Part II

Wavelets and applications to image processing

Chapter 5

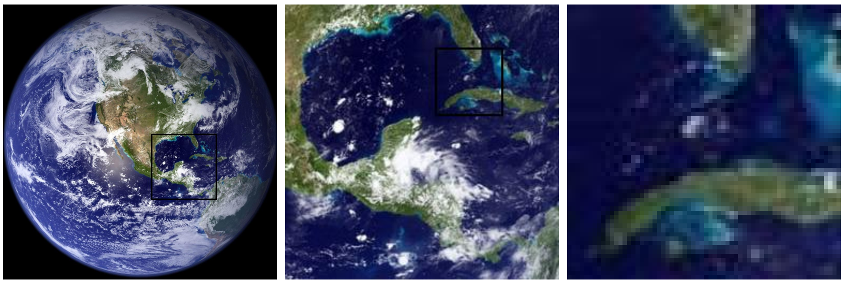
Motivation for wavelets

In Part I our focus was to approximate functions or vectors with trigonometric functions. We saw that the Discrete Fourier transform could be used to obtain the representation of a vector in terms of such functions, and that computations could be done efficiently with the FFT algorithm. This was useful for analyzing, filtering, and compressing sound and other discrete data. The approach with trigonometric functions has some limitations, however. One of these is that, in a representation with trigonometric functions, the frequency content is fixed over time. This is in contrast with much sound data, where frequency characteristics are completely different in different parts. We have also seen that, even if a sound has a simple representation in terms of trigonometric functions on two different parts, the representation of the entire function may be more complex. In particular, if the function is nonzero only on a very small interval, a representation of it in terms of trigonometric functions is not so simple.

In this chapter we are going to introduce the basic properties of an alternative to Fourier analysis, namely wavelets. Similar to Fourier analysis, wavelets are also based on the idea of expressing a function in some basis. But in contrast to Fourier analysis, where the basis is fixed, wavelets provide a general framework with many different types of bases. In this chapter we first give a motivation for wavelets, before we continue by introducing the simplest wavelet we have. We then establish a more general framework from the experiences from this wavelet. In the following chapters we then consider more general wavelets.

5.1 Why wavelets?

Figure 5.1(a) shows a view of the entire Earth. The startup image in Google Earth$^{\text{TM}}$, a program for viewing satellite images, maps and other geographic information, is very similar to this. In (b) we have zoomed in on the Mexican Gulff, as marked with a rectangle in (a). Similarly, in (c) we have further zoomed in on Cuba and a small portion of Florida, as marked with a rectangle in (b). There is clearly an amazing amount of information available behind a



(a) Image of the earth

(b) Zoomed in on the rectange marked in (a)

(c) Zoomed further in on the rectange marked in (b)

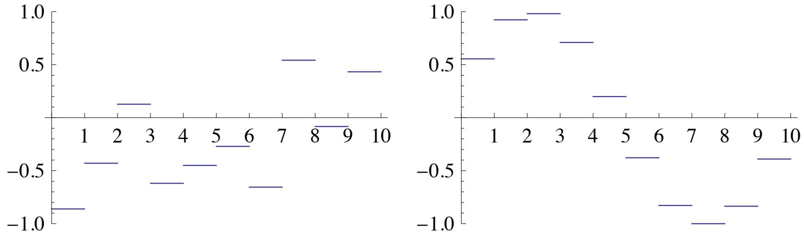
Figure 5.1: A view of Earth from space, together with versions of the image where we have zoomed in.

program like Google Earth$^{\text{TM}}$, since we there can zoom further in, and obtain enough detail to differentiate between buildings and even trees or cars all over the Earth. So, when the Earth is spinning in the opening screen of Google Earth$^{\text{TM}}$, all the Earth's buildings appear to be spinning with it! If this was the case the Earth would not be spinning on the screen, since there would just be so much information to process that a laptop would not be able to display a rotating Earth.

There is a simple reason that the globe can be shown spinning in spite of the huge amounts of information that need to be handled. We are going to see later that a digital image is just a rectangular array of numbers that represent the colour at a dense set of points. As an example, the images in Figure 5.1 are made up of a grid of 1064×1064 points, which gives a total of 1 132 096 points. The colour at a point is represented by three eight-bit integers, which means that the image files contain a total of 3 396 288 bytes each. So regardless of how close to the surface of the Earth our viewpoint is, the resulting image always contains the same number of points. This means that when we are far away from the Earth we can use a very coarse model of the geographic information that is being displayed, but as we zoom in, we need to display more details and therefore need a more accurate model.

Observation 5.1. When discrete information is displayed in an image, there is no need to use a mathematical model that contains more detail than what is visible in the image.

A consequence of Observation 5.1 is that for applications like Google Earth$^{\text{TM}}$ we should use a mathematical model that makes it easy to switch between different levels of detail, or different resolutions. Such models are called *multiresolution models*, and wavelets are prominent examples of this kind of models. We will see that multiresolution models also provide us with means of approximation functions, just as Taylor series and Fourier series. Our new approximation



(a)

(b)

Figure 5.2: Two examples of piecewise constant functions.

scheme differs from these in one important respect, however: When we approximate with Taylor series and Fourier series, the error must be computed at the same data points as well, so that the error contains just as much information as the approximating function, and the function to be approximated. Multiresolution models on the other hand will be defined in such a way that the error and the "approximating function" each contain half of the information from the function we approximate, i.e. their amount of data is reduced. This property makes multiresolution models attractive for the problems at hand, when compared to approaches such as Taylor series and Fourier series.

When we zoom in with Google Earth$^{\mathrm{TM}}$, it seems that this is done continuously. The truth is probably that the program only has representations at some given resolutions (since each representation requires memory), and that one interpolates between these to give the impression of a continuous zoom. In the coming chapters we will first look at how we can represent the information at different resolutions, so that only new information at each level is included.

We will now turn to the specifics of the simplest wavelet we have. Its construction goes in the following steps: First we introduce what we call resolution spaces, then the detail spaces, and the wavelet functions.

5.2 Resolution spaces

The starting point is the space of piecewise constant functions on an interval $[0, N)$.

Definition 5.2 (The resolution space V_0)**.** Let N be a natural number. The resolution space V_0 is defined as the space of functions defined on the interval $[0, N)$ that are constant on each subinterval $[n, n+1)$ for $n = 0, \ldots, N-1$.

Note that this also corresponds to piecewise constant functions which are periodic with period N. We will, just as we did in Fourier analysis, identify a function defined on $[0, N)$ with its (period N) periodic extension. Two examples of functions in V_0 for $N = 10$ are shown in Figure 5.2. It is easy to check that

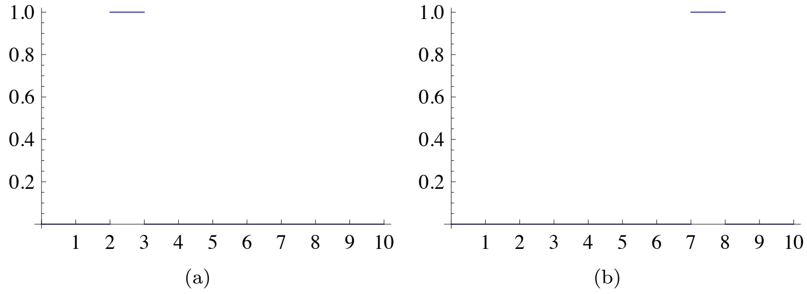


Figure 5.3: The functions ϕ_2 (a) and ϕ_7 (b) in V_0.

V_0 is a linear space, and for computations it is useful to know the dimension of the space and have a basis.

Lemma 5.3. Define the function $\phi(t)$ by

$$\phi(t) = \begin{cases} 1, & \text{if } 0 \leq t < 1; \\ 0, & \text{otherwise;} \end{cases} \tag{5.1}$$

and set $\phi_n(t) = \phi(t-n)$ for any integer n. The space V_0 has dimension N, and the N functions $\{\phi_n\}_{n=0}^{N-1}$ form an orthonormal basis for V_0 with respect to the standard inner product

$$\langle f, g \rangle = \int_0^N f(t)g(t)\,dt. \tag{5.2}$$

In particular, any $f \in V_0$ can be represented as

$$f(t) = \sum_{n=0}^{N-1} c_n \phi_n(t) \tag{5.3}$$

for suitable coefficients $(c_n)_{n=0}^{N-1}$. The function ϕ_n is referred to as the *characteristic* function of the interval $[n, n+1)$

Note the small difference between the inner product we define here from the inner product we used for functions previously: Here there is no scaling $1/T$ involved. Also, for wavelets we will only consider real functions, and the inner product will therefore not be defined for complex functions. Two examples of the basis functions defined in Lemma 5.5 are shown in Figure 5.3.

Proof. Two functions ϕ_{n_1} and ϕ_{n_2} with $n_1 \neq n_2$ clearly satisfy $\int \phi_{n_1}(t)\phi_{n_2}(t)dt = 0$ since $\phi_{n_1}(t)\phi_{n_2}(t) = 0$ for all values of x. It is also easy to check that $\|\phi_n\| = 1$

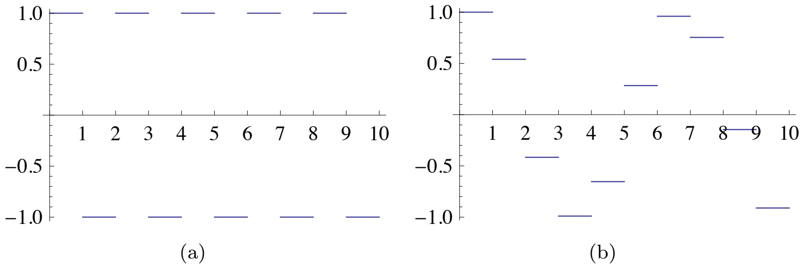


Figure 5.4: The square wave in V_0 (a) and an approximation to $\cos t$ from V_0.

for all n. Finally, any function in V_0 can be written as a linear combination the functions ϕ_0, ϕ_1, …, ϕ_{N-1}, so the conclusion of the lemma follows. □

In our discussion of Fourier analysis, the starting point was the function $\sin 2\pi t$ that has frequency 1. We can think of the space V_0 as being analogous to this function: The function $\sum_{n=0}^{N-1}(-1)^n\phi_n(t)$ is (part of the) square wave that we discussed in Chapter 1, and which also oscillates regularly like the sine function, see Figure 5.4 (a). The difference is that we have more flexibility since we have a whole space at our disposal instead of just one function — Figure 5.4 (b) shows another function in V_0.

In Fourier analysis we obtained a linear space of possible approximations by including sines of frequency 1, 2, 3, …, up to some maximum. We use a similar approach for constructing wavelets, but we double the frequency each time and label the spaces as V_0, V_1, V_2, …

Definition 5.4 (Refined resolution spaces)**.** The space V_m for the interval $[0, N)$ is the space of piecewise linear functions defined on $[0, N)$ that are constant on each subinterval $[n/2^m, (n+1)/2^m)$ for $n = 0, 1, \dots, 2^mN - 1$.

Some examples of functions in the spaces V_1, V_2 and V_3 for the interval $[0, 10]$ are shown in Figure 5.5. As m increases, we can represent smaller details. In particular, the function in (d) is a piecewise constant function that oscillates like $\sin 2\pi 2^2 t$ on the interval $[0, 10]$.

It is easy to find a basis for V_m, we just use the characteristic functions of each subinterval.

Lemma 5.5. Let $[0, N)$ be a given interval with N some positive integer, and let V_m denote the resolution space of piecewise constant functions for some integer $m \geq 0$. Then the dimension of V_m is 2^mN. Define the functions

$$\phi_{m,n}(t) = 2^{m/2}\phi(2^mt - n), \quad \text{for } n = 0, 1, \dots, 2^mN - 1, \tag{5.4}$$

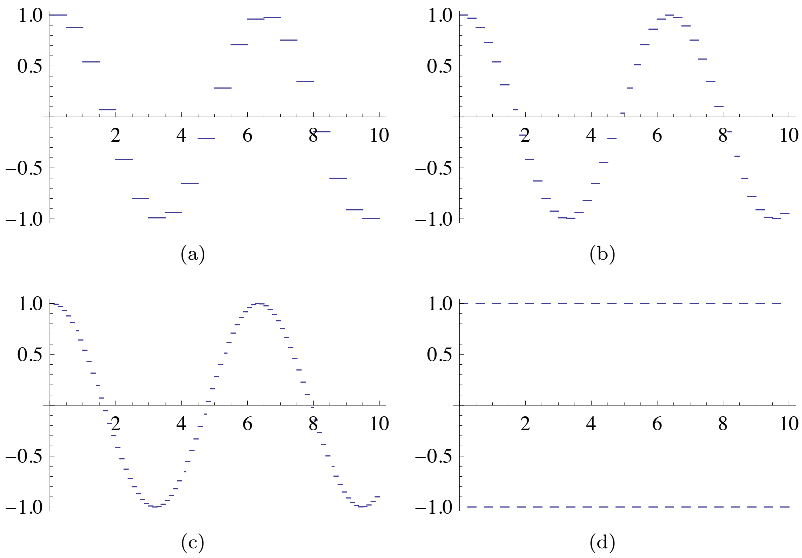


Figure 5.5: Piecewise constant approximations to $\cos t$ on the interval $[0, 10]$ in the spaces V_1 (a), V_2 (b), and V_3 (c). The plot in (d) shows the square wave in V_2.

where ϕ is the characteristic function of the interval $[0, 1]$. The functions $\{\phi_{m,n}\}_{n=0}^{2^m N-1}$ form an orthonormal basis for V_m, and any function $f \in V_m$ can be represented as

$$f(t) = \sum_{n=0}^{2^m N-1} c_n \phi_{m,n}(t)$$

for suitable coefficients $(c_n)_{n=0}^{2^m N-1}$.

Proof. The functions given in (5.19) are exactly the characteristic functions of the subintervals $[n/2^m, (n+1)/2^m)$ which we referred to in Definition 5.4, so the proof is very similar to the proof of Lemma 5.5. The one mysterious thing may be the normalisation factor $2^{-m/2}$. This comes from the fact that

$$\int_0^N \phi(2^m t - n)^2 \, dt = \int_{n/2^m}^{(n+1)/2^m} \phi(2^m t - n)^2 \, dt = 2^{-m} \int_0^1 \phi(u)^2 \, du = 2^{-m}.$$

The normalisation therefore ensures that $\|\phi_{m,n}\| = 1$ for all m. □

In the theory of wavelets, the function ϕ is also called a *scaling function*. The origin behind this name is that the scaled (and translated) functions $\phi_{m,n}$ of ϕ are used as basis functions for the refined resolution spaces. Later on we will see that other scaling functions ϕ can be chosen, where the scaled versions $\phi_{m,n}$ will be used to define similar resolution spaces, with slightly different properties.

5.2.1 Function approximation property

Each time m is increased by 1, the dimension of V_m doubles, and the subinterval on which the functions in V_m are constant are halved in size. It therefore seems reasonable that, for most functions, we can find good approximations in V_m provided m is big enough.

Theorem 5.6. Let f be a given function that is continuous on the interval $[0, N]$. Given $\epsilon > 0$, there exists an integer $m \geq 0$ and a function $g \in V_m$ such that

$$\bigl|f(t) - g(t)\bigr| \leq \epsilon$$

for all t in $[0, N]$.

Proof. Since f is (uniformly) continuous on $[0, N]$, we can find an integer m so that $\bigl|f(t_1) - f(t_2)\bigr| \leq \epsilon$ for any two numbers t_1 and t_2 in $[0, N]$ with $|t_1 - t_2| \leq 2^{-m}$. Define the approximation g by

$$g(t) = \sum_{n=0}^{2^m N-1} f\bigl(t_{m,n+1/2}\bigr) \phi_{m,n}(t),$$

where $t_{m,n+1/2}$ is the midpoint of the subinterval $\big[n2^{-m},(n+1)2^{-m}\big)$,

$$t_{m,n+1/2} = (n+1/2)2^{-m}.$$

For t in this subinterval we then obviously have $|f(t)-g(t)| \leq \epsilon$, and since these intervals cover $[0,N]$, the conclusion holds for all $t \in [0,N]$. □

Theorem 5.6 does not tell us how to find the approximation g although the proof makes use of an approximation that interpolates f at the midpoint of each subinterval. Note that if we measure the error in the L^2-norm, we have

$$\|f-g\|^2 = \int_0^N \big|f(t)-g(t)\big|^2\,dt \leq N\epsilon^2,$$

so $\|f-g\| \leq \epsilon\sqrt{N}$. We therefore have the following corollary.

