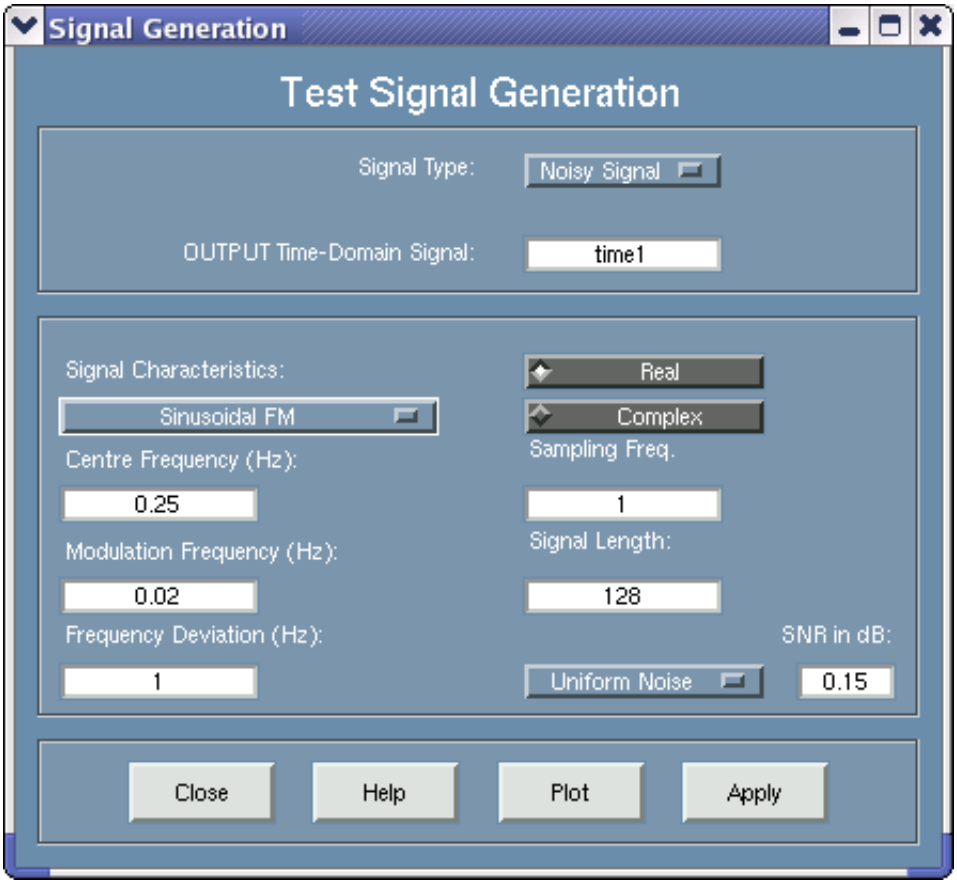
4.1 Running the demo

A good way to gain a feel for the tools and utilities available in TFSA 6.2 is to run the demo. This can be done by:

- Placing the pointer on the **Demo** button contained within the TFSA 6.2 welcome screen and clicking the mouse.
- Typing `tfsademo('initialise')` in the MATLAB command-line window.

4.2 Generating signals

You can generate test signals using the signal generation utility which can be accessed from the **Tools** menu item in the main menu bar. The procedure is as follows:



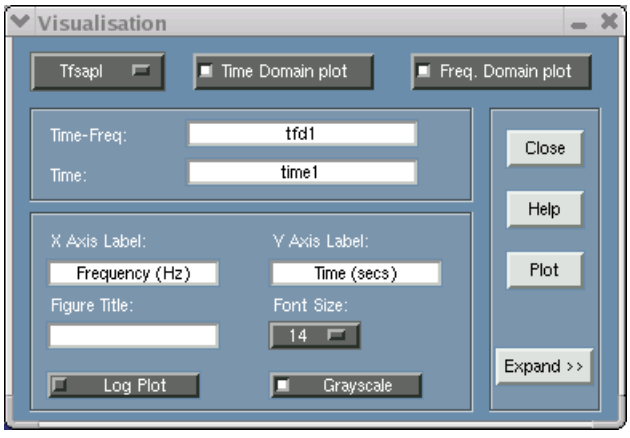
- Place the pointer on the **Tools** menu item and click the mouse.
- Move the pointer to the **Signal Generation** sub-item and click the mouse button. The signal generation screen should then appear (see figure above).
- Observe at the top of the screen the signal type list boxes presents the different types of signal generation options. The first option **Signal Only** sets the necessary fields and inputs for generating a non-stationary signal without noise. Similarly the second and third option, **Noise Only** and **Noisy Signal** enables the necessary fields and inputs for generating for generating uniform or Gaussian distributed white noise alone, and a signal plus noise. The fourth option **Analytic Gen.** generates the analytic associative of the given real signal. The fifth option **Demo Signal** presents the user with a selection of demonstration signal to use.
- The procedure for generating signals using the first three generation options is similar in all cases. Therefore only the third case (noisy signal) will be considered. Consider a real quadratic FM signal of data record length 1024, which has a minimum frequency of 0.15 Hz and maximum frequency of 0.4 Hz. The signal has a sample frequency of 1 Hz and is embedded in a Gaussian noise such that the SNR level is 25 dB. The steps for generating such a signal are as follows:
  - Select the **Noisy Signal** option in **Signal Type** list in the Signal Generation screen.
  - Choose the **Quadratic FM** item from the corresponding pop-up.

- Choose the **Gaussian noise** item from the noise pop-up using a similar procedure.
- Modify the **Start Frequency** and **End Frequency** fields to suit the values given for the minimum and maximum frequency respectively. These fields can be modified by placing the pointer on the respective field and then clicking the mouse button. A vertical cursor flashing inside the field indicates edit mode. Use the arrow keys to move the cursor both left and right, and the **DEL** key to erase any characters which are not needed. After the **Start Frequency** field is edited, move the pointer to the next field and repeat the editing procedure.
- Modify the **SNR**, **Sampling Frequency** and the **Signal Length** fields where appropriate using the above procedure.
- Ensure the generated signal is real (the **real/complex** radio buttons).
- Choose a name for the variable containing the signal data by editing the **Output Signal** field e.g. time1.
- Now generate the signal by clicking the **Apply** button. After the mouse has been clicked, the pointer will become a watch (indicating that computations are being performed) and then change back to a pointer again (indicating that the computations are completed).
- The generated signal is currently saved within the MATLAB workspace under the name found in the **Output Signal** field. This file can be saved to disk by using the MATLAB **SAVE** command.
- To exit the current screen, press the **Close** button.

4.3 Data visualisation

To plot the signal you have just generated:

- Click on the **Plot** button on the **Signal Generation** screen (or click on the **Visualise** menu item and select the **Plotting** sub-item).

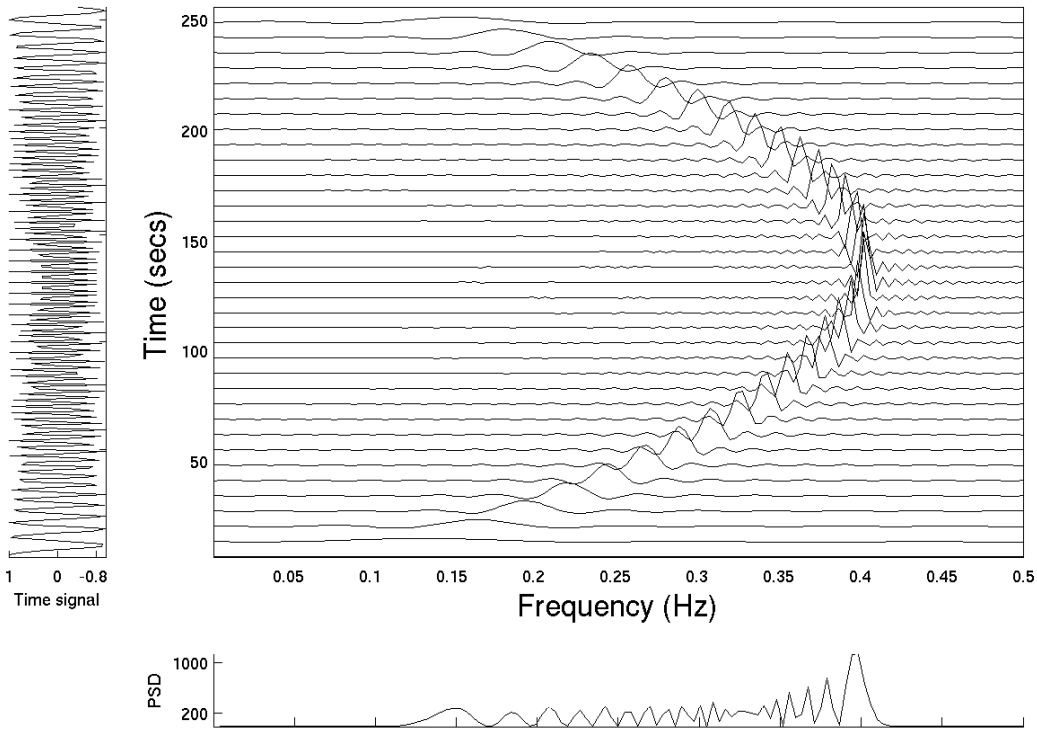


- Ensure that the **Input Data** field contains the appropriate data name.
- Various attributes of the plot can be set from the **Visualise** screen, click on button **Expand** >> to see the entire list.

