The Fast Lifting Wavelet Transform

c.valens@mindless.com

Disclaimer

This tutorial is aimed at the engineer, not the mathematician. This does not mean that there will be no mathematics, it just means that there will be no proofs in the text. In my humble opinion, mathematical papers are completely unreadable because of the proofs that clutter the text. For proofs the reader is pointed to suitable references. The equations presented are there to illustrate and to clarify things, I hope. It should not be necessary to understand all the equations in order to understand the theory. However, to understand this tutorial, a mathematical background on an engineering level is required. Also some knowledge of signal processing theory might come in handy.

The information presented in this tutorial is believed to be correct. However, no responsibilty whatsoever will be accepted for any damage whatsoever due to errors or misleading statements or whatsoever in this tutorial. Should there be anything incorrect, incomplete or not clear in this text, please let me know so that I can improve this tutorial.

Table of Contents

1. Introduction

This tutorial is a sequel to my wavelet tutorial, which will fill in the blank spots on the wavelet transform map, add some detail and even explore the area outside it. We start with taking a closer look at the scaling and wavelet filters in general, what they should look like, what their constraints are and how they can be used in the inverse wavelet transform. Then we will do some algebra and develop a general framework to design filters for every possible wavelet transform. This framework was introduced by Sweldens [Swe96a] and is known as *the lifting scheme* or simply *lifting*. Using the lifting scheme we will in the end arrive at a universal discrete wavelet transform which yields only integer wavelet- and scaling coefficients instead of the usual floating point coefficients. In order to clarify the theory in this tutorial a detailed example will be presented.

In this tutorial we will go into some more detail compared to the wavelet transform tutorial, since the lifting scheme is a quite recent development and especially integer lifting [Cal96], [Uyt97b] and multi-dimensional lifting [Kov97], [Uyt97a] are not (yet) widely known. This tutorial is mainly based on [Dau97], [Cal96], [Swe96a], [Swe96b], [Cla97], [Uyt97b] and [Uyt97c].

Before we start a short note on notation. In order to be compatible with existing lifting literature I will use the same symbols. This means that the analyzing filters are denoted as $\tilde{h}$ and $\tilde{g}$, i.e. with a tilde, while the synthesizing filters are denoted by a plain h and g. In fact, everything that has to do with the forward wavelet transform will carry a tilde. I will however use a slightly different form here due to the fact that in [Dau97] the transposed version of $\tilde{P}(z)$ is used in all calculations and because the filters $\tilde{h}$ and $\tilde{g}$ are there defined as $\tilde{h}(z^{-1})$ and $\tilde{g}(z^{-1})$ instead of $\tilde{h}(z)$ and $\tilde{g}(z)$. In this tutorial λ stands for the scaling function coefficients and γ for the wavelet coefficients.

2. Perfect reconstruction

Usually a signal transform is used to transform a signal to a different domain, perform some operation on the transformed signal and inverse transform it, back to the original domain. This means that the transform has to be invertible. In case of no data processing we want the reconstruction to be perfect, i.e. we will allow only for a time delay. All this holds also in our case of the wavelet transform.

As mentioned before, we can perform a wavelet transform (or subband coding or multiresolution analysis) using a filter bank. A simple one-stage filter bank is shown in figure 1 and it can be made using FIR filters. Although IIR filters can also be used, they have the disadvantage that their infinite response leads to infinite data expansion. For any practical use of an IIR filter bank the output data stream has to be cut which leads to a loss of data. In this text we will only look at FIR filters.
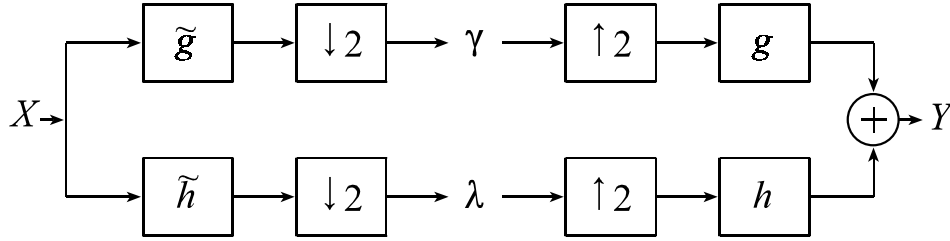
Figure 1

A one-stage filter bank for signal analysis and reconstruction.

Figure 1 shows how a one-stage wavelet transform uses two analysis filters, a low-pass filter $\tilde{h}$ and a high-pass filter $\tilde{g}$ followed by subsampling for the forward transform. From this figure it seems only logical to construct the inverse transform by first performing an upsampling step and then to use two synthesis filters h (low-pass) and g (high-pass) to reconstruct the signal. We need filters here, because the upsampling step is done by inserting a zero in between every two samples and the filters will have to smooth this.

For the filter bank in figure 1 the conditions for perfect reconstruction are now given by [Dau97] as:

$$\begin{aligned} h(z)\tilde{h}(z^{-1}) + g(z)\tilde{g}(z^{-1}) &= 2 \\ h(z)\tilde{h}(-z^{-1}) + g(z)\tilde{g}(-z^{-1}) &= 0 \end{aligned}. \tag{1}$$

The time reversal of the analyzing filters is necessary to compensate for the delays in the filters. Without it, it would be impossible to arrive at an non-delayed perfectly reconstructed signal. If the conditions for perfect reconstruction are fulfilled, then all the aliasing caused by the subsampling will miraculously be canceled during the reconstruction.