Corollary 5.7. Let f be a given continuous function on the interval $[0,N]$ and let $\operatorname{proj}_{V_m}(f)$ denote the best approximation to f from V_m. Then

$$\lim_{m\to\infty} \|f - \operatorname{proj}_{V_m}(f)\| = 0.$$

Figure 5.6 illustrates how some of the approximations of the function $f(x) = x^2$ from the resolution spaces for the interval $[0,1]$ improve with increasing m.

5.2.2 Detail spaces and wavelets

So far we have described a family of function spaces that allow us to determine arbitrarily good approximations to a continuous function. The next step is to introduce the so-called detail spaces and the wavelet functions. For this we focus on the two spaces V_0 and V_1.

We start by observing that since

$$[n,n+1) = [2n/2,(2n+1)/2) \cup [(2n+1)/2,(2n+2)/2)$$

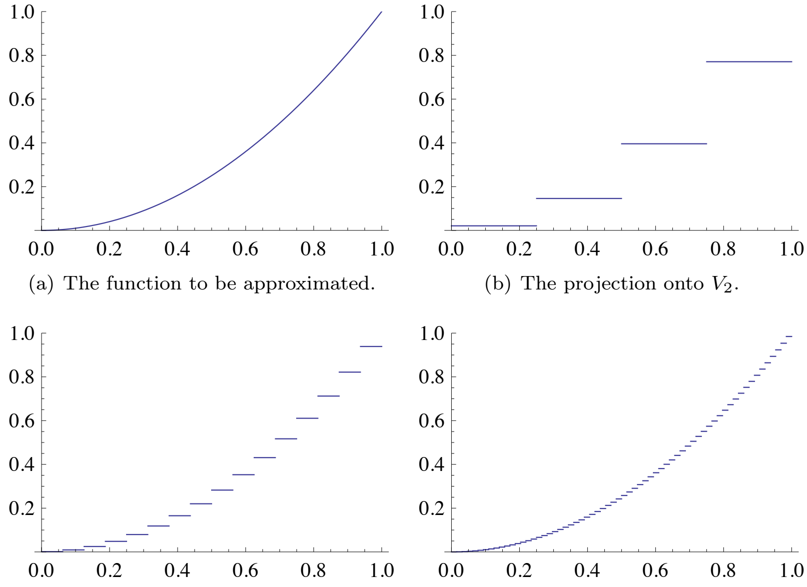
we have

$$\phi_{0,n} = \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{\sqrt{2}}\phi_{1,2n+1}. \tag{5.5}$$

This provides a formal proof of the intuitive observation that $V_0 \subset V_1$. For if $g \in V_0$, then we can write

$$g(t) = \sum_{n=0}^{N-1} c_n\phi_{0,n}(t) = \sum_{n=0}^{N-1} c_n\big(\phi_{1,2n}+\phi_{1,2n+1}\big)/\sqrt{2}.$$

The right-hand side clearly lies in V_1. A similar argument shows that $V_k \subset V_{k+1}$ for any integer $k \geq 0$.



(a) The function to be approximated.

(b) The projection onto V_2.

(c) The projection onto V_4.

(d) The projection onto V_6.

Figure 5.6: Comparison of the function defined by $f(t) = t^2$ on $[0, 1]$ with the projection onto different spaces V_m.

Lemma 5.8. The spaces V_0, V_1, ..., V_m, ... are nested,

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \cdots .$$

The next step is to investigate what happens if we start with a function g_1 in V_1 and project this to an approximation g_0 in V_0.

Lemma 5.9. Let $\operatorname{proj}_{V_0}$ denote the orthogonal projection onto the subspace V_0. Then the projection of a basis function $\phi_{1,n}$ is given by

$$\operatorname{proj}_{V_0}(\phi_{1,n}) = \begin{cases} \phi_{0,n/2}/\sqrt{2}, & \text{if } n \text{ is even;} \\ \phi_{0,(n-1)/2}/\sqrt{2}, & \text{if } n \text{ is odd.} \end{cases} \tag{5.6}$$

If $g_1 \in V_1$ is given by

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n}, \tag{5.7}$$

then

$$\operatorname{proj}_{V_0}(g_1) = g_0 = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n}$$

where $c_{0,n}$ is given by

$$c_{0,n} = \frac{c_{1,2n} + c_{1,2n+1}}{\sqrt{2}}. \tag{5.8}$$

Proof. We first observe that $\phi_{1,n}(t) \neq 0$ if and only if $n/2 \leq t < (n+1)/2$. Suppose that n is even. Then the intersection

$$\left[\frac{n}{2}, \frac{n+1}{2}\right) \cap [n_1, n_1 + 1) \tag{5.9}$$

is nonempty only if $n_1 = \frac{n}{2}$. Using the orthogonal decomposition formula we get

$$\begin{aligned} \operatorname{proj}_{V_0}(\phi_{1,n}) &= \sum_{k=0}^{N-1} \langle \phi_{1,n}, \phi_{0,k} \rangle \phi_{0,k} = \langle \phi_{1,n}, \phi_{0,n_1} \rangle \phi_{0,n_1} \\ &= \int_{n/2}^{(n+1)/2} \sqrt{2}\, dt\, \phi_{0,n/2} = \frac{1}{\sqrt{2}}\phi_{0,n/2}. \end{aligned}$$

When n is odd, the intersection (5.9) is nonempty only if $n_1 = (n-1)/2$, which gives the second formula in (5.6) in the same way.

We project the function g_1 in V_1 using the formulas in (5.6). We split the sum in (5.7) into even and odd values of n,

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} = \sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1}. \tag{5.10}$$

We can now apply the two formulas in (5.6),

$$\begin{aligned}
\mathrm{proj}_{V_0}(g_1) &= \mathrm{proj}_{V_0}\left(\sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1}\right) \\
&= \sum_{n=0}^{N-1} c_{1,2n}\,\mathrm{proj}_{V_0}(\phi_{1,2n}) + \sum_{n=0}^{N-1} c_{1,2n+1}\,\mathrm{proj}_{V_0}(\phi_{1,2n+1}) \\
&= \sum_{n=0}^{N-1} c_{1,2n}\phi_{0,n}/\sqrt{2} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{0,n}/\sqrt{2} \\
&= \sum_{n=0}^{N-1} \frac{c_{1,2n}+c_{1,2n+1}}{\sqrt{2}}\phi_{0,n}
\end{aligned}$$

which proves (5.8) □

When $g_1 \in V_1$ is projected onto V_0, the result $g_0 = \mathrm{proj}_{V_0} g_1$ is in general different from g_0. We can write $g_1 = g_0 + e_0$, where $e_0 = g_1 - g_0$ represents the error we have commited in making this projection. e_0 lies in the ortogonal complement of V_0 in V_1 (in particular, $e_0 \in V_1$).

Definition 5.10. We will denote by W_0 the orthogonal complement of V_0 in V_1. We also call W_0 a *detail space*

The name detail space is used since $e_0 \in W_0$ can be considered as the detail which is left out when considering g_0 instead of g_1 (due to the expression $g_1 = g_0 + e_0$). Since V_0 and W_0 are mutually orthogonal spaces they are also linearly independent spaces, so that we can use the following definition to rewrite the space V_0:

Definition 5.11 (Direct sum of vector spaces)**.** Assume that $U, V \subset W$ are vector spaces, and that U and V are mutually linearly independent. By $U \oplus V$ we mean the vector space consisting of all vectors of the form $\boldsymbol{u} + \boldsymbol{v}$, where $\boldsymbol{u} \in U$, $\boldsymbol{v} \in V$. We will also call $U \oplus V$ the *direct sum* of U and V.

This definition also makes sense if we have several vector spaces, since the direct sum clearly obeys the associate law $U \oplus (V \oplus W) = (U \oplus V) \oplus W$, i.e. we can define $U \oplus V \oplus W = U \oplus (V \oplus W)$. We will have use for this use of direct sum of several vector space in the next section.

We can now write $V_1 = V_0 \oplus W_0$, i.e. the resolution space V_1 is the direct sum of the lower order resolution space V_0, and the detail space W_0. The expression $g_1 = g_0 + e_0$ is thus a decomposition into a low-resolution approximation, and the details which are left out in this approximation. In the context of our Google Earth$^{\mathrm{TM}}$example, in Figure 5.1 you should interpret g_0 as the image in (a), g_1 as the image in (b), and e_0 as the additional details which are needed to reproduce (b) from (a). While Lemma 5.12 explained how we can compute the low level approximation g_0 from g_1, the next result states how we can compute the detail/error e_0 from g_1.

Lemma 5.12. With W_0 the orthogonal complement of V_0 in V_1, set

$$\hat{\psi}_{0,n} = \frac{\phi_{1,2n} - \phi_{1,2n+1}}{2}$$

for $n = 0, 1, \ldots, N-1$. Then $\hat{\psi}_{0,n} \in W_0$ and

$$\operatorname{proj}_{W_0}(\phi_{1,n}) = \begin{cases} \hat{\psi}_{0,n/2}, & \text{if } n \text{ is even;} \\ -\hat{\psi}_{0,(n-1)/2}, & \text{if } n \text{ is odd.} \end{cases} \tag{5.11}$$

If $g_1 \in V_1$ is given by $g_1 = \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n}$, then

$$\operatorname{proj}_{W_0}(g_1) = e_0 = \sum_{n=0}^{N-1} \hat{w}_{0,n}\hat{\psi}_{0,n}$$

where $\hat{w}_{0,n}$ is given by

$$\hat{w}_{0,n} = c_{1,2n} - c_{1,2n+1}. \tag{5.12}$$

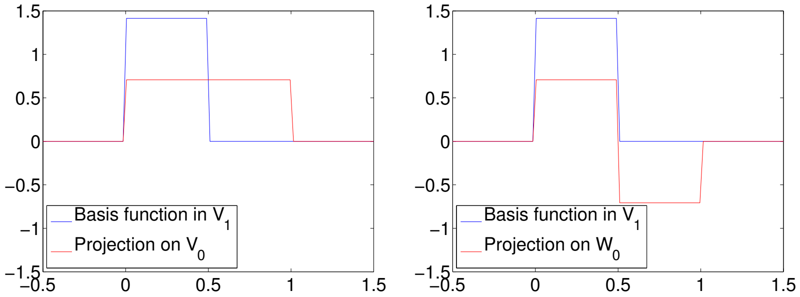
Proof. We start by determining the error when $\phi_{1,n}$, for n even, is projected onto V_0. The error is then

$$\begin{aligned} \operatorname{proj}_{W_0}(\phi_{1,n}) &= \phi_{1,n} - \frac{\phi_{0,n/2}}{\sqrt{2}} \\ &= \phi_{1,n} - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}\phi_{1,n} + \frac{1}{\sqrt{2}}\phi_{1,n+1}\right) \\ &= \frac{1}{2}\phi_{1,n} - \frac{1}{2}\phi_{1,n+1} \\ &= \hat{\psi}_{0,n/2}. \end{aligned}$$

Here we used the relation (5.6) in the second equation. When n is odd we have

$$\begin{aligned} \operatorname{proj}_{W_0}(\phi_{1,n}) &= \phi_{1,n} - \frac{\phi_{0,(n-1)/2}}{\sqrt{2}} \\ &= \phi_{1,n} - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}\phi_{1,n-1} + \frac{1}{\sqrt{2}}\phi_{1,n}\right) \\ &= \frac{1}{2}\phi_{1,n} - \frac{1}{2}\phi_{1,n-1} \\ &= -\hat{\psi}_{0,(n-1)/2}. \end{aligned}$$

For a general function g_1 we first split the sum into even and odd terms as



(a) $\phi_{1,0} \in V_1$ together with its projection onto V_0

(b) $\phi_{1,0} \in V_1$ together with its projection onto W_0

Figure 5.7: The projection of a basis function in V_1 onto V_0 and W_0.

in (5.10) and then project each part onto W_0,

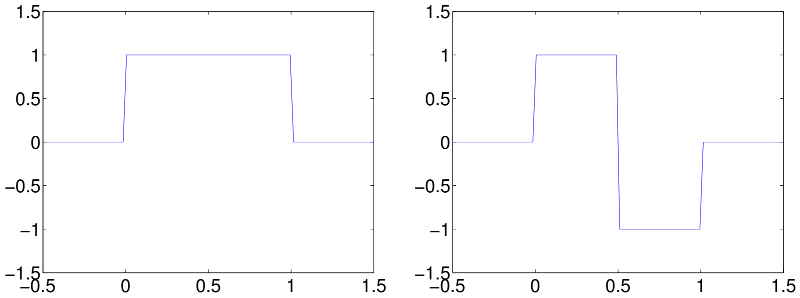
$$\begin{aligned}
\operatorname{proj}_{W_0}(g_1) &= \operatorname{proj}_{W_0}\left(\sum_{n=0}^{N-1} c_{1,2n}\phi_{1,2n} + \sum_{n=0}^{N-1} c_{1,2n+1}\phi_{1,2n+1}\right)\\
&= \sum_{n=0}^{N-1} c_{1,2n}\operatorname{proj}_{W_0}(\phi_{1,2n}) + \sum_{n=0}^{N-1} c_{1,2n+1}\operatorname{proj}_{W_0}(\phi_{1,2n+1})\\
&= \sum_{n=0}^{N-1} c_{1,2n}\hat{\psi}_{0,n} - \sum_{n=0}^{N-1} c_{1,2n+1}\hat{\psi}_{0,n}\\
&= \sum_{n=0}^{N-1} (c_{1,2n} - c_{1,2n+1})\hat{\psi}_{0,n}
\end{aligned}$$

which is (5.12) □

In Figure 5.7 we have useed lemmas 5.9 and 5.12 to plot the projections of $\phi_{1,0} \in V_1$ onto V_0 and W_0. It is an interesting exercise to see from the plots why exactly these functions should be least-squares approximations of $\phi_{1,n}$. It is also an interesting exercise to prove the following from lemmas 5.9 and 5.12:

Proposition 5.13. Let $f(t) \in V_1$, and let $f_{n,1}$ be the value f attains on $[n, n+1/2)$, and $f_{n,2}$ the value f attains on $[n+1/2, n+1)$. Then $\operatorname{proj}_{V_0}(f)$ is the function in V_0 which equals $(f_{n,1} + f_{n,2})/2$ on the interval $[n, n+1)$. Moreover, $\operatorname{proj}_{W_0}(f)$ is the function in W_0 which is $(f_{n,1} - f_{n,2})/2$ on $[n, n+1/2)$, and $-(f_{n,1} - f_{n,2})/2$ on $[n+1/2, n+1)$.

In other words, the projection on V_0 is constructed by averaging on two subintervals, while the projection on W_0 is constructed by taking the difference



(a) The function ϕ

(b) The function ψ

Figure 5.8: The functions we used to analyse the space of piecewise constant functions

from the mean. This sounds like a reasonable candidate for the least-squares approximations. In the exercise we generalize these observations.

Consider the functions $\hat{\psi}_{0,n} = (\phi_{1,2n} - \phi_{1,2n+1})/2$ from Lemma 5.12. They are clearly orthogonal since their nonzero parts do not overlap. We also note that $\|\hat{\psi}_{0,n}\| = \sqrt{2}/2$, since it has absolute value $\sqrt{2}/2$ on two intervals of length $1/2$. The functions defined by $\psi_{0,n}(t) = \sqrt{2}\,\hat{\psi}_{0,n}(t)$ will therefore form an orthonormal set.

Lemma 5.14. Define the function ψ by

$$\psi(t) = \big(\phi_{1,0}(t) - \phi_{1,1}(t)\big)/\sqrt{2} = \phi(2t) - \phi(2t-1) \tag{5.13}$$

and set

$$\psi_{0,n}(t) = \psi(t-n) = \big(\phi_{1,2n}(t) - \phi_{1,2n+1}(t)\big)/\sqrt{2} \quad \text{for } n = 0, 1, \ldots, N-1. \tag{5.14}$$

Then the set $\{\psi_{0,n}\}_{n=0}^{N-1}$ is an orthonormal basis for W_0, the orthogonal complement of V_0 in V_1.

Later we will encounter other functions, which also will be denoted by ψ, and have similar properties as stated in Lemma 5.14. In the theory of wavelets, such ψ are called *mother wavelets*. In Figure 5.8 we have plotted the functions ϕ and ψ. There is one important property of ψ, which we will return to:

Observation 5.15. We have that $\int_0^N \psi(t)dt = 0$.