4.4 Bilinear analysis

In this section the use of time-frequency distributions (TFDs) for analysing signals is described. This tool is invoked by first selecting the **Bilinear Analysis** menu sub-item, and selecting the desired TFD type. The Wigner-Ville distribution (WVD) will be used in this example. The WVD is calculated as follows:

- Select the **Bilinear TF Analysis** main menu sub-item.
- Select the **Wigner-Ville distribution** item from the **Distribution** pop-up.
- Edit both the **INPUT Time Domain Signal** and **Output time-frequency distribution** fields appropriately. If the signal generated above is to be analysed, then enter the string **time1** into the **Input Time Domain Signal** field. A suggested output name for this case might be **output1**.
- The **Time resolution** field should be set to 1, i.e. every time slice is computed. However computational time can be greatly increased by scaling this value by a positive integer.
- Change the **Lag window length** to 127 data points, and the **FFT length** to 128 data points (N.B., The data window length should be odd.).
- Click the **Apply** button to commence calculations.
- Finally use the **Visualisation** utility to view the resulting TFD by click on the **Plot** button. Select the **Tfsapl** option and then click on **OK** to view the plot (see the figure below). The user is advised to familiarise themselves with the different types of plots which are available, by selecting from the various alternatives.



The previous examples explain briefly how to use some TFSA 6.2 utilities. Further details concerning these and other TFSA 6.2 utilities can be found by consulting Part 5.6 TFSA 6.2:Reference Guide.

Part II

TFSA 6.2: Tutorial

Chapter 5

TFSA Tutorial

The following four part tutorial takes a step-by-step approach to the use and understanding of time-frequency signal analysis. It also has an adjunct purpose in that it familiarises the user with TFSA 6.2 for MATLAB. The tasks presented below illustrate various aspects of time-frequency analysis, which it is hoped, will also develop the user's diagnostic abilities, since the approach taken is from a practical viewpoint.

Before commencing this tutorial it is recommended that users first familiarise themselves with **Part 1 - TFSA 6.2: Users Guide**. Described in this part is the GUI, which contains the pop-up menus and interface fields which control the various functions/analysis tools available. The GUI parameters are to be varied when undertaking some of the tasks outlined below, and observance made of their resulting effect. Such hands-on work is the best and easiest way to understand and gain experience with the package.

5.1 Tutorial 1

In this tutorial the student is introduced to non-stationary signals, time-frequency distributions (TFDs) and the quantity known as the instantaneous frequency (IF).

5.1.1 Non-stationary signals and time-frequency distributions (TFDs)

Time-frequency distributions (TFDs) are useful for displaying the time-frequency content of a signal. In order to better understand TFDs the following steps should be completed, through the **TFSA Main Menu**:

1. Enter the **Signal Generation** sub-item, and generate a linear FM signal, of arbitrary stop and start frequencies. The signal generation procedure described in the **User's Guide** should be consulted if difficulties arise in doing this. This type of signal may be easily explained through an aural example – imagine singing a note and steadily increasing the pitch.

2. Analyse the signal using a Wigner-Ville distribution (WVD) and a spectrogram (by accessing the **Bilinear TF Analysis** menu sub-item). Display the two-dimensional function and observe the distribution of signal energy in time and frequency. A number of display formats should be investigated, however the **Tfsapl** format is strongly recommended. The advantages of such a display for a signal that has a time-varying spectra should be evident.

3. Vary the analysis window length (**Lag window length**) for the TFDs, and examine what effect this has on the time and frequency resolution of the resulting distribution.

This procedure should be repeated for different signal types and then different TFDs. Many of these TFDs, for example the Choi-Williams Distribution and the B distribution have parameters which control the form of the final distribution. Consult the relevant sections of **part 3 - TFSA 6.2: Reference Guide** in order to better understand the effect that changes in these parameters has on the resulting distributions.

Then add noise to the signal and repeat the exercise. Observe the effect the noise has on the distributions as the signal-to-noise ratio (SNR) decreases. At a certain SNR many TFDs will fail to resolve the time-frequency content of the linear FM signal.

5.1.2 The instantaneous frequency (IF)

The IF is an important quantity that in many cases enables single parameter characterisation of non-stationary signals. The hands-on work from the previous section should have laid the foundations for understanding the IF. The simple signals considered there, and the time-frequency visualisation lucidly illustrate the intuitive nature of IF description. The variation of the frequency over the time evolution of the signal is essentially the IF – this is what one would have considered (possibly without realising), in establishing a "mental picture" of the signal.

The **IF Estimation** pop-up of the TFSA 6.2 package contains many different routines for estimating the IF of a signal. These routines should now be employed to estimate the IF of some test signals. In addition to this, noise (available in the **Signal Generation** module) should also be added to the signals to demonstrate the relative noise performance of the various estimators.

Firstly the **Signal Generation** pop-up should be accessed in order to generate the test signals. The linear FM, cubic FM and stepped FM are suggested for analysing first. Be sure to consider each type of signal by itself and without noise. Then select a few IF estimators (e.g. the peak of the WVD, weighted phase difference and adaptive LMS). Use these techniques to estimate the IF of the test signals. Observe that as more noise is added, the performance of these procedures varies markedly.

What happens if you add two linear FM signals together and estimate the IF of the result?

5.1.3 Whale data

A record of whale data is provided with the TFSA 6.2 package (this can be found under the **Signal Generation** menu item, by selecting the **Demo Signals** option in the **Signal Type** field. This data contains 7000 data points and was collected at a sample rate of 8 kHz and is called **whale1**. Use the TFSA 6.2 **Visualisation** routine or the MATLAB `plot` command in order to observe this data in the time domain. By viewing only short segments of say 512 data points at a time you may be better able to discern the temporal characteristics of the signal. Use the MATLAB command line to do this. As an example, to view the first 512 data points type **plot(whale1(1:512))**.

Observe that it is difficult to infer the time-frequency content from these shorter segments. Now calculate the power spectrum of the complete signal using the **psde** utility. Be sure to set the FFT length **fft_len** and segment length **seg_len** fields to appropriate values (e.g. set both to the data length in order to calculate the periodogram). Observe that although the spectral content of the signal is displayed, no information is given as to what time specific spectral components are present.

Analyse the signal using some of the TFDs available. Use a window length of about 127 data points, and a time resolution of 100 initially. Observe how the time-frequency content of the signal is clearly displayed. A single signal component whose IF changes with time is presented. Estimate this IF using some of the IF estimation tools available. Try the **RLS** (in Adaptive) and **Peak of the WVD** first. For the RLS be sure to try different forgetting factors (suggested range 0.5–0.9875). Why does the forgetting factor of 0.5 give such a noisy IF estimate?

5.2 Tutorial 2

In this tutorial the student will become more familiar with TFDs by using them to analyse a range of signals. The effect of cross-terms due to signal and noise will also be explored.

5.2.1 TFDs and cross-terms

By now, it should be clear that TFDs do not always yield the expected distribution of signal energy in the time-frequency plane. The example of the two linear FM signals should have dramatically illustrated this point. The problem arises due to the quadratic signal product that is formed when calculating any quadratic TFD. This ensures that there are extra terms created (cross-terms) between any two separate signal components. The position and form of these cross-terms varies depending on the type of TFD selected. One should now generate two sinusoids and explore a variety of TFDs to see how the cross-terms manifest themselves. Then employ three sinusoids and see how many cross-terms eventuate.

Next generate a linear FM signal and add noise (try Gaussian noise with 3dB, 0dB and -3dB signal-to-noise ratios). Analyse these signals first with the WVD and vary the data window length. Observe how the noise and cross-terms mask the signal auto-term. Next analyse the same signals using the spectrogram and CWD. For the spectrogram observe the effect that the window length has on the resolution of the IF component. For the CWD vary the smoothing parameter in order to observe how the cross-terms are reduced to give the IF component.