3. The polyphase representation

In the filter stage shown in the left part of figure 1 the signal is first filtered and then subsampled. In other words, we throw away half of the filtered samples and keep only the even-numbered samples, say. Clearly this is not efficient and therefore we would like to do the subsampling before the filtering in order to save some computing time. Let us take a closer look at what exactly is thrown away by subsampling.
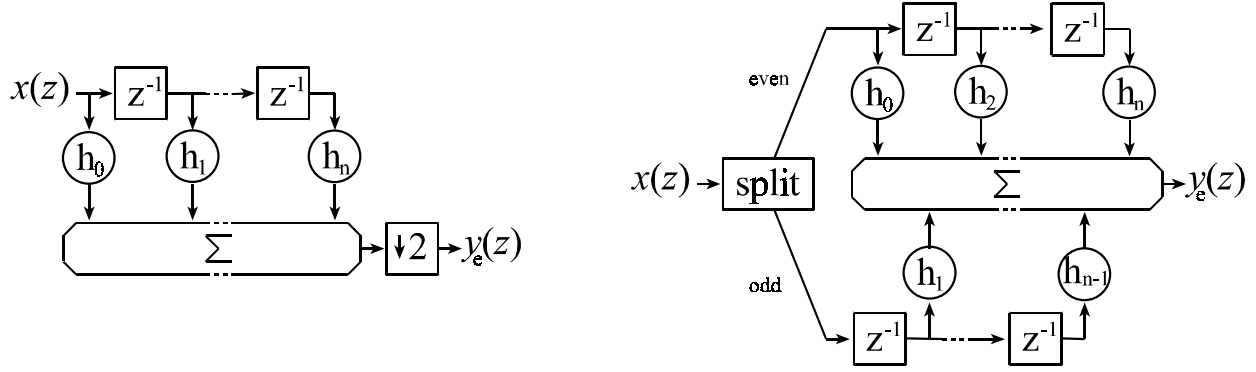
Figure 2

A standard FIR filter with subsampling of the output (left) and a more efficient implementation (right).

In figure 2 a standard FIR filter is drawn, followed by a subsampler. If we write down its output signal $y(z)$, just before the subsampler, for several consecutive samples, we get

$$
\begin{aligned}
&\vdots \\
y_0 &= h_0 x_0 + h_1 x_{-1} z^{-1} + h_2 x_{-2} z^{-2} + \ldots \\
y_1 &= h_0 x_1 + h_1 x_0 z^{-1} + h_2 x_{-1} z^{-2} + \ldots \\
y_2 &= h_0 x_2 + h_1 x_1 z^{-1} + h_2 x_0 z^{-2} + \ldots \\
&\vdots
\end{aligned}
\tag{2}
$$

Subsampling of $y(z)$ will now remove the middle line (row) in (2) and all the other odd lines (rows). We notice that with the odd lines removed the even-numbered filter coefficients h_e are only used with even-numbered samples x_e, while the odd-numbered filter coefficients h_o are only used with odd-numbered samples x_o. If we take the even bits together and name them $h_e(z)x_e(z)$ and do the same with the odd bits to form $h_o(z)x_o(z)$, then we can write the subsampled output signal $y_e(z)$ as

$$
y_e(z) = h_e(z)x_e(z) + z^{-1}h_o(z)x_o(z).
\tag{3}
$$

The delay z^{-1} in front of the odd part in (3) comes from the delay between even and odd samples. Equation (3) shows that we can redraw the FIR filter as shown in the right part of figure 2. In this figure it is assumed (without loss of generality) that n is even.

If we apply this remodeled FIR filter to our wavelet transform filter stage in figure 1 we end up with two equations like (3), one for each filter, so that, if we switch to vector notation, we can write

$$
\begin{pmatrix} \lambda(z) \\ \gamma(z) \end{pmatrix} = \tilde{P}(z) \begin{pmatrix} x_e(z) \\ z^{-1}x_o(z) \end{pmatrix},
\tag{4}
$$

where $\tilde{P}(z)$ is the *polyphase matrix*[1]

$$\tilde{P}(z) = \begin{pmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{pmatrix}. \tag{5}$$

The polyphase matrix now performs the wavelet transform. If we set $\tilde{h}_e(z)$ and $\tilde{g}_o(z)$ to one and we make $\tilde{h}_o(z)$ and $\tilde{g}_e(z)$ zero, i.e. $\tilde{P}(z)$ is the unit matrix, then the wavelet transform is referred to as the *lazy wavelet transform* [Swe96a]. However, I would like to rename it and call it the *femmelet transform*. The femmelet transform does nothing more than splitting the input signal into even and odd components. The polyphase matrix will be used later on to build a very flexible wavelet transform.

Now let's move on to the right part of figure 1, the inverse wavelet transform. Here we have to deal with upsampling after which some filtering is performed. Upsampling is nothing more than inserting a zero in between every two samples and the consequence of this is that the filter will perform a lot of multiplications by zero, again a waste of computing time. Since the idea of moving the subsampling in front of the filters worked rather well for the forward wavelet transform, we will try a similar approach for the inverse wavelet transform, i.e. moving the upsampling behind the filters.

We look again at equation (2) but now imagine it as being the result of filtering an upsampled sequence of samples. If we assume that the inserted zeroes are the odd samples, then all the terms with an odd-numbered $x(z)$ vanish and we can divide the output samples $y(z)$ in odd and even sequences

$$\begin{aligned} y_e(z) &= h_e(z) x_e(z) \\ z \cdot y_o(z) &= h_o(z) x_e(z) \end{aligned}. \tag{6}$$

The "delay" z in front of the odd-numbered output samples is due to the delay between odd and even samples, necessary to merge the two sequences into the output stream $y(z)$. As in the subsampling case we now apply this result to the reconstruction filter stage on the right in figure 1 and write down an equation similar to (4):

$$\begin{pmatrix} y_e(z) \\ z y_o(z) \end{pmatrix} = P(z) \begin{pmatrix} \lambda_e(z) \\ \gamma_e(z) \end{pmatrix}, \tag{7}$$

where $P(z)$ is a second polyphase matrix, the dual of the first,

$$P(z) = \begin{pmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{pmatrix}. \tag{8}$$

---

[1] The term polyphase comes from digital filter theory where it is used to describe the splitting of a sequence of samples into several subsequences which can be processed in parallel. The splitting is done using modulo arithmetic: sample $x(n)$ is routed to subsequence k if $(n+k) \bmod M = 0$, $0 \le k < M$. In our case $M = 2$. The subsequences can be seen as phase-shifted versions of each other, hence the name [Che95].

Note that when we ignore the tildes in (5), (8) is the transposed version of (5). This polyphase matrix performs the inverse wavelet transform. In case of the femmelet transform P(z) will be the unit matrix as well.
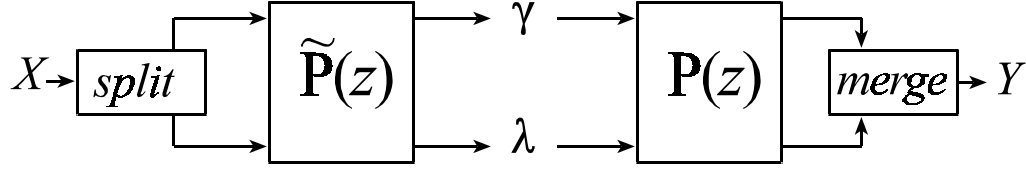

Figure 3

A one-stage filter bank for signal analysis and reconstruction using polyphase matrices.