This can be seen directly from the plot in Figure 5.8, since the parts of the graph above and below the x-axis cancel.

We now have all the tools needed to define the Discrete Wavelet Transform.

Theorem 5.16 (Discrete Wavelet Transform)**.** The space V_1 can be decomposed as the orthogonal sum $V_1 = V_0 \oplus W_0$ where W_0 is the orthogonal complement of V_0 in V_1, and V_1 therefore has the two bases

$$\boldsymbol{\phi}_1 = (\phi_{1,n})_{n=0}^{2N-1} \quad \text{and} \quad (\boldsymbol{\phi}_0, \boldsymbol{\psi}_0) = \big((\phi_{0,n})_{n=0}^{N-1}, (\psi_{0,n})_{n=0}^{N-1}\big).$$

The Discrete Wavelet Transform (DWT) is the change of coordinates from the basis $\boldsymbol{\phi}_1$ to the basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$. If

$$\begin{aligned} g_1 &= \sum_{n=0}^{2N-1} c_{1,n}\phi_{1,n} \in V_1 \\ g_0 &= \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n} \in V_0 \\ e_0 &= \sum_{n=0}^{N-1} w_{0,n}\psi_{0,n} \in W_0 \end{aligned}$$

and $g_1 = g_0 + e_0$, then the DWT is given by

$$\begin{pmatrix} c_{0,n} \\ w_{0,n} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{1,2n} \\ c_{1,2n+1} \end{pmatrix} \tag{5.15}$$

Conversely, the Inverse Discrete Wavelet Transform (IDWT) is the change of coordinates from the basis $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ to the basis $\boldsymbol{\phi}_1$, and is given by

$$\begin{pmatrix} c_{1,2n} \\ c_{1,2n+1} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{0,n} \\ w_{0,n} \end{pmatrix} \tag{5.16}$$

Proof. We have that

$$\begin{aligned} \phi_{1,2n} &= \text{proj}_{V_0}(\phi_{1,2n}) + \text{proj}_{W_0}(\phi_{1,2n}) = \phi_{0,n}/\sqrt{2} + \hat{\psi}_{0,n} \\ &= \phi_{0,n}/\sqrt{2} + \psi_{0,n}/\sqrt{2} \\ \phi_{1,2n+1} &= \text{proj}_{V_0}(\phi_{1,2n+1}) + \text{proj}_{W_0}(\phi_{1,2n+1}) = \phi_{0,n}/\sqrt{2} - \hat{\psi}_{0,n} \\ &= \phi_{0,n}/\sqrt{2} - \psi_{0,n}/\sqrt{2}, \end{aligned}$$

where we have used Equations (5.6) and (5.11). From this it follows that $(\phi_{1,2n}, \phi_{1,2n+1})$ and $(\phi_{0,n}, \psi_{0,n})$ span the same space, and that the corresponding change of coordinate matrix is given by $\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$. This proves Equation (5.15). Equation (5.16) follows immediately since this matrix equals its inverse. □

Exercises for Section 5.2

1. Show that the coordinate vector for $f \in V_0$ in the basis $\{\phi_{0,0}, \phi_{0,1}, \dots, \phi_{0,N-1}\}$ is $(f(0), f(1), \dots. f(N-1))$.

2. In this exercise we will consider the two projections from V_1 onto V_0 and W_0.

a. Consider the projection proj_{V_0} of V_1 onto V_0. Use lemma 5.9 to write down the matrix for proj_{V_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_0$.

b. Consider the projection proj_{W_0} of V_1 onto V_0. Use lemma 5.12 to write down the matrix for proj_{W_0} relative to the bases $\boldsymbol{\phi}_1$ and $\boldsymbol{\psi}_0$.

3. Consider again the projection proj_{V_0} of V_1 onto V_0.

a. Explain why $\mathrm{proj}_{V_0}(\phi) = \phi$ and $\mathrm{proj}_{V_0}(\psi) = 0$.

b. Show that the matrix of proj_{V_0} relative to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ is given by the diagonal matrix where the first half of the entries on the diagonal are 1, the second half 0.

c. Show in a similar way that the projection of V_1 onto W_0 has a matrix relative to $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ given by the diagonal matrix where the first half of the entries on the diagonal are 0, the second half 1.

4. Show that

$$\mathrm{proj}_{V_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1} f(t)dt \right) \phi_{0,n}(t) \tag{5.17}$$

for any f. Show also that the first part of Proposition 5.13 follows from this.

5. Show that

$$\begin{aligned} &\| \sum_n \left(\int_n^{n+1} f(t)dt \right) \phi_{0,n}(t) - f \|^2 \\ &= \langle f, f \rangle - \sum_n \left(\int_n^{n+1} f(t)dt \right)^2 . \end{aligned}$$

This, together with the previous exercise, gives us an expression for the least-squares error for f from V_0 (at least after taking square roots).

6. Show that

$$\mathrm{proj}_{W_0}(f) = \sum_{n=0}^{N-1} \left(\int_n^{n+1/2} f(t)dt - \int_{n+1/2}^{n+1} f(t)dt \right) \psi_{0,n}(t) \tag{5.18}$$

for any f. Show also that the second part of Proposition 5.13 follows from this.

5.3 Higher order resolution spaces

In Section 5 we introduced the important decomposition $V_1 = V_0 \oplus W_0$ which lets us rewrite a function in V_1 as an approximation in V_0 and the corresponding error in W_0, orthogonal to the approximation. The resolution spaces V_m were in fact defined for all integers $m \geq 0$. It turns out that all these resolution spaces can be decomposed in the same way as V_1.

Definition 5.17. The orthogonal complement of V_{m-1} in V_m is denoted W_{m-1}. All the spaces $\{W_k\}_k$ are also called detail spaces.

The first question we will try to answer is how we can, for $f \in V_m$, extract the corresponding detail in W_{m-1}. We first need to define $\psi_{m,n}$ in terms of ψ, similarly to how we defined $\phi_{m,n}$ in terms of ϕ,

$$\psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n), \quad \text{for } n = 0, 1, \ldots, 2^m N - 1. \tag{5.19}$$

As in Lemma 5.14, it is straightforward to prove that $\boldsymbol{\psi}_m = \{\psi_{m,n}\}_{n=0}^{2^m N-1}$ is an orthonormal basis for W_m. Moreover, we have the following result, which is completey analogous to Theorem 5.16.

Theorem 5.18. The space V_m can be decomposed as the orthogonal sum $V_m = V_{m-1} \oplus W_{m-1}$ where W_{m-1} is the orthogonal complement of V_{m-1} in V_m, and V_m has the two bases

$$\boldsymbol{\phi}_m = (\phi_{m,n})_{n=0}^{2^m N-1}$$

and

$$(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) = \big((\phi_{m-1,n})_{n=0}^{2^{m-1}N-1}, (\psi_{m-1,n})_{n=0}^{2^{m-1}N-1}\big).$$

If

$$\begin{aligned} g_m &= \sum_{n=0}^{2^m N-1} c_{m,n}\phi_{m,n} \in V_m, \\ g_{m-1} &= \sum_{n=0}^{2^{m-1}N-1} c_{m-1,n}\phi_{m-1,n} \in V_{m-1}, \\ e_{m-1} &= \sum_{n=0}^{2^{m-1}N-1} w_{m-1,n}\psi_{m-1,n} \in W_{m-1}, \end{aligned}$$

and $g_m = g_{m-1} + e_{m-1}$, then the change of coordinates from the basis $\boldsymbol{\phi}_m$ to the basis $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ is given by

$$\begin{pmatrix} c_{m-1,n} \\ w_{m-1,n} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{m,2n} \\ c_{m,2n+1} \end{pmatrix} \tag{5.20}$$

Conversely, the change of coordinates from the basis $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ to the basis $\boldsymbol{\phi}_m$ is given by

$$\begin{pmatrix} c_{m,2n} \\ c_{m,2n+1} \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} c_{m-1,n} \\ w_{m-1,n} \end{pmatrix} \tag{5.21}$$

We will omit the proof of Theorem 5.20, and only remark that it can be proved by making the substitution $t \to 2^m u$ in Lemma 5.9 and Lemma 5.12, and then following the proof of Theorem 5.16.

Let us return to our interpretation of the Discrete Wavelet Transform as writing a function $g_1 \in V_1$ as a sum of a function $g_0 \in V_0$ at low resolution, and a detail function $e_0 \in W_0$. Theorem 5.20 states similarly how we can write $g_m \in V_m$ as a sum of a function $g_{m-1} \in V_{m-1}$ at lower resolution, and a detail function $e_{m-1} \in W_{m-1}$. The same decomposition can of course be applied to g_{m-1} in V_{m-1}, then to the resulting approximation g_{m-2} in V_{m-2}, and so on,

$$\begin{aligned} V_m &= V_{m-1} \oplus W_{m-1} \\ &= V_{m-2} \oplus W_{m-2} \oplus W_{m-1} \\ &\vdots \\ &= V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}. \end{aligned} \tag{5.22}$$

This change of coordinates corresponds to replacing as many ϕ-functions as we can with ψ-functions, i.e. replacing the original function with a sum of as much detail at different resolutions as possible. Let us give a name to the bases we will use for these direct sums.

Definition 5.19 (Canonical basis for direct sum)**.** Let $C_1, C_2 \ldots, C_n$ be independent vector spaces, and let $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ be corresponding bases. The basis $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$, i.e., the basis where the basis vectors from $\mathcal{B}_i$ are included before $\mathcal{B}_j$ when $i < j$, is referred to as the canonical basis for $C_1 \oplus C_2 \oplus \cdots \oplus C_n$ and is denoted $\mathcal{B}_1 \oplus \mathcal{B}_2 \oplus \ldots \oplus \mathcal{B}_n$.

When we above say "basis for $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}$", we really mean the canonical basis for this space. In general, the Discrete Wavelet Transform is used to denote a change of coordinates from ϕ_m to the canonical basis, for any m.

Definition 5.20 (m-level Discrete Wavelet Transform)**.** Let F_m denote the change of coordinates from $\boldsymbol{\phi}_m$ to the canonical basis

$$\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0 \oplus \boldsymbol{\psi}_1 \oplus \cdots \oplus \oplus \boldsymbol{\psi}_{m-2} \oplus \boldsymbol{\psi}_{m-1}$$

for $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}$. F_m is called a (m-level) *Discrete Wavelet Transform*, or a DWT. After this change of coordinates, the resulting coordinates are called *wavelet coefficients*. The change of coordinates the opposite way is called an (m-level) *Inverse Discrete Wavelet Transform*, or IDWT.

Clearly, this generalizes the Discrete Wavelet Transform defined in Section 5. At each level in a DWT, V_k is split into one part from V_{k-1}, and one part from W_{k-1}. We can visualize this with the following figure, where the arrows represent changes of coordinates:

$$\begin{tikzcd}
V_m \arrow[r] \arrow[dr] & V_{m-1} \arrow[r] \arrow[dr] & V_{m-2} \arrow[r] \arrow[dr] & \cdots \arrow[r] \arrow[dr] & V_0 \\
& W_{m-1} & W_{m-2} & W_{m-3} & W_0
\end{tikzcd}$$

The part from W_{k-1} is not subject to further transformation. This is seen in the figure since W_{m-1} is a leaf node, i.e. there are no arrows going out from W_{m-1}. In a similar illustration for the IDWT, the arrows would go the opposite way. The Discrete Wavelet Transform is the analogue in a wavelet setting to the Discrete Fourier transform. When applying the DFT to a vector of length N, one starts by viewing this vector as coordinates relative to the standard basis. When applying the DWT to a vector of length N, one instead views the vector as coordinates relative to the basis $\boldsymbol{\phi}_m$. This makes sense in light of Exercise 5.1.

The DWT is what is used in practice when transforming a signal using wavelets, and it is straightforward to implement: One simply needs to iterate Equation (5.20) for $m, m-1, \dots, 1$, also at each step, the coordinates in $\boldsymbol{\phi}_{m-1}$ should be placed before the ones in $\boldsymbol{\psi}_{m-1}$, due to the order of the basis vectors in the canonical basis of the direct sum. At each step, only the first coordinates are further transformed. The following function, called `DWTHaarImpl`, follows this procedure. It takes as input the number of levels m, as well as the input vector $\boldsymbol{x}$, runs the m-level DWT on $\boldsymbol{x}$, and returns the result:

```
function xnew=DWTHaarImpl(x,m)
  xnew=x;
  for mres=m:(-1):1
    len=length(xnew)/2^(m-mres);
    c=(xnew(1:2:(len-1))+xnew(2:2:len))/sqrt(2);
    w=(xnew(1:2:(len-1))-xnew(2:2:len))/sqrt(2);
    xnew(1:len)=[c w];
  end
```

Note that this implementation is not recursive, contrary to the FFT. The for-loop here runs through the different resolutions. Inside the loop we perform the change of coordinates from $\boldsymbol{\phi}_k$ to $(\boldsymbol{\phi}_{k-1}, \boldsymbol{\psi}_{k-1})$ by applying Equation (5.20). This works on the first coordinates, since the coordinates from $\boldsymbol{\phi}_k$ are stored first in

$$V_k \oplus W_k \oplus W_{k+1} \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}.$$

Finally, the $\boldsymbol{c}$-coordinates are stored before the $\boldsymbol{w}$-coordinates, again as required by the order in the canonical basis. In this implementation, note that the first levels require the most multiplications, since the latter levels leave an increasing part of the coordinates unchanged. Note also that the change of coordinates matrix is a very sparse matrix: At each level a coordinate can be computed from only two of the other coordinates, so that this matrix has only two nonzero elements in each row/column. The algorithm clearly shows that there is no need to perform a full matrix multiplication to perform the change of coordinates.

The corresponding function for the IDWT, called IDWTHaarImpl, goes as follows:

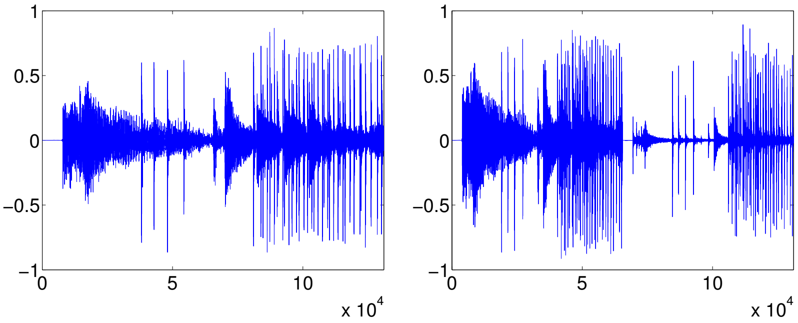
```
function x=IDWTHaarImpl(xnew,m)
  x=xnew;
  for mres=1:m
    len=length(x)/2^(m-mres);
    ev=(x(1:(len/2))+x((len/2+1):len))/sqrt(2);
    od=(x(1:(len/2))-x((len/2+1):len))/sqrt(2);
    x(1:2:(len-1))=ev;
    x(2:2:len)=od;
  end
```

Here the steps are simply performed in the reverse order, and by iterating Equation (5.21).

You may be puzzled by the names DWTHaarImpl and IDWTHaarImpl. In the next sections we will consider other cases, where the underlying function ϕ may be a different function, not necessarily piecewise constant. It will turn out that much of the analysis we have done makes sense for other functions ϕ as well, giving rise to other structures which we also will refer to as wavelets. The wavelet resulting from piecewise constant functions is thus simply one example out of many, and it is commonly referred to as the *Haar wavelet*.

Example 5.21. When you run a DWT you may be led to believe that coefficients from the lower order resolution spaces may correspond to lower frequencies. This sounds reasonable, since the functions $\phi(2^m t - n) \in V_m$ change more quickly than $\phi(t-n) \in V_0$. However, the functions $\phi_{m,n}$ do not correspond to pure tones in the setting of wavelets. But we can still listen to sound from the different resolution spaces. In Exercise 9 you will be asked to implement a function which runs an m-level DWT on the first samples of the sound file castanets.wav, extracts the coefficients from the lower order resolution spaces, transforms the values back to sound samples with the IDWT, and plays the result. When you listen to the result the sound is clearly recognizable for lower values of m, but is degraded for higher values of m. The explanation is that too much of the detail is omitted when you use a higher m. To be more precise, when listening to the sound by throwing away wvereything from the detail spaces $W_0, W_1, \ldots, W_{m-1}$, we are left with a 2^{-m} share of the data. Note that this procedure is mathematically not the same as setting some DFT coefficients to zero, since the DWT does not operate on pure tones.

