TQWT Toolbox Guide

Ivan Selesnick[*]
Electrical and Computer Engineering
Polytechnic Institute of New York University
Email: selesi@poly.edu

October 6, 2011

1 Introduction

The 'Tunable Q-Factor Wavelet Transform' (TQWT) is a flexible fully-discrete wavelet transform [5]. The TQWT toolbox is a set of Matlab programs implementing and illustrating the TQWT. The programs validate the properties of the transform, clarify how the transform can be implemented, and show how it can be used. The TQWT is similar to the rational-dilation wavelet transform (RADWT) [2], but the TQWT does not require that the dilation-factor be rational.

2 Parameters

The main parameters for the TQWT are the Q-factor, the redundancy, and the number of stages (or levels), as listed in Table 1.

The Q-factor, denoted Q, affects the oscillatory behavior the wavelet; specifically, Q affects the extent to which the oscillations of the wavelet are sustained. Roughly, Q is a measure of the number of oscillations the wavelet exhibits. For Q, a value of 1.0 or greater can be specified. (Q can be real-valued.) The definition of the Q-factor of an oscillatory pulse is the ratio of its center frequency to its bandwidth,

$$Q = \frac{f_o}{\mathrm{BW}}.$$

This terminology comes from the design of electronic circuits.

The parameter r is the redundancy of the TQWT when it is computed using infinitely many levels. Here 'redundancy' means total over-sampling rate of the transform (the total number of wavelet coefficients divided by the length of the signal to which the TQWT is applied.) The specified value of r must be greater than 1.0, and a value of 3.0 or greater is recommended. (When r is close to 1.0, the wavelet will not be well localized in time — it will have excessive ringing which is generally considered undesirable.) The actual redundancy will be somewhat different than r because the transform can actually be computed using only a finite number of levels. Moreover, when the radix-2 version of the TQWT is used, the redundancy will

[*]This work is supported in part by NSF under grant CCF-1018020.

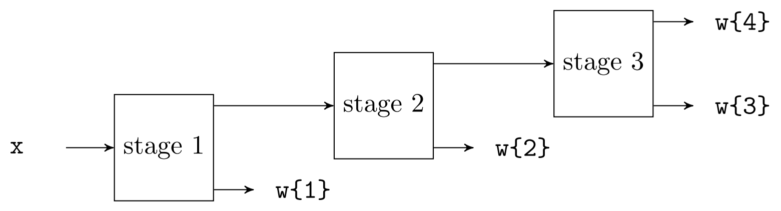


Figure 1: Wavelet transform with three stages ($J = 3$).

be greater than the specified r due to zero-padding performed within the radix-2 TQWT. Therefore, the parameter r affects the redundancy of the TQWT but it is not exactly equal to its redundancy.

The number of stages (or levels) of the wavelet transform is denoted by J. The transform consists of a sequence of two-channel filter banks, with the low-pass output of each filter bank being used as the input to the successive filter bank. The parameter J denotes the number of filter banks. Each output signal constitutes one subband of the wavelet transform. There will be $J + 1$ subbands: the high-pass filter output signal of each filter bank, and the low-pass filter output signal of the final filter bank. For example, a 3-stage wavelet transform is illustrated in Fig. 1.

3 Functions

The main toolbox functions are listed in Table 2. The main functions for the TQWT are `tqwt_radix2` and `itqwt_radix2`. These functions compute the forward and inverse of the radix-2 version of the TQWT. The radix-2 version of the transform uses FFTs which are powers of 2 in length. The functions `tqwt` and `itqwt` also compute the forward and inverse TQWT, but use FFTs of various lengths and are therefore less computationally efficient.

In practice, the radix-2 version of the TQWT should be used because of its relative computationally efficiency. The non-radix-2 functions, `tqwt` and `itqwt`, are provided in the toolbox primarily for completeness: to illustrate an implementation of the TQWT as it is described in [5] on which the radix-2 version is based.

4 Notes

So as to keep the programs minimal, there is almost no error checking of function input parameters. The user of these functions must keep careful track of parameters.

Send comments/questions to `selesi@poly.edu`

Website:
`http://eeweb.poly.edu/iselesni/TQWT/index.html`

The former URL (`http://taco.poly.edu/selesi/TQWT/index.html`) is no longer functional.

Table 1: TQWT Parameters.

Parameter	Description	note
Q	Q-factor Use $Q = 1$ for non-oscillatory signals. A higher value of Q is appropriate for oscillatory signals.	$Q \in \mathbb{R}$, $Q \geq 1$
r	Redundancy The total over-sampling rate when the TQWT is computed over many levels. It is recommended to select $r \geq 3$ in order that the analysis/synthesis functions (wavelets) be well localized.	$r \in \mathbb{R}$, $r > 1$
J	Number of stages (levels) of the TQWT J is the number of times the two-channel filter bank is iterated. There will be a total of $J + 1$ subbands. (The last subband will be the low-pass subband.)	$J \in \mathbb{Z}$, $J \geq 1$

Table 2: List of functions.

Function	Description
`tqwt_radix2.m`	Radix-2 TQWT
`itqwt_radix2.m`	Inverse radix-2 TQWT
`tqwt.m`	TQWT (less efficient than radix-2 version)
`itqwt.m`	Inverse TQWT
	Utility functions
`PlotSubbands.m`	Plot subbands of TQWT
`PlotEnergy`	Plot distribution of signal energy across subbands
`MaxLevels`	Maximum number of levels
`PlotFreqResps.m`	Plot frequency responses of the TQWT
`PlotWavelets.m`	Plot wavelets
`ComputeWavelets.m`	Compute wavelets
`ComputeNow.m`	Compute norms of wavelets
`tqwt_fc.m`	Center frequencies
	Sparsity functions
`tqwt_bp.m`	Sparse representation using basis pursuit (BP)
`tqwt_bpd.m`	Sparse approximation using basis pursuit denoising (BPD)
`dualQ`	Resonance decomposition (BP with dual Q-factors)
`dualQd`	Resonance decomposition (BPD with dual Q-factors)
	Demo functions
`demo1.m`	Illustrate the use of functions and syntax
`demo1_radix2.m`	Illustrate the use of functions and syntax
`sparsity_demo.m`	Reproduces Example 2 in [5]
`resonance_demo.m`	Demonstration of resonance decomposition
	Sub-functions
`afb.m`	Analysis filter bank
`sfb.m`	Synthesis filter bank
`lps.m`	Low-pass scaling
`next.m`	Next power of two
`uDFT.m`	unitary DFT (normalized DFT)
`uDFTinv.m`	unitary inverse DFT (normalized inverse DFT)

5 Examples

The following Matlab code snippet creates a test signal **x**, and calls the forward and inverse TQWT (radix-2 version) and verifies its perfect reconstruction property.

```
% Verify perfect reconstruction property
Q = 1; r = 3; J = 8;            % TQWT parameters
x = test_signal(4);             % Make test signal
N = length(x);                  % Length of test signal
w = tqwt_radix2(x,Q,r,J);       % TQWT
y = itqwt_radix2(w,Q,r,N);      % Inverse TQWT
recon_err = max(abs(x - y))     % Reconstruction error

% recon_err =
%    3.8858e-16
```

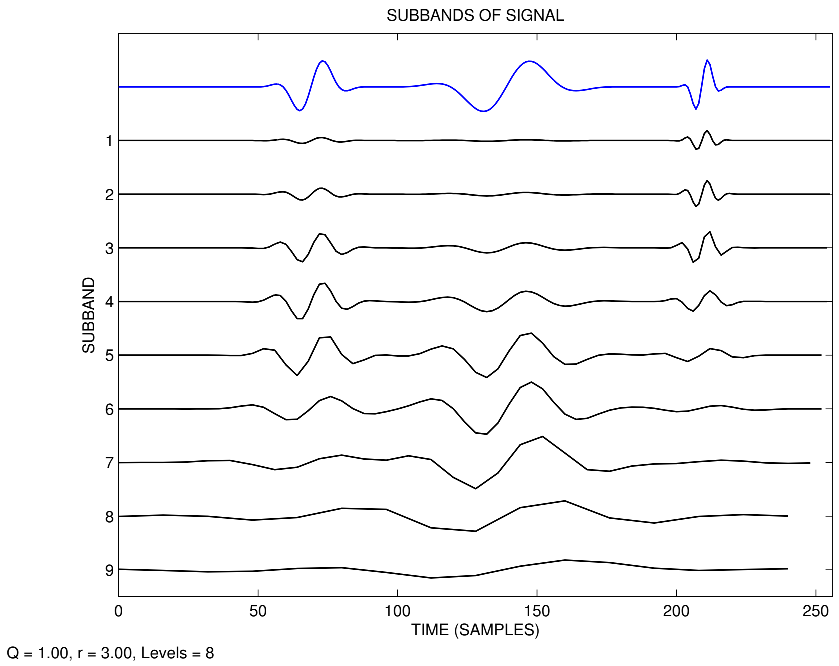
The function `tqwt_radix2` returns a cell array `w` of wavelet coefficients. The first subband (the high-frequency subband) is given by `w{1}`. As eight stages of the TQWT are computed in this example, there are a total of nine subbands. The last subband, `w{9}`, being the low-pass subband. The length of each subband can be seen as follows:

```
>> w'
ans =
    [1x256 double]
    [1x256 double]
    [1x128 double]
    [1x128 double]
    [1x64  double]
    [1x64  double]
    [1x32  double]
    [1x16  double]
    [1x16  double]


>> length(w)        % number of subbands
ans =
    9

>> size(w{4})       % length of subband w{4}
ans =
     1   128

>> length(x)        % length of test signal x
ans =
    256
```

In this case, there is a total of 960 wavelet coefficients. So the actual redundancy is $960/256 = 3.75$ which is more than the parameter r which was set to 3.0 above. The higher redundancy is due primarily to the use of the radix-2 version of the TQWT here.

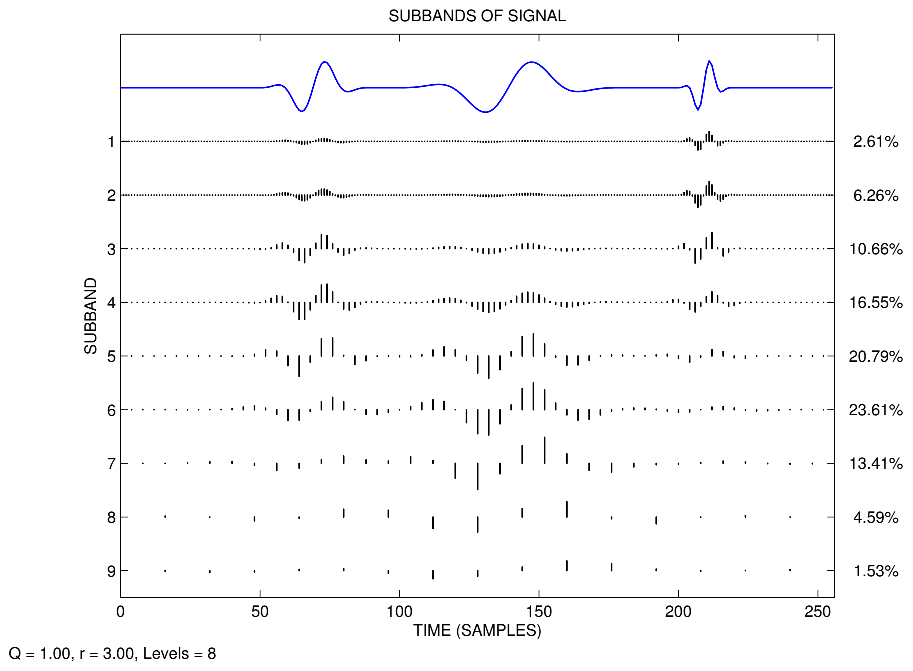
Subbands: The subbands can be displayed with the function `PlotSubbands`. This syntax is `PlotSubbands(x,w,Q,r,J1,J2,fs)`. This function displays subbands `J1` through `J2` as illustrated in the code snippet below. The parameter `fs` refers to the sampling frequency of the signal, which is set to 1 sample/second here. Below, subbands 1 through 9 are displayed, although a smaller range of subbands can be displayed if desired (for example, if some subbands are negligible). In the figure, the signal **`x`** is shown at the top.

```
fs = 1;
figure(1), clf
PlotSubbands(x,w,Q,r,1,J+1,fs);
```