Two additional signals are provided with the TFSA 6.2 package. The first is a synthetic signal containing a variety of components. This signal is called **signal1**. The second signal was produced by a large brown bat[1] (Eptesicus fuscus). This signal was sampled at 142 kHz and is called **bat1**.

First analyse the synthetic signal by applying a few TFDs (i.e. spectrogram, WVD, and CWD). Vary the window length and other pertinent parameters in order to try reveal the time-frequency content of the signal. There should be four distinct IF components. These are: (i) a stationary tone at 0.05Hz, (ii) a linear FM component, (iii) an FM whose IF equals the summation of a linear and sinusoidal component, and (iv) an impulse at sample number 750.

Next add noise to this synthetic signal and repeat the procedure. Observe how the noise can hinder the interpretation of the time-frequency content of the signal. In particular component number (iii) becomes increasingly difficult to resolve as the SNR decreases. Observe also that the stationary tone is always much easy to find. Why is this so?

Finally, analyse **bat1** data using a selection of TFDs. How many signal components are present in this signal? What frequency law does each component have? Try estimating the IF of this signal using the peak of the WVD. Can you explain the result?

[1]The data was kindly provided by C. Condon, K. White and A. Fang from the University of Illinois.

5.3 Tutorial 3

In this tutorial polynomial TFDs will be used to analyse data and the effect of cross-terms further explored. The use of time-scale analysis for detecting transients in real data will also be considered.

5.3.1 Polynomial WVDs (PWVDs)

There is another cross-term effect that is not so obvious. It occurs for single component signals that possess non-linear frequency modulation. It is essentially the same effect as which occurs for two sinusoidal signals summed together, but the cross-terms occur due to intra-component interaction. The action of the quadratic signal product in quadratic TFDs in effect causes a demodulation, where linear FM signals are transformed into sinusoids, and mapped into the time-frequency planes based on their IFs. If, however, the signal's FM is of higher-than-first, or not a polynomial at all, demodulation is not complete and cross-terms exist, as per the examples undertaken in the previous section. This may now be confirmed by generating a quadratic or hyperbolic signal and calculating the WVD. Other TFDs should also be examined.

Cross-terms of this type are particularly troublesome because they distort the fundamental information (namely the time-varying frequency behaviour). It is for this reason that polynomial WVDs have been developed. Essentially they multiply more (than two) signal terms together to demodulate more complicated signals. In the **Multilinear TF Analysis** sub-menu of TFSA 6.2, there are two PWVDs available - a 4th order and a 6th order. Thus the 4th order PWVD yields the appropriate IF law for a signal with frequency modulation up to 4th (polynomial) order with a similar behaviour for the 6th order PWVD. Linear, quadratic and cubic FM signals can therefore be appropriately reconstituted in the time-frequency plane using a 4th order PWVD (you should verify this now).

Further experimentation with varying the parameters should also be undertaken (see the manual for details), as well as investigation of the representation of non-polynomial FM signals like the hyperbolic and sinusoidal available in the **Signal Generation** pop-up window. There is an inevitable drawback to such higher order TFDs, however. As may be expected, they also produce cross-terms. Due to their multilinear nature, they produce many more cross-terms than traditional quadratic TFDs. The signals generated previously to examine the cross-term phenomena may now be used to investigate this effect for PWVDs.

5.3.2 EEG data

Finally, some EEG data which contains transients in the form of spikes has been provided. This data has been sampled at a rate of 50 Hz and is called **eeg1**. Use the Zhao-Atlas Marks (ZAM) distribution and the CWD to analyse this data. Observe how the transients manifest themselves in the resulting distribution. Then use the **Scale Analysis** utility in order to calculate the wavelet transform of this signal. Set the **X Axis Label** to 'scale'. Observe how the transients manifest themselves in the time-scale representation.

5.4 Tutorial 4

In this tutorial the student will become familiar with the discrete wavelet transform and its corresponding time-scale energy distribution known as the scalogram (the proportional bandwidth counterpart of

the spectrogram). The discrete wavelet transform is a linear operation that transforms the input time domain signal into the wavelet domain. The basis functions of the wavelet domain are called the wavelets. An important characteristic of wavelets is that, unlike sines and cosines (i.e. the basis functions of the Fourier transform) the individual wavelets are *localised* in both time and frequency. The local frequency of wavelets, however, is not linked to the frequency modulation, but to the concept of *scale*. Time-Scale analysis using TFSA 6.2 is restricted to one particular class of complete orthonormal wavelets introduced by I. Daubechies[2].

5.4.1 Wavelets

All basis functions of the wavelet domain represent the *scaled* and *translated* versions of the basic (or mother) wavelet. Generate (using MATLAB) unit sample sequences $\delta[n-6]$, $\delta[n-10]$ and $\delta[n-58]$ each of length 1024. Then apply the inverse (Daubechies) wavelet transform D4 (filter with 4 taps) to these three unit sample sequences. What is the result? Repeat the same using D20 wavelet transform and observe that D20 wavelets are much smoother than D4 wavelets. Can you explain why?

5.4.2 Signal reconstruction

The wavelet transform can be used in signal compression. In order to illustrate this, generate using MATLAB a sequence of length 1024: $e^{-\alpha(n-100)}\cos 2\pi f_0 n \cdot u[n-100] + e^{-2\alpha(n-600)}\cos 4\pi f_0 \cdot u[n-600]$, where $\alpha = 0.05$ and $f_0 = 0.1$. Then apply the wavelet transform to this sequence (D4 filter with 4 taps). The majority of the wavelet coefficients will be negligible. Set the smallest (in terms of their absolute values) 512 wavelet coefficients to zero, and reconstruct the original signal. This gives a compression ratio of 2:1. Calculate the signal-to-noise ratio (SNR) of the reconstructed signal where $SNR = 10\log(P_s/P_e)$ where P_s is the power of the reconstructed signal and P_e is the power of the error between the reconstructed signal and the original signal. Repeat this for compression ratios of 3:1, 4:1, 5:1 and 6:1. Plot these results. What can you conclude? Repeat this experiment with the D20 wavelet (filter with 20 taps) and compare the result.

5.4.3 Scalogram

The wavelet transform takes N samples of the input signal and creates N wavelet coefficients. These N coefficients have to be properly arranged and squared in order to form the time-scale energy distribution known as the scalogram. First generate a sinusoid of 1024 data points and then set the middle 15 points of this signal to zero (this effectively produces a signal which has zero amplitude for a short while). Use the "scalogram" function of TFSA 6.2 to generate the time-scale representation. Experiment with different wavelets and compare this representation with that given by the spectrogram (vary the window length of the spectrogram). Next use the "scalogram" function to analyse the demo signal **test signal** ('signal1'). Try all three types of the wavelet transforms (D4, D12 and D20), and compare the results.

[2]M. Vetterli and C. Herley, Wavelets and Filter Banks: Theory and Design, *IEEE Transactions on Signal Processing*, Vol. 40 No. 9, September 1992.

5.5 Summary

These four tutorials have provided a unique introduction to time-frequency signal analysis and the TFSA 6.2 for MATLAB package. They have been designed in order for the new user to experience this type of analysis in a generally unrestrictive way, where some guidelines pointing out significant features and properties were given. This allows the user to personally discover (often through trial and error) the advantages, intuitive notions and drawbacks of time-frequency signal analysis. Such hands-on knowledge and experience is the best place to start one's journey into this realm of non-stationary signal analysis.

5.6 Answers to questions

- Q. What happens if you add two linear FM signals together and estimate the IF of the result?
  A. The signal is no longer mono-component, and depending on the type of IF estimator used the results will be quite different. The concept of the IF was developed for the mono-component signal case, making interpretation of the results of the IF estimation algorithms applied to the multicomponent case quite difficult.

- Q. Why does the forgetting factor of 0.5 give such a noisy IF estimate?
  A. The lower the forgetting factor the less importance is placed on past data values (i.e., less memory is utilised). Therefore the algorithm is better able to track fast changing components, but will however tend to be affected by noise. Since the amplitude of the mono-component signal varies considerably, the SNR of the signal is often quite low. This causes the estimate to have a high variance.

- Q. How many signal components are present in this signal? What frequency law does each component have?
  A. There are three components each with a hyperbolic law.

- Q.Try estimating the IF of this signal using the peak of the WVD. Can you explain the result?
  A. At any time the signal component with the largest amplitude will be selected as the IF estimate when using the peak of the WVD.

- Q. What is the result of applying the wavelet transform to the three unit sequences?
  A. Three of the 1024 possible wavelet functions in the complete orthonormal basis are produced.