In figure 3 the filter stage of figure 1 is redrawn, this time using the polyphase matrices. The delays we had in equations (4) and (7) are incorporated in the *split-* and *merge* boxes. From this figure it will be clear that the condition for perfect reconstruction now can be written as

$$\widetilde{P}(z^{-1})P(z) = I.$$ (9)

Here one of the two polyphase matrices was time-reversed, because in order for (9) to hold we need to cancel the delays caused by the polyphase matrices (a FIR filter is a delay line, see figure 2).

If we assume that P(z) is invertible and if we use Cramer's rule to calculate its inverse, we find

$$P(z)^{-1} = \widetilde{P}(z^{-1}) = \frac{1}{h_e(z)g_o(z) - h_o(z)g_e(z)} \begin{pmatrix} g_o(z) & -g_e(z) \\ -h_o(z) & h_e(z) \end{pmatrix}.$$ (10)

From this it follows that if we demand that the determinant of P(z) = 1, i.e. $h_e(z)g_o(z)$ - $h_o(z)g_e(z)$ = 1, then not only will P(z) be invertible, but also

$$\begin{aligned} \widetilde{h}_e(z) &= g_o(z^{-1}) \\ \widetilde{h}_o(z) &= -g_e(z^{-1}) \\ \widetilde{g}_e(z) &= -h_o(z^{-1}) \\ \widetilde{g}_o(z) &= h_e(z^{-1}) \end{aligned},$$ (11)

which implies that[2]

---

[2] This follows easily once we know that the polyphase representation of a sequence of samples $x(z)$ is given by $x(z) = x_e(z^2) + z^{-1}x_o(z^2)$. This equation has, by the way, an interesting property. If we express $x_e(z^2)$ and $x_o(z^2)$ in $x(z)$ we arrive at the beautiful result $x_e(z^2) = \frac{1}{2}[x(z) + x(-z)]$ and $x_o(z^2) = \frac{1}{2z^{-1}}[x(z) - x(-z)]$, i.e. the polyphase representation can be viewed as a digital equivalent of Euler's formula.

$$\tilde{h}(z) = -z^{-1}g(-z^{-1})$$
$$\tilde{g}(z) = z^{-1}h(-z^{-1}) \quad . \tag{12}$$

In the special case that $h = \tilde{h}$ and $g = \tilde{g}$ the wavelet transform is orthogonal, otherwise it is biorthogonal.

If the polyphase matrix has a determinant of 1, then the filter pair (h,g) is called *complementary*. If the filter pair (h,g) is complementary, so is the filter pair ($\tilde{h}$, $\tilde{g}$).

Note that if the determinant of P(z) = 1 the filters $h_e(z)$ and $h_o(z)$ have to be relatively prime and we will exploit this property in the section on filter factoring. Of course the pairs $g_e(z)$ and $g_o(z)$, $h_e(z)$ and $g_e(z)$ and $h_o(z)$ and $g_o(z)$ will also be relatively prime.

Summarizing we can state that the problem of finding an invertible wavelet transform using FIR filters amounts to finding a matrix P(z) with determinant 1. From this matrix the four filters needed in the invertible wavelet transform follow immediately. Compare this to the definition of the continuous wavelet transform at the beginning of the wavelet tutorial. We sure have come a long way! But there is more to come.

4. Intermezzo: Laurent polynomials

As an intermezzo some algebra will now be presented, which we will need in the following sections.

The z-transform of a FIR filter is given by

$$h(z) = \sum_{k=p}^{q} h_k z^{-k} \quad . \tag{13}$$

This summation is also known as a Laurent polynomial or Laurent series[3]. A Laurent polynomial differs from a normal polynomial in that it can have negative exponents. The degree of a Laurent polynomial h is defined as

$$|h| = q - p \,, \tag{14}$$

so that the length of the filter is equal to the degree of the associated polynomial plus one. Note that the Laurent polynomial z^p has degree zero. The sum or difference of two Laurent polynomials is again a Laurent polynomial and the product of two Laurent polynomials of degree a and b is a Laurent polynomial of degree $a+b$. Exact division is in general not possible, but division with remainder is. This means that for any two Laurent polynomials $a(z)$ and $b(z) \neq 0$, with $|a(z)| \geq |b(z)|$ there will always exist a Laurent polynomial $q(z)$ with $|q(z)| = |a(z)| - |b(z)|$, and a Laurent polynomial $r(z)$ with $|r(z)| < |b(z)|$ so that

$$a(z) = b(z)q(z) + r(z) \,. \tag{15}$$

---

[3] Like the Taylor series, Laurent series can be used to expand functions in.

This division is not necessarily unique.

Finally we remark that a Laurent polynomial is invertible if and only if it is of degree zero, i.e. if it is a monomial.

In the following sections we will unleash the power of algebra on the polyphase matrices. The result will be an extremely powerful algorithm to build wavelet transforms.

5. Lifting

As will be clear from our intermezzo the polyphase matrix is a matrix of Laurent polynomials and since we demanded that its determinant be equal to 1, we know that the filter pair (h,g) is complementary. The *lifting theorem* [Dau97] now states that any other finite filter g^{new} complementary to h is of the form

$$g^{new}(z) = g(z) + h(z)s(z^2), \tag{16}$$

where $s(z^2)$ is a Laurent polynomial. This can be seen very easily if we write g^{new} in polyphase form (see also the footnote with equation (12)) and assemble the new polyphase matrix as

$$\mathbf{P}^{new}(z) = \begin{pmatrix} h_e(z) & h_e(z)s(z) + g_e(z) \\ h_o(z) & h_o(z)s(z) + g_o(z) \end{pmatrix} = \mathbf{P}(z)\begin{pmatrix} 1 & s(z) \\ 0 & 1 \end{pmatrix}. \tag{17}$$

As can be easily verified the determinant of the new polyphase matrix also equals 1, which proofs (16).