The function `PlotSubbands` has an option, `'E'`, that displays the energy in each subband as a percentage of the total energy. In addition, the subbands can be displayed with a 'stem'-style with the option `'stem'`.

```
figure(1), clf
PlotSubbands(x,w,Q,r,1,J+1,fs,'E','stem');
```

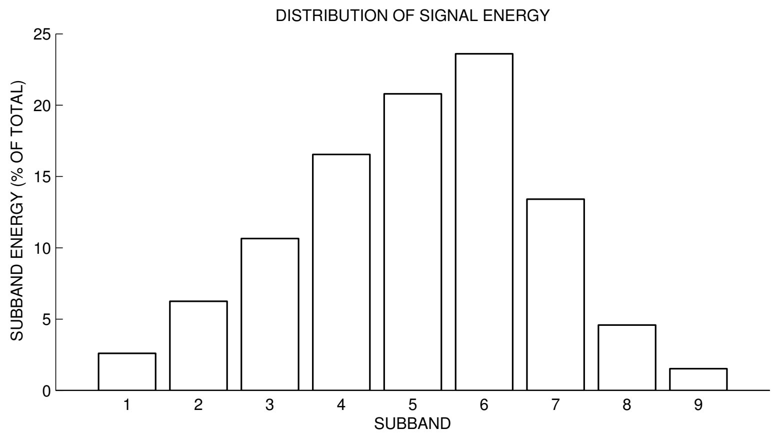


Energy: The TQWT satisfies Parseval's theorem, meaning that the total energy of the wavelet coefficients equals the energy of the signal. This is illustrated in the following code snippet.

```
% Verify Parseval's theorem
E = sum(x.^2);                      % Energy of signal
Ew = 0;                             % Energy in wavelet domain
for j = 1:J+1
    Ew = Ew + sum(abs(w{j}).^2);
end
fprintf('Signal energy = %f\n',E)
fprintf('Energy in wavelet domain = %f\n', Ew)

% >> Signal energy = 20.728991
% >> Energy in wavelet domain = 20.728991
```

It can also be useful to know how the energy of a signal is distributed across the subbands. The function `PlotEnergy` computes the energy in each subband and displays it as a bar graph.

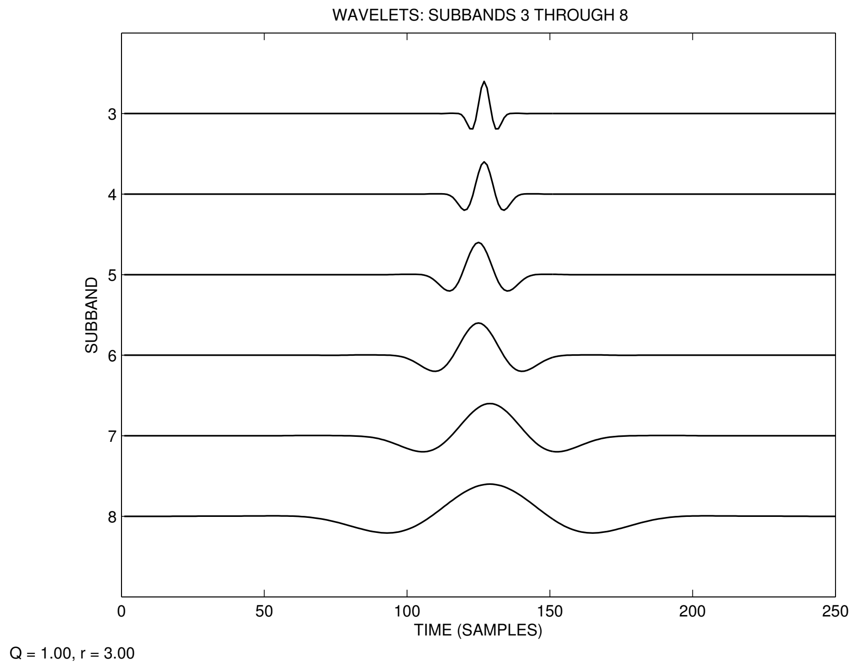
```
% Plot distribution of energy across subbands
figure(1), clf
subplot(3,1,1:2)
e = PlotEnergy(w);
```



Note that subband 1 corresponds to high-frequencies, while subband 9 corresponds to the low-pass band.

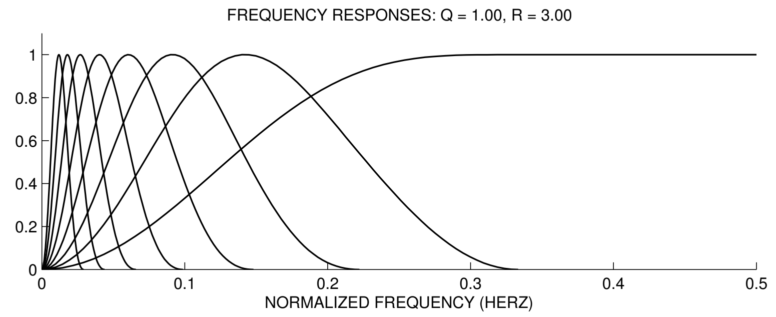
Wavelets: The wavelets may be displayed for a specified set of subbands using the function `PlotWavelets`. The syntax is `PlotWavelets(N,Q,r,J1,J2)` where `N` is the length of the wavelets (in samples), `Q` and `r` are the Q-factor and redundancy parameters, and `J1` and `J2` are the first and last subbands for which the wavelet is to be computed. The syntax `PlotWavelets(...,'radix2')` specifies that the wavelets should be computed using the radix-2 version of the TQWT. For example, the following code snippet displays the wavelets for subbands 3 through 8 (using Q and r previously set).

```
% Display wavelets for subband 3 through 8
Q = 1; r = 3; J = 8;            % TQWT parameters
PlotWavelets(250,Q,r,3,8,'radix2');
```



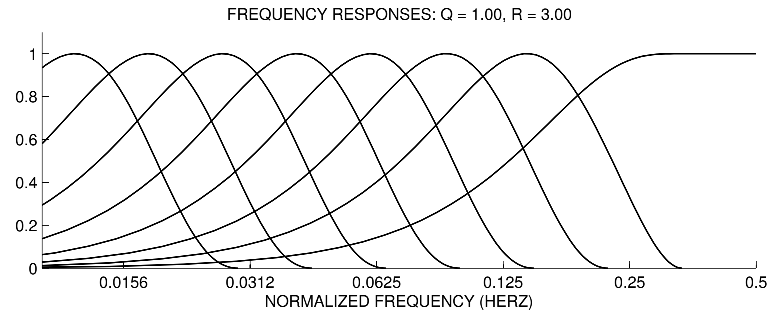
Frequency Responses: The frequency decomposition performed by the TQWT can be displayed using the function `PlotFreqResps`.

```
% Plot frequency response of the TQWT
Q = 1; r = 3; J = 8;            % TQWT parameters
subplot(2,1,1)
PlotFreqResps(Q, r, J)
```

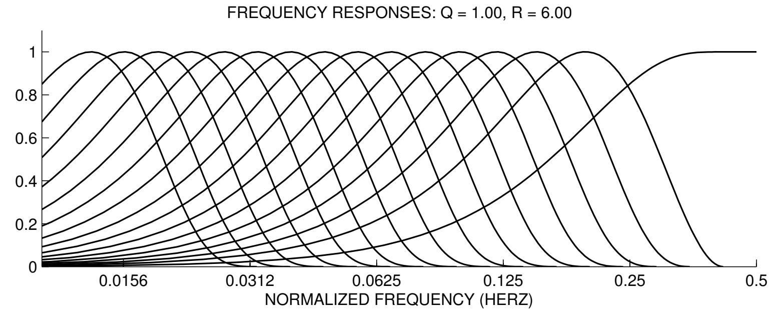


The low Q-factor ($Q = 1.0$) can be recognized in both the shape of the wavelets and in the plot of the frequency responses: The wavelets have a low-oscillation behavior and the frequency responses are wide relative to their center frequencies.
Due to the constant-Q property, the frequency responses have equal width on a log frequency axis. The following Matlab commands change the frequency axis to a log scale.

```
% Plot frequency response of the TQWT on log frequency axis
Q = 1; r = 3; J = 8;            % TQWT parameters
PlotFreqResps(Q, r, J)
set(gca,'xscale','log')
set(gca,'xtick',0.5*2.^(-6:0))
xlim([0.01 0.5])
```



Effect of parameter r: Increasing r, while keeping Q unchanged, has the effect of increasing the overlap between adjacent frequency responses. The parameter r does not effect the general shape of the wavelet of frequency response (they are controlled by Q). With a larger r, the number of levels J should be increased in order to cover the same frequency range, because of the increased overlap. The following figure shows the frequency responses (on log frequency scale) with an r of 6. This is twice the r used above, so twice the number of frequency bands are needed to cover the same frequency range. Comparing the two figures, it can be seen that adjacent bands overlap more when r is larger.



This figure was generated by the following Matlab code.

```
PlotFreqResps(Q, 2*r, 2*J)
set(gca,'xscale','log')
set(gca,'xtick',0.5*2.^(-6:0))
xlim([0.01 0.5])
```

5.1 Sparse signal representation

As the TQWT is over-sampled (redundant), the wavelet coefficients that yield a given signal are not unique. In the examples above, the redundancy parameter r was set to 3.0 (the actual redundancy is greater than this).

In some applications, it is useful to find a sparse set of wavelet coefficients for a given signal $\mathbf{x}$. One approach is *basis pursuit* (BP) [3], the idea of which is to find a set of coefficients with minimal ℓ_1-norm. Given a signal $\mathbf{x}$, one needs to solve the optimization problem:

$$\begin{gathered}\operatorname*{argmin}_{\mathbf{w}} \|\mathbf{w}\|_1 \\ \text{such that } \ \mathrm{TQWT}^{-1}(\mathbf{w}) = \mathbf{x}.\end{gathered}$$

For the radix-2 TQWT, the synthesis functions (wavelets) do not all have the same energy (l_2-norm squared). Specifically, the energy is different in different subbands. Therefore, a suitable modification of the above problem is:

$$\begin{gathered}\operatorname*{argmin}_{\mathbf{w}} \sum_{j=1}^{J+1} \lambda_j \|\mathbf{w}_j\|_1 \\ \text{such that } \ \mathrm{TQWT}^{-1}(\mathbf{w}) = \mathbf{x}\end{gathered} \tag{1}$$

where $\mathbf{w}_j$ represents subband j, and λ_j are regularization parameters. The vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{J+1})$ can be used to take into account the fact that the ℓ_2-norm of the wavelet is different in different subbands. For example, it is appropriate that each λ_j be set proportional to the ℓ_2-norm of the wavelet at subband j.

The program `tqwt_bp` solves problem (1) using a variation of SALSA [1]. As shown in the code snippet below, `tqwt_bp` requires several parameters. First: the signal $\mathbf{x}$, the Q-factor Q, the redundancy parameter r, and the number of stages J. The function also requires three additional parameters:

`lambda`: a vector of length $J+1$ of regularization parameters λ_j.

`mu`: parameter related to the SALSA algorithm (affects convergence speed).

`Nit`: number of iterations of the SALSA algorithm.

We will set λ_j to be proportional the ℓ_2-norm of the wavelet at level j. The vector `now` in the code snippet below stands for 'norm of wavelets'. The choice of `mu` can be selected by 'trial and error'. Generally, `mu` may be roughly proportional to the signal energy.
The basis pursuit function `tqwt_bp` returns a sparse set of wavelet coefficients and, optionally, a vector of the cost function values per iteration.
The following code snippet illustrates the use of `tqwt_bp`.