It is of interest to plot the samples of our test audio file castanets.wav, and compare it with the first order DWT coefficients of the same samples. This is



(a) The 2^{17} first sound samples

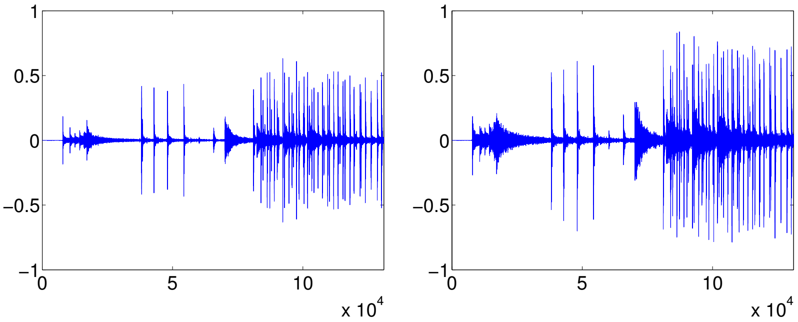
(b) The first order DWT coefficients

Figure 5.9: The sound samples and the DWT coefficients of the sound `castanets.wav`.

shown in Figure 5.9. The first half part of the plot represents the low-resolution approximation of the sound, the second half part represents the detail/error. We see that the detail is quite significant in this case. This means that the first order wavelet approximation does not give a very good approximation to the sound. In the exercises we will experiment more on this.

It is also interesting to plot only the detail/error in the sound, for different resolutions. For this, we must perform a DWT so that we get a representation in the basis $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$, set the coefficicents from V_0 to sero, and transform back with the IDWT. In figure 5.10 the error is shown for the test audio file `castanets.wav` for $m = 1$, $m = 2$. This clearly shows that the error is larger when two levels of the DWT are performed, as one would suspect. It is also seen that the error is larger in the part of the file where there are bigger variations. This also sounds reasonable. ♣

The previous example illustrates that wavelets as well may be used to perform operations on sound. As we will see later, however, our main application for wavelets will be images, where they have found a more important role than for sound. Images typically display variations which are less abrupt than the ones found in sound. Just as the functions above had smaller errors in the corresponding resolution spaces than the sound had, images are thus more suited for for use with wavelets. The main idea behind why wavelets are so useful comes from the fact that the detail, i.e., wavelet coefficients corresponding to the spaces W_k, are often very small. After a DWT one is therefore often left with a couple of significant coefficients, while most of the coefficients are small. The approximation from V_0 can be viewed as a good approximation, even though it contains much less information. This gives another reason why wavelets are popular for images: Detailed images can be very large, but when they are downloaded to a web browser, the browser can very early show a low-resolution of the image, while waiting for the rest of the details in the image to be downloaded. When



(a) $m = 1$

(b) $m = 2$

Figure 5.10: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.

we later look at how wavelets are applied to images, we will need to handle one final hurdle, namely that images are two-dimensional.

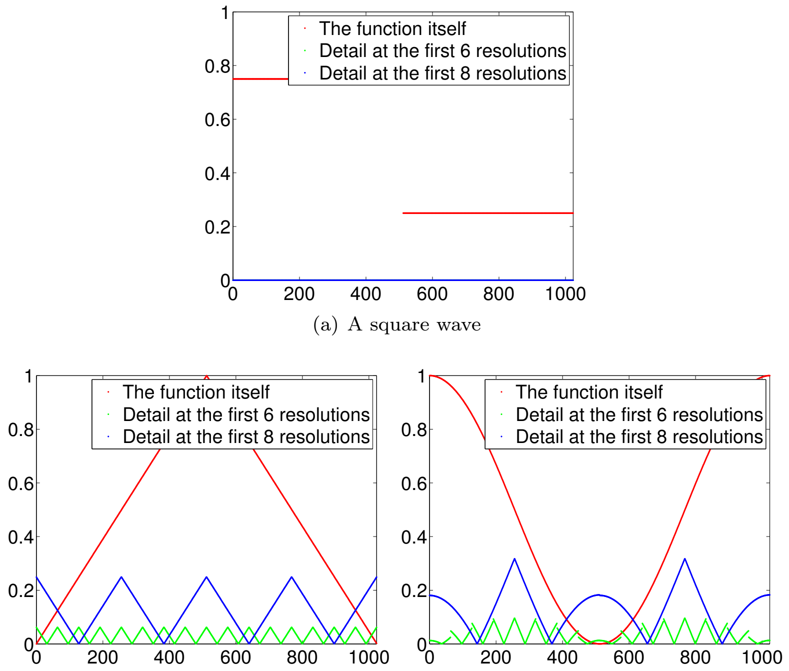
Example 5.22. Above we plotted the DWT coefficients of a sound, as well as the detail/error. We can also experiment with samples generated from a mathematical function. Figure 5.11 plots the error for different functions, with $N = 1024$. In these cases, we see that we require large m before the detail/error becomes significant. We see also that there is no error for the square wave. The reason is that the square wave is a piecewise constant function, so that it can be represented exactly by the ϕ-functions. For the other functions, however, this is not the case, so we here get an error. ♣

Above we used the functions `DWTHaarImpl`, `IDWTHaarImpl` to plot the error. For the functions we plotted in the previous example it is also possible to compute the wavelet coefficients, which we previously have denoted by $w_{m,n}$, exactly. You will be asked to do this in exercises 12 and 13. The following example shows the general procedure which can be used for this:

Example 5.23. Let us compute the wavelet coefficients $w_{m,n}$ for the function $f(t) = 1 - t/N$. This function decreases linearly from 1 to 0 on $[0, N]$. Since the $w_{m,n}$ are coefficients in the basis $\{\psi_{m,n}\}$, it follows by the orthogonal decomposition formula that $w_{m,n} = \langle f, \psi_{m,n} \rangle = \int_0^N f(t)\psi_{m,n}(t)dt$. Using the definition of $\psi_{m,n}$ we get that

$$w_{m,n} = \int_0^N (1 - t/N)\psi_{m,n}(t)dt = 2^{m/2} \int_0^N (1 - t/N)\psi(2^m t - n)dt.$$

Moreover $\psi_{m,n}$ is nonzero only on $[2^{-m}n, 2^{-m}(n+1))$, and is 1 on $[2^{-m}n, 2^{-m}(n+$



(a) A square wave

(b) $f(t) = 1 - 2|1/2 - t/N|$

(c) $f(t) = 1/2 + \cos(2\pi t/N)/2$

Figure 5.11: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m.

$1/2)$), and -1 on $[2^{-m}(n+1/2), 2^{-m}(n+1))$. We can therefore write

$$\begin{aligned}
w_{m,n} &= 2^{m/2}\int_{2^{-m}n}^{2^{-m}(n+1/2)}(1-t/N)dt - 2^{m/2}\int_{2^{-m}(n+1/2)}^{2^{-m}(n+1)}(1-t/N)dt \\
&= 2^{m/2}\left[t-\frac{t^2}{2N}\right]_{2^{-m}n}^{2^{-m}(n+1/2)} - 2^{m/2}\left[t-\frac{t^2}{2N}\right]_{2^{-m}(n+1/2)}^{2^{-m}(n+1)} \\
&= 2^{m/2}\left(2^{-m}(n+1/2)-\frac{2^{-2m}(n+1/2)^2}{2N}-2^{-m}n++\frac{2^{-2m}n^2}{2N}\right) \\
&- 2^{m/2}\left(2^{-m}(n+1)-\frac{2^{-2m}(n+1)^2}{2N}-2^{-m}(n+1/2)+\frac{2^{-2m}(n+1/2)^2}{2N}\right) \\
&= 2^{m/2}\left(\frac{2^{-2m}n^2}{2N}-\frac{2^{-2m}(n+1/2)^2}{N}+\frac{2^{-2m}(n+1)^2}{2N}\right) \\
&= \frac{2^{-3m/2}}{2N}\left(n^2-2(n+1/2)^2+(n+1)^2\right) \\
&= \frac{1}{N2^{2+3m/2}}.
\end{aligned}$$

We see in particular that $w_{m,n} \to 0$ when $m \to \infty$. We see also that there were a lot of computations even in this very simple example. For most functions we therefore usually do not compute $w_{m,n}$ exactly. Instead we use implementations like `DWTHaarImpl`, `IDWTHaarImpl`, and run them on a computer. ♣

Exercises for Section 5.3

1. Generalize exercise 3 to the projections from V_{m+1} onto V_m amd W_m.

2. Show that $f(t) \in V_m$ if and only if $g(t) = f(2t) \in V_{m+1}$.

3. Let $C_1, C_2 \ldots, C_n$ be independent vector spaces, and let $T_i : C_i \to C_i$ be linear transformations. The direct sum of T_1, T_2,…,T_n, written as $T_1 \oplus T_2 \oplus \ldots \oplus T_n$, denotes the linear transformation from $C_1 \oplus C_2 \oplus \cdots \oplus C_n$ to itself defined by

$$T_1 \oplus T_2 \oplus \ldots \oplus T_n(\boldsymbol{c}_1 + \boldsymbol{c}_2 + \cdots + \boldsymbol{c}_n) = T_1(\boldsymbol{c}_1) + T_2(\boldsymbol{c}_2) + \cdots + T_n(\boldsymbol{c}_n)$$

when $\boldsymbol{c}_1 \in C_1$, $\boldsymbol{c}_2 \in C_2$, …, $\boldsymbol{c}_n \in C_n$. Similarly, when $A_1, A_2, \ldots, A_n$ are square matrices, $A_1 \oplus A_2 \oplus \cdots \oplus A_n$ is defined as the block matrix where the blocks along the diagonal are $A_1, A_2, \ldots, A_n$, and where all other blocks are 0. Show that, if $\mathcal{B}_i$ is a basis for C_i then

$$[T_1 \oplus T_2 \oplus \ldots \oplus T_n]_{\mathcal{B}_1 \oplus \mathcal{B}_2 \oplus \ldots \oplus \mathcal{B}_n} = [T_1]_{\mathcal{B}_1} \oplus [T_2]_{\mathcal{B}_2} \oplus \cdots \oplus [T_n]_{\mathcal{B}_n},$$

Here three new concepts are used: a direct sum of matrices, a direct sum of bases, and a direct sum of linear transformations.

4. Assume that T_1 and T_2 are matrices, and that the eigenvalues of T_1 are equal to those of T_2. What are the eigenvalues of $T_1 \oplus T_2$? Can you express the eigenvectors of $T_1 \oplus T_2$ in terms of those of T_1 and T_2?

5. Assume that A and B are square matrices which are invertible. Show that $A \oplus B$ is invertible, and that $(A \oplus B)^{-1} = A^{-1} \oplus B^{-1}$.

6. Let A, B, C, D be square matrices of the same dimensions. Show that $(A \oplus B)(C \oplus D) = (AC) \oplus (BD)$.

7. Assume that you run an m-level DWT on a vector of length r. What value of N does this correspond to? Note that an m-level DWT performs a change of coordinates from V_m to $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}$.

8. Run a 2-level DWT on the first 2^{17} sound samples of the audio file `castanets.wav`, and plot the values of the resulting DWT-coefficients. Compare the values of the coefficients from V_0 with those from W_0 and W_1.

9. In this exercise we will experiment with applying an m-level DWT to a sound file.

a. Write a function

```
function playDWTlower(m)
```

which

1. reads the audio file `castanets.wav`,
2. performs an m-level DWT to the first 2^{17} sound samples of $\boldsymbol{x}$ using the function `DWTHaarImpl`,
3. sets all wavelet coefficients representing detail to zero (i.e. keep only wavelet coefficients from V_0 in the decomposition $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-2} \oplus W_{m-1}$),
4. performs an IDWT on the resulting coefficients using the function `IDWTHaarImpl`,
5. plays the resulting sound.

b. Run the function `playDWTlower` for different values of m. For which m can you hear that the sound gets degraded? How does it get degraded? Compare with what you heard through the function `playDFTlower` in Example 2.27, where you performed a DFT on the sound sample instead, and set some of the DFT coefficients to zero.

c. Do the sound samples returned by `playDWTlower` lie in $[-1,1]$?

10. Attempt to construct a (nonzero) sound where the function `playDWTlower` form the previous exercise does not change the sound for $m = 1, 2$.

11. Repeat Exercise 9, but this time instead keep only wavelet coefficients from the detail spaces $W_0, W_1, \ldots$. Call the new function `playDWTlowerdifference`. What kind of sound do you hear? Can you recognize the original sound in what you hear?

12. Compute the wavelet detail coefficients analytically for the functions in Example 5.22, i.e. compute the quantities $w_{m,n} = \int_0^N f(t)\psi_{m,n}(t)dt$ similarly to how this was done in Example 5.23.

13. Compute the wavelet detail coefficients analytically for the functions $f(t) = \left(\frac{t}{N}\right)^k$, i.e. compute the quantities $w_{m,n} = \int_0^N \left(\frac{t}{N}\right)^k \psi_{m,n}(t)dt$ similarly to how this was done in Example 5.23. How do these compare with the coefficients from the Exercise 12?

14. (Exam UIO V2012) Suppose that we have the vector $\boldsymbol{x}$ with length $2^{10} = 1024$ defined by $x_n = 1$ for $0 \leq n \leq 511$, while $x_n = 0$ for all other values of n. What will be the result if you run a 10-level DWT with the Haar-wavelet on $\boldsymbol{x}$?

5.4 Multiresolution analysis: A generalization

Let us summarize the properties of the spaces V_m. We showed that they were nested, i.e.

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots .$$

We also showed that continuous functions could be approximated arbitrarily well from V_m, as long as m was chosen large enough. Moreover it is clear that the space V_0 is closed under all translates, at least if we view the functions in V_0 as periodic with period N. In the following we will always identify a function with this periodic extension, just as we did in Fourier analysis. When performing this identification, we also saw that $f(t) \in V_m$ if and only if $g(t) = f(2t) \in V_{m+1}$. We have therefore shown that the scaling funtion ϕ fits in with the following general framework.

Definition 5.24 (Multiresolution analysis)**.** A Multiresolution analysis, or MRA, is a nested sequence of function spaces

$$V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_m \subset \cdots \tag{5.23}$$

so that

1. Any function can be approximated arbitrarily well from V_m, as long as m is large enough,

2. $f(t) \in V_0$ if and only if $f(2^m t) \in V_m$,
3. $f(t) \in V_0$ if and only if $f(t-n) \in V_0$ for all n.
4. There is a function ϕ, called a scaling function, so that $\{\phi(t-n)\}_{0\leq n<N}$ is an orthonormal basis for V_0. .

Also in this more general setting the spaces V_m are called resolution spaces. Due to the orthonormality of $\{\phi(t-n)\}_{0\leq n<N}$, we say that we have an *orthonormal MRA*. MRA's are much used, and one can find a wide variety of functions ϕ, not only piecewise constant functions, which give rise to MRA's.

In the MRA-setting it helps to think about the continuous-time function $f(t)$ as the model for an image, which is the object under study. f itself may not be in any V_m (this corresponds to that detail is present in the image for infinitely many m), and increasing m corresponds to that we also include the detail we see when we zoom in on the image. The best we can do is thus to look at the least squares approximation of f at resolution m, $f^{(m)} \in V_m$, which can be written as

$$f^{(m)}(t) = \sum_n c_{m,n}\phi_{m,n}(t),$$

where the coordinates $c_{m,n}$ are given by

$$c_{m,n} = \int_0^N f(t)\phi_{m,n}(t)dt.$$

Finding $f^{(m)}$ thus requires the computation of integrals where all function values are needed. However, as before we have only access to some samples $f(2^{-m}n)$, $0 \leq n < 2^m N$. These are called pixel values in the context of images, so that we can only hope to obtain a good approximation to $f^{(m)}$ (and thus f) from the pixel values. The following result explains how we can obtain this.

