TFSAP 7.0

Time-Frequency Signal Analysis Toolbox

Developed by Professor Boualem Boashash
IEEE Fellow, 1999

28 January, 2016

Acknowledgments

Developed within the Signal Processing Research Center with the principal assistance of my former RAs and Postdocs Peter Boles, Andrew Reilly, Branko Ristic, John O' Toole and Samir Ouelha. Other less junior or less involved RAs and PhD students were also involved since the 1980s. The funding for this project came from Prof Boashash's research grants obtained mostly from the ARC and NHMRC prior to 2009 and QNRF after 2010.

Licence agreement

TFSA is the former name of the current TFSAP toolbox.

Support policy

It is the intent of Prof. Boashash to continue to update TFSAP to reflect new ideas and algorithms, and to correct bugs which may be discovered. Bug reports are welcome, and should contain sufficient information to reliably reproduce the aberrant behaviour. Our address for matters concerning the TFSAP package is:

TFSAP
c/o Prof. B. Boashash
`tfsap.research@gmail.com>`

For fast technical support, contact the maintainer at:
`tfsap.research@gmail.com>`.

Part I

TFSAP[1] 7.0: Tutorial

[1]The name ”TFSA” has been replaced by ”TFSAP” to take into account new advances in TF processing. P stands for ”processing”.

Chapter 1

TFSAP Tutorial

The following four part tutorial takes a step-by-step approach to the use and understanding of time-frequency signal analysis and processing. It also has an adjunct purpose in that it familiarises the user with TFSAP 7.0 for MATLAB. The tasks presented below illustrate various aspects of time-frequency analysis, which it is hoped, will also develop the user's interpretation abilities, since the approach taken is from a practical viewpoint.

Before commencing this tutorial it is recommended that users first familiarize themselves with Chapter 17 of the 2$^{\text{nd}}$ Edition TFSAP book. 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.

1.1 Tutorial 1

In this tutorial the user is introduced to non-stationary signals, time-frequency distributions (TFDs) and the concept of the instantaneous frequency (IF).

1.1.1 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 the concept of TFDs and their use, the following steps should be completed, using the **TFSAP 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 **Chapter 17** should be consulted if difficulties arise in doing this. This type of signal may be easily explained through an oral 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 recommended. The advantages of such a display for a signal that has a time-varying spectra should become soon 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 B distribution have parameters which control the form of the final distribution. Consult the relevant sections of **part 3 - TFSAP 7.0: 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 (t, f) representation 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.

1.1.2 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 TFSAP 7.0 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 now 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 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.

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

1.1.3 Whale data

A record of whale data is provided with the TFSAP 7.0 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 TFSAP 7.0 **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?

1.2 Tutorial 2

In this tutorial the user 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.

1.2.1 TFDs and cross-terms

By now, it should be clear that TFDs do not always yield exactly 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 MBD. 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 TFSAP 7.0 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.

1.3 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.

1.3.1 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 that 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 TFSAP 7.0, 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.

1.3.2 EEG data[2]

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 EMBD, CKD and MDD 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.

[2]The data were collected at the Royal Brisbane and Womens hospital in Brisbane, Australia as part of several ARC, NHMRC and QNRF grants.

1.4 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 TFSAP 7.0 is restricted to one particular class of complete orthonormal wavelets introduced by I. Daubechies[3] (for more details, see Chapter 2 and Section 4.1 of the TFSAP book).

1.4.1 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?

1.4.2 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.

1.4.3 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 TFSAP 7.0 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

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

test signal ('signal1'). Try all three types of the wavelet transforms (D4, D12 and D20), and compare the results.

1.5 Summary

These four tutorials have provided a unique introduction to time-frequency signal analysis and the toolbox TFSAP 7.0 for MATLAB. They have been designed in order for the new user to experiment with 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 and intuitive notions 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.