- Q. Explain why the D20 wavelets are much smoother than the D4 wavelets?
  A. For a higher number of wavelet filter coefficients it is necessary that a higher order of moments vanish (in order to formulate enough equations). For the case of p vanishing moments this is known as the approximation condition of order p. Hence the D20 wavelets will have higher-order continuous derivatives.

- Q. Plot these results. What can you conclude?
  A. The signal-to-noise ratio decreases as the compression ratio increases. Since the D20 is a smoother wavelet than the D4 it will have a higher numerical accuracy and so will give better compression performance.

Part III

TFSA 6.2: Reference Guide

Chapter 6

Technical Reference

6.1 TFSA 6.2 MATLAB Functions

Internal Function	Description
flatwf	Used in tfsapl to create TFSA plot
getWin	Convert window ID to window string
goodfonts	Select appropriate font types and sizes
helpdata	Help information
oploy	Function to find coefficients for polynomial
tfsademo	TFSA 6.2 demo file
tfsahelp	Help frame
tfsamain	Main TFSA frame
tfsamenu	Handles the menus in the main frame
tfsaopen	Sets up main frame
tfsa_plot2d	TFSA vector plot frame
tfsa_wrn	Handles warnings displayed on command line
uif_base	Template frame called by other uif_* frames
uif_btfd	Callback to manage the bilinear TFD frame
uif_defs	Declared constants used in uif_* frames
uif_dirtfd	Direct method of implementation of some TFDs frame
uif_gsig	Callback to manage the test signal generation frame
uif_ife	Callback to manage the instantaneous frequency estimation frame
uif_mtfd	Callback to manage the multi-linear tfd frame
uif_plot	Callback to manage the plotting frame
uif_synth	Callback to manage the synthesis frame
uif_ts	Callback to manage the time-scale frame
uideflts	Default values for all uicontrols
unphase	Recovers phase of the analyic input signal

The internal TFSA 6.2 functions support the package and are not meant for use by the user and are listed here for reference only.

ambf

Purpose

Computes the ambiguity function of an input signal.

Synopsis

af = ambf(signal);

Parameters

tfrep	The computed time-frequency distribution (ambiguity function) of a signal. size(af) will return [a, b], where a is the next largest power of two above window_length, and b is floor(length(signal)/time_res) - 1.
signal	An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.

Description

The ambiguity function (AF) implemented in the TFSA package is the symmetric ambiguity function, also known as Sussman ambiguity function. The AF for the continuous case is defined as

$$A_z(\theta,\tau) = \int_{-\infty}^{\infty} z\left(t+\frac{\tau}{2}\right) z^*\left(t-\frac{\tau}{2}\right) e^{-j2\pi\theta t} dt.$$

For the implementation, we take the Fourier transform of the kernel $K_z(t,\tau) = z\left(t+\frac{\tau}{2}\right) z^*\left(t-\frac{\tau}{2}\right)$ for each lag τ, i.e.,

$$A_z(\theta,k) = \sum_{n=-M}^{M} z(n+k)z^*(n-k)e^{-j2\pi\theta n},$$

where the signal, $z(n)$, is defined for $n = [-M : M]$. The lag, k, is chosen so that the product of the shifted sequences $z(n+k)$ and $z^*(n-k)$ be non-zero. This results in $k = [-M : M]$.

Therefore, the algorithm is as follows. For the first value of the lag, k, we compute the kernel $K_z(n,k) = z(n+k)z^*(n-k)$ and then take the FFT of the sequence. We choose the second value of the lag, compute the kernel and take the FFT of the sequence. We repeat the procedure for all values of the lag. The result of each FFT is stored in a matrix (as a line column) for the particular value of the lag. The matrix gives the AF.

Figure 6.1 shows the image of the absolute value of the ambiguity function of a linear FM signal.

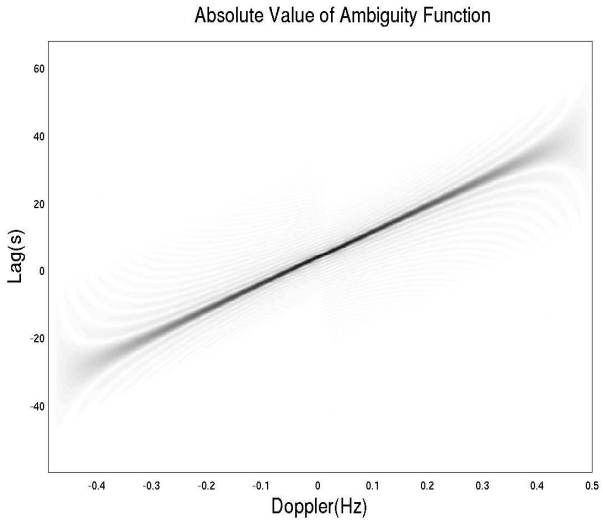


Figure 6.1: Absolute value of the ambiguity function of a linear FM signal, as generated by `ambf` and displayed using `image`.

analyt

Purpose

Compute analytic signal of a real input signal.

Synopsis

output = analyt(signal);

Parameters

output	Generated analytic signal (complex).
signal	Input real one dimensional signal.

Description

This function computes the analytic version of a real signal.

cmpt

Purpose

Generate Time-Frequency Distributions based on Compact Support Kernels.

Synopsis

tfd = cmpt(signal, kernel [, kernel options]);

Parameters

tfd	tfd is the computed time-frequency distribution. Size of the TFD will be [M, N], where M is the next largest power of two of signal length, and N is length of the signal.
signal	Input one dimensional signal to be analysed.
kernel	The determining kernel function. kernel is a string defining a predefined kernel. Predefined types: 'csk': Compact Support Kernel 'ecsk': Extended Compact Support Kernel
kernel options	Parameters to control shape and spread of kernel. 'csk' C: parameters C controls the shape of compact support kernel D: parameters D controls the spread of compact support kernel 'ecsk' C: parameters C controls the shape of extended compact support kernel D, E: parameters D, E controls the spread of extended compact support kernel

Description

This function computes Time-Frequency Distributions of any signal based on Compact Support Kernels.

gsig

Purpose

Generate various time and frequency-varying test signals

Synopsis

output = gsig(data_type1, f1, f2, num_samples, sig_type);
output = gsig(data_type2, cf, mf, num_samples, sig_type, fdev);
output = gsig(data_type3, f1, f2, num_samples, sig_type, ns);

Parameters

output	Generated signal.	
data_type1	One of:	
	'lin'	Linear FM
	'quad'	Quadratic FM
	'cubic'	Cubic FM
	'hyp'	Hyperbolic FM
	with	
	f1	Start frequency (normalised, where sampling frequency = 1).
	f2	End frequency (normalised, where sampling frequency = 1).
data_type2	'sin'	Sinusoidal FM
	with	
	cf	The central frequency (normalised, where sampling frequency = 1).
	mf	The modulation frequency (normalised, where sampling frequency =1).
	fdev	The frequency deviation.
data_type3	'step'	Stepped FM
	with	
	f1	The start frequency (normalised, where sampling frequency = 1).
	f2	The end frequency (normalised, where sampling frequency =1).
	ns	The number of steps.
sig_type	For real data set sig_type=1 otherwise the result is complex.	
num_samples	Length of signal to be produced.	

Description

This function generates various test signals including uniformly distributed white noise. The signals generated can then be analysed using tools available within TFSA 6.2, or saved to disk[1] for use at a later date.

Examples

Generate a 512 point stepped real FM signal, with 3 steps between 10 Hz and 40 Hz, where the sampling frequency is 200Hz:

```
signal = gsig( 'step', 0.05, 0.2, 512, 1, 3);
```

This signal is shown in Figure 6.2, and is used in several other examples in the manual.

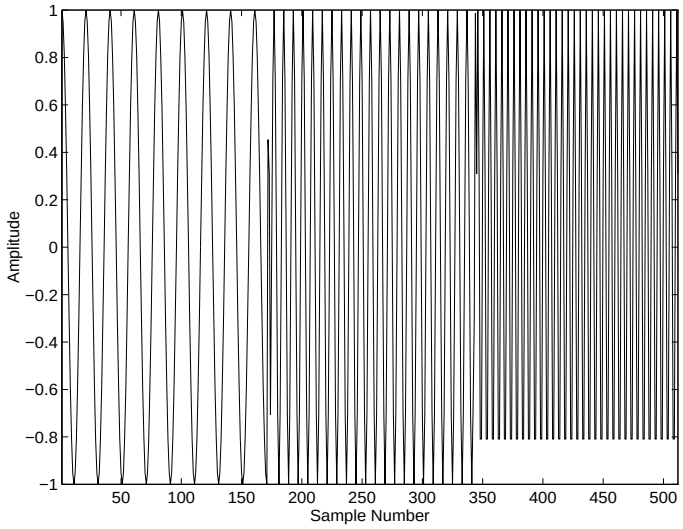


Figure 6.2: Stepped linear FM signal generated by gsig.

