

Scalogram Modification and Inversion

A Wavelet Application in Automotive Industry ¹

Jörg Enders Weihua Geng Peijun Li

April 28, 2002

Abstract: In this paper, we refine a wavelet-based method used by the Ford Motor Company to analyze sounds such as clicks, rattles and taps. This method is based on translation-invariant scalograms computed from recorded signals. Those turn out to be more practical in applications than the standard scalograms obtained from the orthogonal wavelet transform.

We construct an inverse algorithm which computes a signal from a given scalogram. We prove that this algorithm leads to perfect reconstruction of the original signal. Removing features from a scalogram generally results in a scalogram not corresponding to any original waveform. However, if the inverse transform is applied to such a modified scalogram, the signal ζ obtained still reflects the salient characteristics of the modified scalogram. In particular, the differences between the scalogram of ζ and the modified scalogram are localized in time and frequency. We explain this by giving examples of waveforms and their scalograms, and then pointing out where the scalogram of ζ deviates from the scalogram in which certain parts have been deleted.

¹ Work done for the Physical and Environmental Sciences Department of the Ford Research Laboratories under the direction of Technical Specialist David Scholl, in partial fulfillment of the requirement of Michigan State University MTH844. The project is managed by Professor Michael Frazier.

Table of Contents

1. Introduction and Motivation	3
2. Mathematical Background	4
2.1 Mathematical Preliminaries and Notation.....	4
2.2 Standard Discrete Wavelet Transform (DWT) and Scalogram.....	6
2.3 Shift-Invariant Discrete Wavelet Transform (SIDWT) and Scalogram.....	9
2.4 Examples and Comparison.....	10
3. Scalogram Inversion.....	19
4. Scalogram Modification.....	26
4.1 Error Estimate	26
4.2 Examples	28
4.3 Alternative Inverses	37
5. Conclusions	41
6. Future Work	41
Acknowledgments.....	43
References	44
Appendix: MATLAB code.....	45

1. Introduction and Motivation

In his paper on “Translation-Invariant Data Visualization with Orthogonal Discrete Wavelets” [1] David Scholl describes an algorithm yielding scalograms that can be used particularly well to visualize automobile-rattle sounds. This is important in the automotive industry since the sounds recorded inside a moving automobile should be minimized. Therefore it is important to have detailed information about the frequencies appearing as well as a good time-resolution, so that we can find out where and when different rattle sounds occur. Since judging the energy and frequency level of a specific sound by human hearing is not only manpower consuming but also clearly subjective and hence not always accurate, it is necessary to get an objective picture of the frequency content of a recorded signal. Scalograms provide a practical time-frequency analysis. Having computed a scalogram, we can easily look for sounds having short time duration and high frequencies, which might however range over many octaves. Such patterns are typical for any kind of rattle in an automobile. The main advantage of scalograms over spectrograms computed via a windowed Fourier Transformation is the good time resolution. When looking at certain rattle noises, which are spread over several octaves anyways, we do not need a fine frequency resolution, whereas when looking for repeated patterns it is important to know exactly when a rattle-sound appears.

The algorithm given in [1] is the Shift-Invariant Discrete Wavelet Transform (SIDWT). Given a signal z it computes a scalogram, which is a graph with time on the horizontal axis and different octaves (discrete logarithmic frequency scale) on the vertical axis. The energy in a certain octave at a given time is represented by the color of the corresponding block. Instead of a grayscale used in former times, we use a scale from 'blue' (low energy) to 'red' (high energy). If the scalogram has been computed, it can be analyzed for patterns as described above that might be 'responsible' for a certain noise that should not occur in an automobile. Once we are sure that at a known time a rattle of a certain frequency appears, we could look for the part in the vehicle with the given eigenfrequency and fix the problem.

Thus the main goal is to be able to check whether a high-energy pattern occurring in the scalogram of a sound sample is responsible for a certain type of rattling noticeable by our ears. To verify this however, we need to be able to remove the high-energy coefficients of interest and then transform the modified scalogram back to a signal ζ . Then we can listen to the new waveform in the hope that the unwanted noise is removed or at least reduced.

In this paper we give an inverse algorithm that makes it possible to compute the signal ζ from a changed scalogram. This can then be applied to make sure that we have found the pattern in a scalogram corresponding to a specific rattle-sound. Before we start getting into the theory of the transform, an example of a signal z and its corresponding scalogram are shown in Figure 1.

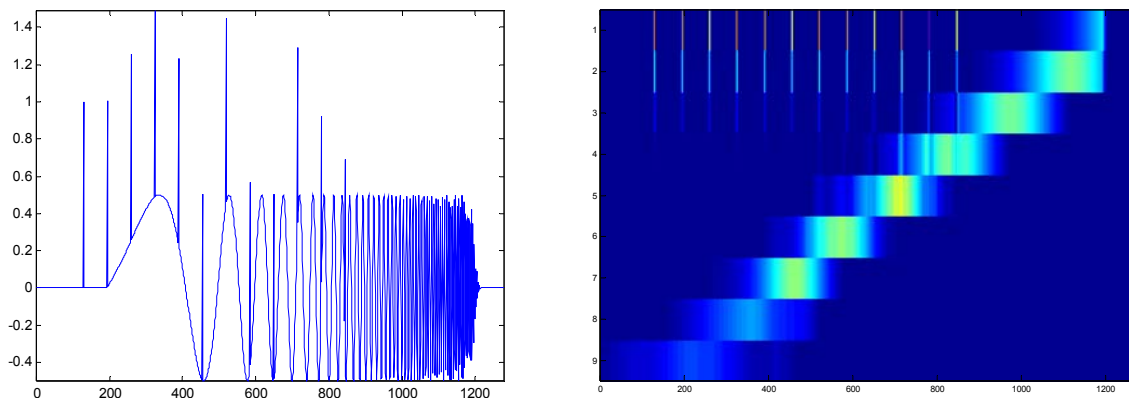


Figure 1. An example of a signal and its corresponding scalogram.

2. Mathematical Background

2.1 Mathematical Preliminaries and Notation

The mathematical theory of wavelets is less than 15 years old, yet already wavelets have become a fundamental tool in many areas of applied mathematics and engineering. Before we explain the algorithms used to compute the scalograms, we set the following notations:

In this paper, we work with discrete signals $z \in l^2(Z_N) = \{z = \{z_0, \dots, z_{N-1}\} : \|z\|_2 < \infty\}$, where $Z_N = \{0, 1, \dots, N-1\}$. We say a vector $z \in l^2(Z_N)$ is localized in space near n_0 if most of the components $z(n)$ of z are 0 or at least relatively small, except for a few values of n close to n_0 . A spatially localized basis $B = \{v_0, v_1, \dots, v_{N-1}\}$ of vectors $v_i \in l^2(Z_N)$ is useful because it provides a local analysis of a given signal. At the same time, we would like our basis to be frequency localized. By this we mean that the Discrete Fourier Transforms (DFTs) of our basis vectors should be negligibly small except near one particular region. Thus, our ultimate goal is to obtain a basis, whose elements are both spatially and frequency localized. Then a vector's expansion coefficients in this basis will provide both time and frequency information. Hence, we would obtain a simultaneous space and frequency analysis of this vector. Wavelets provide such a basis ([3]) and therefore yield the scalograms that are practical in applications.

To simplify notation, we will use operator notation to represent several frequently used terms. In the following we introduce the most often used notations and operators:

Consider the signal $z = \{z(0), \dots, z(N-1)\}$ representing N spaced time samples. Then:

\hat{z} is the Discrete Fourier transform of z .

\bar{z} is the complex conjugate of z taken by component.

\tilde{z} is the conjugate reflection of z defined by $\tilde{z}(n) = \overline{z(-n)} = \overline{z(N-n)}$ for all n .

Note that $\tilde{z} = (\hat{z})^\vee$.

Let n be an integer such that $0 < n < N$. Let m be an integer factor of N . To **left shift** a signal by n places using periodic boundary conditions we define

$$S_n(z) = \{z(n), z(n+1), \dots, z(N-1), z(0), z(1), \dots, z(n-1)\}.$$

Dividing z into consecutive groups of m samples, we define the **downsampling operator**

$$D_m(z) = \{z(0), z(m), z(2m), \dots, z(N-m)\}$$

as the operator that selects the first entry of each group. An **upsampling operator** increases the total number of samples by a factor of n by inserting $n-1$ additional copies of every sample from z and dividing every element by n :

$$U_n(z) = \frac{1}{n} \{y(0), y(1), \dots, y(nN-1)\}$$

where $y(j) = z(i)$, $i \cdot n \leq j \leq (i+1)n-1$. A **reordering** by m is given by

$$R_m(z) = \{z(0), z(m), \dots, z(N-m), z(1), z(m+1), \dots, z(N-m+1), \dots, z(m-1), z(2m-1), \dots, z(N-1)\}.$$

Sometimes we will use R_{-m} to denote the inverse operator of R_m , i.e., $R_{-m} \cdot R_m = I$, where I is

the identity operator. In fact $R_{-m} = R_{\frac{N}{m}}$.

When plotting the scalograms, we will use different colors for different energies. The **energy** of the signal z is defined as

$$E(z) = \{z(0)^2, z(1)^2, \dots, z(N-1)^2\}.$$

However, in fact we will use the analytic signal energy that is often used in signal processing. It returns a smoother scalogram [1] and uses the discrete Hilbert transform z' of z :

$$E(z) = \frac{1}{2} \{z(0)^2 + z'(0)^2, z(1)^2 + z'(1)^2, \dots, z(N-1)^2 + z'(N-1)^2\}.$$

2.2 Standard Discrete Wavelet Transform (DWT) and Scalogram

The advantage of the wavelet transform lies not only in its ability to provide a basis whose elements are both spatially and frequency localized, but also in the fact that it can be computed by a fast algorithm. In the following we give a brief description of the standard discrete wavelet Transform (DWT), to clarify how the advantages are guaranteed.

Suppose $w \in l^2(Z_N)$ is such that the set $B = \{T_k w\}_{k=0}^{N-1}$ of all translates by k is an orthonormal basis for $l^2(Z_N)$. Then the coefficients of the expansion of a vector $z \in l^2(Z_N)$ are the inner products $\langle z, T_k w \rangle$. According to the connection between inner product and convolution $\langle z, T_k w \rangle = z * \tilde{w}$ we see that the inner products can be calculated rapidly by the FFT: $z * \tilde{w} = (\hat{z} \cdot \hat{\tilde{w}})^\vee$. However, as shown in [3], $B = \{T_k \omega\}_{k=0}^{N-1}$ being an orthonormal basis is equivalent to $|\hat{w}(n)| = 1$ for all $n \in Z_N$, which means we cannot obtain a frequency localized orthonormal basis of the form $B = \{T_k w\}_{k=0}^{N-1}$. Therefore, instead of looking for one vector w whose full set of translates form an orthonormal basis, we look for two vectors, u and v , such that the set of their translates by even integers forms an orthonormal basis. The following examples of the real Shannon basis, the Daubechies D6 wavelet basis and D4 wavelet basis are orthonormal bases of this kind.

In this paper we will mainly work with Daubechies (D4) wavelets on Z_N . In addition, in section 2.4 we also give the scalograms computed using the real Shannon wavelets and Daubechies D6 wavelets. The Shannon wavelets are to our knowledge not used in applications. The reason is mainly the fact that although they are indeed localized in frequency, their spatial localization is not sufficient in most applications. In contrast to that the D4 and D6 wavelets form a well-localized basis of $l^2(Z_N)$.

Now let us give the so-called filters u and v (their DFTs respectively) that are used to compute the standard wavelet transform and the related scalogram [3]:

- First-stage real Shannon basis (N divisible by 4):

$$\hat{u}(n) = \begin{cases} \sqrt{2}, & n = 0, 1, \dots, \frac{N}{4} - 1, n = \frac{3N}{4} + 1, \dots, N - 1 \\ i, & n = \frac{N}{4} \\ -i, & n = \frac{3N}{4} \\ 0, & n = \frac{N}{4} + 1, \dots, \frac{3N}{4} - 1 \end{cases}$$

$$\hat{v}(n) = \begin{cases} 0, & n = 0, 1, \dots, \frac{N}{4} - 1, n = \frac{3N}{4} + 1, \dots, N - 1 \\ 1, & n = \frac{N}{4}, n = \frac{3N}{4} \\ \sqrt{2}, & n = \frac{N}{4} + 1, \dots, \frac{3N}{4} - 1. \end{cases}$$

- Daubechies D6 wavelets: Let $a = 1 - \sqrt{10}, b = 1 + \sqrt{10}, c = \sqrt{2 + 2\sqrt{10}}$.

Then

$$u = \frac{\sqrt{2}}{32} (b + c, 2a + 3b + 3c, 6a + 4b + 2c, 6a + 4b - 2c, 2a + 3b - 3c, b - c, 0, \dots, 0)$$

$$v = (-u(1), u(0), 0, \dots, 0, -u(5), u(4), -u(3), u(2)).$$

- Daubechies D4 wavelets:

$$u = \frac{\sqrt{2}}{8} (1 + \sqrt{3}, 3 + \sqrt{3}, 3 - \sqrt{3}, 1 - \sqrt{3}, 0, \dots, 0)$$

$$v = \frac{\sqrt{2}}{8} (-3 - \sqrt{3}, 1 + \sqrt{3}, 0, \dots, 0, -1 + \sqrt{3}, 3 - \sqrt{3}).$$

The central statement about when this set of the even translates of u and v forms a basis of $l^2(Z_N)$ is given by the following lemma from [3]:

Lemma 1.

Suppose $M \in \mathbb{N}, N = 2M$, and $u, v \in l^2(Z_N)$. For $n \in Z$, define $A(n)$, the system matrix of u and v , by

$$A(n) = \frac{1}{\sqrt{2}} \begin{bmatrix} \hat{u}(n) & \hat{v}(n) \\ \hat{u}(n+M) & \hat{v}(n+M) \end{bmatrix}.$$

Then

$$\begin{aligned} B &= \{T_{2^k} v\}_{k=0}^{M-1} \cup \{T_{2^k} u\}_{k=0}^{M-1} \\ &= \{v, T_2 v, T_4 v, \dots, T_{N-2} v, u, T_2 u, T_4 u, \dots, T_{N-2} u\} \end{aligned}$$

is an orthonormal basis for $l^2(Z_N)$ if and only if the system matrix $A(n)$ of u and v is unitary for each $n = 0, 1, \dots, M-1$.

Equivalently, B is a first-stage wavelet basis for $l^2(Z_N)$ if and only if

$$\begin{aligned} |\hat{u}(n)|^2 + |\hat{v}(n)|^2 &= 2, \\ |\hat{u}(n+M)|^2 + |\hat{v}(n+M)|^2 &= 2, \\ \hat{u}(n) \cdot \bar{\hat{v}}(n) + \hat{u}(n+M) \cdot \bar{\hat{v}}(n+M) &= 0 \end{aligned}$$

for all $n = 0, 1, \dots, M-1$.

So far we have looked at the so-called first-stage orthonormal wavelet basis of $l^2(Z_N)$ in the above-mentioned three examples. It is of the form

$$B = \{T_{2^k} v\}_{k=0}^{M-1} \cup \{T_{2^k} u\}_{k=0}^{M-1}.$$

Based on this we can compute the coefficients of a signal in the new basis. This change of basis can be computed as described in filter bank schemes using a sequence of convolution- and downsampling-operations (see Figure 2). Note that this change of basis means that we are mapping an n -dimensional vector (signal z) to another n -dimensional vector that contains more information about frequency. Since this mapping is a linear change of basis, there is an inverse transform taking a vector represented in the first-stage orthonormal wavelet basis to the unique corresponding signal. We get this change from a sequence of upsampling- and convolution-operations. Moreover, because each of those transforms is accomplished by a pair of convolutions, which can be done via FFT, the change of basis can be computed quickly.

As shown in [3], u and v are originally chosen to split the frequency scale in a lower and upper half. From music we know that it is more natural to consider frequencies on a logarithmic scale, in octaves. This suggests that we should leave the terms carrying the top half of the frequencies, but that we should subdivide the frequencies in the lower half into two equal parts again, the lowest quarter of frequencies and the next quarter. Then we can split the lowest quarter

again, and so on. In this way our basis decomposition could give us a more refined frequency analysis of a signal, where the number of basis vectors is decreasing with the frequency. In other words, if we split z into $P(z)$ and $Q(z)$, where $P(z)$ are the terms coming from the translates of u and $Q(z)$ the coefficients with respect to the translates of v , both vectors have length $N/2$. If we go further, by splitting $P(z)$ again the length of the vectors is halved in every step. This motivates us to perform a process of iteration, in other words, apply the wavelet transform on the output again and again in order to get more and more detailed information on the energy of the lower frequencies. However as mentioned we lose spatial information in the lower octaves. In order for the scalogram to have the same size in each row (=octave) the algorithm in Figure 3 uses the upsampling operator to insert an additional copy of every sample. In the same way as in the analysis phase (decomposition of the signal), the iteration of the inverse transform perfectly reconstructs the signal. Figure 2 shows the iteration process.