1.6 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 II

TFSAP 7.0: Reference Guide

Chapter 2

Technical Reference

2.1 TFSAP 7.0 MATLAB Functions

Internal Function	Description
flatwf	Used in tfsapl to create TFSAP 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	TFSAP 7.0 demo file
tfsahelp	Help frame
tfsamain	Main TFSAP frame
tfsamenu	Handles the menus in the main frame
tfsaopen	Sets up main frame
tfsa_plot2d	TFSAP 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 TFSAP 7.0 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

<u>tfrep</u> 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.

<u>signal</u> 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 TFSAP 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 2.1 shows the image of the absolute value of the ambiguity function of a linear FM signal.

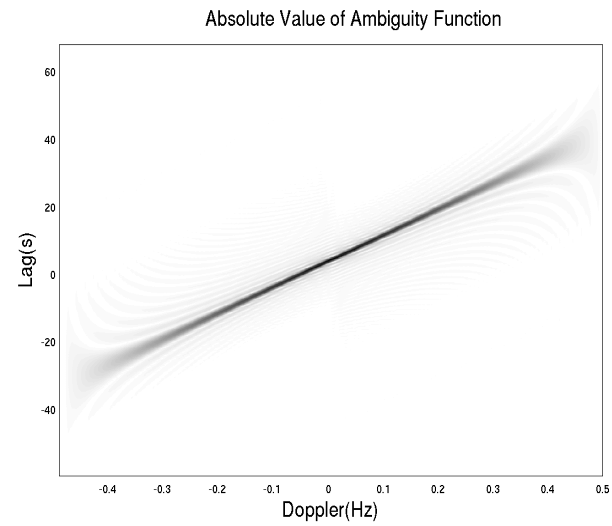

Figure 2.1: Absolute value of the ambiguity function of a linear FM signal, as generated by `ambf` and displayed using `image`.

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 TFSAP 7.0, 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 2.2, and is used in several other examples in the manual.

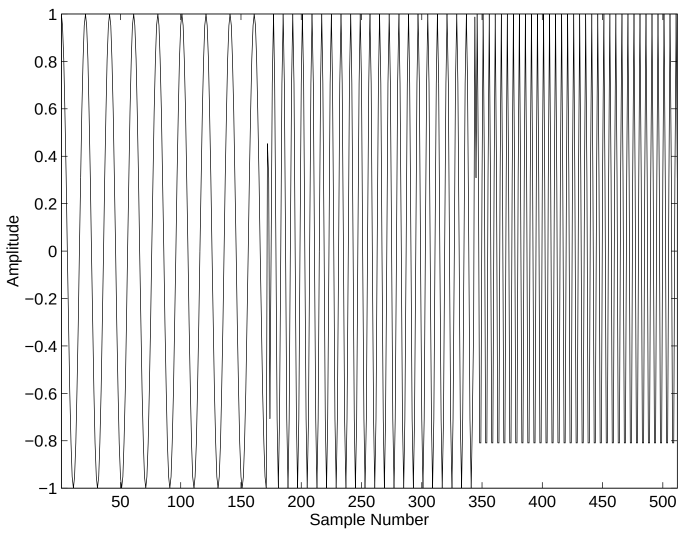

Figure 2.2: Stepped linear FM signal generated by gsig.

[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

mdd

Purpose

Computes the multi-directional distribution of an input signal.

Synopsis

[tfd, kernel, amb] = mdd(signal, C, D, E, theta, tr);

Parameters

signal	Input one dimensional signal to be analyzed.(real or analytic) .
C	Slope adjustment parameter for each branch of the multi directional kernel (MDK).
D	Half support of the MDK along the direction perpendicular to the i^{th} branch.
E	Half support of the MDK along the direction parallel to the i^{th} branch.
theta	Direction of the i^{th} branch.
tr	Time resolution
kernel	MD Kernel
amb	Ambiguity domain function

Description

This function generates a (t, f) representation based on the multi-directional kernel.