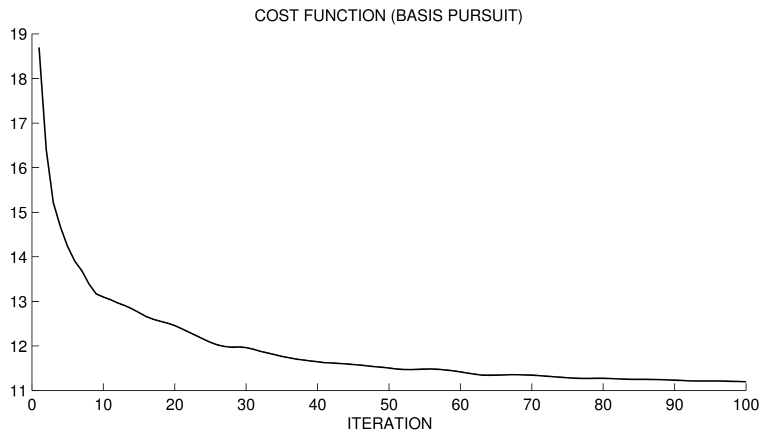
```
% Sparse signal representation (Basis pursuit)
now = ComputeNow(N,Q,r,J,'radix2');                   % Compute norms of wavelets
mu = 2.0;                                             % SALSA parameter
Nit = 100;                                            % Number of iterations
lambda = now;                                         % Regularization parameters
[w2, costfn] = tqwt_bp(x, Q, r, J, lambda, mu, Nit);  % Basis pursuit
y = itqwt_radix2(w2, Q, r, N);                        % Signal reconstruction
err = x - y;                                          % Reconstruction error
max(abs(err))
% ->   3.3307e-16
```

Note that in the code snippet it is verified that the returned wavelet coefficients `w2` correctly reconstructs the signal `x`.
The following code snippet shows how the cost function is computed. This is the cost function that `tqwt_bp` minimizes.

```
cost = 0;
for j = 1:J+1
    cost = cost + lambda(j)*sum(abs(w2{j}));
end
cost
% ->  11.1948
```

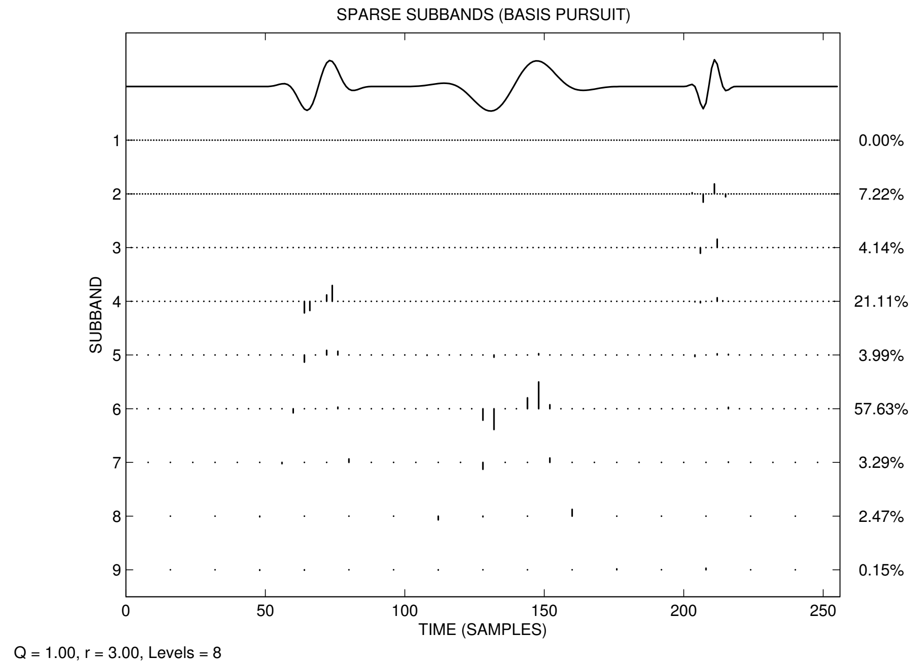
It can be useful to plot the cost function versus iteration as in the code snippet below. This plot verifies that the iterative minimization algorithm is converging.

```
% Display cost function
figure(2), clf
subplot(3,1,1:2)
plot(costfn);
title('COST FUNCTION (BASIS PURSUIT)')
xlabel('ITERATION')
box off
```



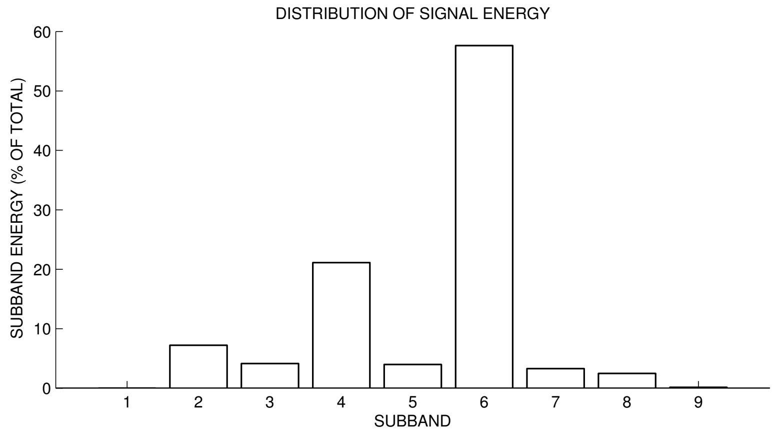
Does 'basis pursuit' actually produce a sparse set of wavelet coefficients? The function `PlotSubbands` can be used as above, to display of the subbands. The resulting figure demonstrates the enhanced sparsity obtained by ℓ_1 norm minimization. (Compare the subbands illustrated below to the subbands obtained above for the same signal.)

```
% Plot sparse subbands
figure(3), clf
PlotSubbands(x,w2,Q,r,1,J+1,fs,'E','stem');
title('SPARSE SUBBANDS (BASIS PURSUIT)')
```



Sparsification of the subbands can also lead to a different distribution of signal energy across subbands. The function `PlotEnergy` can be used as above. The resulting plot shows that sparsification can have the effect of compressing the energy of the wavelet representation into fewer subbands. The plot below shows that much of the energy is now concentrated in subbands 2, 4, and 6, in contrast to the broad distribution of energy previously seen. In the previous bar graph of energy distribution, note that no subband had more than 25% of the total energy; while for the sparsified wavelet representation a single subband (subband 6) accounts for more than half the total energy (57%).

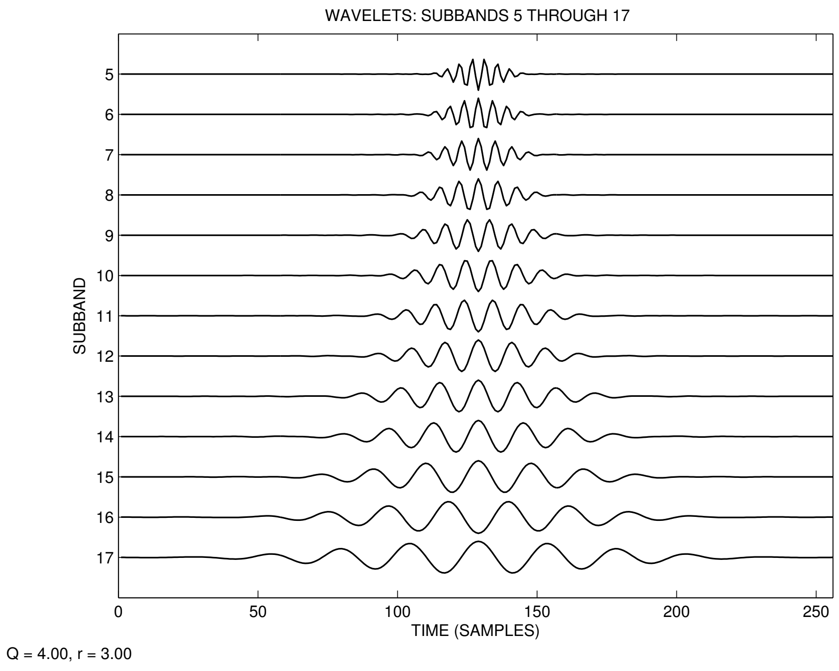
```
figure(3), clf
subplot(3,1,1:2)
e = PlotEnergy(w2);
```

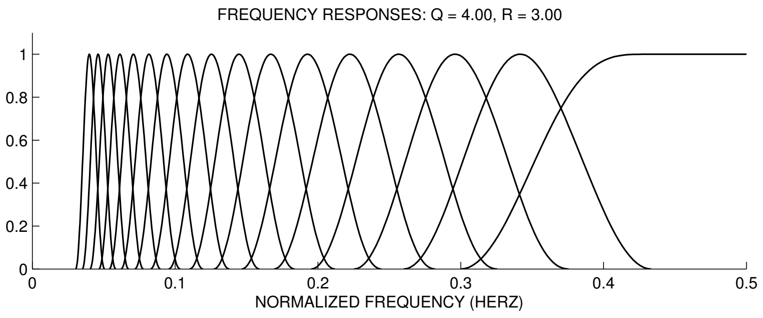


5.2 Higher Q-factor

The TQWT allows the user to specify the Q-factor. By increasing Q, the wavelets become more oscillatory (due to the higher Q-factor). The following code snippet displays the wavelets and the frequency responses when Q is set to 4.0. Note that the frequency responses are more narrow now, compared to above where Q was set to 1.0. With Q increased from 1.0 to 4.0, more stages are needed in order to span the same frequency range because each frequency response is narrower. Here we have used 17 stages instead of 8 above.

```
Q = 4; r = 3; J = 17;             % TQWT parameters
figure(1), clf
N = 256;
PlotWavelets(N, Q, r, 5, J, 'radix2');

figure(2),clf
subplot(2,1,1)
PlotFreqResps(Q, r, J);
```

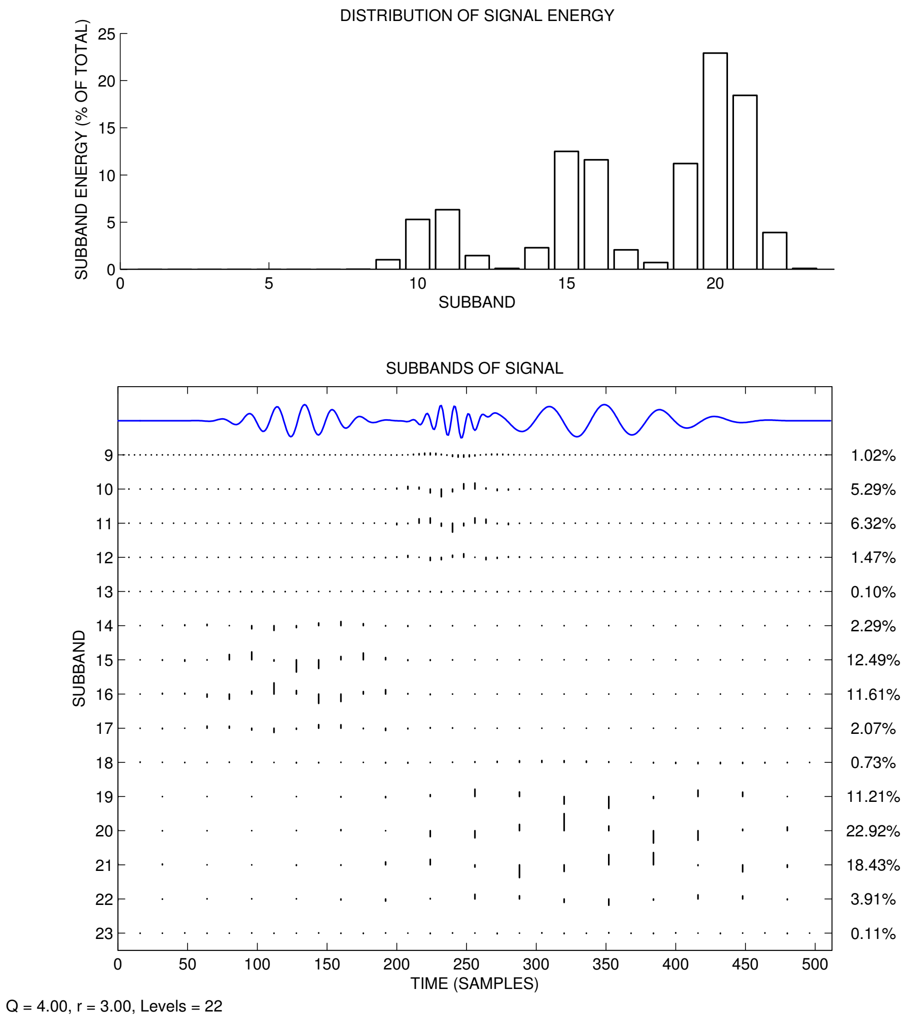




The following code snippet creates a new test signal, applies the TQWT, displays the subbands, and displays the distribution of energy across subbands. This test signal has a more oscillatory behavior than the test signal used above. Note that sufficiently many stages are specified (22 stages) so that the low-pass subband (subband 23) has negligible energy. Because the first eight subbands have essential zero energy, it is chosen in the code snippet not to display them by setting `J1` in `PlotSubbands` to 9.

```
x = test_signal(2);             % Make test signal
N = length(x);
Q = 4; r = 3; J = 22;           % TQWT parameters
w = tqwt_radix2(x, Q, r, J);

% Plot energy distribution
figure(1), clf
subplot(2,1,1)
PlotEnergy(w);
```