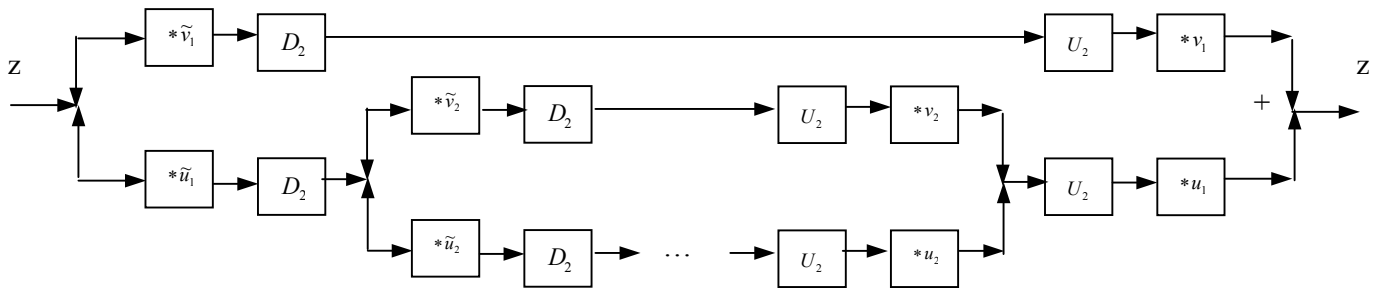


Figure 2. The iteration process: analysis and synthesis of a signal.

2.3 Shift-Invariant Discrete Wavelet Transform (SIDWT) and Scalogram

The class of shift invariant systems is characterized by the property that if $y(n)$ is the response to $x(n)$, then $y(n-k)$ is the response to $x(n-k)$ for this system, where k is a positive or negative integer. In other words, the system does not behave differently at different time. Mathematically, if we let $L:l^2(Z_N) \rightarrow l^2(Z_N)$ be the linear transformation the system represents, then the fact that L is translation invariant is equivalent to the fact that L is a convolution operator or the matrix A representing L in the standard Euclidean basis is circulant (see [3]).

The above-mentioned standard discrete wavelets have compact support in time; therefore, edge effects and other localized features have only a local influence. However, this character

limits their application to data visualization and causes the problem that time-shifted copies of identical signals have different scalograms. Therefore it is necessary for us to maintain the shift invariance by keeping redundant information and thus sacrificing the orthogonality property. A conventional orthogonal discrete wavelet transform can be made exactly shift invariant by computing the wavelet coefficients for all possible circularly shifted copies of the input waveform.

The algorithm for the shift-invariant wavelet transform is similar to that for the shift-variant transform but with reordering operators taking the place of downsampling operators. The additional coefficients that result from this change are causing the redundancy and thus the shift-invariance. In section 2.4, we will provide the detailed comparison of the algorithms of the shift-variant transform and shift-invariant transform. In later experiments we also show the advantages of the shift-invariant wavelet transform in data visualization.

2.4 Examples and Comparison

We give the two algorithms of the DWT and the SIDWT used to compute the scalograms, then compare them by looking at an example giving both the shift-variant and the shift-invariant scalograms using the three different wavelets Shannon, D4 and D6.

<u>Shift-variant algorithm</u>	<u>Shift-invariant algorithm</u>
For an input z , define	For an input z , define
$x_1 = D_2(z * \tilde{v}_1)$	$x_1 = R_2(z * \tilde{v}_1)$
$y_1 = D_2(z * \tilde{u}_1)$	$y_1 = R_2(z * \tilde{u}_1)$
Define $x_2, y_2, \dots, x_p, y_p$ inductively	Define $x_2, y_2, \dots, x_p, y_p$ inductively
$x_l = D_2(y_{l-1} * \tilde{v}_l)$	$x_l = R_2(y_{l-1} * \tilde{v}_l)$
$y_l = D_2(y_{l-1} * \tilde{u}_l)$	$y_l = R_2(y_{l-1} * \tilde{u}_l)$
For $n \in \{1, \dots, p\}$ the n^{th} row of the shift-variant scalogram matrix M is	For $n \in \{1, \dots, p\}$ the n^{th} row of the shift-invariant scalogram matrix M is
$M_n = U_b E(x_n)$	$M_n = ES_a R_b(2^{-(n/2)} x_n)$
where $b = 2^{n-1}$.	where $b = N / 2^n$ and $a = 2^{n-1} - 1$.
Thus M is an $(p \times N)$ - matrix.	Thus M is an $(p \times N)$ - matrix.

Table 1. Comparison of the shift-variant and shift-invariant algorithms.

We gave the so-called shift-invariant algorithm. The following simple sine-waveform illustrates the difference to the shift-variant algorithm and hence makes clear that it is important to use the translation-invariant version when looking for features in a scalogram. Figure 3 contains a sine-waveform with its shift-invariant scalogram (D4).

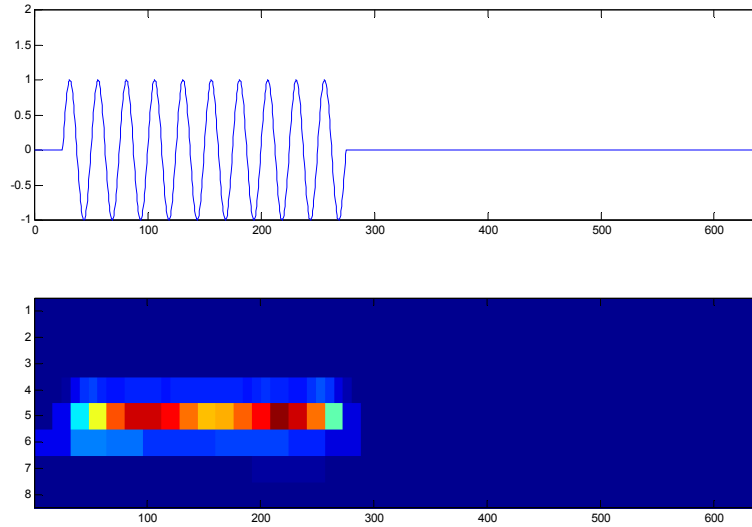


Figure 3. Sine-waveform and its translation-variant scalogram before the right-shift.

Now in Figure 4 we look at the same signal and its shift-variant scalogram again after it has been shifted to the right by 400 steps, i.e. it appears later in time.

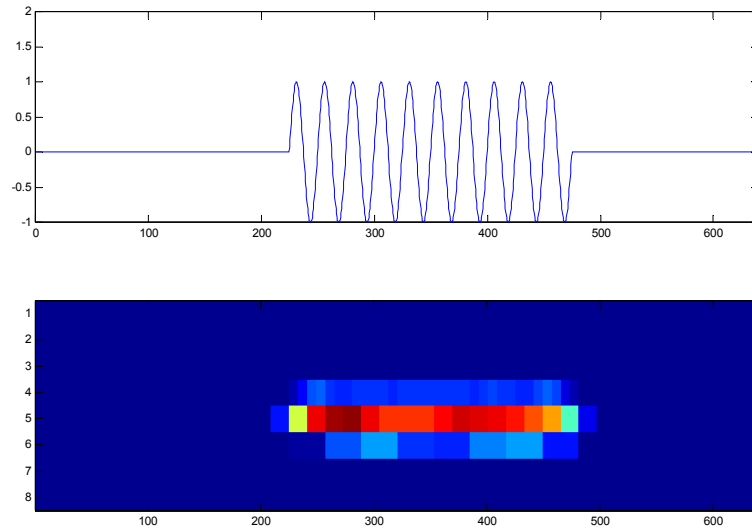


Figure 4. Sine-waveform and its translation-variant scalogram after the right-shift.

As we can see, the scalograms are not exactly the same. Besides the shift in time, we notice that also the coefficients (colors) change. This is exactly what shift-variance means. As a consequence a comparison of the scalograms is possible only in a restricted way. It is still clear to us that the two scalograms look similar, however it might be impossible for a computer to find the similarity. This change in the scalogram caused by the time-shift of a signal is what the SIDWT prevents. Figures 5 and 6 are the corresponding graphs using the shift-invariant algorithm however. We see that the scalograms look exactly the same.

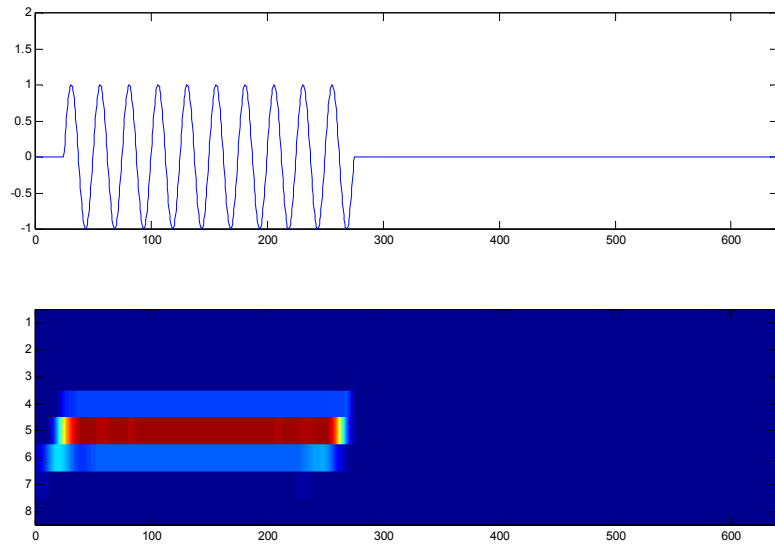


Figure 5. Sine-waveform and its translation-invariant scalogram before the right-shift.

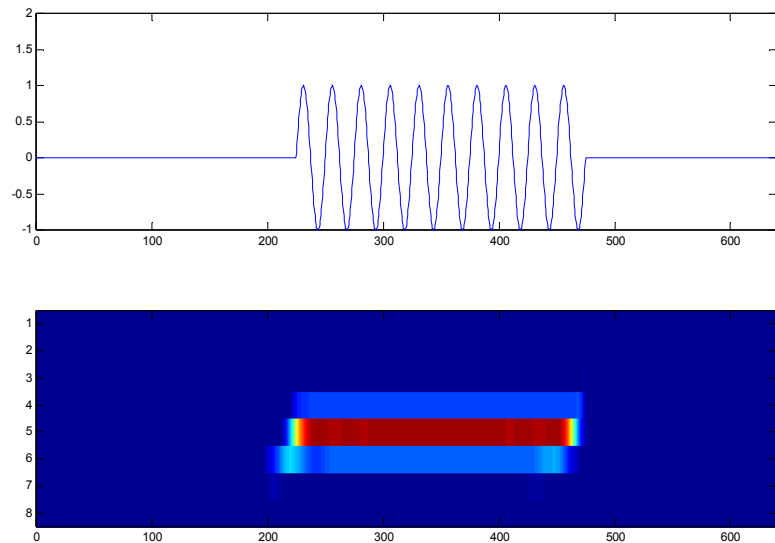


Figure 6. Sine-waveform and its translation-invariant scalogram after the right-shift.

In the following, we look at a very similar waveform as in [1], since it is a very telling example. Figure 7 shows the 1280-sample test waveform composed of a windowed exponential chirp with 12 added transients. The transients are added to the chirp waveform as the series $z(i) = \delta_{i,j}$, where $i = \{1,2,\dots,1280\}$ and $j = \{130,130 + 65,130 + 2 \times 65,\dots,130 + 11 \times 65\}$.

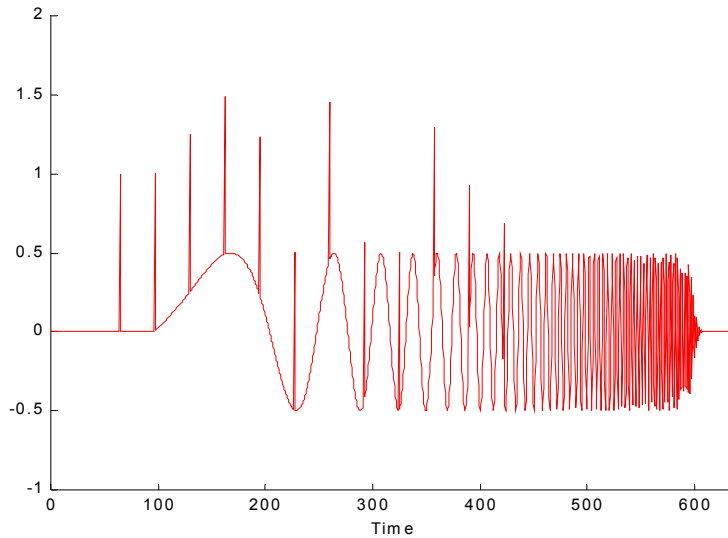


Figure 7. The 1280-sample test waveform composed of a windowed exponential chirp with 12 added transients.

In Figure 8 we computed the spectrogram of this waveform. It was produced with 64-point Hanning window zero-padded to 256 points and shifted by four examples for each time step. We constructed the test waveform so that the frequency increases exponentially. This can be seen from the spectrogram.

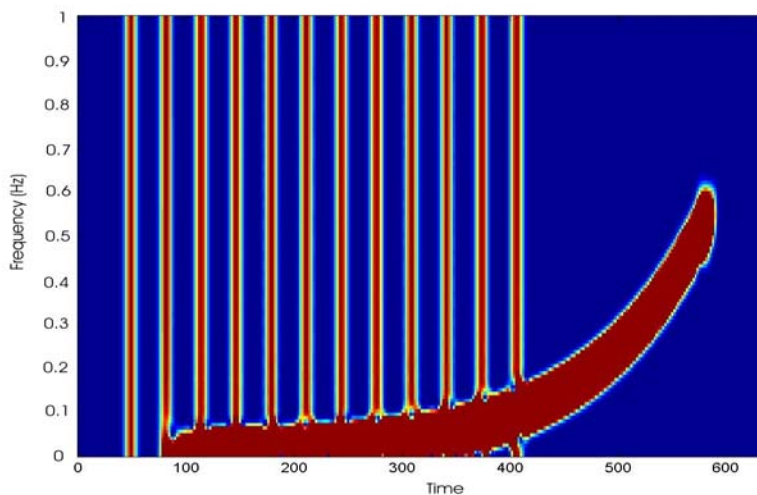


Figure 8. The spectrogram of the test waveform.

Now we compare the shift-variant scalograms in Figures 9(j), 11(j) and 13(j) with the corresponding shift-invariant scalograms in Figures 10(j), 12(j) and 14(j). Shift-invariant scalograms are clearly better for data visualization than shift-variant scalograms, both for the transients and the chirp. In the shift-variant scalograms (j) in Figures 9, 11 and 13 each identical but shifted copy of the transients produces a different pattern on the scalograms. In addition, the smooth sinusoidal chirp waveform looks noisy and irregular. In contrast to that, Figures (j) in 10, 12 and 14 show the shifted transients as identical patterns and the chirp as a smoothly varying series of horizontal bands.

As far as the shift-invariant scalograms in Figures 12 and 14 using D4 and D6 and the shift-invariant scalogram Figure 10 using Shannon wavelets are concerned, D4 and D6 yield a better result, because D4 and D6 produce more smoothly varying series of horizontal bands than Shannon does. This is not a surprise since we know that Daubechies wavelets have better frequency and spatial localization. Also, the difference between D4 and D6 is not noticeable. The two wavelets differ primarily in the fact that the number of non-zero entries is 4 and 6, respectively. This difference cannot be seen in the scalograms, which is to be expected.

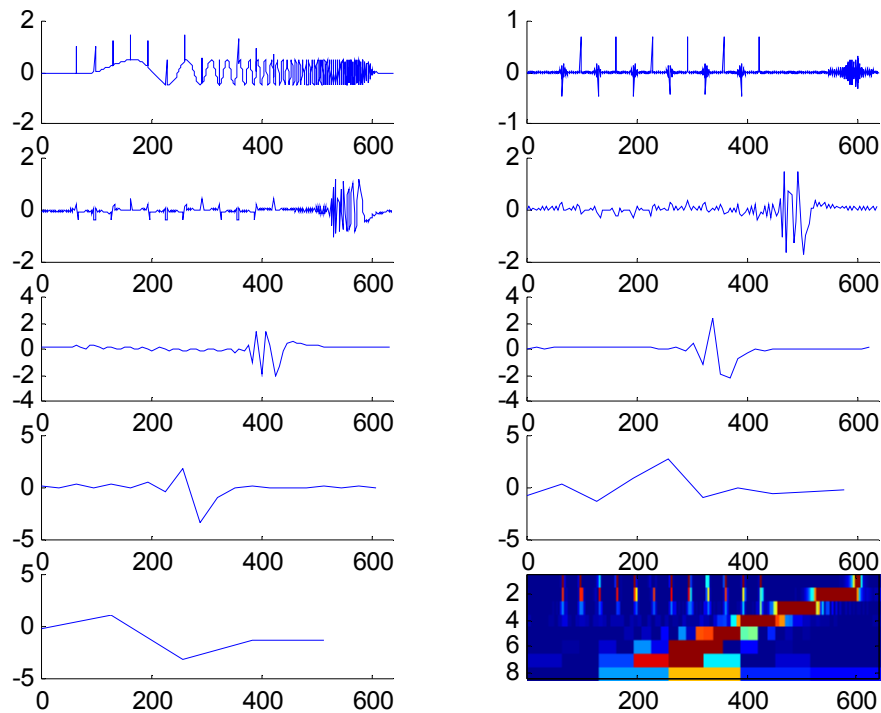


Figure 9. Shift-variant scalogram by using Shannon wavelets. (first row: (a) and (b), second: (c) and (d), ...)