[1] Use the MATLAB `save` command to do this.

lms

Purpose

Estimate the instantaneous frequency of the input signal using the least mean square adaptive algorithm.

Synopsis

ife = lms(signal, mu);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal (real or analytic).
mu	Adaptation Constant.

Description

This function estimates the instantaneous frequency of an input signal using the least mean square adaptive algorithm. A one-tap transversal filter is used to achieve this.

Examples

Compute the instantaneous frequency estimate of the signal `time1` by using the `lms` function with an adaption constant of 0.8.

```
ife_lms = lms( time1, 0.8);
```

See Also

rls, pde, sfpe, wvpe, zce

pde

Purpose

Estimates the instantaneous frequency of the input signal using general phase difference (FFD, CFD, 4th, 6th orders) and weighted phase difference estimation.

Synopsis

ife = pde(signal, order, [,window_length]);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal (real or analytic).
order	Order of the finite phase difference estimator. Available estimator orders are: 1, 2, 4, 6.
window_length	Kay smoothing window length in the case of weighted phase difference estimator.

Description

This function estimates the instantaneous frequency of the input signal using either the general phase difference estimation approach or Kay's frequency estimator (weighted phase difference estimator). The order of the phase difference selected should reflect the signal phase law and the signal-to-noise ratio.

Examples

Compute the instantaneous frequency estimate of the signal `time1` by using the 4th order general phase difference estimator.

```
ife = pde( time1, 4);
```

Compute the instantaneous frequency estimate of the signal `time1` by using the Kay smoothing weighted phase difference estimate with a smoothing window length of 32 data points.

```
ife  = pde( signal, 2, 32);
```

See Also

lms, rls, sfpe, wvpe, zce

psde

Purpose

Estimate the power spectrum of a signal.

Synopsis

PSD = psde(signal, seg_len, fft_len, overlap, window_type);

Parameters

signal	The signal one dimensional signal which may be real or complex.
seg_len	The length of each segment. NB seg_len must be less than or equal to the fft_len.
fft_len	The length of the fft. If this is not radix two in will be shifted up to the next radix two number.
overlap	The size of the overlap between the segments.
window_type	The window type can by hamm, hann, bart or rect. The length of window is determined by the segment length.

Description

Estimates the power spectral density of input data stream. The data is divided into segments. The periodogram of each segment is calculated and the result is the average of the periodograms.

pwvd4

Purpose

Computes the polynomial Wigner-Ville distribution

Synopsis

tfrep = pwvd4(signal, lag_win_len, time_res [, fft_length]);

Parameters

tfrep	The computed time-frequency distribution. size(tfrep) will return [a, b], where a is the next largest power of two above lag_win_len, and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
lag_win_len	The length of the data window used for analysis..
time_res	The number of time samples to skip between successive slices of the analysis.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the polynomial Wigner-Ville distribution (fourth order kernel) of the input signal. An analytic signal generator is called if the input signal is real. The supplied length of the analysis window defines whether the "true" pwvd or a pseudo (windowed) pwvd is computed. This function is fully optimised for speed.

Examples

Compute the PWVD4 of a 1024 point signal using a time-resolution of 20:

```
tf = pwvd4( signal, 1023, 20);
```

pwvd6

Purpose

Computes the polynomial Wigner-Ville distribution

Synopsis

tfrep = pwvd6(signal, lag_win_len, time_res, interp [,fft_length]);

Parameters

tfrep	The computed time-frequency distribution. size(tfrep) will return [a, b], where a is the next largest power of two above lag_win_len, and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
lag_win_len	The length of the data window used for analysis.
time_res	The number of time samples to skip between successive slices of the analysis.
interp	Number of times to interpolate the input signal before computing the kernel. If this value is not a power of 2, it will be replaced by the radix 2 value above it.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the polynomial Wigner-Ville distribution (sixth order kernel) of the input signal. An analytic signal generator is called if the input signal is real. The supplied length of the analysis window defines whether the "true" pwvd or a pseudo (windowed) pwvd is computed. This function interpolates the signal in the time domain to get the required values of the signal at the fractional time lags specified by the sixth order kernel. This function is fully optimised for speed.

Examples

Compute the PWVD6 of a 1024 point quadratic FM signal a time-resolution of 10. The interpolation degree is chosen to be 8.

```
signal = gsig( 'quad', 0.1, 0.4, 1024, 1 );
tf = pwvd6( signal, 511, 20, 8, 1024 );
tfsapl( signal, tf, 'TimePlot', 'on', 'FreqPlot', 'on' );
```

The resulting PWVD6 is shown in figure 6.3.

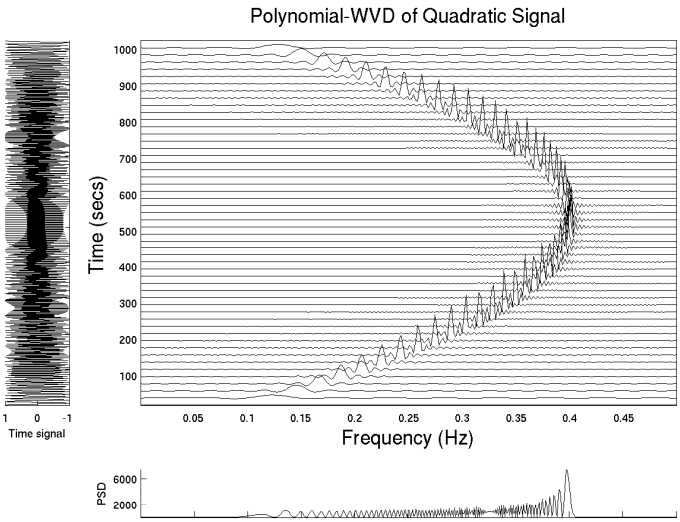


Figure 6.3: Sixth order kernel PWVD of a quadratic FM signal. The plot was generated using the `tfsapl` utility.

pwvpe

Purpose

Estimates the instantaneous frequency of the input signal by extracting the peaks of the sixth order kernel polynomial Wigner-Ville distribution.

Synopsis

ife = pwvpe(signal, lag_win_len, time_res, interp [,fft_length]);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
lag_win_len	The length of the data window used for analysis.
time_res	The number of time samples to skip between successive instantaneous frequency estimates.
interp	Number of times to interpolate the input signal before computing the kernel. If this value is not a power of 2, it will be replaced by the radix 2 value above it.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the polynomial Wigner-Ville distribution (sixth order kernel) of the input signal and then takes the peak of this distribution in order to form the instantaneous frequency estimate. An analytic signal generator is called if the input signal is real. The supplied length of the analysis window defines whether the "true" pwvd or a pseudo (windowed) pwvd is computed. This function interpolates the signal in the time domain to get the required values of the signal at the fractional time lags specified by the sixth order kernel. This function is fully optimised for speed.

See Also

pwvd6

quadknl

Purpose

Generate Quadratic Class Time-Lag Kernel Functions

Synopsis

kernel = quadknl(kernel_type, window_length, full_kernel [,kernel_options]);

Parameters

kernel — The generated time-frequency kernel, indexed as kernel(time, lag).

The kernel is arranged in memory such that the zero time, zero lag point is located at kernel(1,1), and the positive time-lag quadrant extends to kernel((window_length+1)/2, (window_length+1)/2). Negative values are indexed from window_length down to (window_length+1)/2 + 1.

kernel_type — The determining kernel function of type:

'wvd'	Wigner-Ville
'smoothed'	Smoothed Wigner-Ville
'specX'	Spectrogram estimate
'rm'	Rihaczek-Margenau-Hill
'cw'	Choi-Williams
'bjc'	Born-Jordan
'zam'	Zhao-Atlas-Marks
'b'	B-distribution
'mb'	Modified B-distribution

window_length — Size of the generated kernel. Kernel will be defined in both time and lag dimensions from -(window_length/2) to +(window_length/2). See above for storage map.

full_kernel — A boolean, indicating whether the full kernel is to be generated, or just the first quadrant (positive time and lag only). Passing 1 indicates the former case, while 0 indicates the latter. In the latter case, the returned size of the kernel will be (window_length+1)/2 in both dimensions.

kernel_options — Some kernels require extra parameters:

'smoothed'	kernel_option = length of smoothing window kernel_option2 = type of smoothing window, one of: 'rect', 'hann', 'hamm', 'bart'
'stft'	kernel_option = type of smoothing window, one of: 'rect', 'hann', 'hamm', 'bart'
'cw'	kernel_option = Smoothing parameter
'zam'	kernel_option = ZAM 'a' parameter

'b'	kernel_option = B-distribution smoothing parameter σ
'mb'	kernel_option = modified B-distribution parameter $alpha$

Description

This function allows the user to generate the stand-alone kernel function associated with a particular quadratic time-frequency distribution.