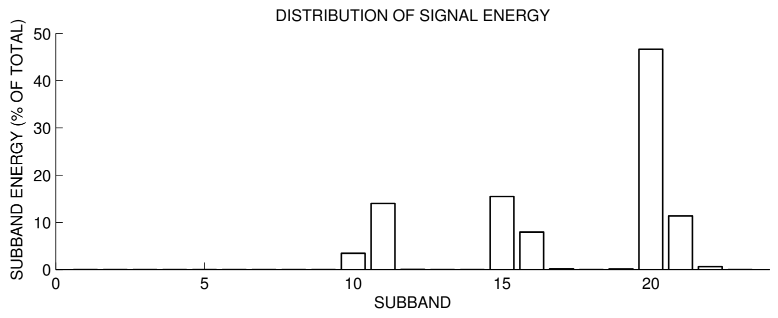
```
% Plot subbands
figure(2), clf
PlotSubbands(x, w, Q, r, 9, J+1, 1, 'E', 'stem')
```

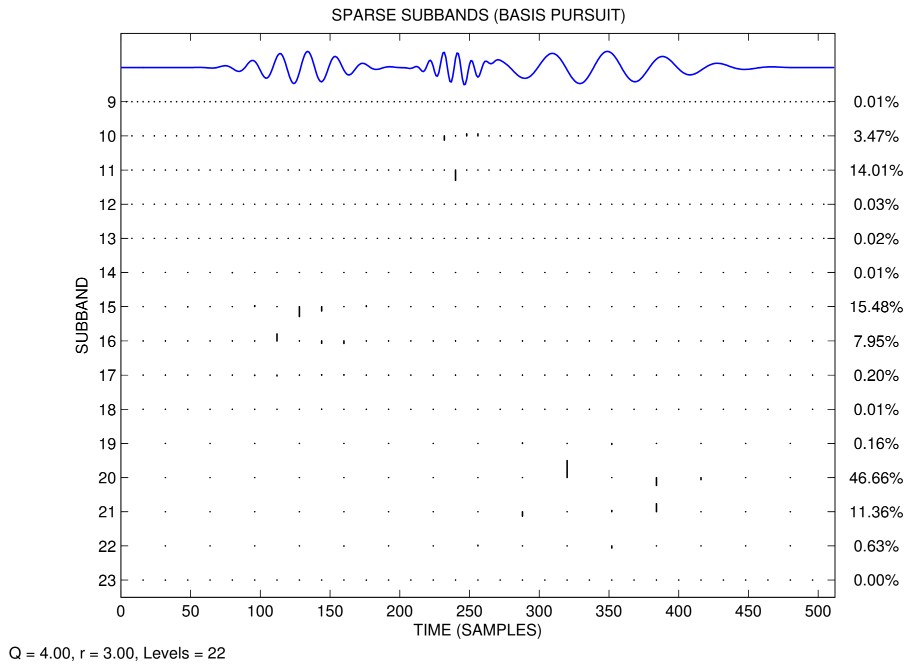


As above, we can apply basis pursuit (`tqwt_bp`) to find a sparse set of wavelet coefficients.

```
% Sparse signal representation (Basis pursuit)
now = ComputeNow(N,Q,r,J,'radix2');                 % Compute norms of wavelets
mu = 2.0;                                           % SALSA parameter
Nit = 100;                                          % Number of iterations
lambda = now;                                       % Regularization parameters
[w2, costfn] = tqwt_bp(x, Q, r, J, lambda, mu, Nit);   % Basis pursuit

% Plot sparse subbands
figure(1), clf
PlotSubbands(x,w2,Q,r,9,J+1,fs,'E','stem');
```

```
title('SPARSE SUBBANDS (BASIS PURSUIT)')

% Plot energy distribution
figure(2), clf
subplot(2,1,1)
e = PlotEnergy(w2);
```





Note that the wavelet coefficients `w2` obtained by basis pursuit are indeed substantially more sparse than the wavelet coefficients `w`, even though both sets of wavelet coefficients exactly represents the signal $\mathbf{x}$. As above, basis pursuit also leads to a compression of the energy into fewer subbands.

5.3 Sparse signal approximation

Previously, we saw that a sparse representation of a signal can be obtained by solving the basis pursuit problem. A related approach can be used for signal denoising, where the observed signal $\mathbf{y}$ has been corrupted by additive noise,

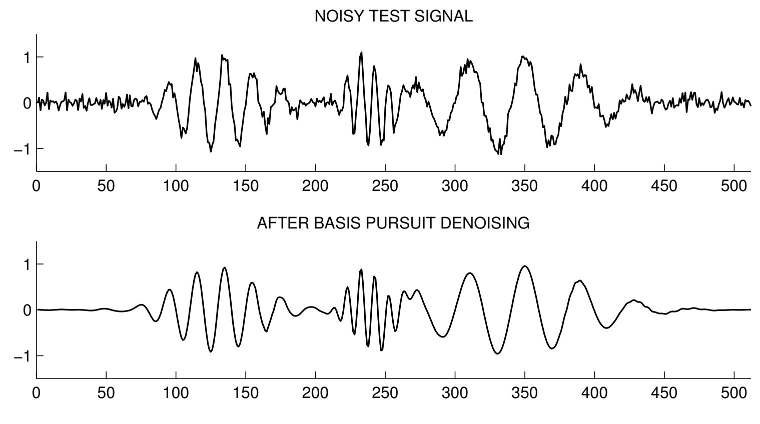
$$\mathbf{y} = \mathbf{x} + \mathbf{n}.$$

The problem is to estimate $\mathbf{x}$ from the observed signal $\mathbf{y}$. If it is known that $\mathbf{x}$ has a sparse representation with respect to a wavelet transform (or other transform), then it can be estimated via sparsity-based methods. One approach is *basis pursuit denoising* (BPD) [3] which minimizes the sum of the ℓ_1-norm of the transform coefficients and the energy of the residual:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \; \|\mathbf{y} - \mathrm{TQWT}^{-1}(\mathbf{w})\|_2^2 + \sum_{j=1}^{J+1} \lambda_j \|\mathbf{w}_j\|_1 \tag{2}$$

Then $\mathbf{x}$ can be estimated as $\mathrm{TQWT}^{-1}(\mathbf{w})$.
Basis pursuit denoising does not find an exact representation for $\mathbf{y}$ — instead the resulting wavelet coefficients give an approximation to the given signal $\mathbf{y}$. It is appropriate when noise (or general stochastic component) is present in the observed signal $\mathbf{y}$. The function `tqwt_bpd` solves the BPD problem with the TQWT. It is based again on a variant of SALSA.
The following code snippet illustrates the use of BPD for signal denoising with the TQWT.

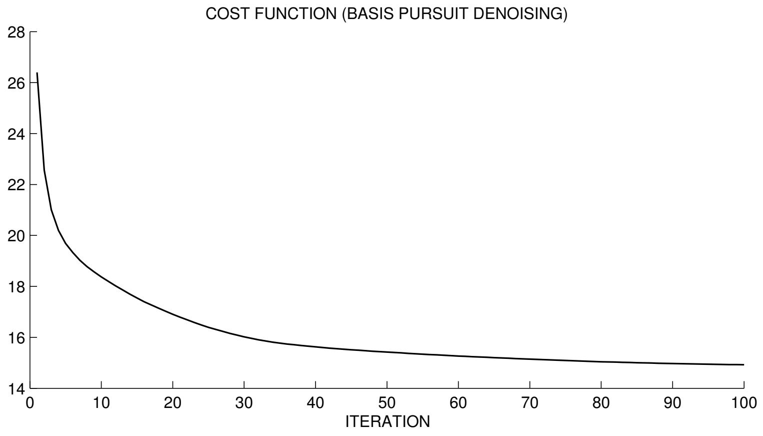
```
% Basis pursuit denoising example
y = test_signal(2);             % Make test signal
N = length(y);                  % Signal length
y = y + 0.1*randn(1,N);         % Noisy test signal

% Call basis pursuit denoising function
now = ComputeNow(N,Q,r,J,'radix2');                 % Compute norms of wavelets
Q = 4; r = 3; J = 22;                               % TQWT parameters
lambda = 0.5*now;                                   % Regularization parameters
mu = 2.0;                                           % SALSA parameter
Nit = 100;                                          % Number of iterations
[w, costfn] = tqwt_bpd(y,Q,r,J,lambda,mu,Nit);      % Basis pursuit denoising
x = itqwt_radix2(w, Q, r, N);                       % Denoised signal
```



To clarify the cost function minimized by the function `tqwt_bpd`, it is computed as shown in the following code snippet.

```
cost = sum(abs(y - x).^2);
for j = 1:J+1
    cost = cost + lambda(j)*sum(abs(w{j}));
end
cost
% ->  14.7165
```

The convergence of `tqwt_bpd` can be assessed by plotting the cost function versus iteration, `costfn`.



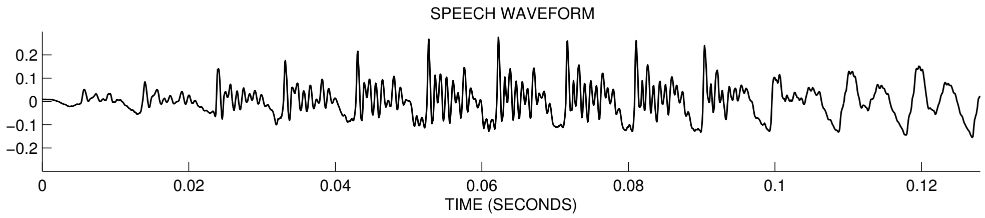
In this example we have used `lambda = 0.5*now`. Using a smaller multiplier (say 0.3 instead of 0.5) will result in less noise reduction because the sparsity of the wavelet coefficients is de-emphasized. Using a greater multiplier (say 0.7 instead of 0.5) will result in more noise reduction but will also result in distortion of the signal. There is a trade-off between noise reduction and signal distortion.