(a) Sampled waveform composed of an exponential chirp with 12 identical transients (b) first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-variant scalogram.

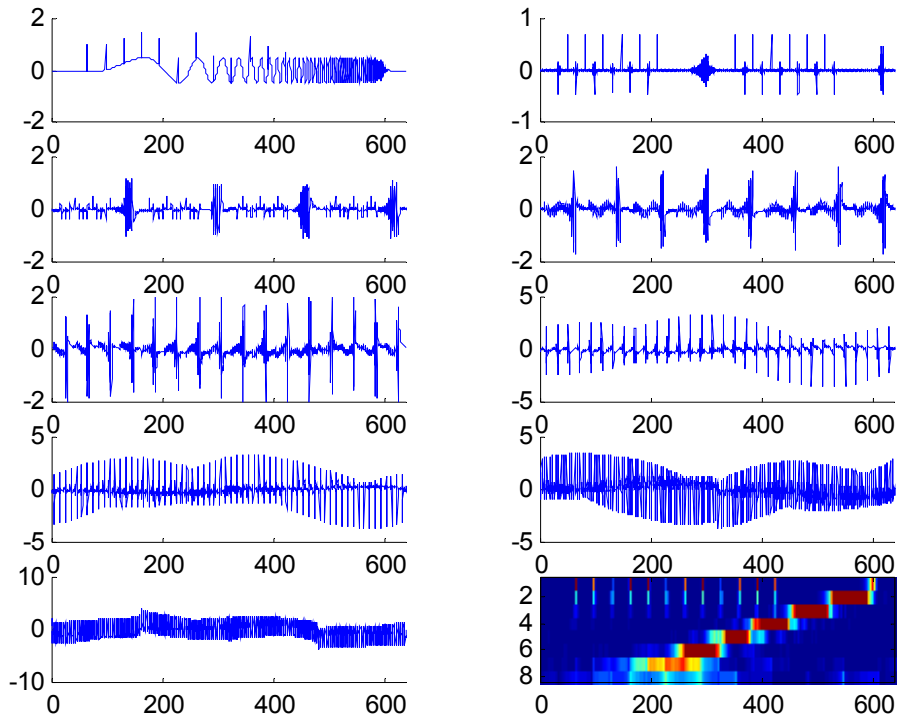


Figure 10. Shift-invariant scalogram by using Shannon wavelets.

(a) Sampled waveform composed of an exponential chirp with 12 identical transients (b) the first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-invariant scalogram.

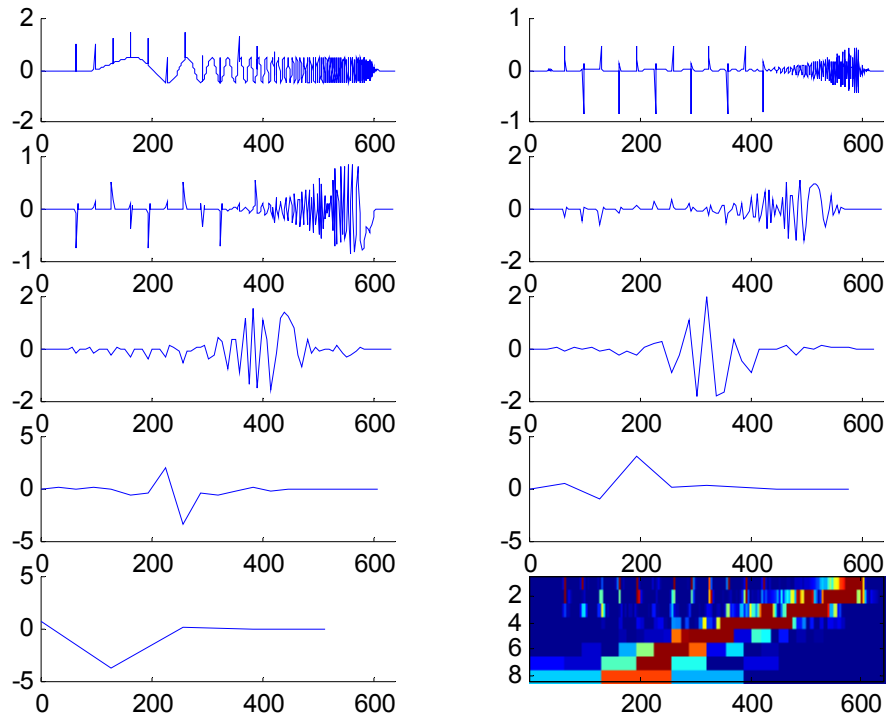


Figure 11. Shift-variant scalogram by using D4 wavelets.

- (a) Sampled waveform composed of an chirp with 12 transients (b) the first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet of coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-variant scalogram.

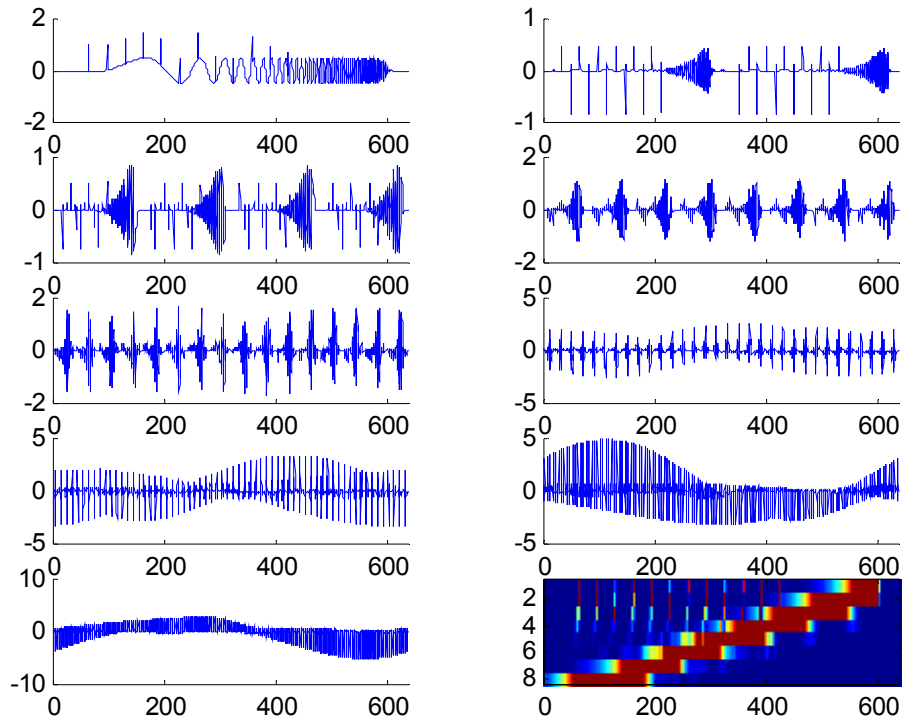


Figure 12. Shift-invariant scalogram by using D4 wavelets.

(a) Sampled waveform composed of an exponential chirp with 12 identical transients (b) the first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-invariant scalogram.

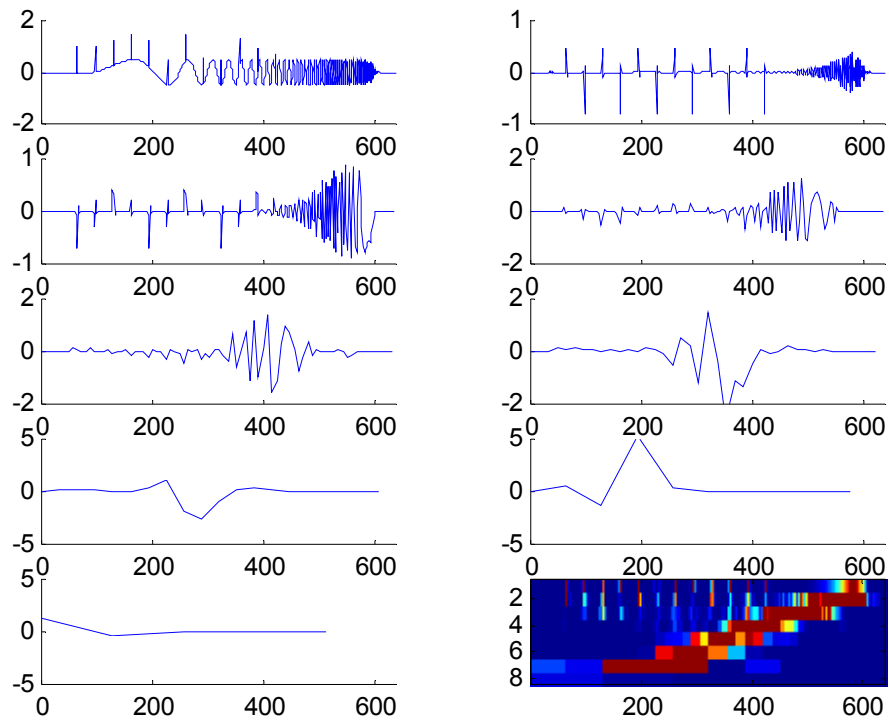


Figure 13. Shift-variant scalogram by using D6 wavelets.

(a) Sampled waveform composed of an exponential chirp with 12 identical transients (b) the first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-variant scalogram.

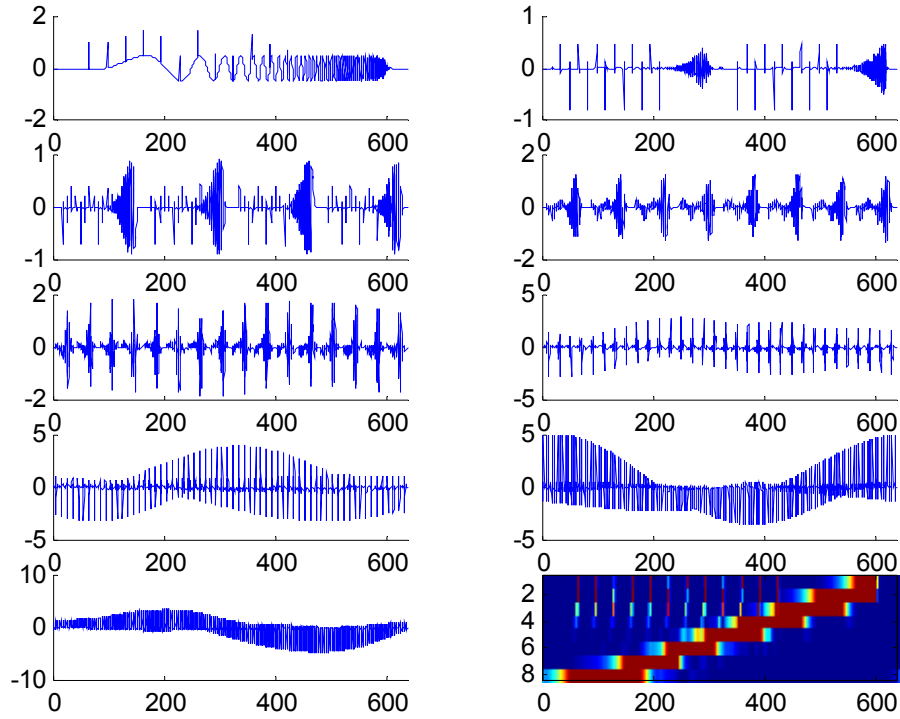


Figure 14. Shift-invariant scalogram by using D6 wavelets.

(a) Sampled waveform composed of an exponential chirp with 12 identical transients (b) the first generation wavelet coefficients of z (c) second generation wavelet coefficients of z (d) third generation wavelet coefficients of z (e) fourth generation wavelet coefficients of z (f) fifth generation wavelet coefficients of z (g) sixth generation wavelet coefficients of z (h) seventh generation wavelet coefficients of z (i) eighth generation wavelet coefficients of z (j) shift-invariant scalogram.

3. Scalogram Inversion

So far we looked at how a signal can be analyzed using wavelet techniques. As described in the introduction, we are interested in whether it is possible to find an inverse mapping from a scalogram back to the original signal. Due to the translation-invariance of this modified Wavelet Transform we lose the uniqueness of the inverse. Whereas the standard DWT is linear mapping from an N -dimensional space into an N -dimensional space, the SIDWT is a linear mapping from an N -dimensional space into an $(N \log_2 N)$ -dimensional space (if $N = 2^k$ for some k). Thus there is no unique inverse function to compute a signal from an arbitrary scalogram. However, in the following it turns out that the mapping is injective, thus it can be inverted on its image – the set of

all scalograms that belong to given input waveforms. This is shown in the next algorithm, which we will call the Inverse Shift-Invariant Discrete Wavelet Transform (ISIDWT).

Suppose we are given the shift-invariant discrete wavelet transform of a vector z , in other words the vectors $x_1, x_2, \dots, x_p, y_p$, where p is the stage of shift-invariant discrete wavelet transform. We want to reconstruct z . As we will prove, this can be done by a sequence of reordering and convolution steps, as further illustrated in Figure 15 (one stage SIDWT and ISIDWT) and Figure 16 (multi-stage SIDWT and ISIDWT).

We define the ISIDWT by

$$y_{p-1} = \frac{R_{-2}(y_p)}{2} * u + \frac{R_{-2}(x_p)}{2} * v,$$

$$y_{p-2} = \frac{R_{-2}(y_{p-1})}{2} * u + \frac{R_{-2}(x_{p-1})}{2} * v,$$

and so on, until the final step is

$$y_0 = \frac{R_{-2}(y_1)}{2} * u + \frac{R_{-2}(x_1)}{2} * v.$$

This is the algorithm, that makes is possible to invert the SIDWT on its image. For the algorithm to be useful, y_0 should agree again with the original vector z , which we will prove in the following theorem:

Theorem 1.

The algorithm Inverse Shift-Invariant Discrete Wavelet Transform (ISIDWT) yields a perfect reconstruction of the original waveform z , if we apply it to its SIDWT $x_1, x_2, \dots, x_p, y_p$.