Theorem 5.25. If f is continuous, and ϕ has compact support, we have that, for all t,

$$f(t) = \lim_{m\to\infty} \sum_{n=0}^{2^m N-1} \frac{2^{-m}}{\int_0^N \phi_{m,0}(t)dt} f(n/2^m)\phi_{m,n}(t).$$

Proof. We have that

$$2^{-m}\sum_{n=0}^{2^m N-1} \phi_{m,n} = \sum_{n=0}^{2^m N-1} 2^{-m}\phi_{m,0}(t-2^{-m}n).$$

We recognize this as a Riemann sum for the integral $\int_0^N \phi_{m,0}(t)dt$. Therefore, $\lim_{m\to\infty}\sum_{n=0}^{2^m N-1} 2^{-m}\phi_{m,n} = \int_0^N \phi_{m,0}(t)dt$. Also, finitely many n contribute in this sum since ϕ has compact support. We now get that

$$\begin{aligned}\sum_{n=0}^{2^mN-1} 2^{-m}f(n/2^m)\phi_{m,n}(t) &= \sum_{n \text{ so that } 2^{-m}n\approx t} 2^{-m}f(n/2^m)\phi_{m,n}(t)\\ &\approx \sum_{n \text{ so that } 2^{-m}n\approx t} 2^{-m}f(t)\phi_{m,n}(t)\\ &= f(t)\sum_{n \text{ so that } 2^{-m}n\approx t} 2^{-m}\phi_{m,n}(t) \approx f(t)\int_0^N \phi_{m,0}(t)dt.\end{aligned}$$

where we have used the continuity of f and that $\lim_{m\to\infty}\sum_{n=0}^{2^mN-1}2^{-m}\phi_{m,n} = \int_0^N \phi_{m,0}(t)dt$. The result follows. Note that here we have not used the requirement that $\{\phi(t-n)\}_n$ are orthogonal. □

The coordinate vector $\boldsymbol{x} = \left(\frac{2^{-m}}{\int_0^N \phi_{m,0}(t)dt}f(n/2^m)\right)_{n=0}^{2^mN-1}$ in $\boldsymbol{\phi}_m$ is therefore a candidate to an approximation of both f and $f^{(m)}$ from V_m, using only the pixel values. Normally one drops the leading constant $\frac{2^{-m}}{\int_0^N \phi_{m,0}(t)dt}$, so that one simply considers the sample values $f(n/2^m)$ as a coordinate vector in $\boldsymbol{\phi}_m$. This is used as the input to the DWT.

5.4.1 Connection between wavelet coefficients and anlog filters

We can further write

$$\begin{aligned}c_{m,n} &= \int_0^N f(t)\phi_{m,n}(t)dt = 2^{m/2}\int_0^N f(t)\phi(2^mt-n)dt\\ &= 2^{m/2}\int_0^N \phi(2^m(t-2^{-m}n))f(t)dt = 2^{m/2}\int_0^N \phi(-2^mt)f(2^{-m}n-t)dt\\ &= \int_0^N(-\phi_{m,0}(-t))f(2^{-m}n-t)dt.\end{aligned}$$

This gives the first connection with wavelets and filters:

Observation 5.26 (Connection between analog filters and the DWT)**.** Define the analog filter

$$s_{\phi,m}(f(t)) = \int_0^N(-\phi_{m,0}(-u))f(t-u)du \tag{5.24}$$

(i.e. the filter with convolution kernel $g(t) = -\phi_{m,0}(-t)$). The coefficients $c_{m,n}$ of an approximation at resolution m can be computed by sampling $s_{\phi,m}(f(t))$ at the points $2^{-m}n$.

5.4.2 Mother wavelet

With an MRA there is another important thing we also need: As in the case of piecewise constant functions, we need to be able to efficiently compute the decomposition of $g_m \in V_m$ into a low resolution approximation and a detail/error. By the low resolution approximation we mean an element $g_{m-1} \in V_{m-1}$, as with previous notation. Previous notation also indicates that we should write $e_{m-1} \in W_{m-1}$ for the detail, but we have not defined the detail spaces W_m yet. Previously we just defined W_{m-1} as the orthogonal complement of V_{m-1} in V_m. This is what we do in the general setting of MRA as well. We also would like to find a simple basis for W_m. Once we have this, we can perform a change of coordinates to find the detail and low resolution approximations. Let us summarize this with the following recipe:

Idea 5.27 (Recipe for constructing wavelets)**.** In order to construct MRA's which are useful for practical purposes, we need to do the following:

1. Find a function ϕ which can serve as the scaling function for an MRA,
2. Find a function ψ so that $\boldsymbol{\psi} = \{\psi(t-n)\}_{0\leq n<N}$ and $\boldsymbol{\phi} = \{\phi(t-n)\}_{0\leq n<N}$ together form an orthonormal basis for V_1. The function ψ is also called a mother wavelet.

With V_0 the space spanned by $\boldsymbol{\phi} = \{\phi(t-n)\}_{0\leq n<N}$, and W_0 the space spanned by $\boldsymbol{\psi} = \{\psi(t-n)\}_{0\leq n<N}$, ϕ and ψ should be chosen so that we easily can compute the decomposition of $g_1 \in V_1$ into g_0+e_0, where $g_0 \in V_0$ and $e_0 \in W_0$. If we can achieve this, the m-level Discrete Wavelet Transform can be defined and computed similarly as in the case when ϕ is a piecewise constant function, with the obvious replacements.

Note that there may be many possible choices of ψ so that $\boldsymbol{\psi}$ span W_0. In general we choose one which easily helps us compute the decomposition $g_m = g_{m-1} + e_{m-1}$, but we will see later that it also is desirable to choose a ψ with other properties. The term wavelet is used in very general terms. However, the term mother wavelet is quite concrete, and is what gives rise to the theory of wavelets. With the help of ψ and the DWT, $f^{(m)}$ is instead written as

$$f^{(m)} = \sum_n c_{0,n}\phi_{0,n} + \sum_{m'<m,n} w_{m',n}\psi_{m',n}, \tag{5.25}$$

In this decomposition, it is useful to interpret m as frequency, n as time, and $w_{m,n}$ as the contribution in f at frequency m and time n. In this sense, wavelets provide a *time-frequency representation* of signals. This is what can make them more useful than Fourier analysis, which only provides a frequency representation.

As before we can write

$$w_{m,n} = \int_0^N f(t)\psi_{m,n}(t)dt = \int_0^N (-\psi_{m,0}(-t))f(2^{-m}n - t).$$

Thus, if we define the analog filter

$$s_{\psi,m}(f(t)) = \int_0^N (-\psi_{m,0}(-u))f(t-u)du \tag{5.26}$$

(which is similar to how we defined $s_{\phi,m}$), $w_{m,n}$ are the samples of $s_{\psi,m}$ at $2^{-m}n$. If we combine this with what we previously found, the DWT of the vector $\{c_{m,n}\}_n$ can be obtained by applying the analog filters given by equations 5.24 and 5.26 for $m' < m$, to $f = \sum_n c_{m,n}\phi_{m,n} \in V_m$. The entire DWT can thus be expressed in terms of analog filters. It is important to note that the input to the DWT are typically samples from a function $f(t)$ (the model), and what we aim for is to have the DWT mimic what the analog filter (the operation) does, to as high precision as possible. If the functions ϕ, ψ are symmetric around 0, these filters are symmetric (since a filter is symmetric if and only if the convolution kernel is symmetric around 0), in which case we know that such a high precision implementation is possible using the simple technique of symmetric extension. Let us summarize this as the following idea.

Idea 5.28. If the functions ϕ, ψ in a wavelet are symmetric around 0, then we can obtain an implementation of the DWT with higher precision when we consider symmetric extensions of the input.

Unfortunately the filter is not symmetric for the ϕ we have considered in this chapter, since ϕ is not symmetric around 0. We will return to this later when we consider wavelets where the filters are symmetric.

With the Haar wavelet we succeeded in finding a function ψ which could be used in the recipe above. Note, however, that there may be many other ways to define a function ψ which can be used in the recipe. In the next chapter we will follow the recipe in order to contruct other wavelets, and we will try to express which pairs of function ϕ, ψ are most interesting, and which resolution spaces are most interesting.

5.5 Summary

We started this chapter by motivating the theory of wavelets as a different function approximation scheme, which solved some of the shortcomings of Fourier series. While one approximates functions with trigonometric functions in Fourier theory, with wavelets one instead approximates a function in several stages, where one at each stage attempts to capture information at a given resolution, using a function prototype. This prototype is localized in time, contrary to the Fourier basis functions, and this makes the theory of wavelets suitable for making time-frequency representations of signals. We used an example based on Google Earth$^{\text{TM}}$to illustrate that the wavelet-based scheme can represent an image at different resolutions in a scalable way, so that passing from one resolution to a better resolution simply mounts to adding some detail information to the lower resolution version of the image.

We defined the simplest wavelet, the Haar wavelet, which is a function approximation scheme based on piecewise constant functions. We deduced the properties of the Haar wavelet, and established a general framework by summarizing these. We defined the Discrete Wavelet Transform as a change of basis corresponding to the function spaces we defined. We will continue in the next chapter to construct more general wavelets.

Chapter 6

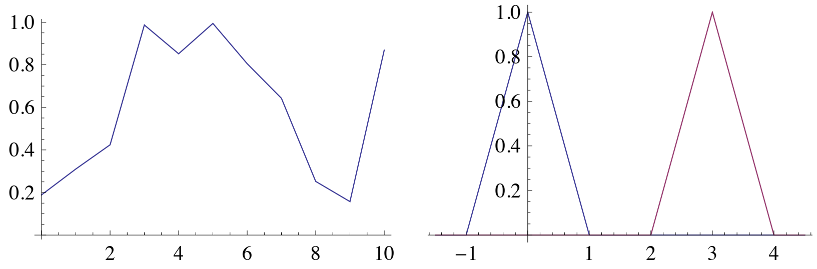
Wavelets constructed from piecewise linear functions

In Section 5.3 we started with the simple space of functions that are constant on each interval between two integers, which has a very simple orthonormal basis given by translates of the characteristic function of the interval $[0, 1)$. From this we constructed a so-called multiresolution analysis of successively refined spaces of piecewise constant functions that may be used to approximate any continuous function arbitrarily well. We then saw how a given function in a fine space could be projected orthogonally into the preceding coarser space. The computations were all taken care of with the Discrete Wavelet Transform.

Unfortutately, piecewise constant functions are too simple to provide good approximations. In this chapter we are going to extend the construction of wavelets to piecewise linear functions. The advantage is that piecewise linear functions are better for approximating smooth functions and data than piecewise constants, which should translate into smaller components (errors) in the detail spaces in many practical situations. As an example, this would be useful if we are interested in compression. In this new setting it turns out that we loose the orthonormality we had for the Haar wavelet. On the other hand, we will see that the new scaling functions and mother wavelets are symmetric functions, so that, as commented, the corresponding DWT and IDWT have simple implementations with higher precision.

6.1 A first construction of a wavelet for piecewise constant functions

Our experience from deriving Haar wavelets will guide us in the construction of piecewise linear wavelets. The first task is to define the underlying function spaces.



(a) A piecewise linear function.

(b) The two functions $\phi(t)$ and $\phi(t-3)$.

Figure 6.1: Some piecewise linear functions.

Definition 6.1 (Resolution spaces of piecewise linear functions)**.** The space V_m is the subspace of continuous functions on $\mathbb{R}$ which are periodic with period N, and linear on each subinterval of the form $[n2^{-m}, (n+1)2^{-m})$.

Any $f \in V_m$ is uniquely determined by its values on $[0, N)$. Figure 6.1 (a) shows an example of a piecewise linear function in V_0 on the interval $[0, 10]$. We note that a piecewise linear function in V_0 is completely determined by its value at the integers, so the functions that are 1 at one integer and 0 at all others are particularly simple and therefore interesting, see Figure 6.1 (b). These simple functions are all translates of each other and can therefore be built from one scaling function, as is required for a multiresolution analysis.

Recall that the *support* of a function f defined on a subset I of $\mathbb{R}$ is given by the closure of the set of points where the function is nonzero,

$$\operatorname{supp}(f) = \overline{\{t \in I \mid f(t) \neq 0\}}.$$

Lemma 6.2. Let the function ϕ be defined by

$$\phi(t) = \begin{cases} 1+t, & \text{if } -1 \le t < 0; \\ 1-t, & \text{if } 0 \le t < 1; \\ 0, & \text{otherwise;} \end{cases} \tag{6.1}$$

and for any $m \ge 0$ set

$$\phi_{m,n}(t) = 2^{m/2}\phi(2^m t - n) \quad \text{for } n = 0,\, 1,\, \ldots,\, 2^m N - 1,$$

or in vector notation

$$\boldsymbol{\phi}_m = (\phi_{m,0}, \phi_{m,1}, \ldots, \phi_{m,2^m N-1}).$$

The functions $\{\phi_{m,n}\}_{n=0}^{2^m N-1}$, restricted to the interval $[0, N]$, form a basis for the space V_m for this interval. In other words, the function ϕ is a scaling function for the spaces V_0, V_1, ... Moreover, the function $\phi_{0,n}(t)$ is the function in V_0 with smallest support that is nonzero at $t = n$.

Proof. The proof is similar for all the resolution spaces, so it is sufficient to consider the proof in the case of V_0. The function ϕ is clearly linear between each pair of neighbouring integers, and it is also easy to check that it is continuous. Its restriction to $[0, N]$ therefore lies in V_0. And as we noted above $\phi_{0,n}(t)$ is 0 at all the integers except at $t = n$ where its value is 1.

A general function f in V_0 is completely determined by its values at the integers in the interval $[0, N]$ since all straight line segments between neighbouring integers are then fixed. Note that we can also write f as

$$f(t) = \sum_{n=0}^{N-1} f(n)\phi_{0,n}(t) \tag{6.2}$$

since this function agrees with f at the integers in the interval $[0, N]$ and is linear on each subinterval between two neighbouring integers. This means that V_0 is spanned by the functions $\{\phi_{0,n}\}_{n=0}^{N-1}$. On the other hand, if f is identically 0, all the coefficients in (6.2) are also 0, so $\{\phi_{0,n}\}_{n=0}^{N-1}$ are linearly independent and therefore a basis for V_0.

Suppose that the function $g \in V_0$ has smaller support than $\phi_{0,n}$, but is nonzero at $t = n$. Then g must be identically zero either on $[n-1, n)$ or on $[n, n+1]$, since a straight line segment cannot be zero on just a part of an interval between integers. But then g cannot be continuous, which contradicts the fact the it lies in V_0. □

The function ϕ and its translates and dilates are often referred to as hat functions for obvious reasons.

A formula like (6.2) is also valid for functions in V_m.

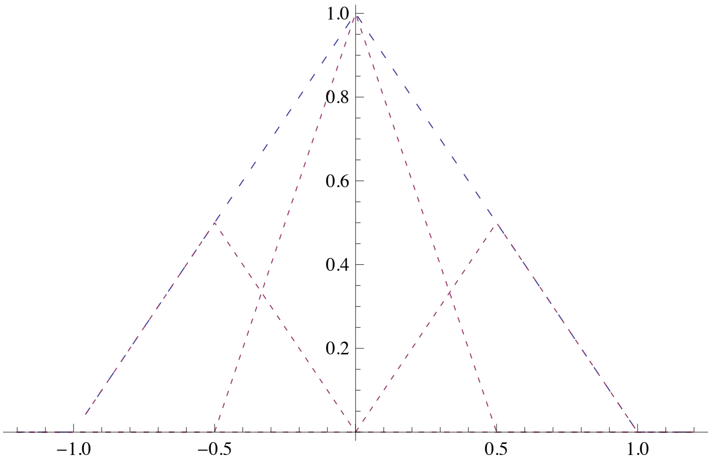
Lemma 6.3. A function $f \in V_m$ may be written as

$$f(t) = \sum_{n=0}^{2^m N-1} f(n/2^m)2^{-m/2}\phi_{m,n}(t). \tag{6.3}$$

An essential property of a multiresolution analysis is that the spaces should be nested.

Lemma 6.4. The piecewise linear resolution spaces are nested,

$$V_0 \subset V_1 \subset \cdots \subset V_m \subset \cdots .$$



Proof. We only need to prove that $V_0 \subset V_1$ since the other inclusions are similar. But this is immediate since any function in V_0 is continuous, and linear on any subinterval in the form $[n/2, (n+1)/2)$. □

In the piecewise constant case, we saw in Lemma 5.5 that the scaling functions were automatically orthogonal since their supports did not overlap. This is not the case in the linear case, but we could orthogonalise the basis $\boldsymbol{\phi}_m$ with the Gram-Schmidt process from linear algebra. The disadvantage is that we lose the nice local behaviour of the scaling functions and end up with basis functions that are nonzero over all of $[0, N]$. And for most applications, orthogonality is not essential; we just need a basis.

Let us sum up our findings so far.

Observation 6.5. The spaces V_0, V_1, $\ldots$, V_m, $\ldots$ form a multiresolution analysis generated by the scaling function ϕ.