Examples

Read the value of a 255 point Choi-Williams kernel (smoothing value = 10) at time = 12, lag = -50:

```
k = quadknl( 'cw', 255, 1, 10);
val = k(1+12, (255+1)-50);
```

Note that the 1 added on to the 12 and 255 is necessary because MATLAB matrices begin indexing at 1, instead of zero. Hence k(1,:) is at time 0, k(2,:) at time 1, . . ., k(13,:) at time 12.

Alternatively, since the Choi-Williams kernel is symmetric, we can use:

```
k = quadknl( 'cw', 255, 0, 2);
val = k(13, 51);
```

Figure 6.4 shows the output of the command:

```
k = quadknl( 'cw', 63, 1, 10);
mesh(k);
```

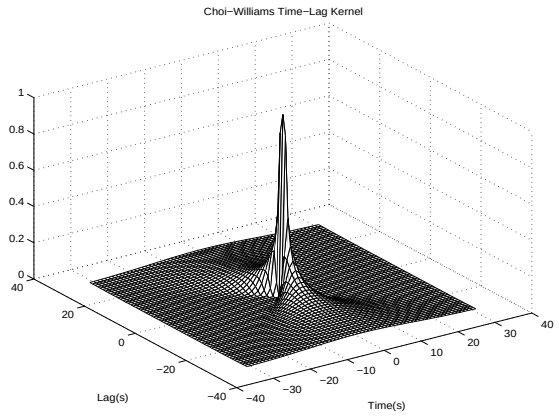


Figure 6.4: Time-lag Choi-Williams kernel, as generated by `quadknl` and displayed using `mesh`.

See Also

quadtfd

quadtfd

Purpose

Generates various quadratic time-frequency distributions.

Synopsis

tfrep = quadtfd(signal, lag_win_len, time_res, kernel [, kernel_options], [fft_length]);

Parameters

tfrep — The computed time-frequency distribution. size(tfrep) will return [a, b], where a is the next largest power of two above lag_win_len, and b is floor(length(signal)/time_res) - 1.

signal — Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.

lag_win_len — This is the lag window length and controls the size of the kernel used for analysis. The parameter lag_win_len must be odd. The kernel used will be defined from -(lag_win_len+1)/2 to +(lag_win_len+1)/2 in both time and lag dimensions, although the time dimension may be further modified by the smoothing window length for relevant distributions.

time_res — The number of time samples to skip between successive slices of the analysis.

kernel — The determining kernel function. kernel is a string defining a predefined kernel of type:

'wvd'	Wigner-Ville
'smoothed'	Smoothed Wigner-Ville
'specX'	Spectrogram estimate
'rm'	Rihaczek-Margenau-Hill
'cw'	Choi-Williams
'bjc'	Born-Jordan
'zam'	Zhao-Atlas-Marks
'b'	B-distribution
'mb'	Modified B-distribution
'emb'	Extended Modified B-distribution

kernel_options — Some kernels require extra parameters:

'smoothed'	kernel_option = length of smoothing window kernel_option2 = type of smoothing window, one of: 'rect', 'hann', 'hamm', 'bart'
'specx'	kernel_option = type of smoothing window, one of: 'rect', 'hann', 'hamm', 'bart'

'cw'	kernel_option = Smoothing parameter
'zam'	kernel_option = ZAM 'a' parameter
'b'	kernel_option = B-distribution smoothing parameter σ
'mb'	kernel_option = Modified B-distribution parameter α
'emb'	kernel_option = Extended Modified B-distribution parameters α and σ

fft_length — Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

This function generates various quadratic time-frequency distributions (these are listed under **Parameters**- kernel). The code has been optimised for computational efficiency. For example, the use of symmetry and realness for a particular distribution has been utilised where possible in order to reduce the number of computations.

Examples

Generate the smoothed Wigner-Ville distribution of signal `t1`, using an analysis (lag) window length of 127, a time resolution of 15 points and a α parameter value of 0.05. This is illustrated in Figure 6.5.

```
t1 = gsig( 'step',0.05, 0.45, 512, 1, 3 );
tf1 = quadtfd( t1, 127, 15, 'mb', 0.05, 512 );
tfsapl( t1, tf1, 'plotfn', 'wfall' );
```

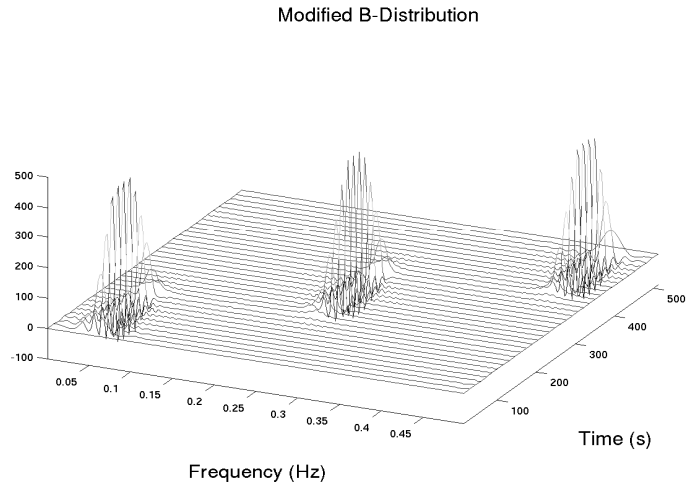


Figure 6.5: Waterfall plot of the Modified B-Distribution of a stepped FM signal.

See Also

quadknl

rihaczek

Purpose

Calculates the Rihaczek, Levin or windowed-Rihaczek/Levin time-frequency distribution.

Synopsis

tfrep = rihaczek(signal [,time_res] [,fft_length] [,rih_levin] [,window_length] [,window_type]);

Parameters

tfrep	The computed time-frequency distribution. size(tfrep) will return [a/2+1, b], where a is the next largest power of two above signal_length, and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
time_res	The number of time samples to skip between successive slices of the analysis.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the window_length, then the next highest power of two above window_length is used. If fft_length is not a power of two, the next highest power of two is used.
rih_levin	Option to specify whether to return the Rihaczek distribution or the Levin distribution.

0	Rihaczek (default)
1	Levin

window_length	If this parameter is specified then the windowed distribution is used. window_length must be odd.
window_type	One of 'rect', 'hann', 'hamm', 'bart' .

Description

Computes the Rihaczek or windowed-Rihaczek time-frequency distribution. Can also return the Levin distribution which is simply the real part of the Rihaczek distriubtion. The windowed-Rihaczek distribution uses the Short-Time Fourier Transform (by calling the `spec` function) inplace of the Fourier Transform of the signal. If the input signal is real it is replaced by it's analytic associate.

Examples

Compute the Rihaczek distribution of a 128 point signal `time1` by using the `rihaczek` function (time-resolution is set to 1).

```
tfrep = rihaczek( time1, 1, 128 );
```

Compute the windowed-Levin distribution of a 128 point signal `time1` with a 21 point Hamming window by using the `rihaczek` function (time-resolution is set to 2).

```
tfrep = rihaczek( time1, 2, 128, 1, 21, 'hamm'  );
```

See Also

spec

rls

Purpose

Estimate the instantaneous frequency of the input signal using the recursive least square adaptive algorithm.

Synopsis

ife = rls(signal, alpha);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal (real or analytic).
alpha	Forgetting factor.

Description

This function estimates the instantaneous frequency of an input signal using the recursive least square adaptive algorithm. A one-tap transversal filter is used to achieve this.

Examples

Compute the instantaneous frequency estimate of the signal `time1` by using the `rls` function with a forgetting factor of 0.8.

```
ife_rls = rls( time1, 0.8);
```

See Also

lms, pde, sfpe, wvpe, zce

sfpe

Purpose

Estimates the instantaneous frequency of the input signal by extracting the peaks of the spectrogram.

Synopsis

ife = sfpe(signal, window_length, time_res [,fft_length]);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
window_length	The length of the data window used for analysis.
time_res	The number of time samples to skip between successive instantaneous frequency estimates.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the window_length, then the next highest power of two above window_length is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the spectrogram of the input signal and then takes the peak of this distribution in order to form the instantaneous frequency estimate. An analytic signal generator is called if the input signal is real. This function is fully optimised for speed.