Proof: Without loss of generality, we consider the first stage reconstruction

$$y_0 = \frac{R_{-2}(y_1)}{2} * u + \frac{R_{-2}(x_1)}{2} * v$$

$$= \frac{z * \tilde{u}}{2} * u + \frac{z * \tilde{v}}{2} * v.$$

Taking the Fourier transform on both sides and using the property that the system matrix $A(n)$ is unitary (see Lemma 1), we get

$$\begin{aligned}
\hat{y}_0(n) &= \left(\frac{z^* \tilde{u}}{2} * u + \frac{z^* \tilde{v}}{2} * v \right)^\wedge(n) \\
&= \frac{1}{2} (\hat{z}(n) \cdot \hat{\tilde{u}}(n) \cdot \hat{u}(n) + \hat{z}(n) \cdot \hat{\tilde{v}}(n) \cdot \hat{v}(n)) \\
&= \frac{1}{2} (\hat{z}(n) \cdot \overline{\hat{u}}(n) \cdot \hat{u}(n) + \hat{z}(n) \cdot \overline{\hat{v}}(n) \cdot \hat{v}(n)) \\
&= \frac{1}{2} (|\hat{u}(n)|^2 + |\hat{v}(n)|^2) \cdot \hat{z}(n) \\
&= \hat{z}(n).
\end{aligned}$$

By taking the inverse Fourier transform, we complete the proof of this theorem. \square

Using the diagrams as used in [3], our algorithm can be illustrated as follows:

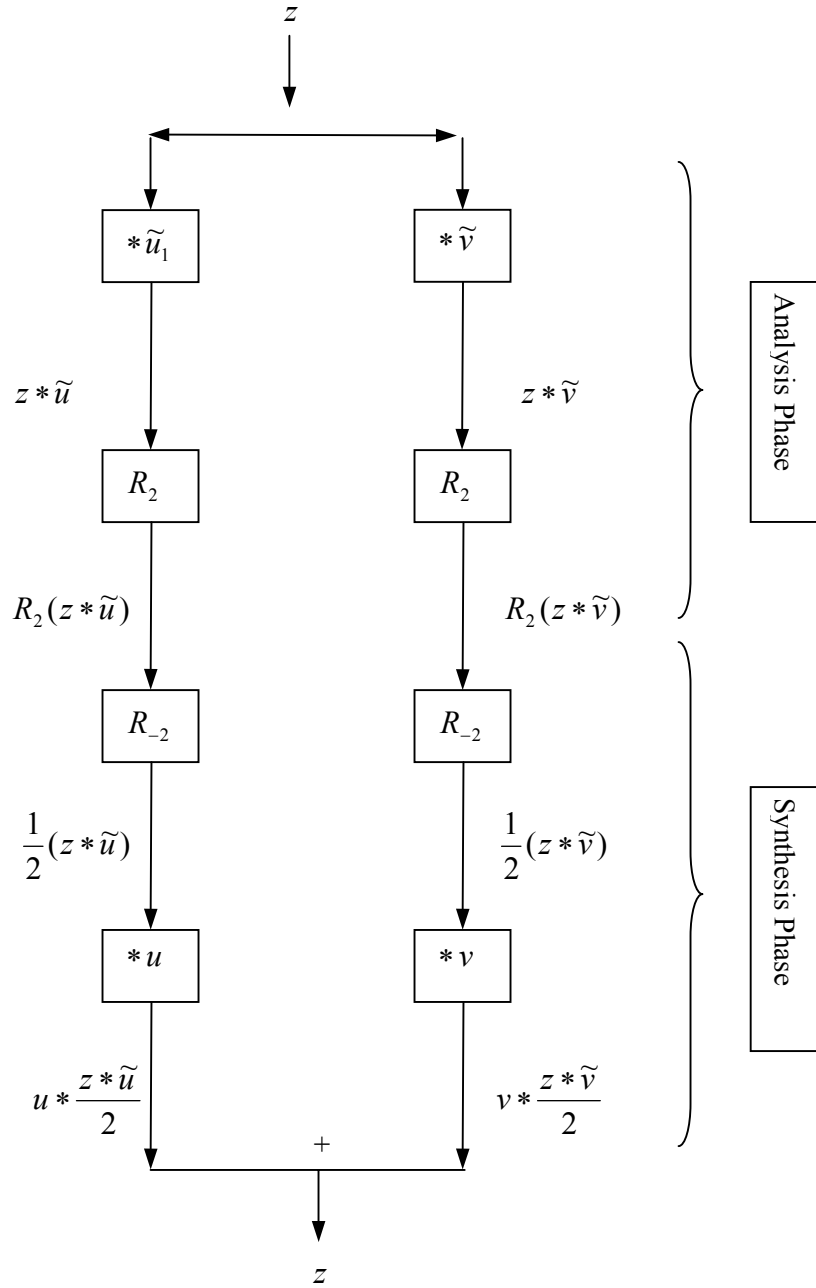


Figure 15. The output of the right branch is $v * \frac{z * \tilde{v}}{2}$ and the output of the left branch is $u * \frac{z * \tilde{u}}{2}$. Lemma 1

gives the perfect reconstruction condition, Theorem 1 shows that the ISIDWT reconstructs the original waveform.

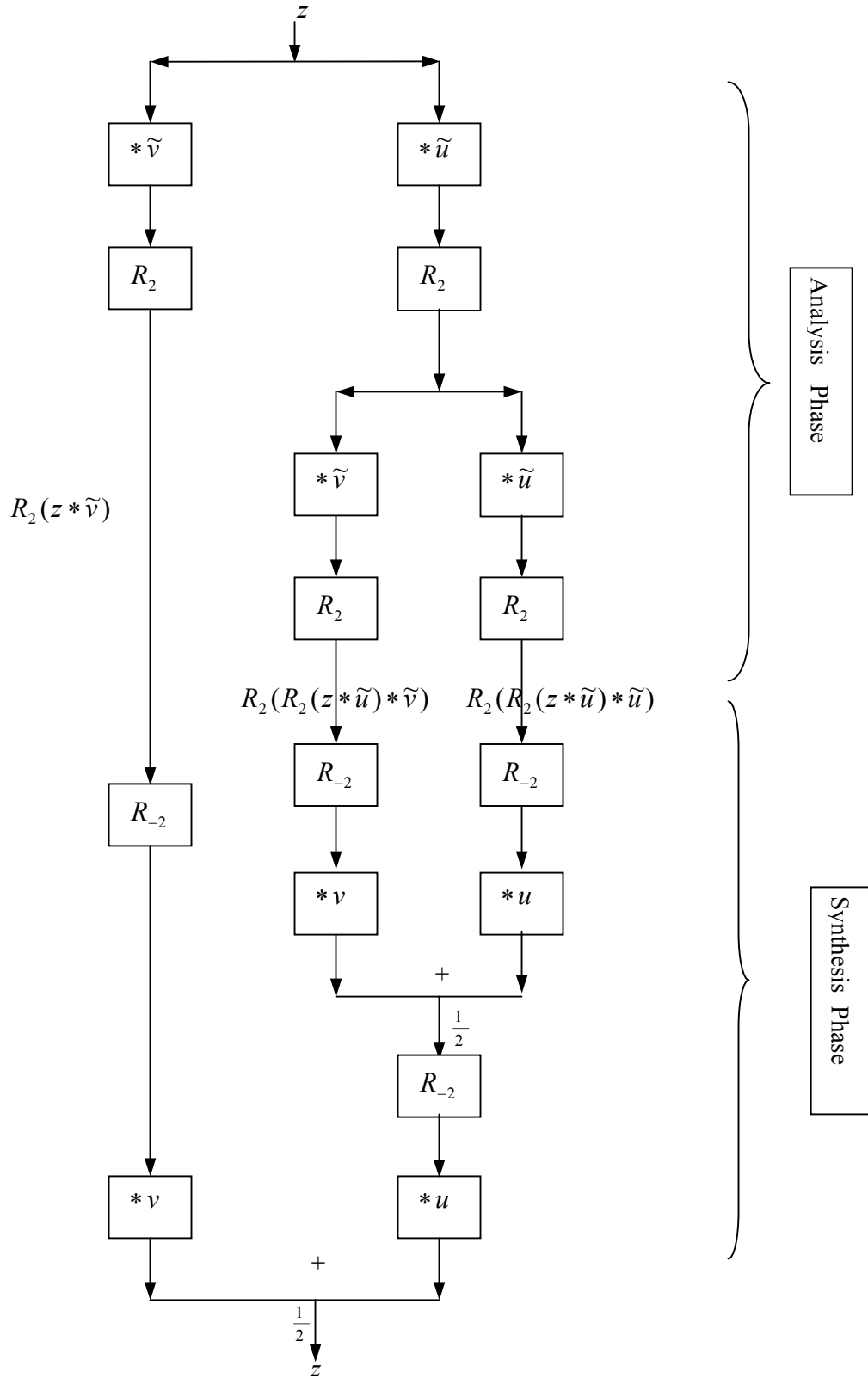


Figure 16. The multi-stage Shift-Invariant Wavelet Transform and its inversion.

So we found an algorithm to invert the shift invariant wavelet transform. However, in order for the algorithm to be useful, it must be computable in a reasonable amount of time. We know from [3] that the standard wavelet transform and its inverse can be computed rapidly. Now we consider the amount of computation required for the ISIDWT. To be precise, we would have to count the number of additions and reorderings as well. However, a complex multiplication is a much slower process on a computer than addition and reordering. So we get a sufficient idea of the speed of computation by just counting the number of complex multiplications involved in the computation. For an input sequence of length N , we define $\#_N$ to be the least number of complex multiplications needed to carry out an algorithm. Our inversion algorithm is based on the FFT, so first we give a lemma on the computational complexity of the FFT as well as the computation of a convolution. Those two results are from [3].

Lemma 2.

Suppose $N = 2^n$ for some $n \in \mathbb{N}$. Then the number of complex multiplications required to compute the DFT of a vector of length N using the FFT is

$$\#_N \leq \frac{1}{2} N \log_2 N.$$

The number of complex multiplications needed to compute a convolution (using the FFT) is

$$\#_N \leq N + \frac{3N}{2} \log_2 N.$$

Using those two results we can prove the following theorem:

Theorem 2.

Suppose $N = 2^n$ for some $n \in \mathbb{N}$. Then the total number of complex multiplications required to reconstruct a waveform using the ISIDWT algorithm is

$$\#_N \leq 2N \log_2 N + 3N(\log_2 N)^2.$$

Proof: Suppose we are given the wavelet transform of a vector z , in other words the vectors $x_1, x_2, \dots, x_p, y_p$, where p is the number of stages. We want to reconstruct z . This is done by a sequence of reordering and convolution steps, as described above (see Figures 10 and 11). Firstly, we need to have

$$p \leq \log_2 N$$

for the algorithm to make sense.

We consider the number of complex multiplications at one stage since the number of complex multiplications is the same at each stage. At the i^{th} stage, we have

$$y_{i-1} = \frac{1}{2}[R_{-2}(y_i) * u + R_{-2}(x_i) * v].$$

This is mainly composed of two convolution operations, which according to Lemma 2 cost

$$2 \cdot [N + \frac{3N}{2} \log_2 N] = 2N + 3N \log_2 N$$

complex multiplications.

Thus the total number of complex multiplication for ISIDWT is

$$\begin{aligned} \#_N &\leq p \cdot [2N + 3N \log_2 N] \\ &\leq 2N \log_2 N + 3N(\log_2 N)^2. \end{aligned}$$

This completes the proof. \square

Remark: In particular, if we take Daubechies D4 u and v then the computational complexity is just $O(p \cdot N)$.

Now, let us have a closer look at the properties of the mappings SIDWT and ISIDWT. Having found an inverse-algorithm that yields a complete reconstruction of the original signal, there is one question that occurs: What happens if we apply the ISIDWT-algorithm to a scalogram-matrix, which is not in the image? This question has a direct application: When having computed the scalogram, we would like to delete some patterns in it that might be ‘responsible’ for a certain rattling. The deletion is not a problem, however after the deletion we need to apply the ISIDWT to the changed scalogram. This however is might not be in the image of SIDWT. In the next section we first prove a theorem that states that the error between the waveform we get applying the ISIDWT to a modified scalogram and the original signal is small if the modification is small.

4. Scalogram Modification

4.1 Error Estimate

In the following, we will give an error estimate for the ISIDWT. First we need to introduce some definitions:

Given the vector $z = (z_0, z_1, \dots, z_{N-1})$ and the matrix $A = (a_{ij})_{i,j=1,N}$, define the vector norm

$$\|z\|_1 = \sum_{i=0}^{N-1} |z_i|,$$

and the subordinate matrix norm

$$\|A\|_1 = \max_{j=1, \dots, N} \left(\sum_{i=1}^N |a_{ij}| \right).$$

Moreover, let u, v be the high-pass and low-pass wavelet filters

$$u = (u(0), u(1), \dots, u(N-1))$$

$$v = (v(0), v(1), \dots, v(N-1))$$

and denote the two circular matrices C_u, C_v as follows

$$C_u = \begin{bmatrix} u(0) & u(N-1) & \dots & u(1) \\ u(1) & u(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & u(N-1) \\ u(N-1) & \dots & u(1) & u(0) \end{bmatrix},$$

$$C_v = \begin{bmatrix} v(0) & v(N-1) & \dots & v(1) \\ v(1) & v(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & v(N-1) \\ v(N-1) & \dots & v(1) & v(0) \end{bmatrix}.$$

Recall the definition of the reordering operator R_m . We can rewrite this operator R_2 in the

form of a matrix

$$R_2 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}.$$

Easily, we can get the inverse matrix R_{-2} as

$$R_{-2} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Without loss of generality, we only consider the first stage of the ISIDWT. Let z be the original waveform. Then we can get x_1, y_1 by using the shift invariant discrete wavelet transform as in Table 1. Now, suppose there exist some perturbations $\varepsilon_x, \varepsilon_y$ with respect to the given data x_1 and y_1 . We will apply our ISIDWT to the vectors $x_1 + \varepsilon_x, y_1 + \varepsilon_y$ and get a waveform ζ . This is the mathematical model for the modification of the scalogram and the generation of a waveform that we hope to be close to what we expect. The problem of this kind of modification is: Is the difference between the waveform ζ derived from the modified scalogram and the original signal z small if the perturbation $\varepsilon_x, \varepsilon_y$ are small? The answer is YES, as we can see from the following error estimate:

Theorem 3 (Error Estimate).

If the perturbation terms $\varepsilon_x, \varepsilon_y$ satisfy $\|\varepsilon_x\|_1 \leq \varepsilon$ and $\|\varepsilon_y\|_1 \leq \varepsilon$, the error estimate holds

$$\|\zeta - z\|_1 \leq \left(\frac{\|C_u\|_1 + \|C_v\|_1}{2} \right) \cdot \varepsilon.$$

Proof: Using the result of Theorem 1 and the definition of the matrices C_u, C_v and R_{-2} we get

$$\begin{aligned} \zeta &= \frac{R_{-2}(y_1 + \varepsilon_y)}{2} * u + \frac{R_{-2}(x_1 + \varepsilon_x)}{2} * v \\ &= \frac{R_{-2}(y_1)}{2} * u + \frac{R_{-2}(x_1)}{2} * v + \frac{1}{2} (R_{-2}(\varepsilon_y) * u + R_{-2}(\varepsilon_x) * v) \\ &= z + \frac{1}{2} (C_u R_{-2}(\varepsilon_y) + C_v R_{-2}(\varepsilon_x)). \end{aligned}$$

Looking at the definition of the inverse matrix of the ordering operator, we find $\|R_{-2}\|_1 = 1$.

Subtracting z from ζ and using the typical inequalities for the matrix norm, we have

$$\begin{aligned}
\|\zeta - z\|_1 &= \left\| \frac{1}{2}(C_u R_{-2}(\varepsilon_y) + C_v R_{-2}(\varepsilon_x)) \right\|_1 \\
&\leq \frac{1}{2}(\|C_u R_{-2}(\varepsilon_y)\|_1 + \|C_v R_{-2}(\varepsilon_x)\|_1) \\
&\leq \frac{1}{2}(\|C_u\|_1 \cdot \|R_{-2}\|_1 \cdot \|\varepsilon_y\|_1 + \|C_v\|_1 \cdot \|R_{-2}\|_1 \cdot \|\varepsilon_x\|_1) \\
&\leq \frac{1}{2}(\|C_u\|_1 + \|C_v\|_1) \cdot \varepsilon.
\end{aligned}$$

This completes the proof. \square

Theorem 3 shows that small changes in the scalogram of a signal z result in small differences between the original waveform z and the signal ζ computed by the ISIDWT, which guarantees the stability of the algorithm. However, deleting a high-energy part in the scalogram is in general not a small change. Therefore, in the next section we have a look at how localized the changes are with respect to frequency and time.

4.2 Examples

We are going to look at examples to analyze the extent to which changes in the scalogram of a signal correspond to changes of the signal. We first consider again our example of the exponential chirp with 12 transients, in the scalogram of which we delete different features.

Remark: Since SIDWT is a linear mapping from dimension N to dimension $N \log_2 N$ changing some entries of the scalogram most likely yields a vector which is not in the image. In fact the probability for that is zero.

(Technical remark: Notice that blue parts in the scalogram do not necessarily mean that there is no energy at all. However those parts have very low energy compared to the red parts. Also, sometimes the coloring that can be seen in the scalograms seems inconsistent, especially when looking at the different transients. They are all the same color when zooming in. This is a MATLAB problem that we cannot avoid. Hence one has to be careful when looking at scalograms and has a look at the numerical values of the matrices to be precise. However, for our purposes that is not necessary, since low energy frequency content cannot be perceived anyways.)

Starting with the signal, we compute the related scalogram using SIDWT (Figure 17).

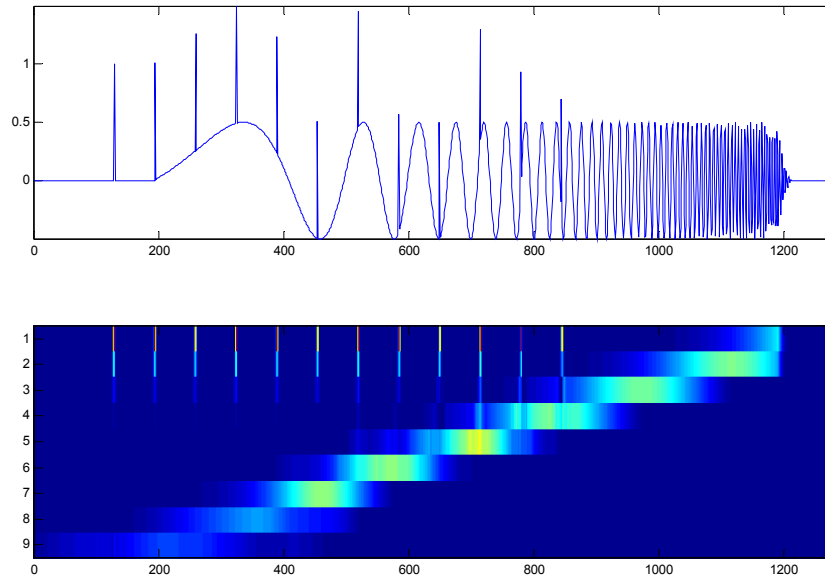


Figure 17. Chirp signal and its scalogram.