Similarly, we can apply the lifting theorem to create the filter $\tilde{h}^{new}(z)$ complementary to $\tilde{g}(z)$ (recall the identities from (11))

$$\tilde{h}^{new}(z) = \tilde{h}(z) + \tilde{g}(z)\tilde{s}(z^2), \tag{18}$$

with the new dual polyphase matrix given by

$$\tilde{\mathbf{P}}^{new}(z) = \begin{pmatrix} \tilde{h}_e(z) + \tilde{g}_e(z)\tilde{s}(z) & \tilde{h}_o(z) + \tilde{g}_o(z)\tilde{s}(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{pmatrix} = \begin{pmatrix} 1 & \tilde{s}(z) \\ 0 & 1 \end{pmatrix}\tilde{\mathbf{P}}(z). \tag{19}$$

What we just did is called *primal lifting*, we lifted the low-pass subband with the help of the high-pass subband. Figure 4 shows the effect of primal lifting graphically.
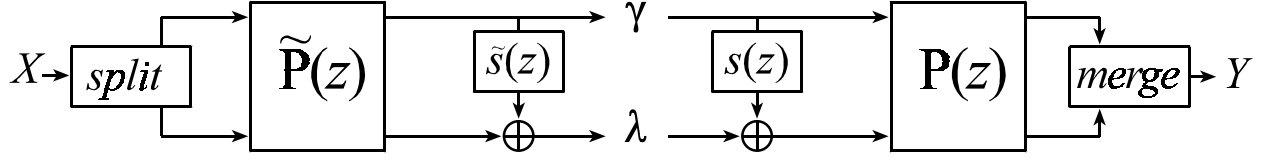
Figure 4

Primal lifting, the low-pass subband is lifted with the help of the high-pass subband.

We can also go the other way, that is lifting the high-pass subband with the help of the low-pass subband and then it is called *dual lifting*. For dual lifting the equations become

$$h^{new}(z) = h(z) + g(z)t(z^2), \tag{20}$$

$$P^{new}(z) = \begin{pmatrix} h_e(z) + g_e(z)t(z) & g_e(z) \\ h_o(z) + g_o(z)t(z) & g_o(z) \end{pmatrix} = P(z)\begin{pmatrix} 1 & 0 \\ t(z) & 1 \end{pmatrix}, \tag{21}$$

and

$$\widetilde{g}^{new}(z) = \widetilde{g}(z) + \widetilde{h}(z)\widetilde{t}(z^2), \tag{22}$$

$$\widetilde{P}^{new}(z) = \begin{pmatrix} \widetilde{h}_e(z) & \widetilde{h}_o(z) \\ \widetilde{g}_e(z) + \widetilde{h}_e(z)\widetilde{t}(z) & \widetilde{g}_o(z) + \widetilde{h}_o(z)\widetilde{t}(z) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \widetilde{t}(z) & 1 \end{pmatrix}\widetilde{P}(z). \tag{23}$$

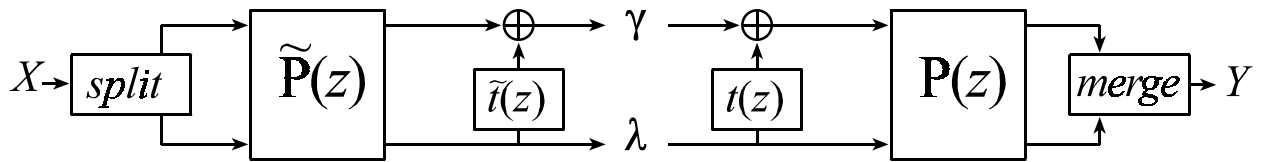Dual lifting can be graphically displayed as in figure 5.

Figure 5

Dual lifting, the high-pass subband is lifted with the help of the low-pass subband.

After all these equations and figures it should be clear how things work in the lifting scheme and we can explain why this technique is called lifting. If we start for example with the femmelet transform then both the polyphase matrices are simply equal to the unit matrix. When we apply a primal- and/or a dual lifting step to the femmelet transform we get a new wavelet transform which is a little more sophisticated. In other words, we have *lifted* the wavelet transform to a higher level of sophistication. We can perform as many lifting steps as we like and therefore we can build highly sophisticated wavelet transforms.

The inverse lifting transform now also begins to take shape. If we start with a femmelet transform we only split the input stream into an even and an odd stream. Then we lift one of these streams as in the left of figures 4 and 5 by applying a Laurent polynomial to the other and adding it to the first. We can very easily undo this lifting step by again applying the same Laurent polynomial to the other stream and then subtract it from the first. In other words, inverting a lifting transform is the same as changing all the signs of the lifting Laurent polynomials in figures 4 and 5 and run it backwards, i.e. start at the output. Inverting the lifting scheme this way will always work! From this we can conclude that $t(z) = -\tilde{t}(z)$ and $s(z) = -\tilde{s}(z)$.

From the figures 4 and 5 we can see another interesting property of lifting. Every time we apply a primal or dual lifting step we add something to one stream. All the samples in the stream are replaced by new samples and at any time we need only the current streams to update sample values. In other words, the whole transform can be done in-place, without the need for auxiliary memory. This is the same as with the fast Fourier transform, where the transformed data also takes the same place as the input data. This in-place property makes the lifting wavelet transform very attractive for use in embedded applications, where memory and board space are still expensive.

We conclude this section with a note on terminology. In lifting literature the dual lifting step is also referred to as the *predict step*, while the primal lifting step is also referred to as the *update step*. The idea behind this terminology is that lifting of the high-pass subband with the low-pass subband can be seen as prediction of the odd samples from the even samples. One assumes that, especially at the first steps, consecutive samples will be highly correlated so that it should be possible to predict the odd ones from the even ones (or the other way around). The update step, i.e. lifting the low-pass subband with the high-pass subband, then is done to keep some statistical properties of the input stream, usually at least the average, of the low-pass subband.

6. Factoring filters

In the previous section we lifted a wavelet transform to a more sophisticated level. Of course we can also do the opposite, i.e. we can factor the FIR filters of an existing wavelet transform into lifting steps. This would be very useful because a lot of research has already been performed on designing wavelet filters for all kinds of applications and factoring these filters will allow us to benefit easily from this research. So how do we go about?