The next step in the derivation of wavelets is to find formulas that let us express a function given in the basis $\boldsymbol{\phi}_0$ for V_0 in terms of the basis $\boldsymbol{\phi}_1$ for V_1.

Lemma 6.6. The function $\phi_{0,n}$ satisfies the relation

$$\phi_{0,n} = \frac{1}{\sqrt{2}}\left(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}\right). \tag{6.4}$$

Proof. Since $\phi_{0,n}$ is in V_0 it may be expressed in the basis $\boldsymbol{\phi}_1$ with formula (6.3),

$$\phi_{0,n}(t) = 2^{-1/2}\sum_{k=0}^{2N-1}\phi_{0,n}(k/2)\phi_{1,k}(t).$$

The relation (6.4) now follows since

$$\phi_{0,n}\big((2n-1)/2\big) = \phi_{0,n}\big((2n+1)/2\big) = 1/2, \quad \phi_{0,n}(2n/2) = 1,$$

and $\phi_{0,n}(k/2) = 0$ for all other values of k. □

6.1.1 Detail spaces and wavelets

The next step in our derivation of wavelets for piecewise linear functions is the definition of the detail spaces. In the case of V_0 and V_1, we need to determine a space W_0 so that V_1 is the direct sum of V_0 and W_0. In the case of piecewise constants we started with a function g_1 in V_1, computed the least squares approximation g_0 in V_0, and then defined the space W_0 as the space of all possible error functions. This is less appealing in the linear case since we do not have an orthogonal basis for V_0.

As in the case of piecewise constants we start with a function g_1 in V_1, but we use an extremely simple approximation method, we simply drop every other coefficient.

Definition 6.7. Let g_1 be a function in V_1 given by

$$g_1 = \sum_{n=0}^{2N-1} c_{1,n}\, \phi_{1,n}. \tag{6.5}$$

The approximation $g_0 = S(g_1)$ in V_0 interpolates g_1 at the integers,

$$g_0(n) = g_1(n), \quad n = 0, 1, \ldots, N-1. \tag{6.6}$$

It is very easy to see that the coefficients of g_0 actually can be obtained by dropping every other coefficient:

Lemma 6.8. Let g_1 be given by (6.5) and suppose that $g_0 = \sum_{n=0}^{N-1} c_{0,n}\phi_{0,n}$ in V_0 interpolates g_1 at the integers in $0, \ldots, N-1$. Then

$$S(\phi_{1,n}) = \begin{cases} \sqrt{2}\phi_{0,n/2}, & \text{if } n \text{ is an even integer;} \\ 0, & \text{otherwise.} \end{cases}$$

Once the method of approximation is determined, it is straightforward to determine the detail space as the space of error functions. With the notation from Definition 6.7, the error is given by $e_0 = g_1 - g_0$. Since g_0 interpolates g_1 at the integers, the error is 0 there,

$$e_0(n) = 0, \quad \text{for } n = 0, 1, \ldots, N-1.$$

Conversely, any function in V_1 which is 0 at the integers may be viewed as an error function in the above sense. This provides the basis for a precise description of the error functions.

Lemma 6.9. Suppose the function g_0 in V_0 interpolates a function g_1 in V_1 at the integers. Then the error $e_0 = g_1 - g_0$ lies in the space W_0 defined by

$$W_0 = \{f \in V_1 \mid f(n) = 0, \quad \text{for } n = 0, 1, \ldots, N-1.$$

A basis for W_0 is given by the wavelets $\{\psi_{0,n}\}_{n=0}^{N-1}$ defined by

$$\psi_{0,n} = \frac{1}{\sqrt{2}}\phi_{1,2n+1}, \quad \text{for } n = 0, 1, \ldots, N-1.$$

Proof. Since $g_0(n) = g_1(n)$ for all integers n, $e_0(n) = 0$. This means that the error functions are precisely the piecewise linear functions at the half intervals which are zero at the integers. Clearly this is the same space as that spanned by the $\psi_{0,n} = \frac{1}{\sqrt{2}}\phi_{1,2n+1}$. □

We now have all the ingredients to formulate an analog of Theorem 5.20 that describes how V_m can be expressed as a direct sum of V_{m-1} and W_{m-1}. The formulas for $m = 1$ generalise without change, except that the upper bound on the summation indices must be adjusted.

Theorem 6.10. The space V_m can be decomposed as the direct sum $V_m = V_{m-1} \oplus W_{m-1}$ where W_{m-1} is the space of all functions in V_m that are zero at the points $\{n/2^{m-1}\}_{n=0}^{N2^{m-1}-1}$. The space V_m has the two bases

$$\boldsymbol{\phi}_m = (\phi_{m,n})_{n=0}^{2^m N-1}$$

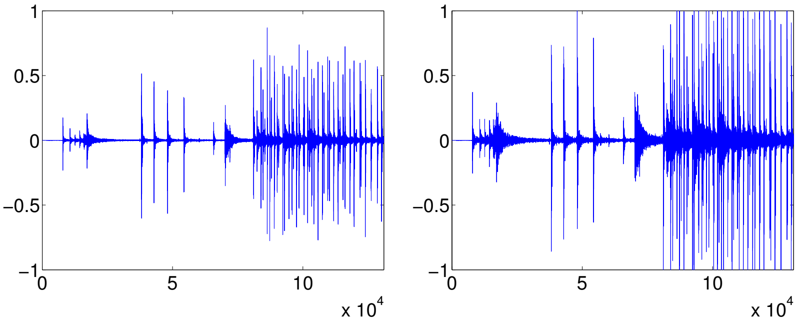
and

$$(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1}) = \big((\phi_{m-1,n})_{n=0}^{2^{m-1}N-1}, (\psi_{m-1,n})_{n=0}^{2^{m-1}N-1}\big).$$

The equations above take the form

$$\begin{aligned}\phi_{0,n} &= \frac{1}{2\sqrt{2}}\phi_{1,2n-1} + \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{2\sqrt{2}}\phi_{1,2n+1} \\ \psi_{0,n} &= \frac{1}{\sqrt{2}}\phi_{1,2n+1}.\end{aligned} \tag{6.7}$$

These two relations together give all columns in the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow (\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0)}$ (i.e. the IDWT), when the spaces $\boldsymbol{\phi}_m$, $\boldsymbol{\psi}_m$ instead are defined in terms of the function ψ, and the normalized ϕ. Similarly we can compute the



(a) $m = 1$

(b) $m = 2$

Figure 6.2: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.

change of coordinate matrix the opposite way by rewriting equations (6.7) as

$$\begin{aligned}\frac{1}{\sqrt{2}}\phi_{1,2n} &= \phi_{0,n} - \frac{1}{2\sqrt{2}}\phi_{1,2n-1} - \frac{1}{2\sqrt{2}}\phi_{1,2n+1}\\ \frac{1}{\sqrt{2}}\phi_{1,2n+1} &= \psi_{0,n},\end{aligned}$$

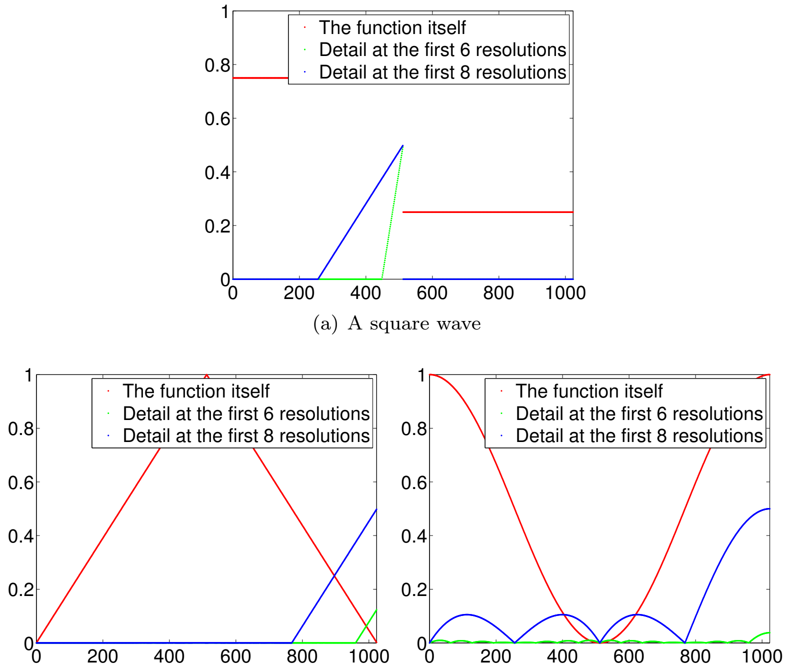
from which it follows that

$$\begin{aligned}\phi_{1,2n} &= \sqrt{2}\phi_{0,n} - \frac{1}{2}\phi_{1,2n-1} - \frac{1}{2}\phi_{1,2n+1}\\ &= -\frac{\sqrt{2}}{2}\psi_{0,n-1} + \sqrt{2}\phi_{0,n} - \frac{\sqrt{2}}{2}\psi_{0,n} \qquad (6.8)\\ \phi_{1,2n+1} &= \sqrt{2}\psi_{0,n}. \qquad (6.9)\end{aligned}$$

This gives us the columns in the change of coordinate matrix $P_{(\boldsymbol{\phi}_0\oplus\boldsymbol{\psi}_0)\leftarrow\boldsymbol{\phi}_1}$ as well, i.e. the DWT.

Example 6.11. In Section 7 we will construct an algorithm which performs DWT/IDWT, for a general wavelet. In particular, this algorithm can be used for the wavelet we constructed in this section. Let us also for this wavelet plot the detail/error in the test audio file `castanets.wav` for different resolutions, as we did in Example 5.21. The result is shown in Figure 6.2. When comparing with Figure 5.10 we see much of the same, but it seems here that the error is bigger than before. In the next section we will try to explain why this is the case, and construct another wavelet based on piecewise linear functions which remedies this. ♣

Example 6.12. Let us also repeat Exercise 5.22, where we plotted the detail/error at different resolutions, for the samples of a mathematical function.



(a) A square wave

(b) $f(t) = 1 - 2|1/2 - t/N|$

(c) $f(t) = 1/2 + \cos(2\pi t/N)/2$

Figure 6.3: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m.

Figure 6.3 shows the new plot. With the square wave we see now that there is an error. The reason is that a piecewise constant function can not be represented exactly by piecewise linear functions, due to discontinuity. For the second function we see that there is no error. The reason is that this function is piecewise constant, so there is no error when we represent the function from the space V_0. With the third function, hoewever, we see an error. ♣

Exercises for Section 6.1

1. Show that, for $f \in V_0$ we have that $[f]_{\phi_0} = (f(0), f(1), \dots, f(N-1))$. This generalizes the result for piecewise constant functions.

2. Show that

$$\begin{aligned} \langle \phi_{0,n}, \phi_{0,n} \rangle &= \frac{2}{3} \\ \langle \phi_{0,n}, \phi_{0,n\pm 1} \rangle &= \frac{1}{6} \\ \langle \phi_{0,n}, \phi_{0,n\pm k} \rangle &= 0 \text{ for } k > 1. \end{aligned}$$

As a consequence, the $\{\phi_{0,n}\}_n$ are neither orthogonal, nor have norm 1.

3. The convolution of two functions defined on $(-\infty, \infty)$ is defined by

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt.$$

Show that we can obtain the piecewise linear ϕ we have defined as $\phi = \chi_{[-1/2,1/2)} * \chi_{[-1/2,1/2)}$ (recall that $\chi_{[-1/2,1/2)}$ is the function which is 1 on $[-1/2, 1/2)$ and 0 elsewhere). This gives us a nice connection between the piecewise constant scaling function (which is similar to $\chi_{[-1/2,1/2)}$) and the piecewise linear scaling function in terms of convolution.

6.2 Multiresolution analysis: Another generalization

Let us generalize our definition of Multiresolution analysis from Section 5.4 so that the theory also fits with what we have found so far in this chapter. The first difference we have is that the $\{\phi(t-n)\}_{0 \leq n < N}$ are not orthogonal anymore. We will remedy this by changing requirement 4 for a Multiresolution analysis in Definition 5.24 to the following:

Definition 6.13 (Alternative requirement 4 for a Multiresolution Analysis)**.** There is a function ϕ, called a scaling function, so that $\{\phi(t-n)\}_{0 \leq n < N}$ is a basis for V_0.

As with the example we started this chapter with, it turns out to be easier to construct interesting MRA's when we do not assume orthonormality as above.

6.2.1 Dual scaling function

One other difference from before is that we can not easily compute the best approximation $f^{(m)}$ to f from V_m anymore, since that would require that we create an orthogonal basis. In fact we will not compute the best approximation anymore, but something similar. This new type of approximation is obtained in terms of what is called a dual scaling function:

Theorem 6.14. Assume that we have a an MRA with scaling function ϕ. There exists a function $\tilde{\phi}$ so that

$$f(t) = \lim_{m\to\infty} \sum_n c_{m,n}\phi_{m,n}(t),$$

with $c_{m,n} = \int_0^N f(t)\tilde{\phi}_{m,n}dt$. $\tilde{\phi}$ is also called the *dual scaling function.*

This definition has a clear interpretation for orthonormal MRA's, since $\tilde{\phi} = \phi$ then. With $c_{m,n}$ defined as above, one may believe that $\sum_n c_{m,n}\phi_{m,n}$ actually is the best approximation to f from V_m. This is certainly the case for orthonormal MRA's, but it may not hold generally. So, for non-orthonormal MRA's, we have no procedure for computing best approximations. Anyway, we have mentioned that Theorem 5.25 was valid also when $\{\phi(t-n)\}_n$ were not orthogonal, i.e. $\frac{2^{-m}}{\int_0^N \phi_{m,0}(t)dt} f(n/2^m)\phi_{m,n}(t)$, has validity as an approximation to f also in this more general setting.

6.2.2 Construction of the detail spaces

Now let us consider the more general construction of the detail spaces W_m. Since we do not assume orthogonality anymore, we will not require that W_{m-1} is the orthogonal complement of V_{m-1} in V_m either. Rather we will allow W_{m-1} to be any space independent from V_{m-1} so that V_{m-1} and W_{m-1} together span V_m (i.e. we require that $V_m = V_{m-1} \oplus W_{m-1}$). Even though W_{m-1} is unique when we require orthogonality, it, and hence also the decomposition $g_m = g_{m-1} + e_{m-1}$, is not unique when we require that the two spaces together span V_m. We thus have some freedom in the design of W_m. We also need to choose a ψ so that $\boldsymbol{\psi} = \{\psi(t-n)\}_{0\leq n<N}$ is a basis for W_0 (again, there is no requirement on orthogonality here). The recipe for constructing wavelets is thus rewritten as follows:

Idea 6.15 (Recipe for constructing wavelets, non-orthonormal case)**.** In order to construct MRA's which are useful for practical purposes, we need to do the following

1. Find a vector space W_0, linearly independent from V_0, and so that $V_1 = V_0 \oplus W_0$.

2. Find a function ψ so that $\boldsymbol{\psi} = \{\psi(t-n)\}_{0 \leq n < N}$ is a basis for W_0. The function ψ is also called a mother wavelet.

Again, ϕ and ψ should be chosen so that we easily can compute the decomposition of $g_1 \in V_1$ into $g_0 + e_0$, where $g_0 \in V_0$ and $e_0 \in W_0$. This gives rise to the m-level Discrete Wavelet Transform as before.

6.2.3 Dual mother wavelet

Now, assume that we have found a pair (ϕ, ψ) for an MRA. We state the following without proof, for how the DWT can be expressed.

Theorem 6.16. Assume that $f^{(m)}(t) = \sum_n c_{m,n}\phi_{m,n} \in V_m$, and assume that $\tilde{\phi}$ is as in Theorem 6.14. There exists a function $\tilde{\psi}$ so that

$$f^{(m)}(t) = \sum_n c_{0,n}\phi_{0,n} + \sum_{m'<m,n} w_{m',n}\psi_{m',n},$$

with $w_{m,n} = \int_0^N f^{(m)}(t)\tilde{\psi}_{m,n}(t)dt$.