In the coefficients of the scalogram (before squaring and smoothing) we delete the 1st, 3rd and 8th transient as well as the full 6th octave. This yields the scalogram shown in Figure 18.

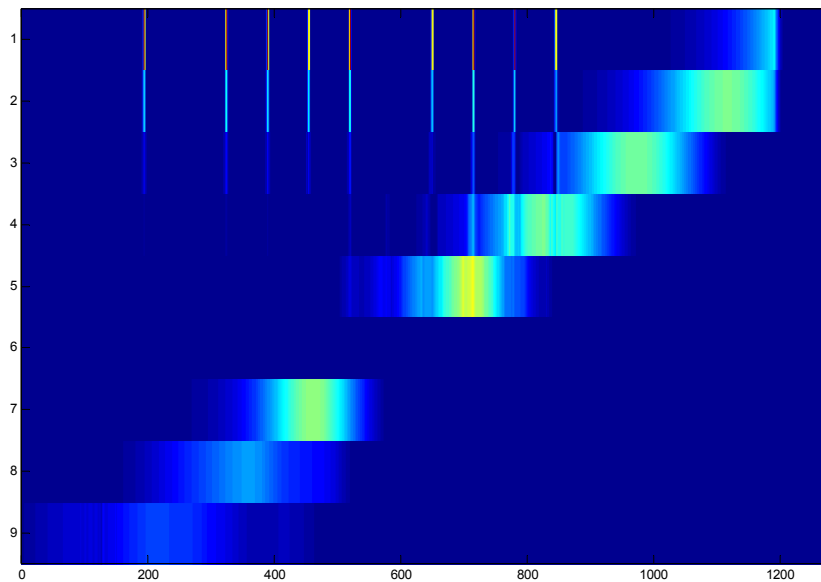


Figure 18. The scalogram with the 1st, 3rd and 8th transients and the full 6th octave deleted.

From this changed scalogram we compute the signal using ISIDWT, even though this scalogram is not in the image of SIDWT (see remark above). This returns the following signal:

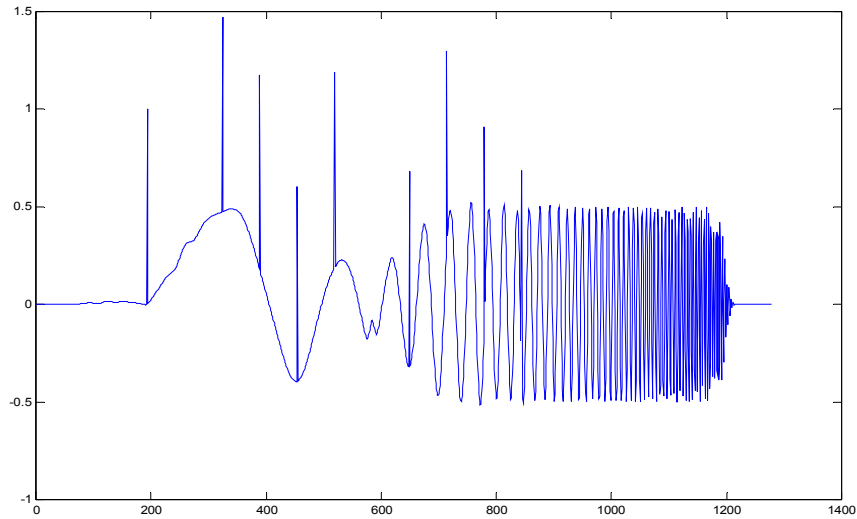


Figure 19. Signal obtained by applying ISIDWT to the modified scalogram in Figure 18.

Now, when applying the SIDWT algorithm to that modified signal we do not get the scalogram we started with, since this is not in the image. But instead we receive:

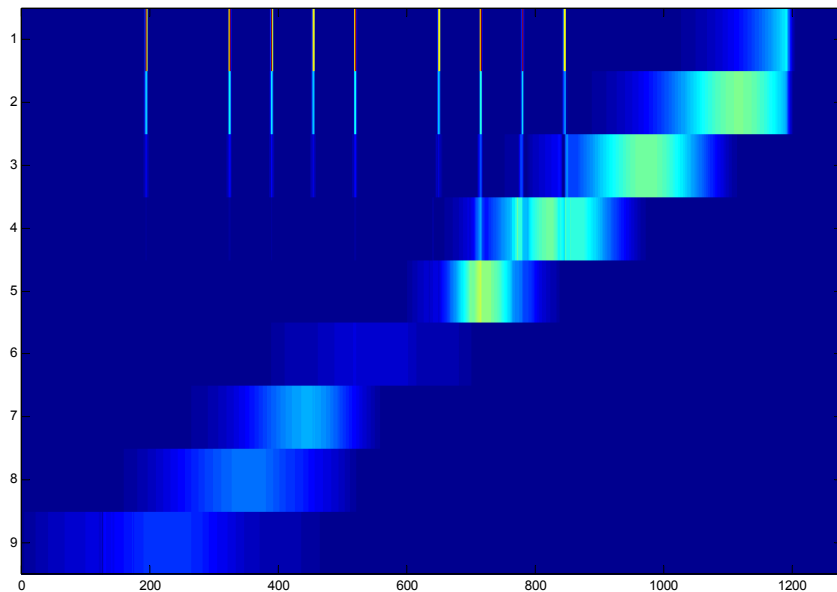


Figure 20. The scalogram belonging to the signal in Figure 19.

This differs from the above scalogram that we started with by setting some coefficients to zero. The difference is plotted in Figure 17, however the energy has been rescaled, that means the colors are in no correlation to the original energies. However we can see very well where the major changes occurred.

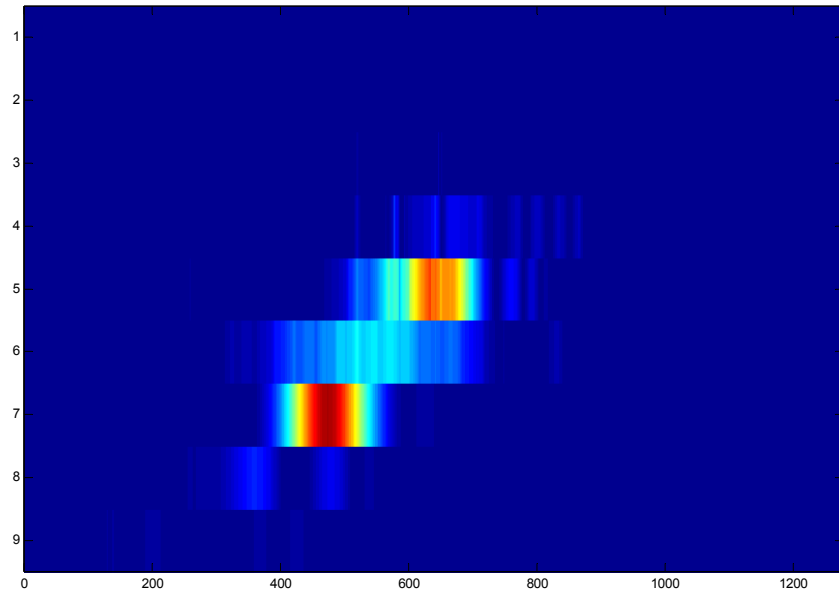


Figure 21. The difference between the modified scalogram and the scalogram computed from the modified signal.

It can be seen that the biggest change occurs in the area around the part where coefficients were removed, in the area of the deleted octave, which is — as can be seen in the figure on the previous page — partly rebuilt, meaning it does not remain zeroed out after the “transition” to a signal scalogram in the image of the SIDWT mapping.

It is interesting that especially in the neighboring frequency areas the changes are big (red). This is the case since the scalogram has to be converted to a signal $z \in l^2(Z_N)$ which has to “make sense”. Therefore it has to also have some coefficients directly below and above the deleted octave. This is underlined by the fact that the changes below are shifted to the left, the ones above to the right — exactly like the original scalogram.

Altogether the area of change is localized around the area where the coefficients have been zeroed. This shows that the method can be applied in the way we have described. The removal of

the chirps (which have the shape of the rattle-sounds we are looking for) works without any problems.

The following graphic shows the difference between the original signal and the changed one when only removing the 6th octave:

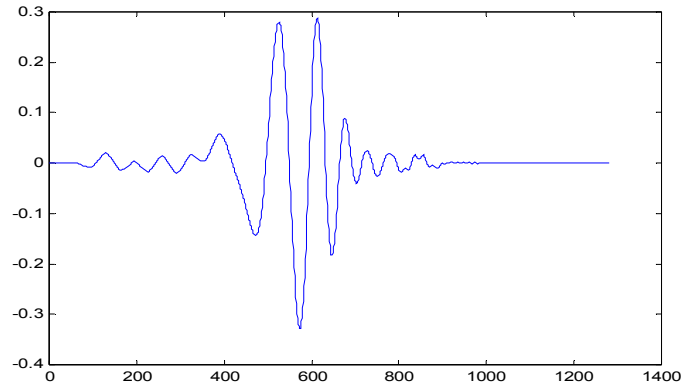


Figure 22. The difference between the original signal and the changed one when only removing the 6th octave.

Let us now have a look at “zero-scalogram” where we add energy at different places to get an impression how the inverse algorithm works. This method of adding entries to a zero-scalogram matrix is in some sense equivalent to deleting certain entries from a scalogram. It therefore shows some properties of the ISIDWT.

1. We start with the following scalogram, with a single high-energy block in a high octave:

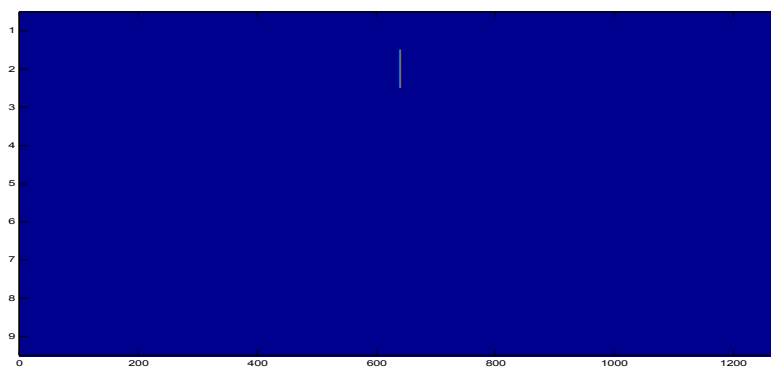


Figure 23. The scalogram with a single high-energy entry in a high octave.

The inverse algorithm yields the signal of Figure 24, which has a scalogram close to the one we started with.

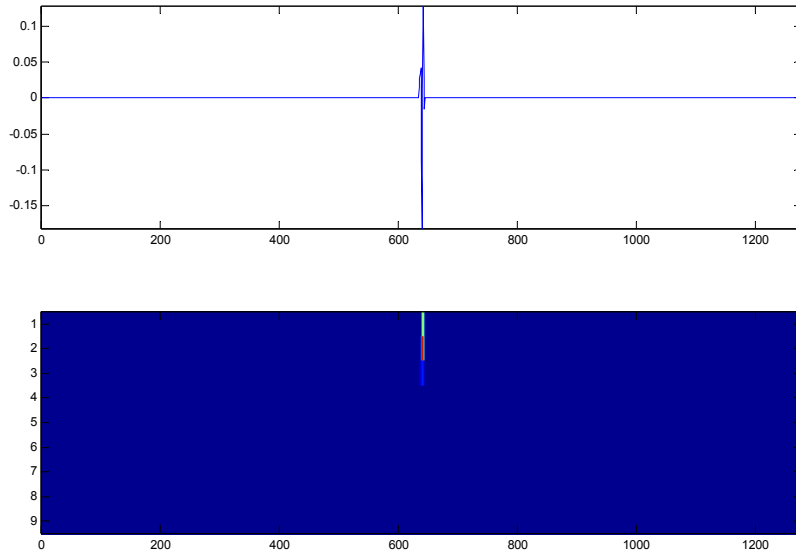


Figure 24. The signal together with its scalogram obtained from applying ISIDWT to the scalogram in Figure 23.

2. Next we do the same thing with a block in a lower octave as seen in Figure 25. Due to the periodicity, the spatial position is of no relevance.

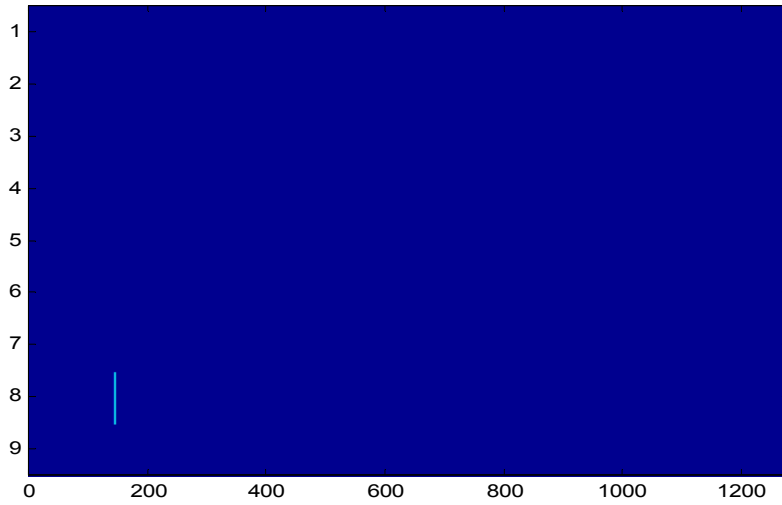


Figure 25. The scalogram with a single high-energy block in a low octave.

This scalogram of Figure 25 yields the signal of Figure 26 when the ISIDWT is applied:

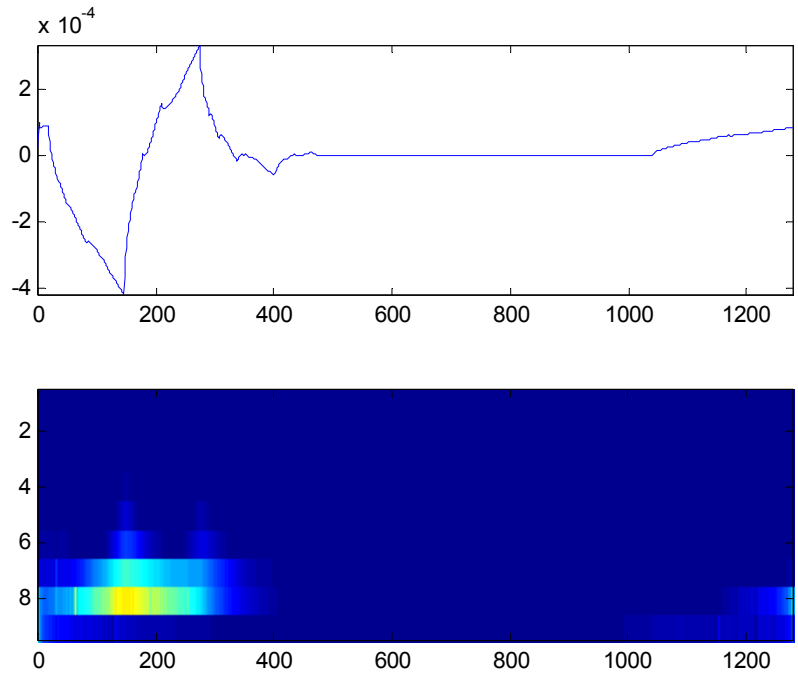


Figure 26. The signal (obtained by applying ISIDWT to Figure 25) and its related scalogram.

As we can see, the fact that the wavelets are less localized in the lower frequency range causes the bigger difference between the scalograms. But still the changes occurring are localized.

3. Now we consider bars over a certain time range — see Figure 27.

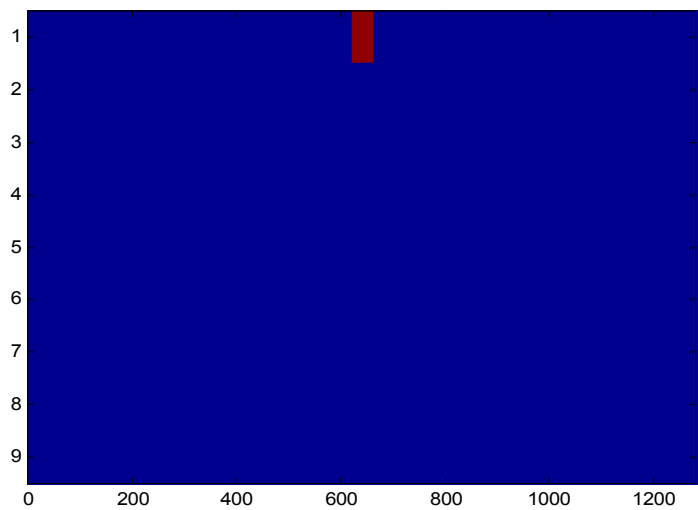


Figure 27. The scalogram with a block over a certain time range in a high octave.