Starting with for instance equation (22) we can go the other way by writing

$$\tilde{g}(z) = \tilde{h}(z)\tilde{t}(z^2) + \tilde{g}^{new}(z) . \tag{24}$$

The form of (24) is identical to a long division with remainder of Laurent polynomials, where $\tilde{g}^{new}(z)$ is the remainder. If we rewrite equation (23) a little as well, we obtain

$$\tilde{P}(z) = \begin{pmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{h}_e(z)t(z) + \tilde{g}_e^{new}(z) & \tilde{h}_o(z)t(z) + \tilde{g}_o^{new}(z) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \tilde{t}(z) & 1 \end{pmatrix} \tilde{P}^{new}(z) , \tag{25}$$

and we see that for the polyphase matrix we have to perform two long divisions in order to extract one lifting step. Once we have extracted one such lifting step we can continue by extracting more lifting steps from the new polyphase matrix until we end up with the unit matrix, or a matrix with only two constants on its main diagonal. In fact, in [Dau97] it is proven that it will always be possible to do this when we start with a complementary filter pair (h,g), i.e. P(z) can always be factored into lifting steps:

$$P(z) = \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix} \prod_{i=1}^{m} \left\{ \begin{pmatrix} 1 & s_i(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ t_i(z) & 1 \end{pmatrix} \right\}. \tag{26}$$

In equation (26) K_1 and K_2 are scaling constants unequal to zero. If scaling is not desired for some reason, it is even possible to factor the scaling matrix into four more lifting steps, one of which can be combined with the last real lifting step so that factoring a scaling matrix costs three extra lifting steps [Dau97].

To perform a long division with remainder on Laurent polynomials we can use the Euclidean algorithm for Laurent polynomials[4]. The Euclidean algorithm was originally developed to find the greatest common divisor (*gcd*) of two natural numbers, but we can extend it to Laurent polynomials as well. In [Dau97] it is given as:

Take two Laurent polynomials a(z) and b(z) ≠ 0 with |a(z)| ≥ |b(z)|. Let $a_0(z) = a(z)$ and $b_0(z) = b(z)$ and iterate the following steps starting from i = 0

$$\begin{aligned} a_{i+1}(z) &= b_i(z) \\ b_{i+1}(z) &= a_i(z) \% b_i(z) \\ q_{i+1}(z) &= a_i(z)/b_i(z) \end{aligned} \tag{27}$$

Then $a_n(z) = gcd(a(z),b(z))$ where n is the smallest number for which $b_n(z) = 0$.

The '%'-symbol in the second line of (27) means mod, i.e. integer divide with remainder but only keeping the remainder, and it is the same symbol as used in the C programming language for the mod operation. The '/'-symbol in the third line is the C-language div operator. The result of this algorithm can be written as

$$\begin{pmatrix} a(z) \\ b(z) \end{pmatrix} = \prod_{i=1}^{n} \begin{pmatrix} q_i(z) & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_n(z) \\ 0 \end{pmatrix}, \tag{28}$$

which looks very much like a series of lifting steps. The gcd found might not be unique since it is defined only up to a factor z^p, i.e. there are several factorizations possible. This turns out to be an advantage because it allows us to select the factoring which best suits our needs.

---

[4] Why? We write the long division with remainder of a_0 and a_1 as $a_0 = q_1 \cdot a_1 + a_2$. But we can express a_1 in a similar way as $a_1 = q_2 \cdot a_2 + a_3$ and a_2 also, and so on. The row of remainders will eventually reach zero, $a_1 > a_2 > \ldots > a_n > a_{n+1} = 0$, and this is where it stops. The gcd of a_0 and a_1 is now a_n. However, we are more interested in the intermediate results $a_0 = q_1 \cdot (q_2 \cdot (\ldots (q_n \cdot a_n + a_{n+1}) + \ldots) + a_3) + a_2$ (remember $a_{n+1} = 0$) or written as in (28) [Coh95].

7. Example

To summarize the theory of the previous sections now a detailed example will be presented.

Suppose we are given the following wavelet transform filters:

$$\tilde{h}(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4} + \frac{1}{4}z - \frac{1}{8}z^2$$
$$\tilde{g}(z) = \frac{1}{4}z^{-2} - \frac{1}{2}z^{-1} + \frac{1}{4}$$

$$(29)$$

which we want to use in a lifting scheme. What do we have to do?

The first step is to assemble the corresponding polyphase matrix. Because we have read all the footnotes we recall that the polyphase representation is given by

$$x(z) = x_e(z^2) + z^{-1}x_o(z^2),$$

$$(30)$$

and apply this to the two analyzing filters to obtain

$$\tilde{h}(z) = \underbrace{\left\{ -\frac{1}{8}z^{-2} + \frac{3}{4} - \frac{1}{8}z^2 \right\}}_{\tilde{h}_e(z^2)} + z^{-1}\underbrace{\left\{ \frac{1}{4} + \frac{1}{4}z^2 \right\}}_{\tilde{h}_o(z^2)}$$
$$\tilde{g}(z) = \underbrace{\left\{ \frac{1}{4}z^{-2} + \frac{1}{4} \right\}}_{\tilde{g}_e(z^2)} + z^{-1}\underbrace{\left\{ -\frac{1}{2} \right\}}_{\tilde{g}_o(z^2)}$$

$$(31)$$

which means that

$$\tilde{h}_e(z) = -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z \quad \tilde{h}_o(z) = \frac{1}{4} + \frac{1}{4}z$$
$$\tilde{g}_e(z) = \frac{1}{4}z^{-1} + \frac{1}{4} \quad\quad \tilde{g}_o(z) = -\frac{1}{2}$$

$$(32)$$

and thus

$$\tilde{P}(z) = \begin{pmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{1}{4}z \\ \frac{1}{4}z^{-1} + \frac{1}{4} & -\frac{1}{2} \end{pmatrix}.$$

$$(33)$$

With the help of (11) we can now assemble the synthesizing polyphase matrix as well. First we check the determinant of (33):

$$\left(-\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z \right)\left(-\frac{1}{2} \right) - \left(\frac{1}{4} + \frac{1}{4}z \right)\left(\frac{1}{4}z^{-1} + \frac{1}{4} \right) = -\frac{1}{2},$$

$$(34)$$

so we have to scale (11) a bit before using the equalities to get:

$$P(z) = \begin{pmatrix} 1 & \frac{1}{2}z^{-1} + \frac{1}{2} \\ \frac{1}{2} + \frac{1}{2}z & \frac{1}{4}z^{-1} - \frac{3}{2} + \frac{1}{4}z \end{pmatrix}.$$

(35)