Together, theorem 6.14 and 6.16 help us express the DWT for a non-orthonormal wavelet in terms of analog filters, thereby generalizing what we previously did for orthonormal wavelets:

Observation 6.17. Let

$$f^{(m)} = \sum_n c_{m,n}\phi_{m,n} = \sum_n c_{0,n}\phi_{0,n} + \sum_{m'<m,n} w_{m',n}\psi_{m',n}$$

be the approximation of f from V_m we obtain as above. Regardless of whether the wavelet is orthogonal or not, $c_{m,n}$ and $w_{m,n}$ can be computed by sampling the output of an analog filter. To be more precise, there exist functions $\tilde{\phi}$, $\tilde{\psi}$ so that

$$\begin{aligned} c_{m,n} = \langle f, \tilde{\phi}_{m,n}\rangle &= \int_0^N f(t)\tilde{\phi}_{m,n}(t)dt = \int_0^N (-\tilde{\phi}_{m,0}(-t))f(2^{-m}n - t)dt \\ w_{m,n} = \langle f, \tilde{\psi}_{m,n}\rangle &= \int_0^N f(t)\tilde{\psi}_{m,n}(t)dt = \int_0^N (-\tilde{\psi}_{m,0}(-t))f(2^{-m}n - t)dt. \end{aligned}$$

In other words, if we define the analog filters $s_{\tilde{\phi},m}$, $s_{\tilde{\psi},m}$ by

$$s_{\tilde{\phi},m}(f(t)) = \int_0^N (-\tilde{\phi}_{m,0}(-u))f(t-u)du \tag{6.10}$$

$$s_{\tilde{\psi},m}(f(t)) = \int_0^N (-\tilde{\phi}_{m,0}(-u))f(t-u)du \tag{6.11}$$

$c_{m,n}$ can be obtained by sampling $s_{\tilde{\phi},m}(f(t))$ at the points $2^{-m}n$, and $w_{m,n}$ can be obtained by sampling $s_{\tilde{\psi},m}(f(t))$ at the points $2^{-m}n$.

Here the convolution kernels of the filters were as before, with the exception that ϕ, ψ were replaced by $\tilde{\phi}, \tilde{\psi}$. Note also that, if the functions $\tilde{\phi}$, $\tilde{\psi}$ are symmetric, we can increase the precision in the DWT with the method of symmetric extension also in this more general setting. It is possible to show that the pair $(\tilde{\phi}, \tilde{\psi})$ can be used to define another multiresolution analysis, called the dual wavelet. We will not go into the theory of this here.

6.2.4 Vanishing moments

The direct sum decomposition with piecewise linear functions we started this chapter with was very simple, but also has its shortcomings. To see this, set $N = 1$ and consider the space V_{10}, which has dimension 2^{10}. When we apply a DWT, we start with a function $g_{10} \in V_{10}$. This may be a very good representation of the underlying data. However, when we compute g_{m-1} we just pick every other coefficient from g_m. By the time we get to g_0 we are just left with the first and last coefficient from g_{10}. In some situations this may be adequate, but usually not. The following idea addresses this shortcoming:

Idea 6.18. We would like a wavelet basis to be able to represent f efficiently. By this we mean that the approximation $f^{(m)} = \sum_n c_{0,n}\phi_{0,n} + \sum_{m'<m,n} w_{m',n}\psi_{m',n}$ to f from Observation 6.17 should converge quickly for the f we work with, as m increases. This means that, with relatively few $\psi_{m,n}$, we can create good approximations of f.

To address this, let us use Observation 6.17 and write for $f \in V_m$

$$f = \sum_{n=0}^{N-1} \langle f, \tilde{\phi}_{0,n}\rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle f, \tilde{\psi}_{r,n}\rangle \psi_{r,n}. \tag{6.12}$$

If f is s times differentiable, it can be represented as $f = P_s(x) + Q_s(x)$, where P_s is a polynomial of degree s, and Q_s is a function which is very small (P_s could for instance be a Taylor series expansion of f). If in addition $\langle t^k, \tilde{\psi}\rangle = 0$, for $k = 1, \ldots, s$, we have also that $\langle t^k, \tilde{\psi}_{r,t}\rangle = 0$ for $r \leq s$, so that $\langle P_s, \tilde{\psi}_{r,t}\rangle = 0$ also. This means that Equation (6.12) can be written

$$\begin{aligned}
f &= \sum_{n=0}^{N-1} \langle P_s + Q_s, \tilde{\phi}_{0,n}\rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle P_s + Q_s, \tilde{\psi}_{r,n}\rangle \psi_{r,n} \\
&= \sum_{n=0}^{N-1} \langle P_s + Q_s, \tilde{\phi}_{0,n}\rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle P_s, \tilde{\psi}_{r,n}\rangle \psi_{r,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle Q_s, \tilde{\psi}_{r,n}\rangle \psi_{r,n} \\
&= \sum_{n=0}^{N-1} \langle f, \tilde{\phi}_{0,n}\rangle \phi_{0,n} + \sum_{r=0}^{m-1} \sum_{n=0}^{2^r N-1} \langle Q_s, \tilde{\psi}_{r,n}\rangle \psi_{r,n}.
\end{aligned}$$

Here the first sum lies in V_0. We see that the wavelet coefficients from W_r are $\langle Q_s, \tilde{\psi}_{r,n}\rangle$, which are very small since Q_s is small. This means that the detail in the different spaces W_r is very small, which is exactly what we aimed for. Let us summarize this as follows:

Theorem 6.19 (Vanishing moments)**.** We say that ψ has k vanishing moments if the integrals $\int_{-\infty}^{\infty} t^l \psi(t)dt = 0$ for all $0 \leq l \leq k-1$. If a function $f \in V_m$ is r times differentiable, and $\tilde{\psi}$ has r vanishing moments, then f can be approximated well from V_0. Moreover, the quality of this approximation improves when r increases.

Having many vanishing moments is thus very good for compression, since the corresponding wavelet basis is very efficient for compression. In particular, if f is a polynomial of degree less than or equal to $k-1$ and $\tilde{\psi}$ has k vanishing moments, then the detail coefficients $w_{m,n}$ are exactly 0. Since (ϕ, ψ) and $(\tilde{\phi}, \tilde{\psi})$ both are wavelet bases, it is equally important for both to have vanishing moments. We will in the following concentrate on the number of vanishing moments of ψ.

The Haar wavelet has one vanishing moment, since $\tilde{\psi} = \psi$ and $\int_0^N \psi(t)dt = 0$ as we noted in Observation 5.15. It is an exercise to see that the Haar wavelet has only one vanishing moment, i.e. $\int_0^N t\psi(t)dt \neq 0$.

6.3 Alternative wavelets for piecewise linear functions

Now consider the wavelet we have used up to now for piecewise linear functions, i.e. $\psi(t) = \phi_{1,1}(t)$. Clearly this has no vanishing moments, since $\psi(t) \geq 0$ for all t. This is thus not a very good choice of wavelet. Let us see if we can construct an alternative function $\hat{\psi}$, which has two vanishing moments, i.e. one more than the Haar wavelet.

Idea 6.20. Adjust the wavelet construction in Theorem 6.10 so that the new wavelets $\{\hat{\psi}_{m-1,n}\}_{n=0}^{N2^m-1}$ in W_{m-1} satisfy

$$\int_0^N \hat{\psi}_{m-1,n}(t)\,dt = \int_0^N t\hat{\psi}_{m-1,n}(t)\,dt = 0, \tag{6.13}$$

for $n = 0, 1, \ldots, N2^m - 1$.

As usual, it is sufficient to consider what happens when V_1 is written as a direct sum of V_0 and W_0. From Idea 6.20 we see that we need to enforce two conditions for each wavelet function. If we adjust the wavelets in Theorem 6.10

by adding multiples of the two neighbouring hat functions, we have two free parameters,

$$\hat{\psi}_{0,n} = \psi_{0,n} - \alpha\phi_{0,n} - \beta\phi_{0,n+1} \tag{6.14}$$

that we may determine so that the two conditions in (6.13) are enforced. If we do this, we get the following result:

Lemma 6.21. The function

$$\hat{\psi}_{0,n}(t) = \psi_{0,n}(t) - \frac{1}{4}\big(\phi_{0,n}(t) + \phi_{0,n+1}(t)\big) \tag{6.15}$$

satisfies the conditions

$$\int_0^N \hat{\psi}_{0,n}(t)\,dt = \int_0^N t\hat{\psi}_{0,n}(t)\,dt = 0.$$

Using Equation (6.4), which stated that

$$\phi_{0,n}(t) = \frac{1}{\sqrt{2}}(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1}) \tag{6.16}$$

we get

$$\begin{aligned}\hat{\psi}_{0,n}(t) &= \psi_{0,n}(t) - \frac{1}{4}\big(\phi_{0,n}(t) + \phi_{0,n+1}(t)\big)\\ &= \frac{1}{\sqrt{2}}\phi_{1,2n+1}(t) - \frac{1}{4}\frac{1}{\sqrt{2}}\big(\frac{1}{2}\phi_{1,2n-1} + \phi_{1,2n} + \frac{1}{2}\phi_{1,2n+1} + \frac{1}{2}\phi_{1,2n+1} + \phi_{1,2n+2} + \frac{1}{2}\phi_{1,2n+3}\big)\\ &= -\frac{1}{8\sqrt{2}}\phi_{1,2n-1} - \frac{1}{4\sqrt{2}}\phi_{1,2n} + \frac{3}{4\sqrt{2}}\phi_{1,2n+1} - \frac{1}{4\sqrt{2}}\phi_{1,2n+2} - \frac{1}{8\sqrt{2}}\phi_{1,2n+3}\end{aligned} \tag{6.17}$$

Note that what we did here is equivalent to finding the coordinates of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_1$: Equation (6.15) says that

$$[\hat{\psi}]_{\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0} = (-1/4, -1/4, 0, \ldots, 0) \oplus (1, 0, \ldots, 0). \tag{6.18}$$

Since the IDWT is the change of coordinates from $\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0$ to $\boldsymbol{\phi}_1$, we could also have computed $[\hat{\psi}]_{\boldsymbol{\phi}_1}$ by taking the IDWT of $(-1/4, -1/4, 0, \ldots, 0) \oplus (1, 0, \ldots, 0)$. In the next section we will consider more general implementations of the DWT and the IDWT, which we thus can use instead of performing the computation above.

In summary we have

$$\begin{aligned}\phi_{0,n}(t) &= \frac{1}{2\sqrt{2}}\phi_{1,2n-1} + \frac{1}{\sqrt{2}}\phi_{1,2n} + \frac{1}{2\sqrt{2}}\phi_{1,2n+1}\\ \hat{\psi}_{0,n}(t) &= -\frac{1}{8\sqrt{2}}\phi_{1,2n-1} - \frac{1}{4\sqrt{2}}\phi_{1,2n} + \frac{3}{4\sqrt{2}}\phi_{1,2n+1} - \frac{1}{4\sqrt{2}}\phi_{1,2n+2} - \frac{1}{8\sqrt{2}}\phi_{1,2n+3},\end{aligned} \tag{6.19}$$

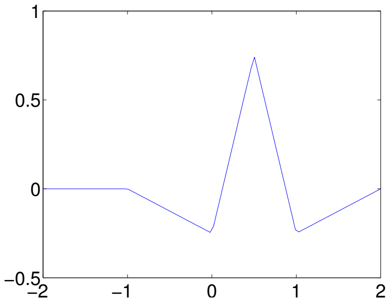
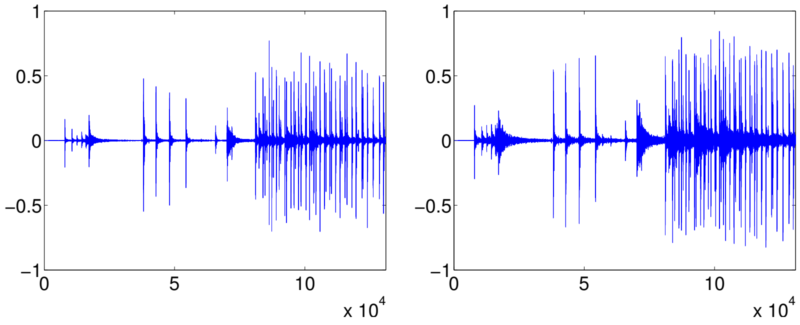


Figure 6.4: The function ψ we constructed as an alternative wavelet for piecewise linear functions.



(a) $m = 1$

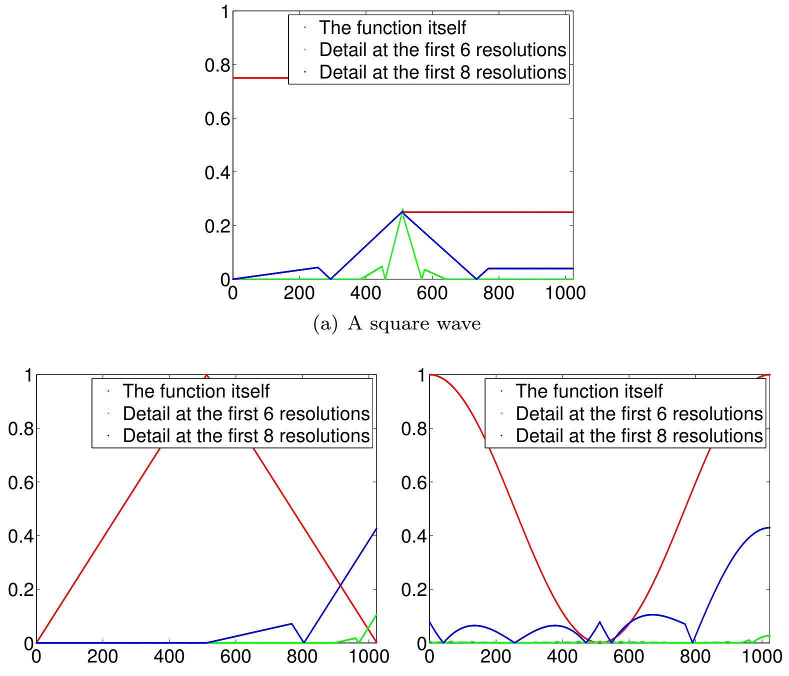
(b) $m = 2$

Figure 6.5: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) in the sound file `castanets.wav`, for different values of m.

which gives us the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow (\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0)}$. The new function $\psi = \hat{\psi}$ is plotted in Figure 6.4.

Example 6.22. Let us also plot the detail/error in the test audio file `castanets.wav` for different resolutions for our alternative wavelet, as we did in Example 5.21. The result is shown in Figure 6.5. Again, when comparing with Figure 5.10 we see much of the same. It is difficult to see an improvement from this figure. However, this figure also clearly shows a smaller error than the wavelet of the preceding section. A partial explanation is that the wavelet we now have constructed has two vanishing moments. ♣

Example 6.23. Let us also repeat Exercise 5.22 for our alternative wavelet, where we plotted the detail/error at different resolutions, for the samples of a mathematical function. Figure 6.6 shows the new plot. Again for the square wave there is an error, which seems to be slightly lower than for the previous



(a) A square wave

(b) $f(t) = 1 - 2|1/2 - t/N|$

(c) $f(t) = 1/2 + \cos(2\pi t/N)/2$

Figure 6.6: The error (i.e. the contribution from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$) for $N = 1024$ for different functions $f(t)$, for different values of m.

wavelet. For the second function we see that there is no error, as before. The reason is the same as before, since the function is piecewise constant. With the third function there is an error. The error seems to be slightly lower than for the previous wavelet, which fits well with the number of vanishing moments. ♣

Exercises for Section 6.3

1. In this exercise we will show that there is a unique function on the form (6.14) which has two vanishing moments.

a. Show that, when $\hat{\psi}$ is defined by (6.14), we have that

$$\hat{\psi}(t) = \begin{cases} -\alpha t - \alpha & \text{for } -1 \le t < 0 \\ (2+\alpha-\beta)t - \alpha & \text{for } 0 \le t < 1/2 \\ (\alpha-\beta-2)t - \alpha + 2 & \text{for } 1/2 \le t < 1 \\ \beta t - 2\beta & \text{for } 1 \le t < 2 \\ 0 & \text{for all other } t \end{cases}$$

b. Show that

$$\int_0^N \hat{\psi}(t)dt = \frac{1}{2} - \alpha - \beta$$
$$\int_0^N t\hat{\psi}(t)dt = \frac{1}{4} - \beta.$$

c. Explain why there is a unique function on the form (6.14) which has two vanishing moments, and that this function is given by Equation (6.15).