The high peaks that in Figure 28 occur are caused by the jumps in energy at both ends of the bar. In between there is a frequency rustle, which cannot be seen in the scalogram.

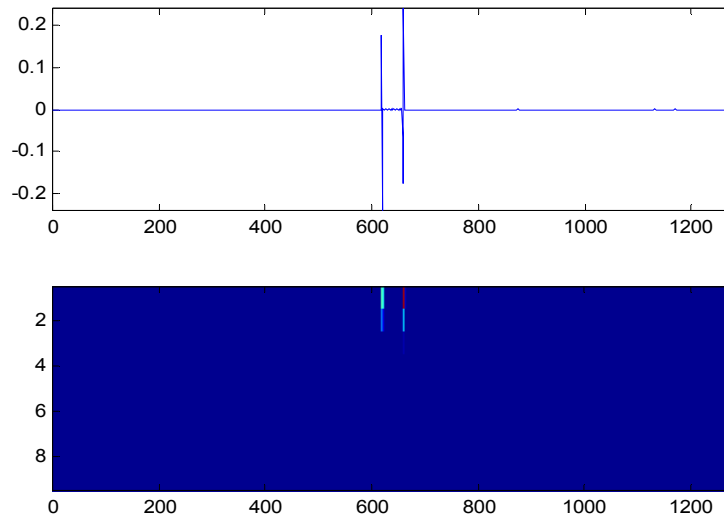


Figure 28. The signal and its related scalogram. The signal is obtained from applying ISIDWT to Figure 27.

Thus even this behavior of the inverse algorithm could have been predicted.

Whereas in this high-frequency case the changes are extremely localized, the next example shows that this localization is less in the lower frequencies — as to be expected.

4. Consider a bar in the 8th octave as shown in Figure 29.

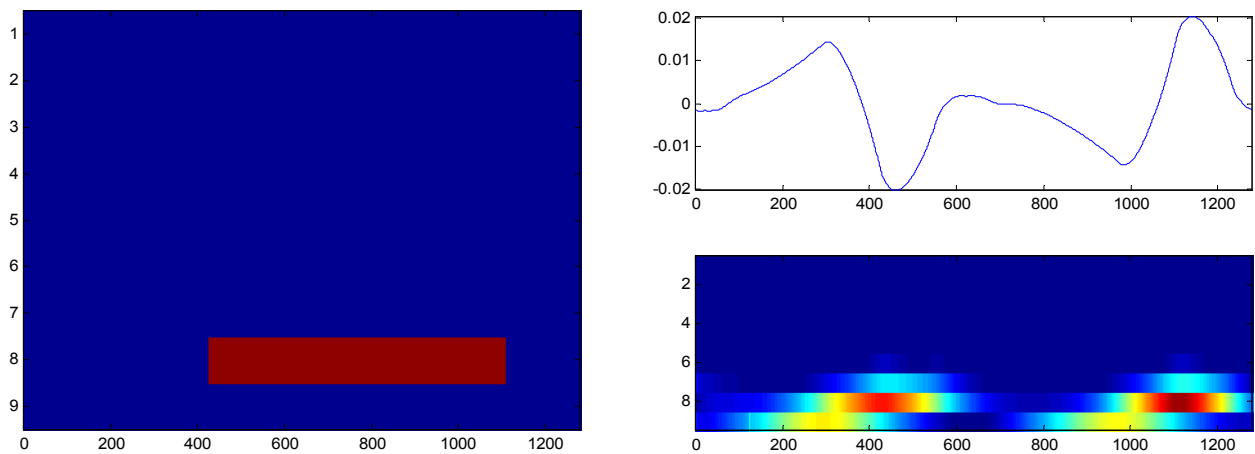


Figure 29. Left: The scalogram with a bar over a certain time range in the 8th octave.

Right: signal and scalogram, obtained via ISDWT.

5. Next we have look at “vertical bars” in a zero-scalogram as shown in Figure 30.

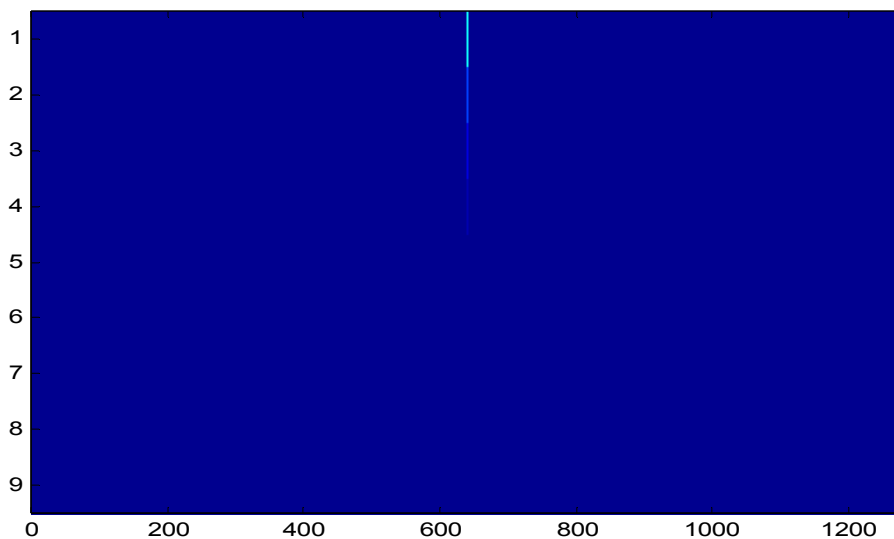


Figure 30. Vertical bar in a zero-scalogram.

Since we are again in the upper frequency part, the result of Figure 31 is as expected, with hardly any changes in the scalogram:

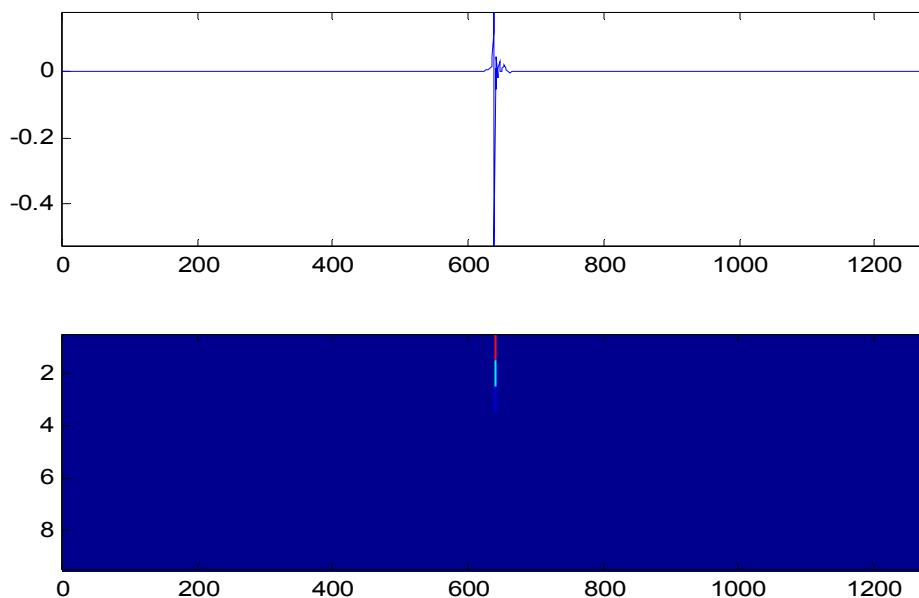


Figure 31. The signal obtained from Figure 30 by using ISIDWT and its related scalogram.

6. Our last example (Figure 32) combines spatial and frequency extension. Both effects discussed above can be seen: Less localization in the lower frequencies and the two peaks at both sides of the bars.

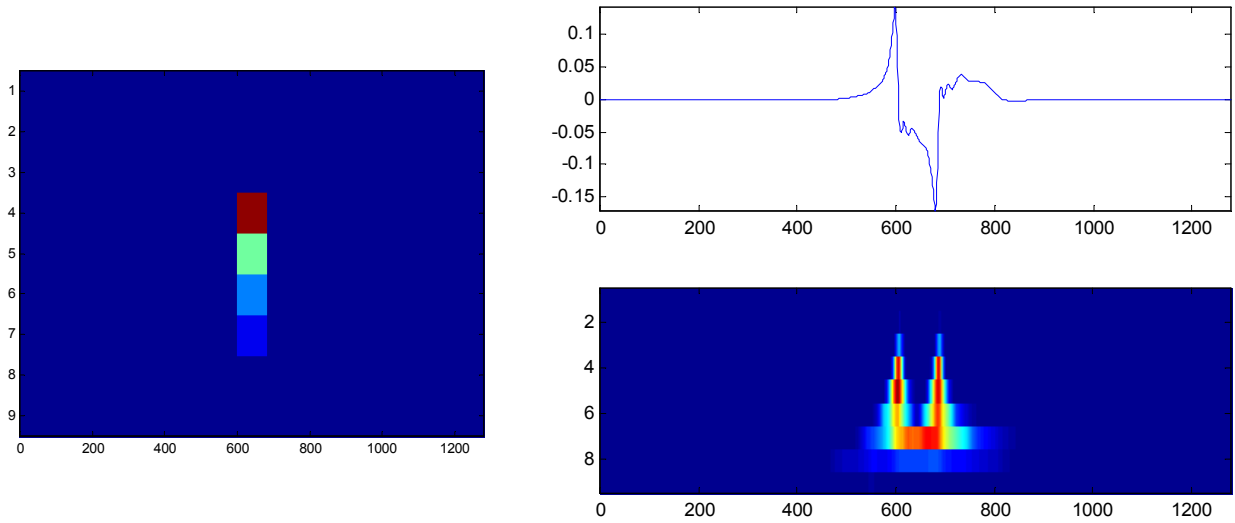


Figure 32. Left: Bar spread over time and frequency. Right: Signal computed by applying ISIDWT to signal on the left. Below, the belonging scalogram (SIDWT).

To come to a conclusion, we notice that those examples show that for the application where we want to remove features of short time duration and mainly higher energy (like the transients in the example of the exponential chirp) our inverse algorithm can be a useful tool.

4.3 Alternative Inverses

In this section we are looking at alternative possibilities to find a way to reconstruct a signal from its shift-invariant scalogram. The SIDWT is a linear mapping from an N -dimensional space to an $N \log_2 N$ -dimensional space. Since the matrices representing this mapping have ‘circulant parts’ and are generally big, looking for an inverse in matrix-form is numerically not promising. The Wavelet-Theory as presented in [3] when modified according to the translation-invariant algorithm makes us look at the class of inverses of a particular form: The proof of Theorem 1 shows that we do not necessarily need to use the wavelets u, v . In fact, if we define two vectors

$$A = (a(0), a(1), \dots, a(N-1)) \quad \text{and} \quad B = (b(0), b(1), \dots, b(N-1))$$

it suffices that they satisfy the equality

$$a(n) \cdot \bar{\hat{u}}(n) + \hat{b}(n) \cdot \bar{\hat{v}}(n) = 2 \quad \text{for all } n = 0, 1, \dots, N-1.$$

Then the proof is still valid and we get a whole family of possible A 's and B 's from the above equation. This family represents a general class of inverse algorithms computable rapidly via the convolution structure as in the algorithm before Theorem 1.

In the following, we try some examples using different choices of A, B to implement the ISIDWT. We remove the first transient in our test-waveform (Figure 7) and plot the difference between the original signal z and signal ζ obtained from applying the ISIDWT to the modified scalogram. In Figure 33 we use the wavelets u, v for the ISIDWT. In the following Figure 34 we use a random choice of A, B . This yields a slightly different result.

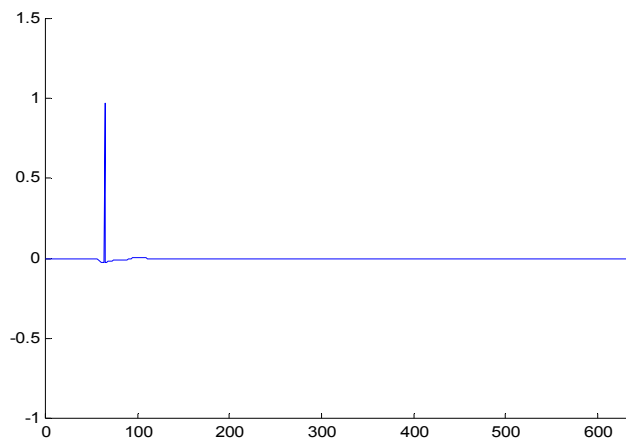


Figure 33. Difference between original signal z and the modified signal ζ using u and v .

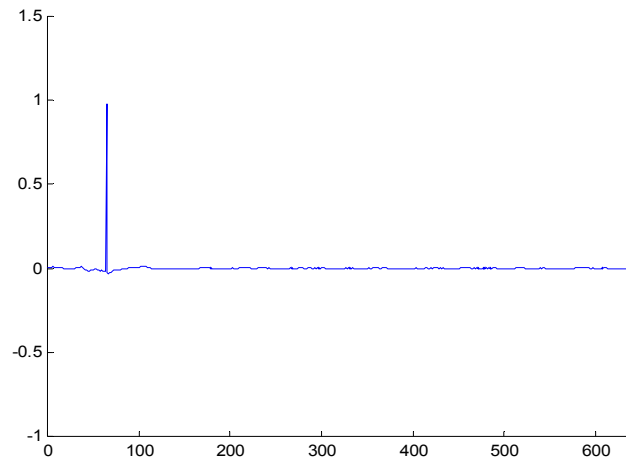


Figure 34. Difference between original signal z and the modified signal ζ using random A and B .

From the examples, we can see that the numerical results are better when using the wavelets u, v than the result when using random A, B . The effects of the deletion of the peak results in only local changes in the first case. However, with the filters A, B we notice slight changes in the full time domain. We rescale the y-axis of the plots in Figures 33 and 34 in Figures 35 and 36. This confirms what we just stated.

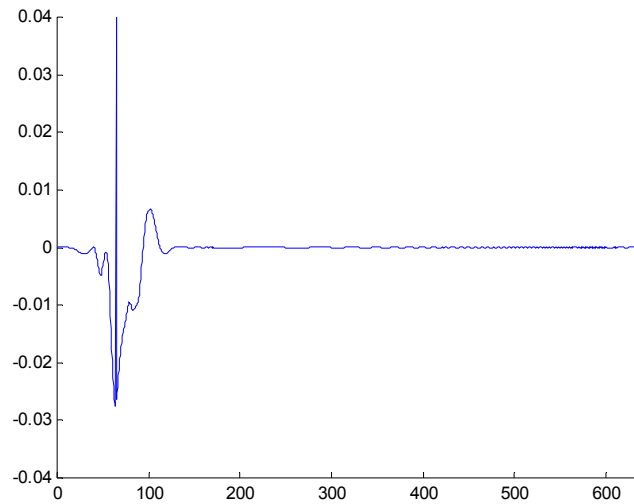


Figure 35. Difference between original signal z and the modified signal ζ using u and v .

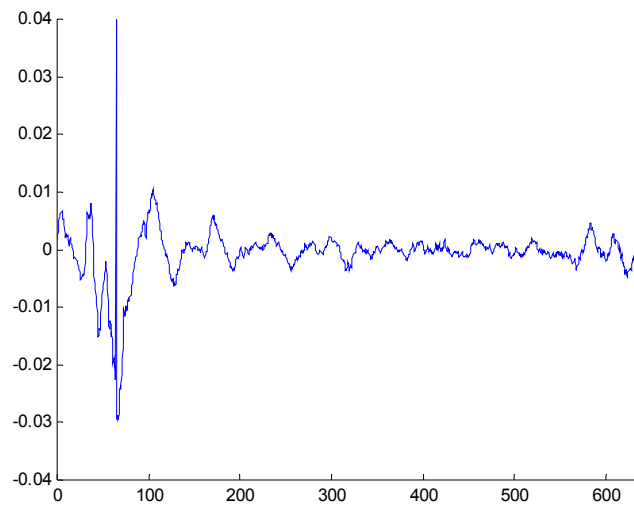


Figure 36. Difference between original signal z and the modified signal ζ using random A and B .

We give one more example before we conclude. Instead of an arbitrary choice of A, B we take the Shannon wavelet. We know that they are well localized in frequency but not particularly

well localized in time. Figure 37 and the rescaled version of it (Figure 38) show what we expect: The changes occurring are not spatially localized as well.