Do not get confused here, $P(z)$ is not time-reversed! Remember, if we want to check (9) we have to time-reverse one of the polyphase matrices. From (35) we can find the synthesizing filters as follows:

$$h_e(z) = 1 \qquad h_o(z) = \frac{1}{2} + \frac{1}{2}z$$
$$g_e(z) = \frac{1}{2}z^{-1} + \frac{1}{2} \qquad g_o(z) = \frac{1}{4}z^{-1} - \frac{3}{2} + \frac{1}{4}z ,$$

(36)

thus

$$h(z) = \underbrace{\{1\}}_{h_e(z^2)} + z^{-1}\underbrace{\left\{\frac{1}{2} + \frac{1}{2}z^2\right\}}_{h_o(z^2)}$$
$$g(z) = \underbrace{\left\{\frac{1}{2}z^{-2} + \frac{1}{2}\right\}}_{g_e(z^2)} + z^{-1}\underbrace{\left\{\frac{1}{4}z^{-2} - \frac{3}{2} + \frac{1}{4}z^2\right\}}_{g_o(z^2)} ,$$

(37)

so that

$$h(z) = \frac{1}{2}z^{-1} + 1 + \frac{1}{2}z$$
$$g(z) = \frac{1}{4}z^{-3} + \frac{1}{2}z^{-2} - \frac{3}{2}z^{-1} + \frac{1}{2} + \frac{1}{4}z .$$

(38)

Note that it is not necessary to actually calculate the synthesizing filters because of the simple reversibility of the forward transform. We have done it here just to illustrate how things work.

The next step is the factoring of the polyphase matrices into lifting steps. We start with the extraction of a dual lifting step[5].

$$\tilde{P}(z) = \begin{pmatrix} \tilde{h}_e^{new}(z) & \frac{1}{4} + \frac{1}{4}z \\ \tilde{g}_e^{new}(z) & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \tilde{t}(z) & 1 \end{pmatrix},$$

(39)

which means that we have to solve the following two equations:

$$-\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z = \tilde{t}(z)\left(\frac{1}{4} + \frac{1}{4}z\right) + \tilde{h}_e^{new}(z)$$
$$\frac{1}{4}z^{-1} + \frac{1}{4} = \tilde{t}(z)\left(-\frac{1}{2}\right) + \tilde{g}_e^{new}(z) .$$

(40)

---

[5] Why? Because $\tilde{h}(z)$ is the longest filter.

We use the Euclidean algorithm with $a_0 = \tilde{h}_e(z)$ and $b_0 = \tilde{h}_o(z)$ and perform one step. Now note that there are three possibilities for q_1, and thus for b_1, depending on which two terms of a_0 you want to match with b_0:

$$-\frac{1}{8}z^{-1}+\frac{3}{4}-\frac{1}{8}z = \begin{cases} \left(-\frac{1}{2}z^{-1}+\frac{7}{2}\right) & \left(\frac{1}{4}+\frac{1}{4}z\right) & -z \\ \left(-\frac{1}{2}z^{-1}-\frac{1}{2}\right) & \left(\frac{1}{4}+\frac{1}{4}z\right) & +1 \\ \left(\frac{7}{2}z^{-1}-\frac{1}{2}\right) & \left(\frac{1}{4}+\frac{1}{4}z\right) & -z^{-1} \end{cases} \tag{41}$$

Note also that we have found three greatest common divisors. If we choose the middle line of (41) as the factorization we have a symmetrical one, which goes nicely with $\tilde{g}(z)$ as well, and we arrive at the following decomposition:

$$\tilde{P}(z) = \begin{pmatrix} 1 & \frac{1}{4}+\frac{1}{4}z \\ 0 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \frac{1}{2}z^{-1}-\frac{1}{2} & 1 \end{pmatrix}. \tag{42}$$

We can continue by extracting a primal lifting step. For this we apply the Euclidean algorithm to $\tilde{g}_e(z)$ and $\tilde{g}_o(z)$ of (42), almost not worth mentioning it, and find:

$$\tilde{P}(z) = \begin{pmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{4}+\frac{1}{4}z \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \frac{1}{2}z^{-1}-\frac{1}{2} & 1 \end{pmatrix}. \tag{43}$$

This equation gives a fully factored version of the filters from (29). If we finally use (43) with (4) we can display our wavelet transform graphically as in figure 6 while the corresponding wavelet and scaling function are displayed in figure 7.
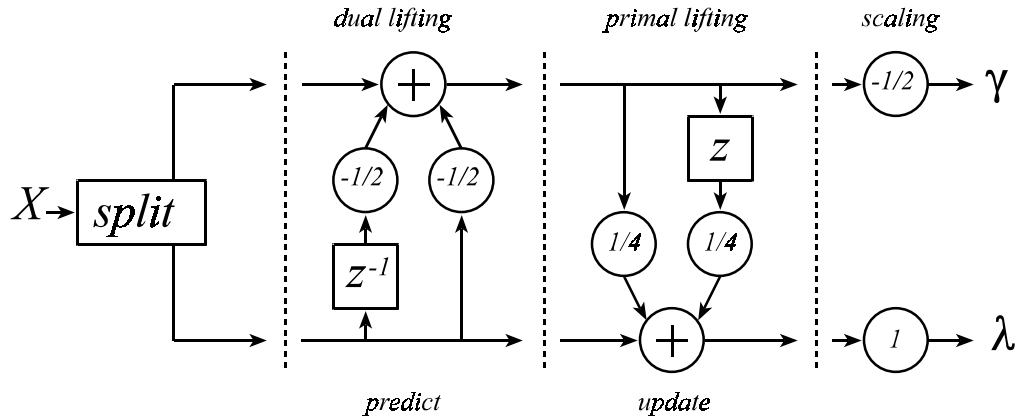

Figure 6
The implementation of the wavelet transform of this example.

From figure 6 we can generalize the lifting steps as:

$$\gamma^{new}(z) = \gamma(z) + \tilde{t}(z)\lambda(z),$$
$$\lambda^{new}(z) = \lambda(z) + \tilde{s}(z)\gamma(z),$$

(44)

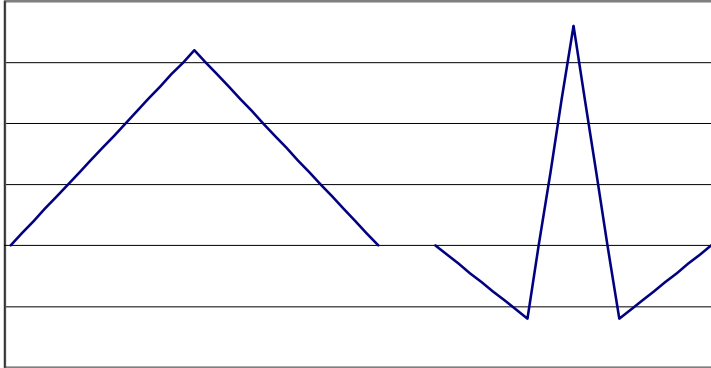to emphasize the in-place calculation property of the lifting transform.