5.4 Speech waveform

In this section, we illustrate the use of the TQWT on a speech waveform. This signal is "I'm" spoken by an adult male, sampled at $16,000$ samples per second. Below, this waveform is analyzed first with a Q-factor of 4.0, then with a Q-factor of 1.0. It will be illustrated in Section 5.5 how the waveform can be sparsely represented using both Q-factors jointly.
The following code snippet loads the waveform and produces a plot of the signal.

```
% Load signal
[x, fs] = test_signal(3);
N = length(x);

% Plot signal
t = (0:N-1)/fs;
figure(1), clf
subplot(4,1,1)
plot(t,x)
xlim([0 N/fs])
ylim([-0.3 0.3])
```

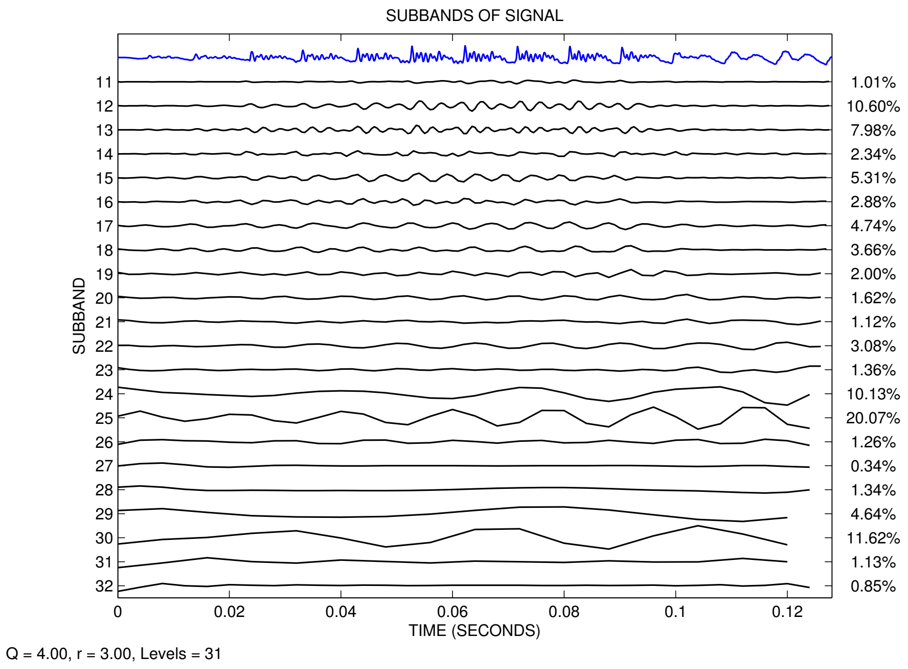
```
xlabel('TIME (SECONDS)')
title('SPEECH WAVEFORM')
box off
```

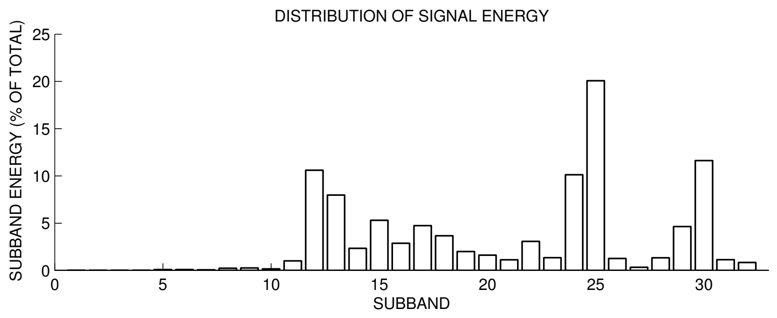


Q-factor = 4.0

The following snippet applies the TQWT with a Q-factor of 4.0. The parameters are denoted by `Q1`, `r1`, and `J1` to distinguish these values from the parameter values to be used subsequently. The resulting set of wavelet coefficients is denoted by `w1`.

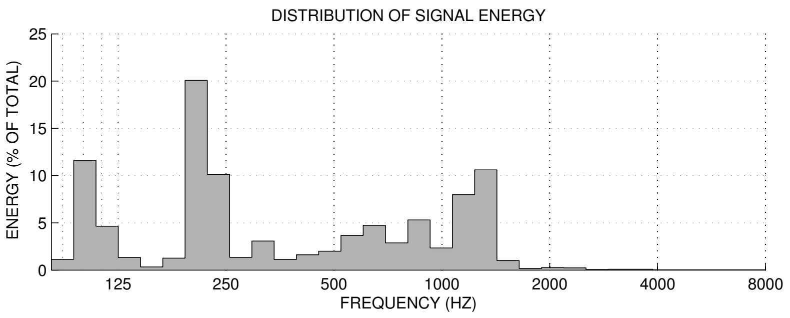
```
% High Q-factor TQWT
Q1 = 4; r1 = 3; J1 = 31;          % TQWT parameters
w1 = tqwt_radix2(x, Q1, r1, J1);

% Plot energy distribution
figure(1), clf
subplot(2,1,1)
PlotEnergy(w1);

% Plot subbands
figure(2), clf
PlotSubbands(x, w1, Q1, r1, 11, J1+1, fs, 'E')
```





The figure above shows the distribution energy across subbands — but the frequencies are not indicated. To make the figure more informative, we can instead plot the distribution of energy versus *frequency*, as follows. Note that in the plot below, the frequency axis is logarithmic, low frequencies are to the left, and the low-pass subband is omitted from the plot.

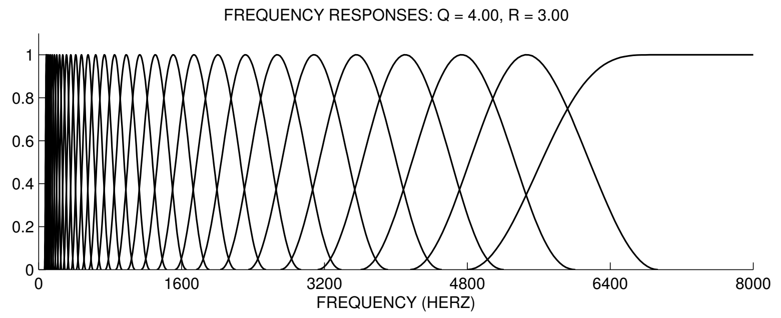
```
% Plot energy versus frequency
figure(1), clf
subplot(2,1,1)
PlotEnergy(w1, Q1, r1, fs);
```



As illustrated previously, the function `PlotFreqResps` displays the frequency response of the TQWT. Optionally, using the command syntax:

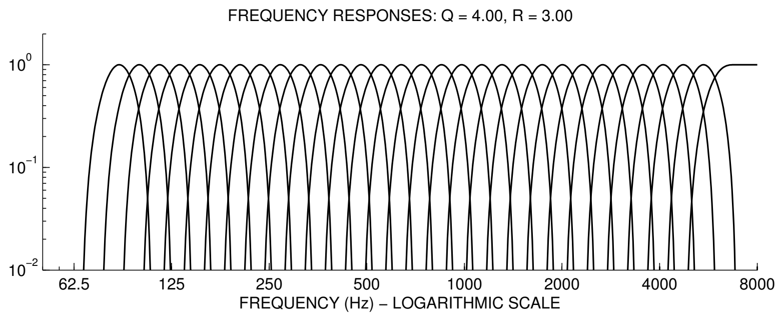
```
PlotFreqResps(Q1, r1, J1, fs);
```

the sampling frequency `fs` can be specified. Note, `fs` is the sampling frequency of the signal to which the TQWT is applied. Here `fs` is 16,000 samples/second, so the Nyquist frequency `fs/2` is 8,000 Hz, as illustrated in the plot.



The constant-Q property of the TQWT means that the frequency responses appear as equal width on a log-frequency axis. By changing the scaling property of the axis in Matlab, the frequency responses can be viewed on a loglog scale as illustrated in the following code snippet.

```
PlotFreqResps(Q1, r1, J1, fs);
set(gca,'Xscale','log')
set(gca,'Yscale','log')
set(gca,'Xtick',fs/2*2.^(-7:0))
xlim([50 fs/2])
ylim([0.01 2])
xlabel('FREQUENCY (Hz) - LOGARITHMIC SCALE')
```

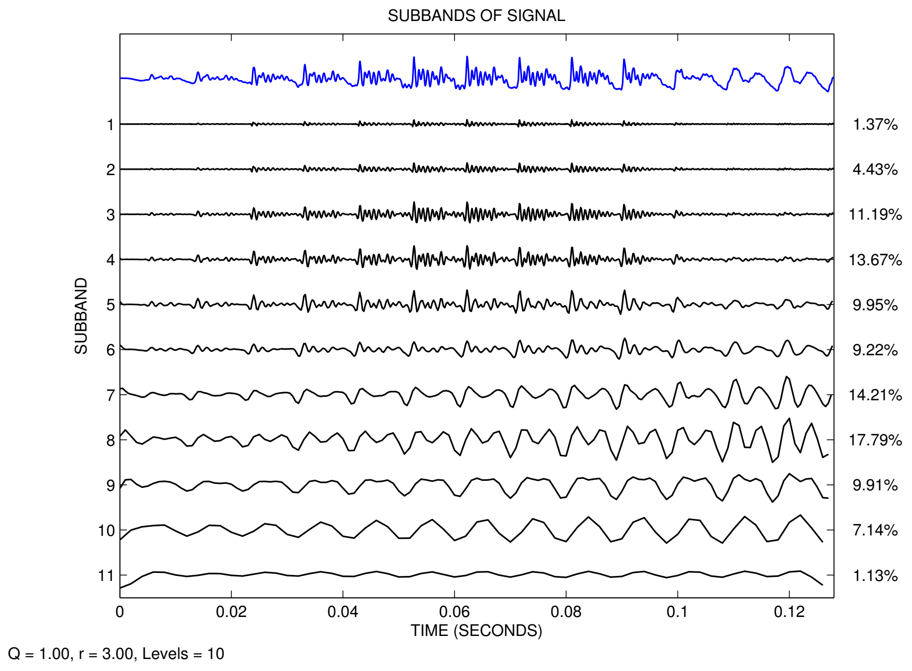


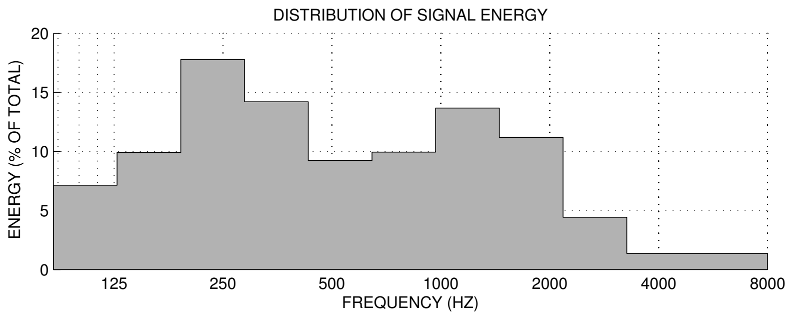
These are the same frequency responses as immediately above, but due to the log-frequency scale, the appearance is quite different. The constant bandwidth on the log-frequency scale is characteristic of constant-Q transforms.

Q-factor = 1.0

The following snippet applies the TQWT with a Q-factor of 1.0. The parameters are denoted by `Q2`, `r2`, and `J2` to distinguish these values from the parameter values used previously. The resulting set of wavelet coefficients is denoted by `w2`.