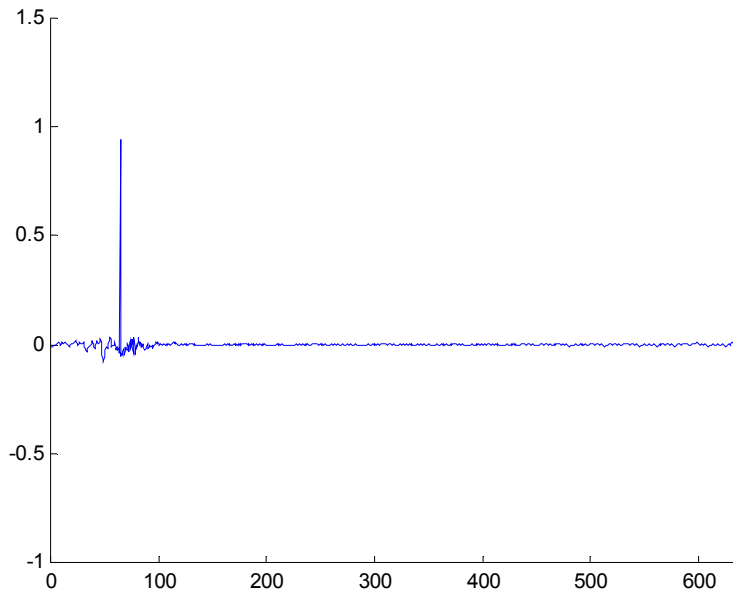


Figure 37. Difference between original signal z and the modified signal ζ using Shannon wavelets as A and B .

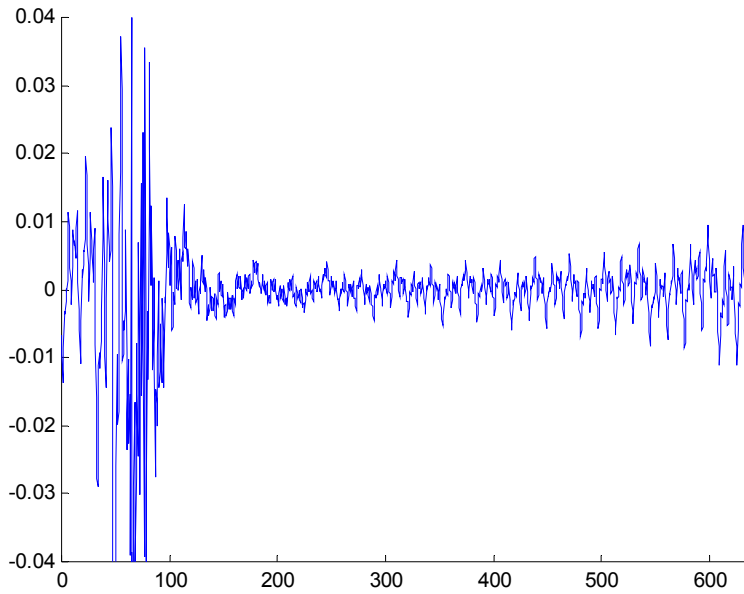


Figure 38. Difference between original signal z and the modified signal ζ using Shannon wavelets as A and B .

The observation that the algorithm works well (maybe best) when $A = u$ and $B = v$ can be explained with the fact that the wavelets u and v have been constructed in the way so that they are both spatially and frequency-localized hence removing features only has local effects.

5. Conclusions

In this paper, we focused on modification and inversion of scalograms. Given the algorithm of the shift-invariant discrete wavelet transform, we constructed a ‘pseudo-inverse’ which is the ‘true’ inverse on the set of all scalograms that can be obtained from input-signals by the shift-invariant discrete wavelet transform. We then looked at the properties of that inverse-algorithm when applied to scalograms which are not in the image of the transform. The main result is that even then the computed signals reflect the scalogram well, i.e. we can in fact delete a specific feature from a scalogram and apply the ‘pseudo-inverse’ to receive a signal that does not have the deleted feature anymore. This process only influences a relatively small neighborhood (in time and frequency) of the spot where the change has been made. Therefore we believe that the algorithm can be used for the purpose explained in section 1, to get an idea of how a recorded rattle-noise sounds after a part in its scalogram has been removed.

6. Future Work

As we said in the beginning, the long-term goal is to develop an algorithm that identifies the parts in a scalogram that correspond to the type of rattle we want to suppress. However, the shape of those features is yet to be determined more precisely, i.e., in a more mathematical way than just describing them as patterns of high energy, short time-duration and range of several octaves. This appears to be a very complicated problem linked to pattern recognition which appears in many different fields of science. However in order to find such an ‘rattle-recognition’ algorithm, the inverse algorithm as described in this paper can be a useful tool since it is necessary to know how rattle-sounds look in the scalogram. We need to delete parts in the scalogram and listen to the modified signal that has been computed by the ISIDWT.

Hence the work that needs to be done next is applying the inverse algorithm to the original automobile-sounds where parts that look like they are ‘causing’ the rattle need to be removed by hand. This is connected with some technical problems due to the size of the sound-samples [2].

We saw in 4.3 that different A, B go along with a different spatial localization of the changes. Although the results using random A, B appear to be worse than those using the original wavelets u, v , they might still be useful. Without having any evidence of that, we think that it might be possible to develop an adaptive method. This however is impossible as long as we cannot even mathematically describe the type of features we are looking for. However one might hope that it is possible to find a ‘best’ A, B which can satisfy our specific requirements, adapted to the kind of pattern we are looking for.

Acknowledgments

The authors express their thanks to several professors and technical specialists, who were extremely generous with their time. We express our deep appreciation to:

Professor Michael Frazier from the Department of Mathematics at Michigan State University, whom we met with weekly. As the manager of our project he conducted the projects process and provided us with many constructive suggestions with great devotion.

Technical Specialist David J. Scholl from the Physical and Environmental Sciences Department of the Ford Research Laboratories in Detroit, who was the proposer of the problem and the company liaison of the project. In addition to meeting with us on campus on two occasions to explain the problem and offer suggestions, he was a permanent contact person who helped us to continue heading in the right direction.

Professor Charles R. MacCluer and Professor Ralph Svetic from the Department of Mathematics at Michigan State University, as the Supervisors of Math 844, who read through our weekly report making corrections and giving valuable suggestions.

Professor Alan G. Haddow from the Mechanical Engineering Department at Michigan State University, who provided us with information on scalograms from an applied point of view and who showed great concern for our progress.

References

- [1] David J. Scholl, *Translation-Invariant Data Visualization with Orthogonal Discrete Wavelets*, IEEE Transactions On Signal Processing, Vol.46, No.7, July 1998.

- [2] David Scholl, *Wavelet-Based Visualization of Impulsive and Transient Sounds in Stationary Background Noise*, Society of Automotive Engineers, Inc., 2001.

- [3] Michael W. Frazier, *An Introduction to Wavelets through Linear Algebra*, New York: Springer-Verlag, 1999.

- [4] G. Beylkin, *On the Representation of Operators in Bases of compactly supported Wavelets*, SIAM J. Numer. Anal., vol ?, No.6, pp. 1716-1740.

- [5] Charles R. MacCluer, *Mathematical Modeling for Industry, Science & Government*, pp36, Prentice Hall, New Jersey, 1999.

- [6] Alan V. Oppenheim & Ronald W. Schaffer, *Digital Signal Processing*, Prentice Hall, New Jersey, 1975.

- [7] *User Guide of Signal Processing Toolbox*, The Mathworks Inc., 2001.

Appendix: MATLAB code

For the project, we have been given the papers [1] and [2]. Using those two references and [3], we implemented all the routines and algorithms as m-files in MATLAB 5.3. In the following, we list all the program codes:

```
% File name : example.m
% Waveform tested in this project
t=0:0.5:639.5;
axis([0 640 -1.0 2.0]);
hold on;
for n=1:192
    z(n)=0;
end;
for n=193:1216
    z(n)=0.5*sin(2*pi*(n-193)*1.004^(n-193)/1023);
end;
for n=1217:1280
    z(n)=0;
end;
for n=1190:1216
    z(n)=z(n)*((1216-n)/(26))^2.0;
end;
for k=0:11
    z(130+k*65)=z(130+k*65)+1.0;
end;
xlabel('Time');
plot(t,z,'r');
hold on;

% File name : D_m.m
% Downsampling operator
function w=D_m(z,m)
l=length(z);
```

```

k=l/m;
for i=1:k;
    w(i)=z(1+(i-1)*m);
end;

```

```

% File name : down.m
% Downsampling operator for m=2
function y=down(z)
n=length(z)/2;
for k=1:n
    y(k)=z(2*k-1);
end

```

```

% File name : E.m
% Energy for a sequence
function w=E(z)
l=length(z);
for j=1:l;
    w(j)=z(j)^2;
end

```

```

% File name : Ea.m
% Analytical energy for a sequence
function w=Ea(x)
l=length(x);
y=imag(hilbert(x));
for i=1:l
    w(i)=0.5*(x(i)^2+y(i)^2);
end

```

```

% File name : fold.m
% Fold a sequence
function w=fold(v)
n=length(v);

```

```

a=v(1:n/2);
b=v(n/2+1:n);
w=a+b;

% File name : Han.m
% Hanning window
function w=Han(N)
for k=0:N-1;
    w(k+1)=0.5*(1-cos(2*pi*k/(N-1)));
end

```

```

% File name : HanPad.m
% Hanning window with padded zeros
function w=HanPad(m,n)
y=blackman(m)';
%y=hanning(m)';
w=[y zeros(1,n-m)];

```

```

% File name : Pad.m
% Pad zeors to a sequence
function w=Pad(M,N)
w=[Han(M) zeros(1,N-M)];

```

```

% File name : R_m.m
% Re-arrange a sequence
function w=R_m(z,m)
l=length(z);
k=l/m;
for j=1:k;
    for i=1:m;
        A(i,j)=z(i+(j-1)*m);
    end
end
end
for i=1:m;

```

```

    for j=1:k;
        w(j+(i-1)*k)=A(i,j);
    end
end

```

```

% File name : realcon.m
% Compute the real convolution of two vectors
function y=realcon(a,b)
y=real(ifft(fft(a).*fft(b)));

```

```

% File name : S_n.m
% Left shift n position for a vector
function w=S_n(z,n)
L=length(z);
a=z(n+1:L);
b=z(1:n);
w=[a,b];

```

```

% File name : swap.m
% Exchange position of elements for a vector
function w=swap(z,m)
l=length(z);
k=l/m;
for i=1:m;
    for j=1:k;
        A(i,j)=z(j+(i-1)*k);
    end
end
for j=1:k;
    for i=1:m;
        w(i+(j-1)*m)=A(i,j);
    end
end
end

```

```

% File name : U_n.m
% Upsampling of a vector
function y=U_n(z,n)
l=length(z);
for j=0:l-1
    for k=1:n
        y(k+j*n)=z(j+1)/n;
    end;
end;

```

```

% File name : Spectro.m
% Compute the spectrogram of a waveform
load example;
N=1280;
z=[z zeros(1,256)];
w=HanPad(64,256);
for i=1:4:1281;
    y=Ea(real(fft(w.*z(i:i+255))));
    j=(i+3)/4;
    for k=1:128
        M(k,j)=y(128-k+1)';
    end
end
X=0:0.5:639.5;
Y=0.0:1.0/256:1.0;
image(X,Y,3000*M);

```

```

% File name : Shan.m
% Compute the Shannon wavelet coefficients.
N=1280;
for k=0:N/4-1;
    a(k+1)=sqrt(2.0);
end;
for k=3*N/4+1:N-1;

```

```

    a(k+1)=sqrt(2.0);
end;
for k=N/4+1:3*N/4-1;
    a(k+1)=0.0;
end;
a(N/4+1)=i;
a(3*N/4+1)=-i;
for k=0:N/4-1;
    b(k+1)=0.0;
end;
for k=3*N/4+1:N-1;
    b(k+1)=0.0;
end;
for k=N/4+1:3*N/4-1;
    b(k+1)=sqrt(2.0);
end;
b(N/4+1)=1.0;
b(3*N/4+1)=1.0;
shanu1=real(iff(a));
shanv1=real(iff(b));
shanu2=fold(shanu1);
shanv2=fold(shanv1);
shanu3=fold(shanu2);
shanv3=fold(shanv2);
shanu4=fold(shanu3);
shanv4=fold(shanv3);
shanu5=fold(shanu4);
shanv5=fold(shanv4);
shanu6=fold(shanu5);
shanv6=fold(shanv5);
shanu7=fold(shanu6);
shanv7=fold(shanv6);
shanu8=fold(shanu7);
shanv8=fold(shanv7);

```

```

shanu1til=real(iff(conj(a)));
shanv1til=real(iff(conj(b)));
shanu2til=fold(shanu1til);
shanv2til=fold(shanv1til);
shanu3til=fold(shanu2til);
shanv3til=fold(shanv2til);
shanu4til=fold(shanu3til);
shanv4til=fold(shanv3til);
shanu5til=fold(shanu4til);
shanv5til=fold(shanv4til);
shanu6til=fold(shanu5til);
shanv6til=fold(shanv5til);
shanu7til=fold(shanu6til);
shanv7til=fold(shanv6til);
shanu8til=fold(shanu7til);
shanv8til=fold(shanv7til);

```

```

% File name : Shantran_variant.m

```

```

% compute the standard 8th stage shift variant shannon wavelet transform of a vector z

```

```

load Shanfil;

```

```

load example;

```

```

x1=down(realcon(z,shanv1til));
y1=down(realcon(z,shanu1til));
x2=down(realcon(y1,shanv2til));
y2=down(realcon(y1,shanu2til));
x3=down(realcon(y2,shanv3til));
y3=down(realcon(y2,shanu3til));
x4=down(realcon(y3,shanv4til));
y4=down(realcon(y3,shanu4til));
x5=down(realcon(y4,shanv5til));
y5=down(realcon(y4,shanu5til));
x6=down(realcon(y5,shanv6til));
y6=down(realcon(y5,shanu6til));
x7=down(realcon(y6,shanv7til));

```

```

y7=down(realcon(y6,shanu7til));
x8=down(realcon(y7,shavu8til));
y8=down(realcon(y7,shanu8til));
n=0:0.5:639.5;
n1=0:1:639;
n2=0:2:639;
n3=0:4:639;
n4=0:8:639;
n5=0:16:639;
n6=0:32:639;
n7=0:64:639;
n8=0:128:639;
subplot(5,2,1),
axis([0 640 -2 2]);
hold on;
plot(n,z);
subplot(5,2,2),
axis([0 640 -1 1]);
hold on;
plot(n1,x1);
subplot(5,2,3),
axis([0 640 -2 2]);
hold on;
plot(n2,x2);
subplot(5,2,4),
axis([0 640 -2 2]);
hold on;
plot(n3,x3);
subplot(5,2,5),
axis([0 640 -4 4]);
hold on;
plot(n4,x4);
subplot(5,2,6),
axis([0 640 -4 4]);

```

```

hold on;
plot(n5,x5);
subplot(5,2,7),
axis([0 640 -5 5]);
hold on;
plot(n6,x6);
subplot(5,2,8),
axis([0 640 -5 5]);
hold on;
plot(n7,x7);
subplot(5,2,9),
axis([0 640 -5 5]);
hold on;
plot(n8,x8);
subplot(5,2,10),
S=[Ea(x1);
   U_n(Ea(x2),2);
   U_n(Ea(x3),4);
   U_n(Ea(x4),8);
   U_n(Ea(x5),16);
   U_n(Ea(x6),32);
   U_n(Ea(x7),64);
   U_n(Ea(x8),128)];
image(1000*S)

```

```

% File name : Shantran_invariant

```

```

% compute the translation-invariant 8th stage shannon wavelet transform of a vector z

```

```

load Shanfil;

```

```

load example;

```

```

L=1280;

```

```

x1=R_m(realcon(z,shanv1til),2);

```

```

y1=R_m(realcon(z,shanu1til),2);

```

```

x2=R_m(realcon(y1,shanv1til),2);

```

```

y2=R_m(realcon(y1,shanu1til),2);
x3=R_m(realcon(y2,shanv1til),2);
y3=R_m(realcon(y2,shanu1til),2);
x4=R_m(realcon(y3,shanv1til),2);
y4=R_m(realcon(y3,shanu1til),2);
x5=R_m(realcon(y4,shanv1til),2);
y5=R_m(realcon(y4,shanu1til),2);
x6=R_m(realcon(y5,shanv1til),2);
y6=R_m(realcon(y5,shanu1til),2);
x7=R_m(realcon(y6,shanv1til),2);
y7=R_m(realcon(y6,shanu1til),2);
x8=R_m(realcon(y7,shanv1til),2);
y8=R_m(realcon(y7,shanu1til),2);
n=0:0.5:639.5;
subplot(5,2,1),
axis([0 640 -2 2]);
hold on;
plot(n,z);
subplot(5,2,2),
axis([0 640 -1 1]);
hold on;
plot(n,x1);
subplot(5,2,3),
axis([0 640 -2 2]);
hold on;
plot(n,x2);
subplot(5,2,4),
axis([0 640 -2 2]);
hold on;
plot(n,x3);
subplot(5,2,5),
axis([0 640 -2 2]);
hold on;
plot(n,x4);