See Also

quadtfd, lms, rls, pde, wvpe, zce

spec

Purpose

Computes the Spectrogram or Short-Time Fourier Transform time-frequency distribution.

Synopsis

tfd = spec(signal, time_res, window_length, window_type, [,fft_length] [,stft_or_spec]);

Parameters

tfd	The computed time-frequency distribution. size(tfrep) will return [a/2+1, b], where a is the next largest power of two above signal_length, and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
time_res	The number of time samples to skip between successive slices of the analysis.
window_length	Length of choosen window.
window_type	One of 'rect', 'hann', 'hamm', 'bart' .
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the window_length, then the next highest power of two above window_length is used. If fft_length is not a power of two, the next highest power of two is used. To avoid periodic effects for a non-periodic signal, the fft_length should be at least twice the signal length.
stft_or_spec	Returns either Short Time Fourier Transform (STFT) or Spectrogram by specifying: 0 Spectrogram (default) 1 STFT

Description

Computes the Spectrogram or Short-Time Fourier Transform time-frequency distribution. If the input signal is real it is replaced by it's analytic associate. The spectrogram can also be computed from the `quadtfd` function, however the `spec` function is fully optimised for speed and is more computationally efficient.

Examples

Compute the STFT of a 1024 point signal `time1` with a 17 point rectangular window by using the `spec` function (time-resolution is set to 20).

```
tfd = spec( time1, 20, 17, 'rect', 1, 2048 );
```

See Also

quadtfd

synthesize

Purpose

Computes a signal synthesized from a given time-frequency distribution. Can be used for STFT, Spectrogram and WVD.

Synopsis

signal = synthesize(tfd, analysis_type, window_length [,analysis_params] [,original_signal]);

Parameters

signal — Synthesized (complex) signal from given `tfd`.

tfd — Matrix containing the time-frequency distribution for a given signal. Can be either a STFT, Spectrogram or Wigner-Ville distribution.

analysis_type — Specifies which type of signal synthesis will be applied to the given `tfd` matrix. The following analysis types are valid:

If the `tfd` is a Short-Time Fourier Transform Distribution:

'idft'	Inverse Discrete Fourier Transform method
'ola'	OverLap-Add method
'mstft'	Modified Short-Time Fourier Transform method

If the `tfd` is a Spectrogram Distribution:

'mspec'	Modified Spectrogram method

If the `tfd` is a Wigner Ville Distribution:

'wvd'	Wigner Ville Distribution method

analysis_params — Some of the anaylsis types require parameters:

'idft'	analysis_params1 = window type
'ola'	analysis_params1 = window type
'mstft'	analysis_params1 = window type
'mspec'	analysis_params1 = window type analysis_params2 = tolerance value for iteration routine

original_signal — Original signal can be supplied to correct the phase of the synthesized signal.

Description

Computes a time-domain signal from a give time-frequency distribution.

Examples

To synthesize a signal from a time-frequency distribution, a 128 $\times$ 128 point WVD `tfd_wvd` is supplied, which is computed as follows:

```
tfd_wvd = quadtfd( signal1, 127, 1, 'wvd', 128 );
```

The data window length is specified at 127 points and the original signal `signal1` is supplied to correct the phase:

```
synth_signal = synthesize( tfd_wvd, 'wvd', 127, signal1 );
```

To synthesize a signal from a given STFT, which uses a 17 point Hamming window and is computed as follows:

```
tfd_stft = quadtfd( signal1, 21, 'hamm', 1, 256 );
```

Using the overlap-add method, the distribution and window information must be supplied:

```
synth_signal = synthesize( tfd_stft, 'ola', 21, 'hamm' );
```

tfsa6

Purpose

Entry routine for the Graphical User Interface (GUI) to TFSA 6.2

Synopsis

tfsa6

Parameters

none

Description

Use the mouse and keyboard to interact with the GUI. The GUI simply provides an alternative to typing TFSA 6.2 commands on the MATLAB command line. Except as noted below, for each operation in the GUI there exists an underlying TFSA 6.2 function which is described in this reference section. Refer to the relevant function description for information on parameters.

Figure 6.6 shows the main menu of the GUI.

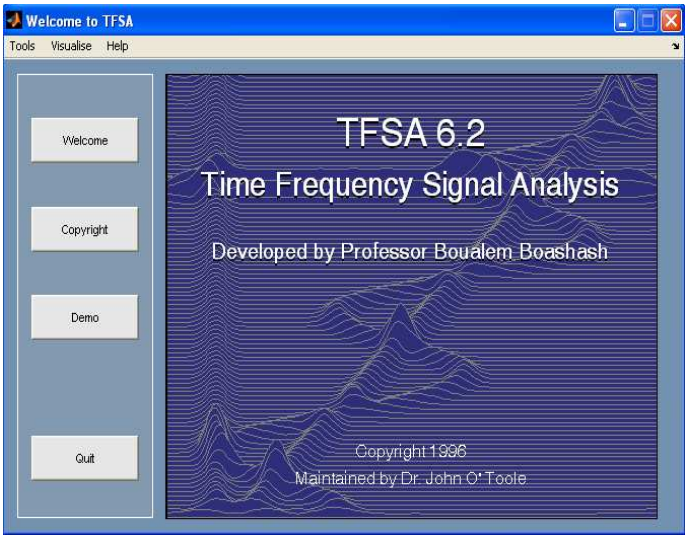


Figure 6.6: Main Menu of the TFSA Graphical User Interface.

tfsapl

Purpose

Time-frequency plotting routine.

Synopsis

p = tfsapl(signal, TFD [, Properties])

Parameters

signal	Time series signal.
TFD	Time-frequency matrix.

Returns

The return value 'p' is a vector containing four graphics axis handles,
p(1) is the time signal
p(2) is the power spectrum
p(3) is the time-frequency plot
p(4) is the title information at the top

These can be used to alter the appearence of the plot after it is complete, with `set( p(i), 'Parameter', value )` commands.

Properities

List of optional properties to set the apperance of the plot. Should be specified as `..., 'ParameterName', 'Value', ...`. They are not case sensitive. E.g.

```
tfsapl( time, TFD, 'TimePlot', 'on', 'FreqPlot', 'on', 'plotfn', 'surf',
'Title', 'Time Frequency Distribution', 'FontSize', 12 )
```

TimePlot	{ 'on' | 'off' } (default 'off') Plot of time domain signal appears along y-axis
FreqPlot	{'on' | 'off'} (default 'off') Plot of spectrum appears along x-axis
SampleFreq	{ numeric value } (default 1) Set sampling frequency.
Res	{ numeric value } (default 1) Time resolution value.
PlotFn	{ 'surf', 'contour', etc. } (default 'tfsapl') Name of function that will plot the TFD. It must be a function that takes three arguments: (xaxis,yaxis,data).

ExtraArgs { string of commands } (default '')
Extra arguments for 'PlotFn'; string of commands that will be executed using the `eval` function.

Title { string } (default '')
String containing title of plot

XLabel { string } (default 'Frequency (Hz)')
String containing label for x-axis.

YLabel { string } (default 'Time (s)')
String containing label for y-axis.

TFfontSize { numeric } (default 14)
Font size in points for 'Title', 'XLabel' and 'YLabel'.

Zoom { vector of length 4 } (default [0 1 0 1])
Start an end magnification of TFD matrix in x and y directions.

TFlog { 'on' | 'off' } (default 'off')
If 'on' the log of the TFD is plotted.

TFGrid { 'on' | 'off' } (default 'off')
Turn on/off the grid on the TFD plot. (Won't effect 'tfsapl' plot)

GrayScale { 'on' | 'off' } (default 'off')
Plots will be specified in grayscale overriding any values relating to colour scheme.

TFShading { 'flat' | 'interp' | 'faceted' } (default 'faceted')
Selects the shading type for the TFD plot. (Won't effect 'tfsapl' plot)

TFColourMap { 'jet', 'bone', etc } (default 'jet')
Colourmap for TFD plot. (Won't effect 'tfsapl' plot)

TFInvert { 'on' | 'off' } (default 'off')
Invert the colourmap for TFD plot. (Won't effect 'tfsapl' plot)

TFLine { 'black', 'white', etc } (default 'cyan')
Line colour for tfsapl plot ONLY.

TFBackGround { 'black', 'white', etc } (default 'black')
Background colour for tfsapl plot ONLY.

TimeLine { 'black', 'white', etc } (default depends of 'plotfn')
Line colour for time domain plot.

TimeBackground { 'black', 'white', etc } (default depends of 'plotfn')
Background colour for time domain plot.