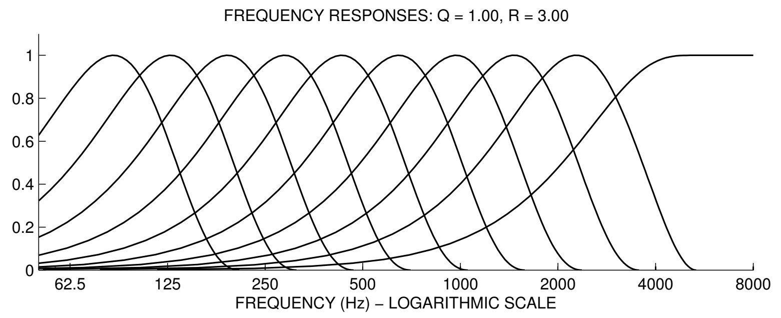
```
% Low Q-factor TQWT
Q2 = 1; r2 = 3; J2 = 10;          % TQWT parameters
w2 = tqwt_radix2(x, Q2, r2, J2);

% Plot energy distribution
figure(1), clf
subplot(2,1,1)
PlotEnergy(w2, Q2, r2, fs);

% Plot subbands
figure(2), clf
PlotSubbands(x, w2, Q2, r2, 1, J2+1, fs, 'E')
```





As above, the frequency responses can be displayed on a log-frequency axis. The sampling frequency `fs` is specified here, which is again reflected in the labeling of the frequency axis. Note that the frequency responses here, with $Q = 1.0$, are much wider than is the case when $Q = 4.0$.

```
PlotFreqResps(Q2, r2, J2, fs);
set(gca,'Xscale','log')
set(gca,'Xtick',fs/2*2.^(-7:0))
xlim([50 fs/2])
xlabel('FREQUENCY (Hz) - LOGARITHMIC SCALE')
```



The center frequency of each subband can be obtained using the function `tqwt_fc`. Recall that the first subband is the high-pass subband, so its center frequency is half the sampling frequency, `fs/2`. Here it is 8,000 Hz. Hence the frequencies listed by `tqwt_fc` are in descending order.

```
fc = tqwt_fc(Q2, r2, J2, fs)
% fc =
%  1.0e+03 *
%    8.0000    2.6667    1.7778    1.1852    0.7901    0.5267    0.3512    0.2341    0.1561    0.1040
```

5.5 Dual Q-factor signal decomposition

This section describes the sparse representation of a signal using two Q-factors simultaneously. This problem can be used for decomposing a signal into high and low 'resonance components' [4].

Consider the problem of writing a given signal $\mathbf{x}$ as the sum of an oscillatory signal $\mathbf{x}_1$ and a non-oscillatory signal $\mathbf{x}_2$,

$$\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2.$$

The signal $\mathbf{x}$ is a measured signal, and $\mathbf{x}_1$ and $\mathbf{x}_2$ are to be determined in such a way that $\mathbf{x}_1$ consists mostly of sustained oscillations and $\mathbf{x}_2$ consists mostly of non-oscillatory transients. As described in [4], such a decomposition is necessarily nonlinear in $\mathbf{x}$, and it can not be accomplished using frequency-based filtering. One approach is to model $\mathbf{x}_1$ and $\mathbf{x}_2$ as having sparse representations using high Q-factor and low Q-factor wavelet transforms respectively [4]. In this case, a sparse representation of the signal $\mathbf{x}$ using both high Q-factor and low Q-factor TQWT jointly, makes the identification of $\mathbf{x}_1$ and $\mathbf{x}_2$ feasible. This approach is based on 'morphological component analysis' (MCA) [6], a general method for signal decomposition based on sparse representations.

Denote TQWT_1 and TQWT_2 as the TQWT with two different Q-factors (high and low Q-factors). Then the sought decomposition can be achieved by solving the constrained optimization problem:

$$\begin{aligned} &\underset{\mathbf{w}_1,\mathbf{w}_2}{\text{argmin}}\ \lambda_1\|\mathbf{w}_1\|_1 + \lambda_2\|\mathbf{w}_2\|_1 \\ &\text{such that}\quad \mathbf{x} = \text{TQWT}_1^{-1}(\mathbf{w}_1) + \text{TQWT}_2^{-1}(\mathbf{w}_2). \end{aligned} \tag{3}$$

For greater flexibility, we will use subband-dependent regularization:

$$\begin{aligned} &\underset{\mathbf{w}_1,\mathbf{w}_2}{\text{argmin}}\ \sum_{j=1}^{J_1+1} \lambda_{1,j}\|\mathbf{w}_{1,j}\|_1 + \sum_{j=1}^{J_2+1} \lambda_{2,j}\|\mathbf{w}_{2,j}\|_1 \\ &\text{such that}\quad \mathbf{x} = \text{TQWT}_1^{-1}(\mathbf{w}_1) + \text{TQWT}_2^{-1}(\mathbf{w}_2), \end{aligned} \tag{4}$$

where $\mathbf{w}_{i,j}$ in (4) denotes subband j of TQWT_i for $i = 1, 2$.
After $\mathbf{w}_1$ and $\mathbf{w}_2$ are obtained, we set

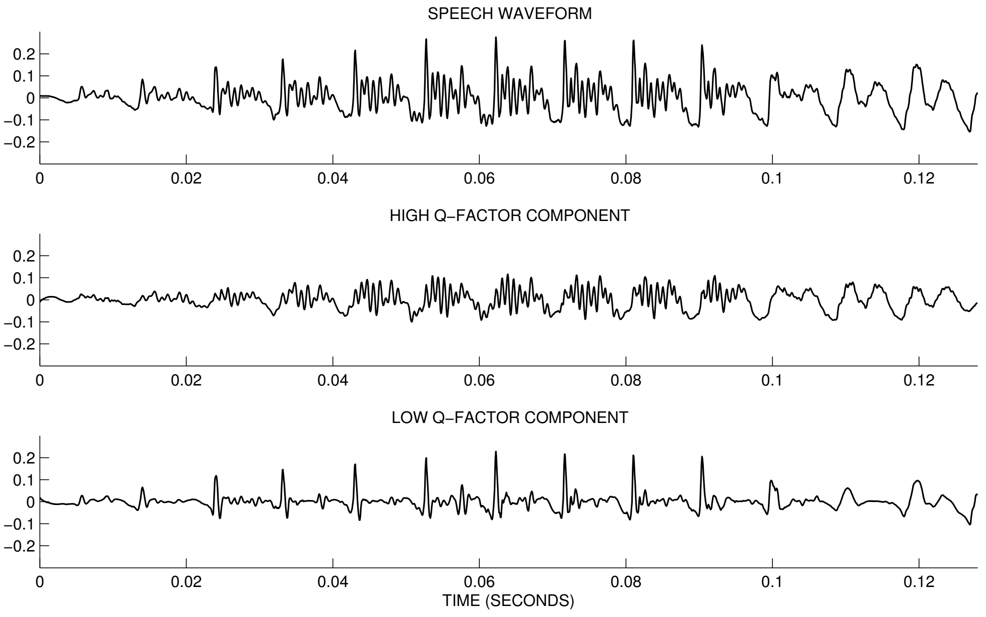
$$\mathbf{x}_1 = \mathrm{TQWT}_1^{-1}(\mathbf{w}_1), \qquad \mathbf{x}_2 = \mathrm{TQWT}_2^{-1}(\mathbf{w}_2).$$

The function `dualQ` solves this optimization problem, as illustrated in the following code snippet. Given the signal $\mathbf{x}$, the function returns signals $\mathbf{x}_1$ and $\mathbf{x}_2$. In addition, it returns sparse wavelet coefficients $\mathbf{w}_1$ and $\mathbf{w}_2$ corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively; the sparse wavelet coefficients are denoted `w1s` and `w2s` in the code.

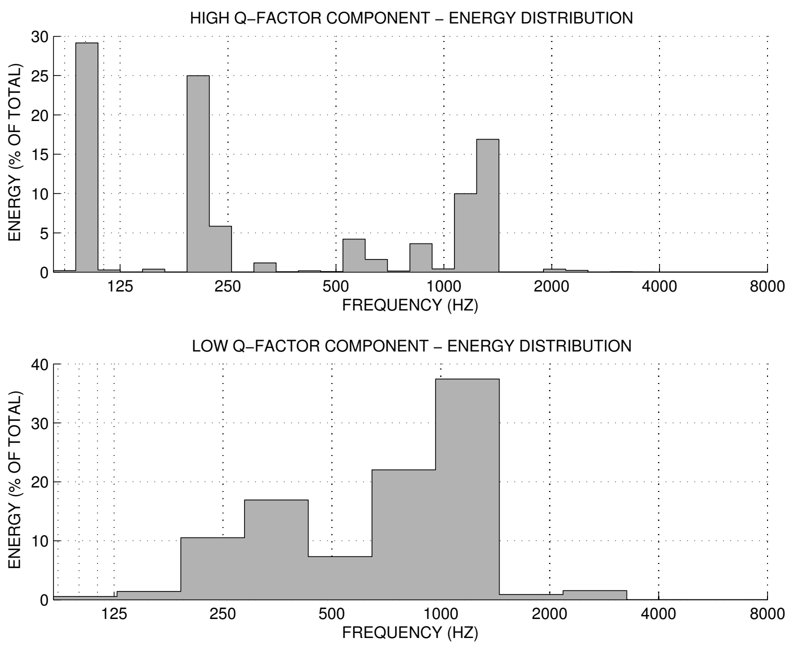
```
% Signal decomposition into high-Q and low-Q components
now1 = ComputeNow(N,Q1,r1,J1,'radix2');     % Compute norms of wavelets
now2 = ComputeNow(N,Q2,r2,J2,'radix2');     % Compute norms of wavelets
lam1 = now1;                                % Regularization params for high Q-factor TQWT
lam2 = now2;                                % Regularization params for low Q-factor TQWT
mu = 2.0;                                   % SALSA parameter
Nit = 100;                                  % Number of iterations
[x1,x2,w1s,w2s,costfn] = dualQ(x,Q1,r1,J1,Q2,r2,J2,lam1,lam2,mu,Nit);

figure(1), clf
subplot(4,1,1)
plot(t,x)
xlim([0 N/fs])
ylim([-0.3 0.3])
title('SPEECH WAVEFORM')
box off
subplot(4,1,2)
plot(t,x1)
xlim([0 N/fs])
ylim([-0.3 0.3])
box off
title('HIGH Q-FACTOR COMPONENT')
subplot(4,1,3)
plot(t,x2)
xlim([0 N/fs])
ylim([-0.3 0.3])
box off
title('LOW Q-FACTOR COMPONENT')
xlabel('TIME (SECONDS)')
```

As above, we set the subband-dependent regularization parameters proportional to the ℓ_2-norms of the wavelets.
It can be seen below that this procedure separates the given signal (the speech waveform) into two signals that have quite different behavior. One signal (the 'high Q-factor component') is sparsely represented by a high Q-factor wavelet transform ($Q = 4.0$). The second signal (the 'low Q-factor component') is sparsely

represented by a low Q-factor wavelet transform ($Q = 1.0$). Note that the high Q-factor component consists largly of sustained oscillatory behavior, while the low Q-factor component consists largely of transients and oscillations that are not sustained.



The distribution of energy across frequency, of both components, is illustrated below. It shows that the components overlap in frequency. Therefore, the decomposition achieved here can not be obtained by conventional frequency-based filtering or Fourier transforms alone. Also note that for both the high Q-factor and low Q-factor components shown here, the energy is more concentrated than in the previous plots, due to the sparsification.



To clarify the cost function that is being minimized, the following code snippet illustrates how the cost function is computed.

```
cost = 0;
for j = 1:J1+1
    cost = cost + lam1(j)*sum(abs(w1s{j}));
end
for j = 1:J2+1
    cost = cost + lam2(j)*sum(abs(w2s{j}));
end
cost
%>>   25.7048
```

We can also check that the two components really add to the input signal.

```
max(abs(x - x1 - x2))
%>>  2.9143e-16
```

In may be the case that the solution to the optimization problem (4) puts too much energy into one of the two components and too little energy into the other component. In that case the relative weight between the two components can be modified by introducing a parameter θ with $0 < \theta < 1$ into the optimization problem as so:

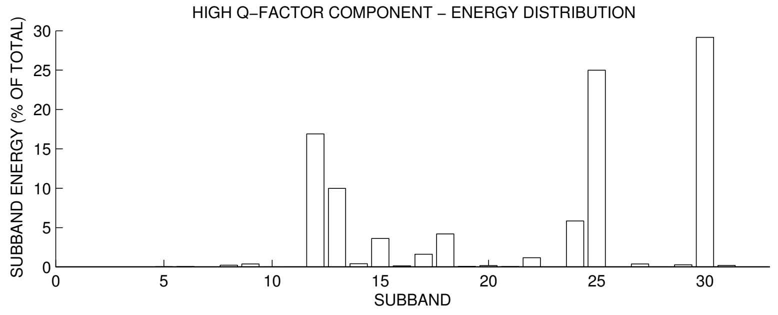
$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}_1,\mathbf{w}_2} \ \theta \sum_{j=1}^{J_1+1} \lambda_{1,j} \|\mathbf{w}_{1,j}\|_1 + (1-\theta) \sum_{j=1}^{J_2+1} \lambda_{2,j} \|\mathbf{w}_{2,j}\|_1 \\ &\text{such that} \quad \mathbf{x} = \mathrm{TQWT}_1^{-1}(\mathbf{w}_1) + \mathrm{TQWT}_2^{-1}(\mathbf{w}_2). \end{aligned} \tag{5}$$

For example, in the code snippet above, one may use instead:

```
  theta = 0.3;
  [x1,x2,w1s,w2s] = dualQ(x,Q1,r1,J1,Q2,r2,J2,theta*lam1,(1-theta)*lam2,mu,Nit);
```

which will put more emphasis on sparsifying the $\mathbf{x}_2$ component (because it is now penalized more than $\mathbf{x}_1$ in the cost function).