```

```

subplot(5,2,6),
axis([0 640 -5 5]);
hold on;
plot(n,x5);
subplot(5,2,7),
axis([0 640 -5 5]);
hold on;
plot(n,x6);
subplot(5,2,8),
axis([0 640 -5 5]);
hold on;
plot(n,x7);
subplot(5,2,9),
axis([0 640 -10 10]);
hold on;
plot(n,x8);
for j=1:8
    a(j)=0;
    b(j)=2^(-j)*L;
end;
subplot(5,2,10),
S=[Ea(S_n(R_m(2^(-1/2))*x1,b(1)),a(1)));
    Ea(S_n(R_m(2^(-2/2))*x2,b(2)),a(2)));
    Ea(S_n(R_m(2^(-3/2))*x3,b(3)),a(3)));
    Ea(S_n(R_m(2^(-4/2))*x4,b(4)),a(4)));
    Ea(S_n(R_m(2^(-5/2))*x5,b(5)),a(5)));
    Ea(S_n(R_m(2^(-6/2))*x6,b(6)),a(6)));
    Ea(S_n(R_m(2^(-7/2))*x7,b(7)),a(7)));
    Ea(S_n(R_m(2^(-8/2))*x8,b(8)),a(8))];
Y=1:1:8;
X=0:0.5:639.5;
image(X,Y,1000*S)

% File name : D4fil.m

```

```
% Compute the D4 wavelet coefficients
```

```
N=1280;
```

```
a=sqrt(2.0)/8.0;
```

```
z1=a*(1.0+sqrt(3.0));
```

```
z2=a*(3+sqrt(3.0));
```

```
z3=a*(3-sqrt(3.0));
```

```
z4=a*(1-sqrt(3.0));
```

```
dauu1=[z1 z2 z3 z4 zeros(1,1276)];
```

```
z1=a*(-3-sqrt(3.0));
```

```
z2=a*(1+sqrt(3.0));
```

```
z3=a*(-1+sqrt(3.0));
```

```
z4=a*(3-sqrt(3.0));
```

```
dauv1=[z1 z2 zeros(1,1276) z3 z4];
```

```
dauu2=fold(dauu1);
```

```
dauv2=fold(dauv1);
```

```
dauu3=fold(dauu2);
```

```
dauv3=fold(dauv2);
```

```
dauu4=fold(dauu3);
```

```
dauv4=fold(dauv3);
```

```
dauu5=fold(dauu4);
```

```
dauv5=fold(dauv4);
```

```
dauu6=fold(dauu5);
```

```
dauv6=fold(dauv5);
```

```
dauu7=fold(dauu6);
```

```
dauv7=fold(dauv6);
```

```
dauu8=fold(dauu7);
```

```
dauv8=fold(dauv7);
```

```
dauu1til=real(iff(conj(fft(dauu1))));
```

```
dauv1til=real(iff(conj(fft(dauv1))));
```

```
dauu2til=fold(dauu1til);
```

```
dauv2til=fold(dauv1til);
```

```
dauu3til=fold(dauu2til);
```

```
dauv3til=fold(dauv2til);
```

```
dauu4til=fold(dauu3til);
```

```
dauv4til=fold(dauv3til);
dauu5til=fold(dauu4til);
dauv5til=fold(dauv4til);
dauu6til=fold(dauu5til);
dauv6til=fold(dauv5til);
dauu7til=fold(dauu6til);
dauv7til=fold(dauv6til);
dauu8til=fold(dauu7til);
dauv8til=fold(dauv7til);
```

```
% File name : D4tran_variant.m
```

```
% compute the standard 8th stage shift variant Daubechies D4 wavelet transform of a vector z
```

```
load D4fil;
```

```
load example;
```

```
x1=down(realcon(z,dauv1til));
```

```
y1=down(realcon(z,dauu1til));
```

```
x2=down(realcon(y1,dauv2til));
```

```
y2=down(realcon(y1,dauu2til));
```

```
x3=down(realcon(y2,dauv3til));
```

```
y3=down(realcon(y2,dauu3til));
```

```
x4=down(realcon(y3,dauv4til));
```

```
y4=down(realcon(y3,dauu4til));
```

```
x5=down(realcon(y4,dauv5til));
```

```
y5=down(realcon(y4,dauu5til));
```

```
x6=down(realcon(y5,dauv6til));
```

```
y6=down(realcon(y5,dauu6til));
```

```
x7=down(realcon(y6,dauv7til));
```

```
y7=down(realcon(y6,dauu7til));
```

```
x8=down(realcon(y7,dauv8til));
```

```
y8=down(realcon(y7,dauu8til));
```

```
n=0:0.5:639.5;
```

```
n1=0:1:639;
```

```
n2=0:2:639;
```

```
n3=0:4:639;
```

```

n4=0:8:639;
n5=0:16:639;
n6=0:32:639;
n7=0:64:639;
n8=0:128:639;
y7=realcon(up(y8),dauu8)+realcon(up(x8),dauv8);
y6=realcon(up(y7),dauu7)+realcon(up(x7),dauv7);
y5=realcon(up(y6),dauu6)+realcon(up(x6),dauv6);
y4=realcon(up(y5),dauu5)+realcon(up(x5),dauv5);
y3=realcon(up(y4),dauu4)+realcon(up(x4),dauv4);
y2=realcon(up(y3),dauu3)+realcon(up(x3),dauv3);
y1=realcon(up(y2),dauu2)+realcon(up(x2),dauv2);
w=realcon(up(y1),dauu1)+realcon(up(x1),dauv1);
plot(n,w-z);
subplot(5,2,1),
axis([0 640 -2 2]);
hold on;
plot(n,z);
subplot(5,2,2),
axis([0 640 -1 1]);
hold on;
plot(n1,x1);
subplot(5,2,3),
axis([0 640 -1 1]);
hold on;
plot(n2,x2);
subplot(5,2,4),
axis([0 640 -2 2]);
hold on;
plot(n3,x3);
subplot(5,2,5),
axis([0 640 -2 2]);
hold on;
plot(n4,x4);

```

```

subplot(5,2,6),
axis([0 640 -2 2]);
hold on;
plot(n5,x5);
subplot(5,2,7),
axis([0 640 -5 5]);
hold on;
plot(n6,x6);
subplot(5,2,8),
axis([0 640 -5 5]);
hold on;
plot(n7,x7);
subplot(5,2,9),
axis([0 640 -5 5]);
hold on;
plot(n8,x8);
subplot(5,2,10),
S=[Ea(x1);
    U_n(Ea(x2),2);
    U_n(Ea(x3),4);
    U_n(Ea(x4),8);
    U_n(Ea(x5),16);
    U_n(Ea(x6),32);
    U_n(Ea(x7),64);
    U_n(Ea(x8),128)];
image(1000*S)

```

```

% File name : D4tran_invariant.m
% compute the translation-invariant 8th stage Daubechies D4 wavelet transform of a vector z
load D4fil;
load example;
L=1280;
x1=R_m(realcon(z,dauv1til),2);

```

```

y1=R_m(realcon(z,dauu1til),2);
x2=R_m(realcon(y1,dauv1til),2);
y2=R_m(realcon(y1,dauu1til),2);
x3=R_m(realcon(y2,dauv1til),2);
y3=R_m(realcon(y2,dauu1til),2);
x4=R_m(realcon(y3,dauv1til),2);
y4=R_m(realcon(y3,dauu1til),2);
x5=R_m(realcon(y4,dauv1til),2);
y5=R_m(realcon(y4,dauu1til),2);
x6=R_m(realcon(y5,dauv1til),2);
y6=R_m(realcon(y5,dauu1til),2);
x7=R_m(realcon(y6,dauv1til),2);
y7=R_m(realcon(y6,dauu1til),2);
x8=R_m(realcon(y7,dauv1til),2);
y8=R_m(realcon(y7,dauu1til),2);
n=0:0.5:639.5;
figure
subplot(5,2,1),
axis([0 640 -2 2]);
hold on;
plot(n,z);
subplot(5,2,2),
axis([0 640 -1 1]);
hold on;
plot(n,x1);
subplot(5,2,3),
axis([0 640 -1 1]);
hold on;
plot(n,x2);
subplot(5,2,4),
axis([0 640 -2 2]);
hold on;
plot(n,x3);
subplot(5,2,5),

```

```

axis([0 640 -2 2]);
hold on;
plot(n,x4);
subplot(5,2,6),
axis([0 640 -5 5]);
hold on;
plot(n,x5);
subplot(5,2,7),
axis([0 640 -5 5]);
hold on;
plot(n,x6);
subplot(5,2,8),
axis([0 640 -5 5]);
hold on;
plot(n,x7);
subplot(5,2,9),
axis([0 640 -10 10]);
hold on;
plot(n,x8);
for j=1:8
    a(j)=0;
    b(j)=2^(-j)*L;
end;
subplot(5,2,10),
S=[Ea(S_n(R_m(2^(-1/2))*x1,b(1)),a(1)));
    Ea(S_n(R_m(2^(-2/2))*x2,b(2)),a(2)));
    Ea(S_n(R_m(2^(-3/2))*x3,b(3)),a(3)));
    Ea(S_n(R_m(2^(-4/2))*x4,b(4)),a(4)));
    Ea(S_n(R_m(2^(-5/2))*x5,b(5)),a(5)));
    Ea(S_n(R_m(2^(-6/2))*x6,b(6)),a(6)));
    Ea(S_n(R_m(2^(-7/2))*x7,b(7)),a(7)));
    Ea(S_n(R_m(2^(-8/2))*x8,b(8)),a(8))];
S=64/max(max(S))*S;    % rescaling of matrix S to assign dark red to the highest energy
Y=1:1:8;

```

```
X=0:0.5:639.5;  
image(X,Y,S)
```

```
% File name : D6fil.m  
% Compute the 8th stage D6 wavelet coefficients  
N=1280;  
a=1.0-sqrt(10.0);  
b=1.0+sqrt(10.0);  
c=sqrt(5.0+2.0*sqrt(10.0));  
d=sqrt(2.0)/32;  
z1=d*(b+c);  
z2=d*(2*a+3*b+3*c);  
z3=d*(6*a+4*b+2*c);  
z4=d*(6*a+4*b-2*c);  
z5=d*(2*a+3*b-3*c);  
z6=d*(b-c);  
dauu1=[z1 z2 z3 z4 z5 z6 zeros(1,1274)];  
z1=-dauu1(2);  
z2=dauu1(1);  
z3=-dauu1(6);  
z4=dauu1(5);  
z5=-dauu1(4);  
z6=dauu1(3);  
dauv1=[z1 z2 zeros(1,1274) z3 z4 z5 z6];  
dauu2=fold(dauu1);  
dauv2=fold(dauv1);  
dauu3=fold(dauu2);  
dauv3=fold(dauv2);  
dauu4=fold(dauu3);  
dauv4=fold(dauv3);  
dauu5=fold(dauu4);  
dauv5=fold(dauv4);  
dauu6=fold(dauu5);  
dauv6=fold(dauv5);
```

```

dauu7=fold(dauu6);
dauv7=fold(dauv6);
dauu8=fold(dauu7);
dauv8=fold(dauv7);
dauu1til=real(iff(conj(fft(dauu1))));
dauv1til=real(iff(conj(fft(dauv1))));
dauu2til=fold(dauu1til);
dauv2til=fold(dauv1til);
dauu3til=fold(dauu2til);
dauv3til=fold(dauv2til);
dauu4til=fold(dauu3til);
dauv4til=fold(dauv3til);
dauu5til=fold(dauu4til);
dauv5til=fold(dauv4til);
dauu6til=fold(dauu5til);
dauv6til=fold(dauv5til);
dauu7til=fold(dauu6til);
dauv7til=fold(dauv6til);
dauu8til=fold(dauu7til);
dauv8til=fold(dauv7til);

% File name : D6tran_variant.m
% compute the translation-invariant 8th stage Daubechies D6 of a vector z
load D6fil ;
load example;
n=0:0.5:639.5;
axis([0 640 -1.0 2.0]);
hold on;
y7=0.5*(realcon(swap(y8,2),dauu1)+realcon(swap(x8,2),dauv1));
y6=0.5*(realcon(swap(y7,2),dauu1)+realcon(swap(x7,2),dauv1));
y5=0.5*(realcon(swap(y6,2),dauu1)+realcon(swap(x6,2),dauv1));
y4=0.5*(realcon(swap(y5,2),dauu1)+realcon(swap(x5,2),dauv1));
y3=0.5*(realcon(swap(y4,2),dauu1)+realcon(swap(x4,2),dauv1));
y2=0.5*(realcon(swap(y3,2),dauu1)+realcon(swap(x3,2),dauv1));

```

```

y1=0.5*(realcon(swap(y2,2),dauu1)+realcon(swap(x2,2),dauv1));
w=0.5*(realcon(swap(y1,2),dauu1)+realcon(swap(x1,2),dauv1));
plot(n,w)

```

```

% File name : inverse.m

```

```

% Inverse Shift-invariant Discrete Wavelet Transform

```

```

load D4tran_invariant;

```

```

load example;

```

```

n=0:0.5:639.5;

```

```

axis([0 640 -1.0 2.0]);

```

```

hold on;

```

```

y7=0.5*(realcon(swap(y8,2),dauu1)+realcon(swap(x8,2),dauv1));

```

```

y6=0.5*(realcon(swap(y7,2),dauu1)+realcon(swap(x7,2),dauv1));

```

```

y5=0.5*(realcon(swap(y6,2),dauu1)+realcon(swap(x6,2),dauv1));

```

```

y4=0.5*(realcon(swap(y5,2),dauu1)+realcon(swap(x5,2),dauv1));

```

```

y3=0.5*(realcon(swap(y4,2),dauu1)+realcon(swap(x4,2),dauv1));

```

```

y2=0.5*(realcon(swap(y3,2),dauu1)+realcon(swap(x3,2),dauv1));

```

```

y1=0.5*(realcon(swap(y2,2),dauu1)+realcon(swap(x2,2),dauv1));

```

```

w=0.5*(realcon(swap(y1,2),dauu1)+realcon(swap(x1,2),dauv1));

```

```

plot(n,w) ;

```

```

% File name : Modification.m

```

```

% Use the ISDWT to invert a changed scalogram to its waveform

```

```

load example ;

```

```

load D4tran_invariant ;

```

```

Q(1,:)=rearrange(x1,b(1));

```

```

Q(2,:)=rearrange(x2,b(2));

```

```

Q(3,:)=rearrange(x3,b(3));

```

```

Q(4,:)=rearrange(x4,b(4));

```

```

Q(5,:)=rearrange(x5,b(5));

```

```

Q(6,:)=rearrange(x6,b(6));

```

```

Q(7,:)=rearrange(x7,b(7));

```

```

Q(8,:)=rearrange(x8,b(8));

```

```

Q(9,:)=rearrange(y8,b(8));

```

```

Q(1,100:155)=zeros(1,56);
Q(2,100:155)=zeros(1,56);
Q(3,100:155)=zeros(1,56);
Q(4,100:155)=zeros(1,56);
Q(5,100:155)=zeros(1,56);
Q(6,100:155)=zeros(1,56);
Q(1,220:290)=zeros(1,71);
Q(2,220:290)=zeros(1,71);
Q(3,220:290)=zeros(1,71);
Q(4,220:290)=zeros(1,71);
x1b=swap(Q(1,:),b(1));
x2b=swap(Q(2,:),b(2));
x3b=swap(Q(3,:),b(3));
x4b=swap(Q(4,:),b(4));
x5b=swap(Q(5,:),b(5));
x6b=swap(Q(6,:),b(6));
x7b=swap(Q(7,:),b(7));
x8b=swap(Q(8,:),b(8));
y8b=swap(Q(9,:),b(8));
y7b=realcon(swap(y8b,2)/2,d4u)+realcon(swap(x8b,2)/2,d4v);
y6b=realcon(swap(y7b,2)/2,d4u)+realcon(swap(x7b,2)/2,d4v);
y5b=realcon(swap(y6b,2)/2,d4u)+realcon(swap(x6b,2)/2,d4v);
y4b=realcon(swap(y5b,2)/2,d4u)+realcon(swap(x5b,2)/2,d4v);
y3b=realcon(swap(y4b,2)/2,d4u)+realcon(swap(x4b,2)/2,d4v);
y2b=realcon(swap(y3b,2)/2,d4u)+realcon(swap(x3b,2)/2,d4v);
y1b=realcon(swap(y2b,2)/2,d4u)+realcon(swap(x2b,2)/2,d4v);
w=realcon(swap(y1b,2)/2,d4u)+realcon(swap(x1b,2)/2,d4v);
figure
plot(n,w);
figure
plot(n,z-w);

```