Figure 7
The scaling function (left) and the wavelet (right) that go with this example.

8. Lifting properties

The lifting scheme has some properties which are not found in many other transforms. Figure 6 shows a few of these properties and we will now take a look at some of the most important ones.

The inverse transform is immediately clear: change the signs of all the scaling factors, replace "*split*" by "*merge*" and go from right to left, i.e. reverse the data flow. This easy invertibility is always true for the lifting scheme.

Lifting can be done in-place (see 42): we never need samples other than the output of the previous lifting step and therefore we can replace the old stream by the new stream at every summation point. Not immediately clear from this figure is that when we iterate a filter bank using in-place lifted filters we end up with *interlaced coefficients*. This can be seen as follows. We split the input in odd- and even-numbered samples and perform the in-place lifting steps. After one complete step the high-pass filtered samples, the wavelet coefficients, sit in the odd-numbered places and the low-pass filtered samples sit in the even-numbered places. Then we perform another transform step, but only using the low-pass filtered samples, so that this sequence will again be divided into odd- and even-numbered samples. Again the odd-numbered samples are transformed into wavelet coefficients, while the even-numbered samples will be processed further so that in the end all wavelet coefficients will be interlaced.

The third important property has not been mentioned yet, but it shows clearly from figure 6: lifting is not causal. Usually this is not really a problem, we can always delay the signal enough to make it causal, but it will never be real-time. In some cases however it is possible to design a causal lifting transform.

The last important property I will mention here is the calculation complexity. In [Dau97] it is proven that for long filters the lifting scheme cuts computation complexity in half, compared to the standard iterated FIR filter bank algorithm. This type of wavelet transform has already a complexity of N, in other words, much more efficient than the FFT with its complexity of $N\log(N)$ and lifting speeds things up with another factor of two. This is where the title of this tutorial comes from: it is a fast wavelet transform and therefore we will refer to it as the *fast lifting wavelet transform* of *FLWT*.

9. Integer lifting

The last stage of our voyage to the ultimate[6] wavelet transform is the stage where we make sure that the wavelet coefficients are integers. In classical transforms, including the non-lifted wavelet transforms, the wavelet coefficients are assumed to be floating point numbers. This is due to the filter coefficients used in the transform filters, which are usually floating point numbers. In the lifting scheme it is however rather easy to maintain integer data, although the dynamic range of the data might increase. That this is possible in the lifting scheme has to do with the easy invertibility property of lifting.

The basic lifting step is given in (44) and we rewrite it here a little modified as [Uyt97]:

$$x^{new}(z) \leftarrow x(z) + s(z)y(z) . \tag{45}$$

Because the signal part $y(z)$ is not changed by the lifting step, the result of the filter operation can be rounded, and we can write:

$$x^{new}(z) \leftarrow x(z) + \lfloor s(z)y(z) \rfloor, \tag{46}$$

where we have used $\lfloor \cdot \rfloor$ to denote the rounding operation. Equation (46) is fully reversible:

$$x(z) \leftarrow x^{new}(z) - \lfloor s(z)y(z) \rfloor, \tag{47}$$

and this shows the most amazing feature of integer lifting: whatever rounding operation is used, the lifting operation will always be reversible.

We have to take care however, because we did not consider the scaling step in the previous paragraph. Scaling usually does not yield integer results but it is a part of the lifting transform. The simplest solution to this problem is to forget all about scaling and just keep in mind that the transform coefficients actually have to be scaled. This is important for instance in denoising applications. If scaling is ignored, then it is desirable to let the scaling factor be

---

[6] That is, in our limited world, i.e. the context of this tutorial.

as close to one as possible. This can be done using the non-uniqueness of the lifting factorization. Another solution is to factor the scaling into lifting steps as well [Dau97].

As mentioned before the integer lifting transform can not guarantee the preservation of the dynamic range of the input signal. Usually the dynamic range doubles [Uyt97c], but there are schemes that can keep the dynamic range. In [Cha96] an interesting lifting transform with the so-called *property of precision preservation* (*PPP*) is described. This transform makes use of the two-complement representation of integers in a computer and the wrap-around overflows cause in this representation. The disadvantage of such a transform is that large coefficients may be represented by small values and it is therefore difficult to take decisions on coefficient values.

10. Coda

We have now finished our self-imposed task of transforming the CWT into a practical implementation. In this tutorial we have seen how we can use the lifting scheme to build a very versatile wavelet transform. After first optimizing the subsampled and upsampled FIR filters from the wavelet tutorial, through the use of some algebra we arrived at a scheme to build a wavelet transform using primal and dual lifting blocks. These modules allowed us to build any wavelet transform, which fits in the classical framework, and more. Adapting the lifting scheme we will be well armed: amongst our weaponry are such elements as[7] easy invertibility of any transform, in-place calculation of the transform and easy integer transform coefficients without losing any of its features. And there are many more features [Cal96], [Uyt97b], [Dau97].

However, this does not mean that this is the only way to go. There are probably as many wavelet transforms as there are wavelets. Due to the infinite variety of wavelets it is possible to design a transform which maximally exploits the properties of a specific wavelet[8], and of course this has been done. While researching wavelet theory I stumbled upon morlets[9], coiflets, wavelants, slantlets, brushlets and wavelet packets to name a few. The lifting scheme on the other hand is a really general scheme, which makes it very suitable for experimenting while the in-place and integer properties make it extremely useful for embedded systems where memory is still expensive. With the application described in this report in mind, it will be clear that these are the reasons for studying the lifting scheme.

Finally, four remarks to conclude this tutorial:

• In [Uyt97b] the filter factoring algorithm is used to split the original filters in simpler filters and one primal and dual lifting step. These lifting steps are then used to make the original wavelet transform integer. This is some kind of hybrid (trans)form but very effective.