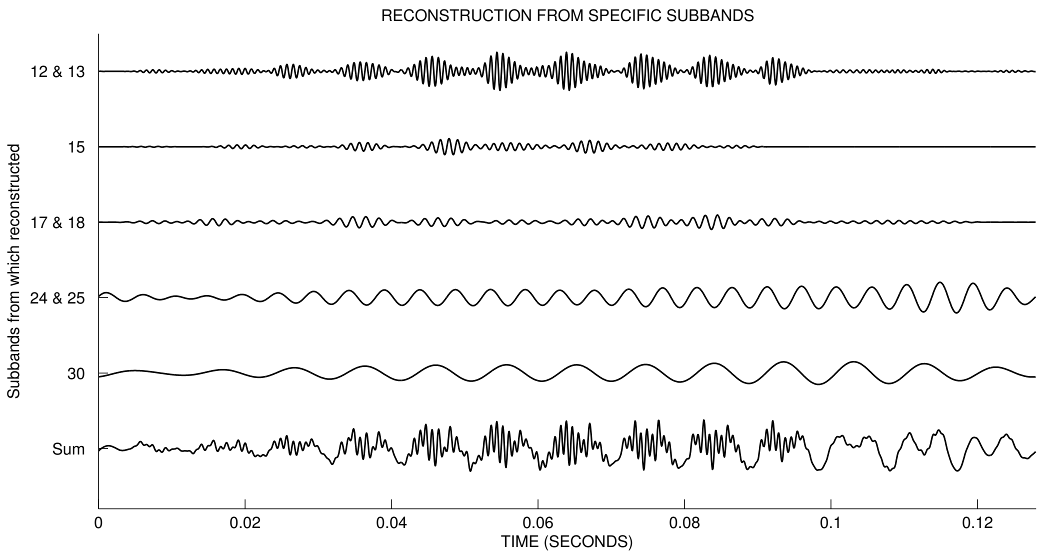
Frequency decomposition: In the plot above, showing the energy distribution for the high Q-factor component, note that the energy is well separated into distinct groups. Plotting the energy versus subband produces the following plot, in which we can identify the specific subbands containing most of the energy.



It can be seen in the plot that most of the energy lies among the five subbands groups: (12,13), (15), (17,18), (24,25), and (30). Therefore, the high Q-factor component can itself be quite accurately written as the sum of five distinct frequency components. To compute these frequency components, we can use the inverse TQWT to reconstruct from specific sets of subbands. The following code snippet illustrates the procedure: set all of the wavelet coefficient to zero, except for the wavelet coefficients in the specified subbands, then compute the inverse TQWT.

```
z = zeros(size(x));                         % All zero signal
w1z = tqwt_radix2(z,Q1,r1,J1);              % All zero wavelet coefficients

w = w1z;
w{12} = w1s{12};
w{13} = w1s{13};
xA = itqwt_radix2(w,Q1,r1,N);               % Reconstruction from subbands 12 and 13

w = w1z;
w{15} = w1s{15};
xB = itqwt_radix2(w,Q1,r1,N);               % Reconstruction from subband 15

w = w1z;
w{17} = w1s{17};
w{18} = w1s{18};
xC = itqwt_radix2(w,Q1,r1,N);               % Reconstruction from subbands 17 and 18

w = w1z;
w{24} = w1s{24};
w{25} = w1s{25};
xD = itqwt_radix2(w,Q1,r1,N);               % Reconstruction from subbands 24 and 25

w = w1z;
w{30} = w1s{30};
xE = itqwt_radix2(w,Q1,r1,N);               % Reconstruction from subband 30

f = xA + xB + xC + xD + xE;                 % Sum of reconstructed signals
```

The following figure illustrates the five reconstructed signals and their sum.



The sum of these five reconstructed signals is not exactly equal to the high Q-factor component because there remains some energy in the omitted subbands; however, as illustrated in the figure, it is quite accurate.

Note that the high frequency component (reconstructed from subbands 12 and 13) has finer temporal resolution than the low frequency components (reconstructed from subbands 24 and 25, or from subband 30). This is due to the constant-Q property of the wavelet transform. For this reason, constant-Q signal analysis is suitable for signals for which the envelope of the high frequency content varies over time more rapidly than the low frequency content, as is the case for the speech waveform here (in particular, its high Q-factor component).

Resonance components of a noisy signal

If the signal under analysis is noisy, then we should not asking for exact equality as in (4). Just as the basis pursuit denoising (BPD) problem adapts the basis pursuit (BP) problem to the noisy signal case, we can also apply resonance decomposition to noisy signals. Suppose $\mathbf{y}$ represents the noisy signal, then the problem can be formulated as the minimization of the following cost function:

$$\underset{\mathbf{w}_1,\mathbf{w}_2}{\operatorname{argmin}} \|\mathbf{y} - \Phi_1\mathbf{w}_1 - \Phi_2\mathbf{w}_2\|_2^2 + \sum_{j=1}^{J_1+1} \lambda_{1,j}\|\mathbf{w}_{1,j}\|_1 + \sum_{j=1}^{J_2+1} \lambda_{2,j}\|\mathbf{w}_{2,j}\|_1 \tag{6}$$

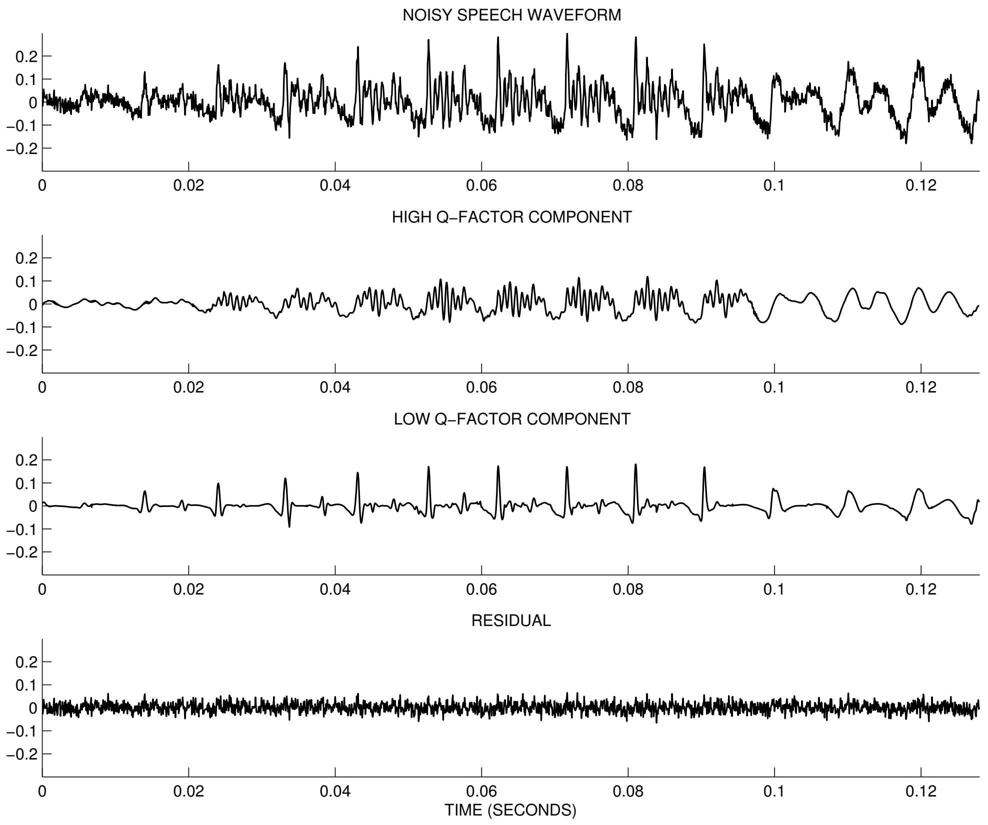
where Φ_1 and Φ_2 represent the inverse TQWT having high and low Q-factors respectively. The regularization parameters $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ are chosen by the user according the power of the noise. After $\mathbf{w}_1$ and $\mathbf{w}_2$ are obtained, we set

$$\mathbf{x}_1 = \text{TQWT}_1^{-1}(\mathbf{w}_1), \qquad \mathbf{x}_2 = \text{TQWT}_2^{-1}(\mathbf{w}_2).$$

The Matlab function `dualQd` addresses this problem, as illustrated in the following code snippet.

```
% Resonance decomposition of a signal with noise
[y, fs] = test_signal(3);            % Load speech waveform
N = length(y);
```

```
t = (0:N-1)/fs;
y = y + 0.02 * randn(1,N);          % Add white Gaussian noise

Q1 = 4; r1 = 3; J1 = 31;            % High Q-factor TQWT
Q2 = 1; r2 = 3; J2 = 10;            % Low Q-factor TQWT

now1 = ComputeNow(N,Q1,r1,J1,'radix2');      % Compute norms of wavelets
now2 = ComputeNow(N,Q2,r2,J2,'radix2');      % Compute norms of wavelets
lam1 = 0.1*now1;                             % Regularization params for high Q-factor TQWT
lam2 = 0.1*now2;                             % Regularization params for low Q-factor TQWT
mu = 2.0;                                    % SALSA parameter
Nit = 100;                                   % Number of iterations
[x1,x2,w1s,w2s,costfn] = dualQd(y,Q1,r1,J1,Q2,r2,J2,lam1,lam2,mu,Nit);

figure(1), clf
subplot(4,1,1)
plot(t,y)
xlim([0 N/fs])
ylim([-0.3 0.3])
title('NOISY SPEECH WAVEFORM')
box off
subplot(4,1,2)
plot(t,x1)
xlim([0 N/fs])
ylim([-0.3 0.3])
box off
title('HIGH Q-FACTOR COMPONENT')
subplot(4,1,3)
plot(t,x2)
xlim([0 N/fs])
ylim([-0.3 0.3])
box off
title('LOW Q-FACTOR COMPONENT')
subplot(4,1,4)
plot(t, y - x1 - x2)
xlim([0 N/fs])
ylim([-0.3 0.3])
box off
title('RESIDUAL')
xlabel('TIME (SECONDS)')
```



Note that the residual ($\mathbf{y} - \mathbf{x}_1 - \mathbf{x}_2$) appears noise-like. For the problem formulated in (6), the residual is the signal component that is sparsely represented in neither the high Q-factor wavelet transform nor the low Q-factor wavelet transform. For this reason, this formulation (6) accounts for the presence of noise in the signal of interest.