Examples

Generate a (t, f) distribution of a signal 'time1' using multi-directional distribution, with two branches MDK.

```
s1= gsig('lin', 0.05, 0.3, 256, 1);
s2= gsig('lin', 0.1, 0.35, 256, 1);
s3= gsig('lin', 0.4,0.36, 256, 1);
sig = s1+s2+s3;
C=[0.1 0.1]; D=[0.1 0.1]; E=[0.23 0.5]; theta=[96 64]; tr=2;
tfd_mdd= mdd(sig, C, D, E,theta, tr);
figure;tfsapl(sig,tfd_mdd);
```

See Also

quadtfd, wd, wvd

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.

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

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 2.3.

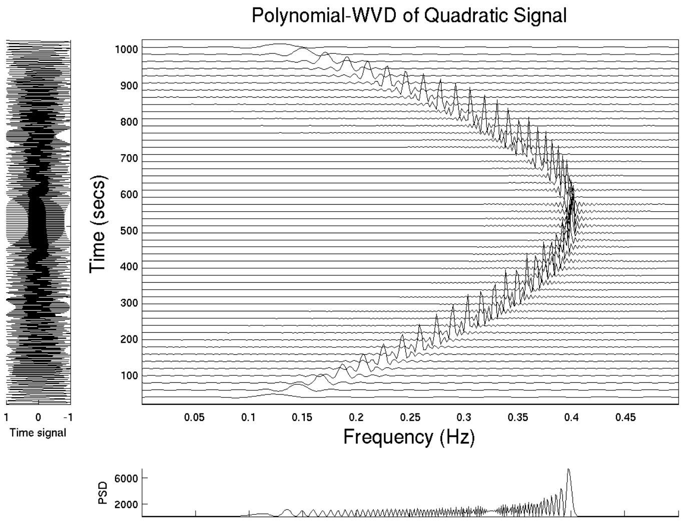

Figure 2.3: Sixth order kernel PWVD of a quadratic FM signal. The plot was generated using the `tfsapl` utility.

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 'emb' Extended 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 β
'mb'	kernel_option = modified B-distribution parameter α
'emb'	kernel_option = extended modified B-distribution parameter α
	kernel_option = extended modified B-distribution parameter β

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 2.4 shows the output of the command:

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

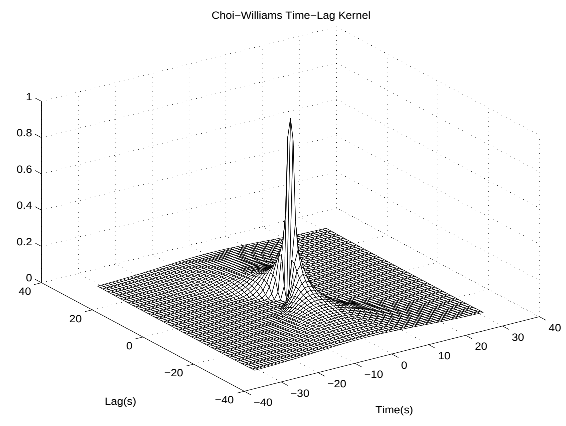

Figure 2.4: Time-lag Choi-Williams kernel, as generated by `quadknl` and displayed using `mesh`.

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 β
'mb'	kernel_option = Modified B-distribution parameter α
'emb'	kernel_option = Extended Modified B-distribution parameters α and β

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 2.5.