TimeGrid { 'on' | 'off' } (default 'off')
Turn on/off grid for time domain plot.

TimeDetails { 'on' | 'off' } (default 'on')
Turn on/off text displaying sampling information of time signal.

FreqLine { 'black', 'white', etc } (default depends of 'plotfn')
Line colour for frequency domain plot.

FreqBackground { 'black', 'white', etc } (default depends of 'plotfn')
Background colour for frequency domain plot.

FreqGrid { 'on' | 'off' } (default 'off')
Turn on/off grid for frequency domain plot.

FigHandle { handle of figure } (default none)
Specify figure handle if plots are to go over whats there. Otherwise a new figure will be created or current figure will be cleared.

Examples

```
signal1 = gsig( 'sin', 0.25, 0.02, 128, 1, 1);
tfd1 = spec( signal1, 2, 31, 'hamm' );
tfsapl( signal1, tfd1, 'Timeplot', 'on', 'Freqplot', 'on',
'Grayscale', 'on', 'Title', 'Spectrogram of Sinusoidal FM Signal' );
```

Figure 6.7 shows the tfsapl plot of a Spectrogram representation of a sinusoidal FM signal.

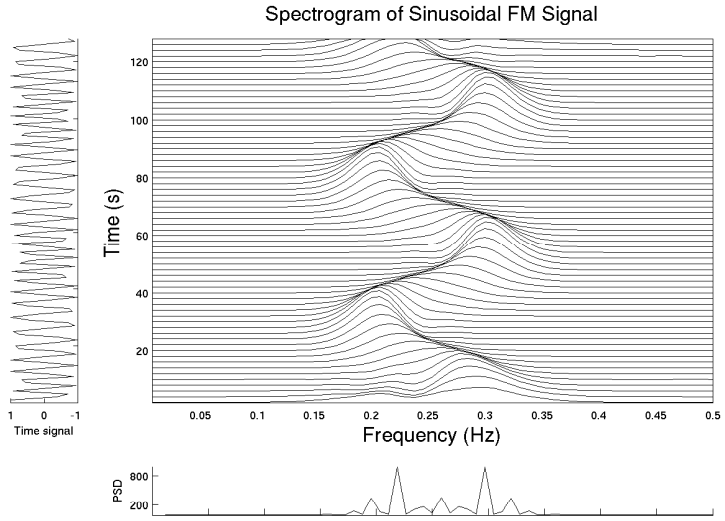


Figure 6.7: Tfsapl style plot for the sinusoidal FM signal.

See Also

wfall

unphase

Purpose

This function recovers the phase of a analytic signal.

Synopsis

phase1 = unphase(input1);

Description

This function recovers the phase of a complex signal. If the signal is real, then the `analyt` function should be utilised to render the complex version.

Examples

To recover the phase of the real signal `signal1` first utilise the `analyt` function i.e.

```
ana_sig1 = analyt(signal1)
```

Then recover the phase by:

```
phase1 = unphase(ana_sig1)
```

See Also

analyt

wfall

Purpose

Matrix waterfall plot, usually called from `tfsapl` function 6.1.

Synopsis

wfall(tfrep', ...);

Parameters

Refer to MATLAB documentation on waterfall and mesh.

Description

This function is a modification of the standard MATLAB function "waterfall". The differences are these:

1. No curtain appears around base of plot.
2. Axis limits are reduced to x and y data limits.
3. Graph is shifted so that it starts at x = y = 0, rather than x = y = 1.
4. Hidden line removal is not used, to prevent visual anomalies which occur with waterfall.
5. Lines along sides of plot are added.

Note that in this function, as in the original waterfall.m, the data is waterfalled row-wise.

Examples

To obtain a waterfall plot of the time-frequency matrix "tfd", with lines connecting frequency values:

```
wfall(tfd');
```

or from the `tfsapl`(see 6.1) function:

```
tfsapl( time1, tfd, 'plotfn', 'wfall' );
```

See Also

tfsapl, mesh (Standard MATLAB function)

wlet

Purpose

Forward and inverse fast wavelet transform using Daubechies wavelets

Synopsis

output = wlet(signal [, output_type [, num_coeff [, direction]]]);

Parameters

signal	Time series to be transformed. Signal must be a one-dimensional signal. The two dimensional signal output from wlet cannot be inverse transformed.
output_type	One of: 1 Output is one dimensional, and represents the raw result from the fast wavelet transform algorithm. This is the default. 2 Output is two dimensional. This output format is convenient for displaying transformed data as a time-scale matrix.
num_coeff	Number of coefficients to use. Possible values are 4, 12 or 20. The default is 20.
direction	One of: 1 Transform is forward, from time domain to time-scale domain. This is the default. -1 Transform is reverse, from time-scale domain to time domain. Two dimensional output can only be used with the forward transform, and the input to both forward and reverse transforms must be one-dimensional.

Description

This function performs the fast wavelet transform. Either forward or reverse transforms may be performed by setting the direction parameter. The algorithm implements decomposition into Daubechies wavelets, and uses a pyramidal filtering scheme. The input signal is filtered using quadrature mirror filters. The high pass output is decimated by 2 and saved in the upper half of the result vector. The lowpass output is decimated by 2 and is then considered as input. This process is iterated until the length of the remaining lowpassed signal is smaller than the number of filter coefficients. The resulting array consists of the concatenation of outputs of each highpass operation, and the remaining lowpass signal at the time of termination.

Examples

Perform the fast wavelet transform using a 12 coefficient Daubechies wavelet and 1-D output.

```
output = wlet(signal, 1, 12, 1);
```

wvd

Purpose

Computes the Wigner-Ville distribution

Synopsis

tfrep = wvd(signal, lag_win_len, time_res [, fft_length]);

Parameters

tfrep	The computed time-frequency distribution. size(tfrep) will return [a, b], where a is the next largest power of two above lag_win_len, and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
lag_win_len	The length of the data window used for analysis.
time_res	The number of time samples to skip between successive slices of the analysis.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the WVD of the input signal. An analytic signal generator is called if the input signal is real. The supplied length of the analysis window defines whether the "true" wvd or a pseudo (windowed) wvd is computed. This function is similar to the `quadtfd` function, except that it is fully optimised for speed and is more computationally efficient.

Examples

Compute the WVD of a 1024 point signal using a time-resolution of 20:

```
tf = wvd(signal, 1023, 20);
```

Compute the Pseudo-WVD of a 1024 point signal using time-resolution 20 and lag window width 63:

```
tf = wvd(signal, 63, 20);
```

See Also

quadtfd, xwvd

wvpe

Purpose

Estimates the instantaneous frequency of the input signal by extracting the peaks of the Wigner-Ville distribution.

Synopsis

ife = wvpe(signal, lag_win_len, time_res [,fft_length]);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal to be analysed. An analytic signal is required for this function, however, if signal is real, a default analytic transformer routine will be called from this function before computing tfrep.
lag_win_len	The length of the data window used for analysis.
time_res	The number of time samples to skip between successive instantaneous frequency estimates.
fft_length	Zero-padding at the FFT stage of the analysis may be specified by giving an fft_length larger than normal. If fft_length is not specified, or is smaller than the lag_win_len, then the next highest power of two above lag_win_len is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the Wigner-Ville distribution of the input signal and then takes the peak of this distribution in order to form the instantaneous frequency estimate. An analytic signal generator is called if the input signal is real. This function is fully optimised for speed.

See Also

wvd, lms, rls, pde, sfpe, zce

xwvd

Purpose

Computes the cross Wigner-Ville distribution

Synopsis

tfrep = xwvd(signal1, signal2, lag_win_len, time_res [, fft_length]);

Parameters

signal1, signal2	Input one dimensional signals to be analysed. These signals must be the same length.

Refer to wvd for information on other parameters.

Description

Refer to wvd.

Examples

Compute the Cross Wigner-Ville Distribution:

```
tfrep = xwvd( signal1, signal2, 1023, 20);
```

See Also

wvd

zce

Purpose

Estimates the instantaneous frequency of the input signal using the zero-crossing instantaneous frequency estimation algorithm.

Synopsis

ife = zce(signal, window_length);

Parameters

ife	Instantaneous frequency estimate (real).
signal	Input one dimensional signal (real or analytic).
window_length	The length of the data window for analysis.

Description

This function estimates the instantaneous frequency of an input signal using the zero-crossing estimator. The number of crossings within the data length `window_length` are counted and used to form the estimate.

Examples

Compute the instantaneous frequency estimate of the signal `time1` by using a window length of 64.

```
ife_zce = zce(time1, 64);
```

See Also

lms, rls, pde, sfpe, wvpe