• Up to now we have only spoken about one-dimensional transforms. It is however easily possible to extend the lifting transform to the multi-dimensional case. Not only can the lifting transform be used in a classical

---

[7] "*Our chief weapon is surprise ... surprise and fear... fear and surprise ... Our* **two** *weapons are fear and surprise ... and ruthless efficiency ... Our* **three** *weapons are fear, surprise, and ruthless efficiency ... and an almost fanatical devotion to the Pope ... Our* **four** *... no ...* **Amongst** *our weapons ... Amongst our weaponry ... are such elements as fear, surprise ...*" From Monthy Python's Flying Circus, series 2, episode 15, "The Spanish Inquisition" (1970).

[8] Compare this to the Fourier transform, where a sine will always be a sine.

[9] Very funny indeed, Morlet [Mor82] being more or less the inventer of wavelets, but used as such in [Wei94].

separable multi-dimensional setting, but it can be made truly multi-dimensional. In [Uyt97a] the lifting transform is extended to a true two-dimensional transform, while in [Kov97] the complete theoretical foundations are laid out for any dimension. The principles behind lifting do not change at all in the multi-dimensional setting.

- One of the advantages of the lifting scheme as pointed out in for instance [Dau97] and [Cal96] is that the lifting scheme allows for an introduction into wavelet theory without the use of Fourier theory. I do not agree with this on the grounds that from the lifting scheme it is totally unclear why there should be wavelets in it at all. The concept of wavelets is completely unnecessary to understand the lifting scheme and therefore, I feel, it should not be used as an introduction to wavelet theory.

- The lifting scheme is constantly under development and is investigated by many. Recent additions are the lifting scheme in a redundant setting in order to improve the translation invariance [Sto98] and adaptive prediction schemes for integer lifting [Cla97].

11. References

Books and papers

[Cal96] Calderbank, A. R. and I. Daubechies, W. Sweldens, B.-L. Yeo
WAVELET TRANSFORMS THAT MAP INTEGERS TO INTEGERS.
Proceedings of the IEEE Conference on Image Processing. Preprint, 1996.
IEEE Press, 1997. To appear.

[Che95] Chen, W.-K., editor.
THE CIRCUITS AND FILTERS HANDBOOK.
Boca Raton, Fl (USA): CRC Press, 1995.
The Electrical Engineering Handbook Series.

[Cla97] Claypoole, R. and G. Davis, W. Sweldens, R. Baraniuk.
NONLINEAR WAVELET TRANSFORMS FOR IMAGE CODING.
Asilomar Conference on Signals, Systems, and Computers. Preprint, 1997.
To appear.

[Dau97] Daubechies, I. and W. Sweldens.
FACTORING WAVELET TRANSFORMS INTO LIFTING STEPS.
J. Fourier Anal. Appl., Vol. 4, Nr. 3, 1998, preprint.

[Kov97] Kovacevic, J. and W. Sweldens
WAVELET FAMILIES OF INCREASING ORDER IN ARBITRARY DIMENSIONS.
To appear in IEEE Transactions on Image Processing. Preprint 1997.

[Mor82] Morlet, J. and G. Arens, I. Fourgeau, D. Giard.
WAVE PROPAGATION AND SAMPLING THEORY.
Geophysics, Vol. 47 (1982), p. 203-236.

[Sto98] Stoffel, A.

REMARKS ON THE UNSUBSAMPLED WAVELET TRANSFORM AND THE LIFTING SCHEME.
Elsevier Science. Preprint, 1998.

[Swe96a]   Sweldens, W.
THE LIFTING SCHEME: A CONSTRUCTION OF SECOND GENERATION WAVELETS.
Siam J. Math. Anal, Vol. 29, No. 2 (1997). Preprint, 1996.

[Swe96b]   Sweldens, W.
BUILDING YOUR OWN WAVELETS AT HOME.
In: Wavelets in Computer Graphics.
ACM SIGGRAPH Course Notes, 1996.

[Uyt97a]   Uytterhoeven G. and A. Bultheel.
THE RED-BLACK WAVELET TRANSFORM.
Technical report TW271, Department of Computer Science.
Leuven: Katholieke Universiteit Leuven, 1997.

[Uyt97b]   Uytterhoeven G. and F. Van Wulpen, M. Jansen, D. Roose, A. Bultheel.
WAILI: WAVELETS WITH INTEGER LIFTING.
Technical report TW262, Department of Computer Science.
Leuven: Katholieke Universiteit Leuven, 1997.

[Uyt97c]   Uytterhoeven G. and D. Roose, A. Bultheel.
WAVELET TRANSFORMS USING THE LIFTING SCHEME.
Report ITA-Wavelets-WP1.1, Department of Computer Science.
Leuven: Katholieke Universiteit Leuven, 1997.

[Wei94]   Weiss, L. G.
WAVELETS AND WIDEBAND CORRELATION PROCESSING.
IEEE Signal Processing Magazine, January (1994), p. 13-32.

Internet resources

Besides classical references there are many Internet sources that deal with wavelets. Here I have listed a few that have proved to be useful. With these links probably every other wavelet related site can be found. Keep in mind however that this list was compiled in July 1999.

- The *Wavelet Digest*, a monthly electronic magazine currently edited by Wim Sweldens [Swe96a,b], is a platform for people working with wavelets. It contains announcements of conferences, abstracts of publications and preprints and questions and answers of readers. It can be found at www.wavelet.org/wavelet/index.html and it is *the* site for wavelets.

- *Mathsoft®*, the makers of *Mathcad®*, maintain a wavelet site called wavelet resources, which contains a huge list of wavelet-related papers and links. It is located at www.mathsoft.com/wavelets.html

- Rice University, the home of Burrus [Bur98] et al, keeps a list of publications and makes available the Rice Wavelet Toolbox for *MatLab®* at www-dsp.rice.edu/publications/

- The Katholieke Universiteit of Leuven, Belgium, is active on the net with wavelets, publications and the toolbox WAILI, at www.cs.kuleuven.ac.be/~wavelets/

- Amara Graps maintains a long list of links, publications and tools besides explaining wavelet theory in a nutshell at www.amara.com/current/wavelet.html

- There is a real lifting page, dedicated to Liftpack, a lifting toolbox at
  www.cs.sc.edu/~fernande/liftpack/index.html

- Finally I would like to mention an interesting tutorial aimed at engineers by Robi Polikar from Iowa State University at www.public.iastate.edu/~rpolikar/WAVELETS/