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

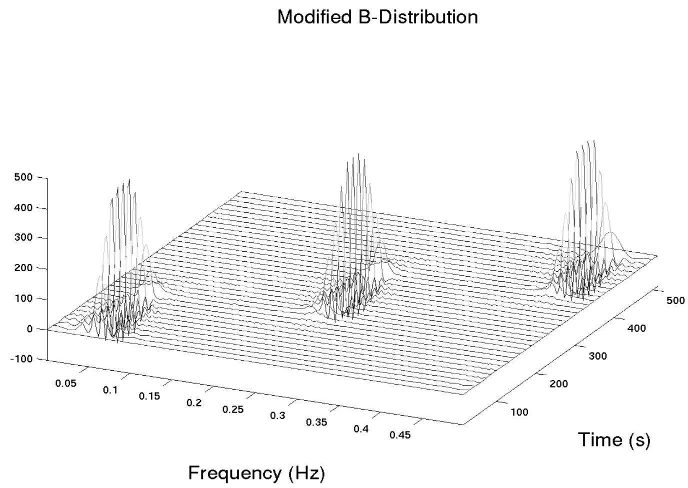

Figure 2.5: Waterfall plot of the Modified B-Distribution of a stepped FM signal.

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

specSM

Purpose

Computes the enhanced spectrogram of an input signal, using the S-method.

Synopsis

tfd = specSM(signal, L, wl, wtype, overlap, fftl);

Parameters

signal	Input one dimensional signal to be analyzed.(real or analytic) .
L	Define the length of frequency window in the specSM equals to $(2 * L + 1)$.
wl	Length of chosen window .
wtype	One of 'rect', 'hann','hamm', 'bart'.
overlap	Number of samples in common between two consecutives windows. 0 <overlap<wl
fftl	Zero-padding at the fft stage of the analysis may be specified by giving an fftl larger than normal. If fftl is not specified then the next highest power of two above the signal length will be used. If fftl is not a power of two, then the next highest power of two is used.

Description

This function computes the enhanced spectrogram using S-method. The size of tfd is [a/2+1,b], where a is the next largest power of two above fftl, and b is $\lceil x(\frac{N-overlap}{wl-overlap}\rceil$, where N is the length of the input signal.

Examples

Generate a (t, f) distribution of a 512 points signal time1 using specSM, with a 3 points smoothing window, hanning window with 63 points and an overlapping of length 60.

```
time1 = gsig('lin', 0.1, 0.4, 512, 1);
tf1 =  specSM( time1, 3, 63, 'hann', 60, 128);
tfsapl( time1, tf1);
```

See Also

stft, spec, quadtfd

stft

Purpose

Computes the time frequency distribution of an input signal using the short time Fourier Transform. This is a direct implementation using sliding windows and overlap.

Synopsis

[tfd, t, f] = stft (signal,Fe , wl, wtype, overlap [, fftl] [, stft_or_spec])

Parameters

signal	Input one dimensional signal to be analyzed.(real or analytic) .
Fe	Sampling frequency of the input signal .
wl	Length of chosen window .
wtype	One of 'rect', 'hann','hamm', 'bart'.
overlap	Number of samples in common between two consecutives windows. 0 <overlap<wl
fftl	Zero-padding at the fft stage of the analysis may be specified by giving an fftl larger than normal. If fftl is not specified then the next highest power of two above the signal length will be used. If fftl is not a power of two, then the next highest power of two is used.
stft_or_spec	Returns either Short Time Fourier Transform (STFT) or Spectrogram by specifying: 0 Spectrogram (default) 1 STFT

Description

This function computes the short time Fourier transform (Stft). The size of tfd is [a/2+1,b], where a is the next largest power of two above fftl, and b is $\lceil \frac{N-overlap}{wl-overlap} \rceil$, where N is the length of the input signal.

Examples

Generate a (t, f) distribution of a signal time1 using Short time Fourier transform (stft), with hanning window length of 63 points and an overlap window length of 60.

```
time1 = gsig('lin', 0.1, 0.4, 512, 1);
tf1 = stft( time1, 1, 63, 'hann', 60, 128);
tfsapl( time1, tf1);
```