To clarify the cost function that `dualQd` minimizes, the following code snippet illustrates the computation of the cost function.

```
cost = sum(abs(y - x1 - x2).^2);
for j = 1:J1+1
    cost = cost + lam1(j)*sum(abs(w1s{j}));
end
for j = 1:J2+1
    cost = cost + lam2(j)*sum(abs(w2s{j}));
end
cost
% >>  2.7207
```

6 Some formulas

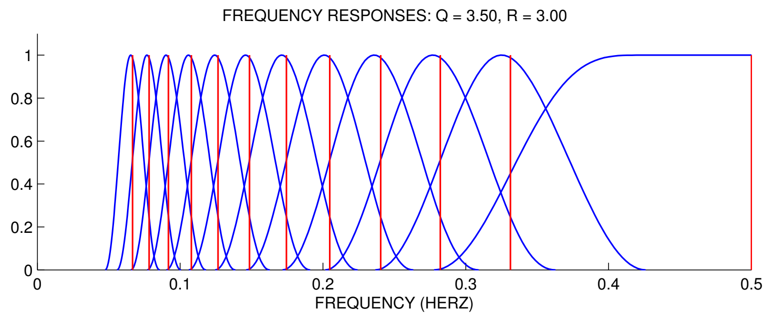
The center frequency of subband j is given by

$$f_c(j) \approx 0.25\,\alpha^{j-1}\,(2-\beta)\,f_s \tag{7}$$

for $j > 1$, where f_s is the sampling frequency of the input signal, and

$$\beta = \frac{2}{Q+1}, \quad \alpha = 1 - \frac{\beta}{r}. \tag{8}$$

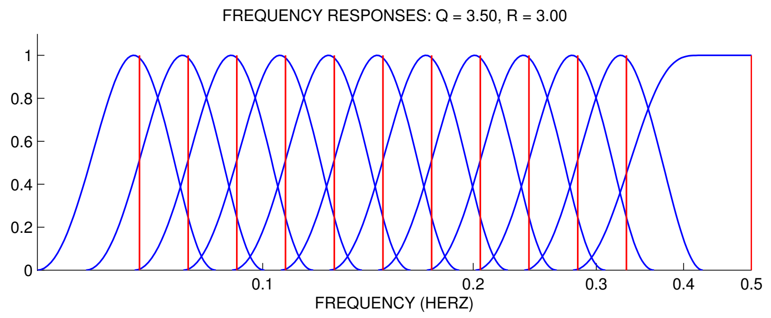
These formulas are from [5]. It should be noted that (7) is not accurate when Q is close to unity. For higher Q, such as $Q > 3$, the accuracy of (7) improves. The following figure shows the frequency responses together with the center frequencies calculated using (7).



This plot was produced using the following code. The Matlab command `tqwt_fc` calculates the center frequencies using (7).

```
Q = 3.5; r = 3; J = 12; fs = 1;
PlotFreqResps(Q, r, J, fs)
fc = tqwt_fc(Q, r, J, fs);                    % center frequencies
for j = 1:J
    line(fc(j)*[1 1], [0 1], 'color', 'r');   % indicate fc on plot
end
```

The frequency axis can be converted to a log scale using the command `set(gca, 'xscale', 'log')`. This gives the following figure.



7 Implementation in C/Mex

In order to reduce the execution time of the TQWT functions we have implemented[1] the main functions in C. In the C programs, we have attempted to reduce unnecessary computation and to optimize the memory usage so as to avoid unnecessary data shuffling. Using the *mex* utility in Matlab, we have compiled the C programs so that they can be called from Matlab. As shown in the tables and graphs, the Matlab mex/C implementations run substantially faster than the Matlab mfiles. The speed up depends on the length of the signal. For lengths up to 4092, the mex functions run more than 20 times faster than the mfiles. The timings were performed on a 2010 base-model Apple MacBook Pro (2.4 GHz Intel Core 2 Duo) with Matlab version 7.8 (R2009a).

The mex files are included in the toolbox in the folder `mex_functions`. They have the same functionality as the mfiles, except that they work only for real signals (the mfiles work for both real and complex signals). In order to use the mex files, the user can move the mex files into the main TQWT toolbox folder so that Matlab will execute the mex files instead of the mfiles (mex files have precedence over mfiles).

To determine which file Matlab will execute, use the `which` command. For example, `which tqwt_radix2` will determine if the mfile (.m) or mex file (.mex—) will be executed.

The toolbox currently provides mex files for the following systems.

Computer	Mex file extension	Matlab version
pcwin	mexw32	7.4 (R2007a)
pcwin64	mexw64	7.11 (R2010b) 64 bit
maci	mexmaci	7.8 (R2009a)
maci64	mexmaci64	7.12 (R2011a) 64 bit

Let us know if you would like mex files for another system. The Matlab command `computer` returns the operating system in use.

[1] Thanks to Faruk Uysal for implementing the functions in C and mex.

`tqwt_radix2` run-time (ms)			
N	mfile	mex	speed up
32	0.970	0.034	28.5
64	1.959	0.050	39.1
128	2.861	0.070	40.6
256	4.212	0.101	41.9
512	5.410	0.157	34.4
1024	7.013	0.256	27.4
2048	9.161	0.446	20.6
4096	11.805	0.836	14.1
8192	16.499	1.648	10.0
16384	25.207	3.577	7.0
32768	43.415	8.742	5.0

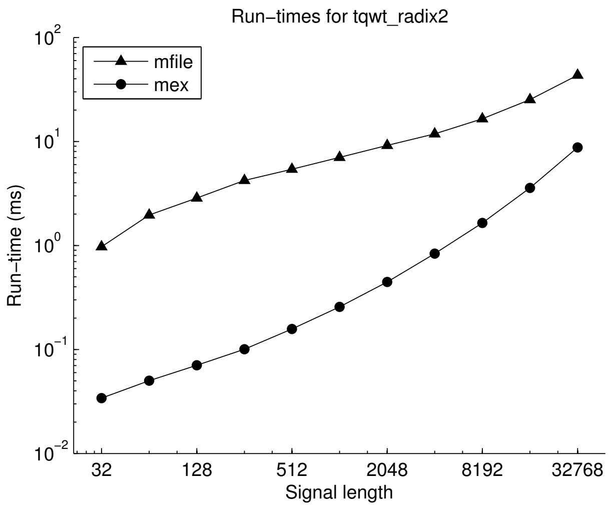


Figure 2: Run-times for forward TQWT (`tqwt_radix2`). Run-times performed using parameters $Q = 4.0$, $r = 3.0$, and the maximum number of levels for each signal length N.

`itqwt_radix2` run-time (ms)			
N	mfile	mex	speed up
32	1.069	0.034	31.2
64	1.933	0.047	41.4
128	3.050	0.062	49.6
256	4.081	0.089	45.9
512	5.588	0.142	39.4
1024	7.131	0.238	30.0
2048	9.320	0.429	21.7
4096	12.594	0.815	15.5
8192	18.023	1.612	11.2
16384	28.441	3.404	8.4
32768	53.592	7.405	7.2

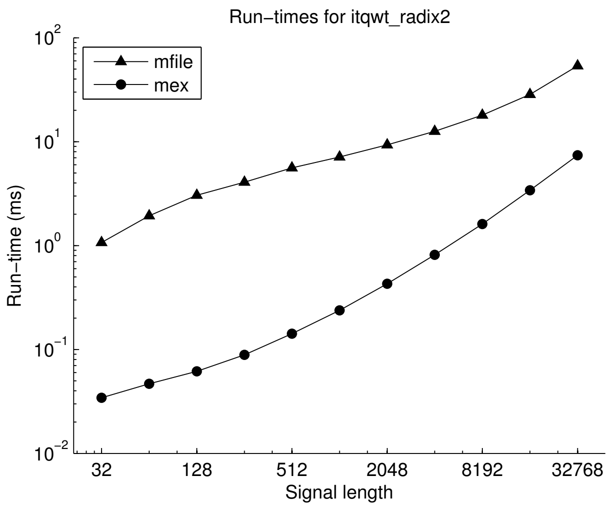


Figure 3: Run-times for inverse TQWT (`itqwt_radix2`). Run-times performed using parameters $Q = 4.0$, $r = 3.0$ and maximum levels.

`tqwt_bp` run-time (ms)			
N	mfile	mex	speed up
32	89.307	0.334	267.6
64	211.519	0.666	317.4
128	327.122	1.394	234.6
256	454.524	2.888	157.4
512	606.455	6.044	100.3
1024	784.550	12.161	64.5
2048	1014.531	24.724	41.0
4096	1356.525	50.108	27.1
8192	1939.807	105.199	18.4
16384	3031.670	219.502	13.8
32768	5869.573	475.247	12.4

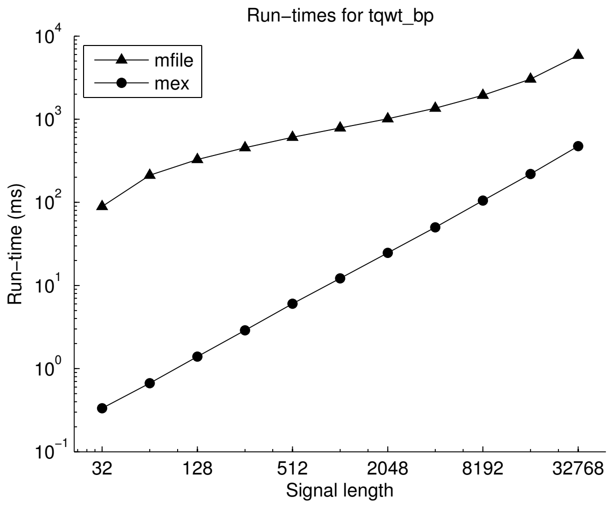


Figure 4: Run-times for sparse signal representation (basis pursuit) using the TQWT (`tqwt_bp`). Run-times performed using parameters $Q = 4.0$, $r = 3.0$, maximum levels, and 50 iterations.

`dualQ` run-time (ms)			
N	mfile	mex	speed up
32	178.020	0.705	252.5
64	328.269	1.340	245.0
128	463.114	2.693	172.0
256	634.967	5.401	117.6
512	824.188	10.854	75.9
1024	1021.563	22.205	46.0
2048	1337.682	44.323	30.2
4096	1871.902	92.403	20.3
8192	2742.564	190.621	14.4
16384	4565.536	409.887	11.1
32768	9299.440	936.065	9.9

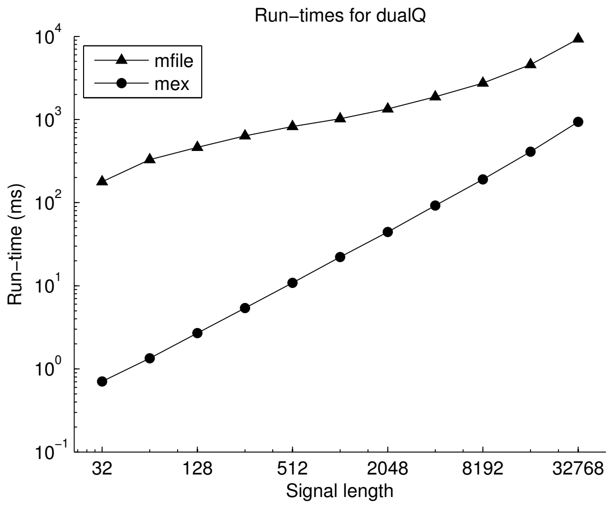


Figure 5: Run-times for dual Q-factor signal decomposition (`dualQ`). Run-times performed using parameters $Q_1 = 3.0$, $r_1 = 3.0$, $Q_2 = 1.0$, $r_2 = 3.0$, maximum levels, and 50 iterations.

References

[1] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo. Fast image recovery using variable splitting and constrained optimization. *IEEE Trans. Image Process.*, 19(9):2345 –2356, September 2010.

[2] I. Bayram and I. W. Selesnick. Frequency-domain design of overcomplete rational-dilation wavelet transforms. *IEEE Trans. Signal Process.*, 57(8):2957–2972, August 2009.

[3] S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1998.

[4] I. W. Selesnick. Resonance-based signal decomposition: A new sparsity-enabled signal analysis method. *Signal Processing*, 91(12):2793 – 2809, 2011.

[5] I. W. Selesnick. Wavelet transform with tunable Q-factor. *Signal Processing, IEEE Transactions on*, 59(8):3560–3575, August 2011.

[6] J.-L. Starck, M. Elad, and D. Donoho. Image decomposition via the combination of sparse representation and a variational approach. *IEEE Trans. Image Process.*, 14(10), 2005.