See Also

spec, specSM

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' );
```

tfsa7

Purpose

Entry routine for the Graphical User Interface (GUI) to TFSAP 7.0

Synopsis

tfsa6

Parameters

none

Description

Use the mouse and keyboard to interact with the GUI. The GUI simply provides an alternative to typing TFSAP 7.0 commands on the MATLAB command line. Except as noted below, for each operation in the GUI there exists an underlying TFSAP 7.0 function which is described in this reference section. Refer to the relevant function description for information on parameters.

Figure 2.6 shows the main menu of the GUI.

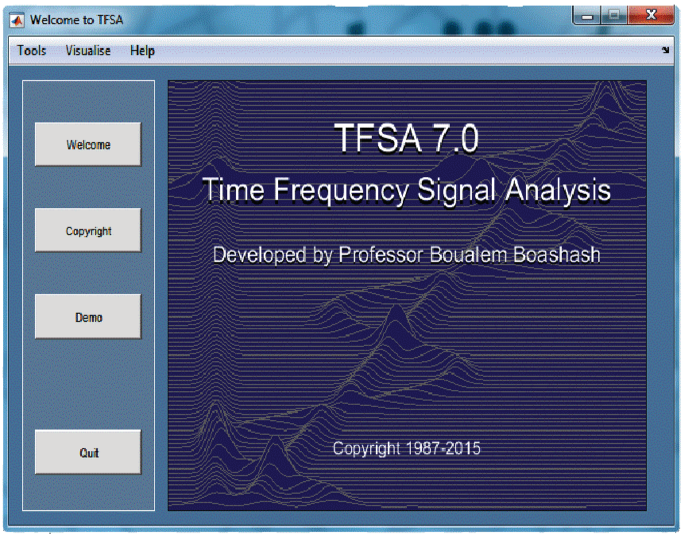

Figure 2.6: Main Menu of the TFSAP Graphical User Interface.

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

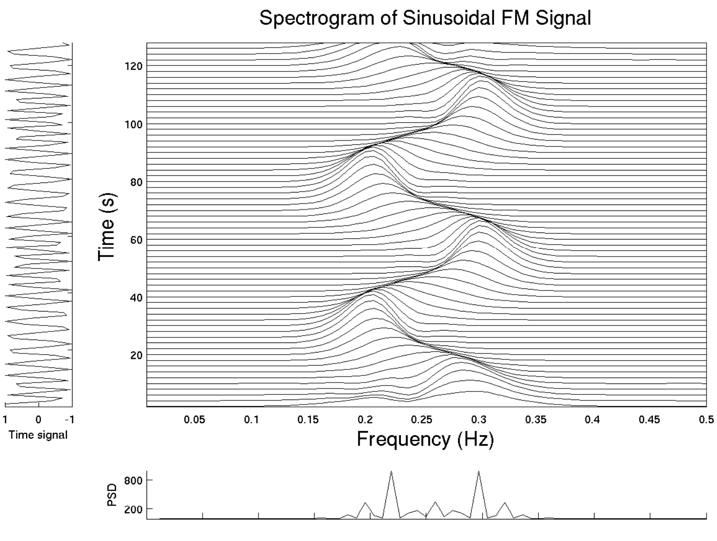

Figure 2.7: Tfsapl style plot for the sinusoidal FM signal.

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

wd

Purpose

Computes the Wigner distribution of an input signal.

Synopsis

tfrep = wd(signal, [,fft_length],time_res);

Parameters

tfrep	The computed time-frequency distribution. size(tfrep) will return [a, b], where a is the next largest power of two above the length of the signal and b is floor(length(signal)/time_res) - 1.
signal	Input one dimensional signal to be analyzed.
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. In the GUI, if fft_length is not specified , or is smaller than the length of the signal then the next highest power of two above the length of the signal is used. If fft_length is not a power of two, the next highest power of two is used.

Description

Computes the Wigner Distribution of the input signal. If the input signal is an analytic signal the produced result will be similar to the Wigner-Ville Distribution.

Examples

Compute the WD of a 256 point signal using a time-resolution of 2:

```
signal = gsig('lin',0.1,0.4,256);
tf = wd(signal, 512, 2);
```

See Also

quadtfd, wvd

wfall

Purpose

Matrix waterfall plot, usually called from `tfsapl` function 2.1.

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 2.1) 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