2. In the previous exercise we ended up with a lot of calculations to find α, β in Equation (6.14). Let us try to make a program which does this for us, and which also makes us able to generalize the result.

a. Define

$$a_k = \int_{-1}^1 t^k(1-|t|)dt$$
$$b_k = \int_0^2 t^k(1-|t-1|)dt$$
$$e_k = \int_0^1 t^k(1-2|t-1/2|)dt,$$

for $k \geq 0$. Explain why finding α, β so that we have two vanishing moments in Equation 6.14 is equivalent to solving the following equation:

$$\begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \end{pmatrix}$$

Write a program which sets up and solves this system of equations, and use this program to verify the values for α, β we previously have found. Hint: recall that you can integrate functions in Matlab with the function `quad`. As an example, the function $\phi(t)$, which is nonzero only on $[-1, 1]$, can be integrated as follows:

```
quad(@(t)t.^k.*(1-abs(t)),-1,1)
```

b. The procedure where we set up a matrix equation in a. allows for generalization to more vanishing moments. Define

$$\hat{\psi} = \psi_{0,0} - \alpha\phi_{0,0} - \beta\phi_{0,1} - \gamma\phi_{0,-1} - \delta\phi_{0,2}. \tag{6.20}$$

We would like to choose $\alpha, \beta, \gamma, \delta$ so that we have 4 vanishing moments. Define also

$$\begin{aligned} g_k &= \int_{-2}^{0} t^k(1 - |t+1|)dt \\ d_k &= \int_{1}^{3} t^k(1 - |t-2|)dt \end{aligned}$$

for $k \geq 0$. Show that $\alpha, \beta, \gamma, \delta$ must solve the equation

$$\begin{pmatrix} a_0 & b_0 & g_0 & d_0 \\ a_1 & b_1 & g_1 & d_1 \\ a_2 & b_2 & g_2 & d_2 \\ a_3 & b_3 & g_3 & d_3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix},$$

and solve this with Matlab.

c. Plot the function defined by (6.20), which you found in b.
Hint: If `t` is the vector of t-values, and you write `(t>=0).*(t<=1).*(1-2*abs(t-0.5))`, you get the points $\phi_{1,1}(t)$.

d. Explain why the coordinate vector of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0$ is

$$[\hat{\psi}]_{\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0} = (-\alpha, -\beta, -\delta, 0, \ldots, 0 - \gamma) \oplus (1, 0, \ldots, 0).$$

Hint: you can also compare with Equation (6.18) here. The placement of $-\gamma$ may seem a bit strange here, and has to with that $\phi_{0,-1}$ is not one of the basis functions $\{\phi_{0,n}\}_{n=0}^{N-1}$. However, we have that $\phi_{0,-1} = \phi_{0,N-1}$, i.e. $\phi(t+1) = \phi(t-N+1)$, since we always assume that the functions we work with have period N.

e. Sketch a more general procedure than the one you found in b., which can be used to find wavelet bases where we have even more vanishing moments.

3. Let $\phi(t)$ be the function we used when we defined the Haar-wavelet.

a. Compute $\mathrm{proj}_{V_0}(f(t))$, where $f(t) = t^2$, and where f is defined on $[0, N)$.

b. Find constants α, β so that $\hat{\psi}(t) = \psi(t) - \alpha\phi_{0,0}(t) - \beta\phi_{0,1}(t)$ has two vanishing moments, i.e. so that $\langle \hat{\psi}, 1\rangle = 0$, and $\langle \hat{\psi}, t\rangle = 0$. Plot also the functiin $\hat{\psi}$.
Hint: Start with computing the integrals $\int \psi(t)dt$, $\int t\psi(t)dt$, $\int \phi_{0,0}(t)dt$, $\int \phi_{0,1}(t)dt$, and $\int t\phi_{0,0}(t)dt$, $\int t\phi_{0,1}(t)dt$.

c. Express ϕ and $\hat{\psi}$ with the help of functions from $\boldsymbol{\phi}_1$, and use this to write down the change of coordinate matrix from $\{\boldsymbol{\phi}_0, \hat{\boldsymbol{\psi}}_0\}$ to $\boldsymbol{\phi}_1$.

4. It is also possible to add more vanishing moments to the Haar wavelet. Define

$$\hat{\psi} = \psi_{0,0} - a_0\phi_{0,0} - \cdots - a_{k-1}\phi_{0,k-1}.$$

Define also $c_{r,l} = \int_l^{l+1} t^r dt$, and $e_r = \int_0^1 t^r \psi(t)dt$.

a. Show that $\hat{\psi}$ has k vanishing moments if and only if $a_0, \ldots, a_{k-1}$ solves the equation

$$\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k-1,0} & c_{k-1,1} & \cdots & c_{k-1,k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} \tag{6.21}$$

b. Write a function

```
function a=vanishingmomshaar(k)
```

which solves Equation 6.21, and returns $a_0, a_1, \ldots, a_{k-1}$ in the vector `a`.

6.4 Summary

We defined another wavelet, which was a function approximation scheme based on piecewise linear functions, instead of piecewise constant functions as in the previous chapter. There were several differences with the new wavelet when compared to the previous one. First of all, the basis functions were not orthonormal, and we did not attempt to make them orthonormal. The resolution spaces are not defined in terms of orthogonal bases, and we had some freedom on how we defined the detail spaces, since they are not defined as orthogonal complements anymore. Similarly, we had some freedom on how we define the mother wavelet, and we saw that we could define it so that it is more suitable for approximation of functions, by adding what we called vanishing moments.

We generalized the concept of multiresolution analysis so that it also applies in this new setting, and established some more properties. We will continue in the next chapter to construct even more general wavelets, based on this extended framework.

Chapter 7

Wavelets and filters

Previously we have seen that analog filters restricted to the Fourier spaces gave rise to digital filters. These digital filters sent the samples of the input function to the samples of the output function, and they are easily implementable in contrast to the analog filters.

We have also seen that wavelets give rise to analog filters. This leads us to believe that the DWT also can be implemented in terms of digital filters. In this chapter we will prove that this is in fact the case. There are some differences, however. First of all, the DWT is not constructed by looking at the samples of a function, but rather by looking at coordinates in a given basis. Secondly, the function spaces we work in (i.e. V_m) are different from the Fourier spaces. Thirdly, we saw that the wavelet transform gave rise to two different types of analog filters: The filter defined by Equation 6.10 for obtaining $c_{m,n}$, and the filter defined by Equation 6.11 for obtaining $w_{m,n}$. We want both to correspond to digital filters. Due to these differences, the way we realize wavelet transformations in terms of digital filters is a bit different from before. Despite the differences, this chapter will make it clear that the output of a wavelet transform can be interpreted as the output of two different filters, and each filter will have a clear interpretation in terms of frequency representations.

7.1 The filters corresponding to a wavelet transformation

We will make the connection with digital filters by looking again at the different examples of wavelet bases we have seen: The one for piecewise constant functions, and the one for piecewise linear functions. We start by reordering the basis vectors in $(\boldsymbol{\phi}_0, \boldsymbol{\psi}_0)$ as

$$\mathcal{C}_1 = \{\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \cdots, \phi_{0,N-1}, \psi_{0,N-1}\}. \tag{7.1}$$

The subscript 1 is used since $\mathcal{C}_1$ is a basis for V_1. It turns out to be useful to reorder of the basis functions since it makes it easier to write down the change of coordinates matrices. To be more precise, let us first consider the Haar wavelet. From formula (5.16) it is apparent that $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ is the matrix where

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

is repeated along the main diagonal N times. Also, from formula (5.15) it is apparent that $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ is the same matrix. Such matrices are called *block diagonal matrices*. This particular block diagonal matrix is clearly orthogonal, since it transforms one orthonormal base to another.

For higher m we also reorder the basis vectors for $(\boldsymbol{\phi}_{m-1}, \boldsymbol{\psi}_{m-1})$ as in Equation (7.1), i.e. we define

$$\mathcal{C}_m = \{\phi_{m-1,0}, \psi_{m-1,0}, \phi_{m-1,1}, \psi_{m-1,1}, \cdots, \phi_{m-1,2^{m-1}N-1}, \psi_{m-1,2^{m-1}N-1}\}. \tag{7.2}$$

The bases $\boldsymbol{\phi}_m$ and $\mathcal{C}_m$ are both referred to as *wavelet bases* . Again, both change of coordinates matrices $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ can be obtained by repeating the matrix from Equation (7.1) along the diagonal, but this time it is repeated $2^{m-1}N$ times.

For the piecewise linear wavelet, if we define the bases $\mathcal{C}_m$ again by Equation 7.1 (i.e. with ϕ and ψ replaced). Equation 6.7 gives that the first two columns in $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ take the form

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 1/2 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & 0 \end{pmatrix}. \tag{7.3}$$

The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly, Equation 6.9 gives that the first two columns in $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ take the form

$$\sqrt{2} \begin{pmatrix} 1 & 0 \\ -1/2 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ -1/2 & 0 \end{pmatrix}. \tag{7.4}$$

Also here, the remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix.

For the alternative piecewise linear wavelet, Equation 6.19 give all columns in the change of coordinate matrix $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ also, when the spaces $\boldsymbol{\phi}_m$, $\mathcal{C}_m$ instead are defined in terms of the function $\hat{\psi}$, and ϕ. In particular, the first two columns in this matrix are

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1/4 \\ 1/2 & 3/4 \\ 0 & -1/4 \\ 0 & -1/8 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & -1/8 \end{pmatrix}. \tag{7.5}$$

The first column is the same as before, since there was no change in the definition of ϕ. The remaining columns are obtained by shifting this, as in a circulant Toeplitz matrix. Similarly we could compute the change of coordinate matrix the opposite way, $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$. We will explain shortly how this can be done.

In each case above it turned out that the change of coordinate matrices $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$ had a special structure: They were obtained by repeating the first two columns in a circulant way, similarly to how we did in a circulant Toeplitz matrix. The matrices were not exactly circulant Toeplitz matrices, however, since there are two different columns repeating. The change of coordinate matrices occuring in the stages in a DWT are thus not digital filters, but they seem to be related. Let us start by giving these new matrices names:

Definition 7.1 (MRA-matrices)**.** An $N \times N$-matrix T, with N even, is called an MRA-matrix if the columns are translates of the first two columns in alternating order, in the same way as the columns of a circulant Toeplitz matrix.

From our previous calculations it is clear that, once ϕ and ψ are given through an MRA, the corresponding change of coordinate matrices will always be MRA-matrices. The MRA-matrices is our connection between filters and wavelets. Let us make the following definition:

Definition 7.2. We denote by H_0 the (unique) filter with the same first row as $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$, and by H_1 the (unique) filter with the same second row as $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$. H_0 and H_1 are also called the *DWT filter components*.

Using this definition it is clear that

$$\begin{aligned} (P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} \boldsymbol{c}_m)_k &= (H_0 \boldsymbol{c}_m)_k && \text{when } k \text{ is even} \\ (P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} \boldsymbol{c}_m)_k &= (H_1 \boldsymbol{c}_m)_k && \text{when } k \text{ is odd} \end{aligned}$$

since the left hand side depends only on row k in the matrix $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$, and this is equal to row k in H_0 (when k is even) or row k in H_1 (when k is odd). This means that $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m} \boldsymbol{c}_m$ can be computed with the help of H_0 and H_1 as follows:

Theorem 7.3 (DWT expressed in terms of filters)**.** Let $\boldsymbol{c}_m$ be the coordinates in $\boldsymbol{\phi}_m$, and let H_0, H_1 be defined as above. Any stage in a DWT can ble implemented in terms of filters as follows:

1. Compute $H_0\boldsymbol{c}_m$. The even-indexed entries in the result are the cordinates $\boldsymbol{c}_{m-1}$ in $\boldsymbol{\phi}_{m-1}$.

2. Compute $H_1\boldsymbol{c}_m$. The odd-indexed entries in the result are the coordinates $\boldsymbol{w}_{m-1}$ in $\boldsymbol{\psi}_{m-1}$.

This gives an important connection between wavelets and filters: The DWT corresponds to applying two filters, H_0 and H_1, and the result from the DWT is produced by assembling half of the coordinates from each. In practice we do not compute the full application of the filters, since only half of the result is needed. We can now complement Figure 5.3 by giving names to the arrows as follows:

$$\begin{tikzcd}
V_m \arrow[r, "H_0"] \arrow[dr, "H_1"] & V_{m-1} \arrow[r, "H_0"] \arrow[dr, "H_1"] & V_{m-2} \arrow[r, "H_0"] \arrow[dr, "H_1"] & \cdots \arrow[r, "H_0"] \arrow[dr, "H_1"] & V_0 \\
 & W_{m-1} & W_{m-2} & W_{m-3} & W_0
\end{tikzcd}$$

Let us make a similar anlysis for the IDWT, and let us first make the following definition:

Definition 7.4. We denote by G_0 the (unique) filter with the same first column as $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, and by G_1 the (unique) filter with the same second column as $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$. G_0 and G_1 are also called the *IDWT filter components.*

These filters are uniquely determined, since any filter is uniquely determined from one of its columns. We can now write

$$\begin{aligned}
P_{\boldsymbol{\phi}_m\leftarrow\mathcal{C}_m}\begin{pmatrix} c_{m-1,0}\\ w_{m-1,0}\\ c_{m-1,1}\\ w_{m-1,1}\\ \cdots\\ c_{m-1,2^{m-1}N-1}\\ w_{m-1,2^{m-1}N-1}\end{pmatrix} &= P_{\boldsymbol{\phi}_m\leftarrow\mathcal{C}_m}\left(\begin{pmatrix} c_{m-1,0}\\ 0\\ c_{m-1,1}\\ 0\\ \cdots\\ c_{m-1,2^{m-1}N-1}\\ 0\end{pmatrix}+\begin{pmatrix} 0\\ w_{m-1,0}\\ 0\\ w_{m-1,1}\\ \cdots\\ 0\\ w_{m-1,2^{m-1}N-1}\end{pmatrix}\right)\\
&= P_{\boldsymbol{\phi}_m\leftarrow\mathcal{C}_m}\begin{pmatrix} c_{m-1,0}\\ 0\\ c_{m-1,1}\\ 0\\ \cdots\\ c_{m-1,2^{m-1}N-1}\\ 0\end{pmatrix}+P_{\boldsymbol{\phi}_m\leftarrow\mathcal{C}_m}\begin{pmatrix} 0\\ w_{m-1,0}\\ 0\\ w_{m-1,1}\\ \cdots\\ 0\\ w_{m-1,2^{m-1}N-1}\end{pmatrix}\\
&= G_0\begin{pmatrix} c_{m-1,0}\\ 0\\ c_{m-1,1}\\ 0\\ \cdots\\ c_{m-1,2^{m-1}N-1}\\ 0\end{pmatrix}+G_1\begin{pmatrix} 0\\ w_{m-1,0}\\ 0\\ w_{m-1,1}\\ \cdots\\ 0\\ w_{m-1,2^{m-1}N-1}\end{pmatrix}.
\end{aligned}$$

Here we have split a vector into its even-indexed and odd-indexed elements, which correspond to the coefficients from $\boldsymbol{\phi}_{m-1}$ and $\boldsymbol{\psi}_{m-1}$, respectively. In the last equation, we replaced with G_0, G_1, since the multiplications with $P_{\boldsymbol{\phi}_m\leftarrow\mathcal{C}_m}$ depend only on the even and odd columns in that matrix (due to the zeros inserted), and these columns are equal in G_0, G_1. We can now state the following characterization of the inverse Discrete Wavelet transform:

Theorem 7.5 (IDWT expressed in terms of filters)**.** Let G_0, G_1 be defined as above. Any stage in an IDWT can be implemented in terms of filters as follows:

$$\boldsymbol{c}_m = G_0\begin{pmatrix} c_{m-1,0}\\ 0\\ c_{m-1,1}\\ 0\\ \cdots\\ c_{m-1,2^{m-1}N-1}\\ 0\end{pmatrix}+G_1\begin{pmatrix} 0\\ w_{m-1,0}\\ 0\\ w_{m-1,1}\\ \cdots\\ 0\\ w_{m-1,2^{m-1}N-1}\end{pmatrix}. \tag{7.6}$$

We can now also complement Figure 5.3 for the IDWT with named arrows as follows:

$$\begin{tikzcd}
V_m & V_{m-1} \arrow[l, "G_0"'] & V_{m-2} \arrow[l, "G_0"'] & \cdots \arrow[l, "G_0"'] & V_0 \arrow[l, "G_0"'] \\
 & W_{m-1} \arrow[ul, "G_1"] & W_{m-2} \arrow[ul, "G_1"] & W_{m-3} \arrow[ul, "G_1"] & W_0 \arrow[ul, "G_1"]
\end{tikzcd}$$

Note that the filters G_0, G_1 were defined in terms of the columns of $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, while the filters H_0, H_1 were defined in terms of the rows of $P_{\mathcal{C}_m \leftarrow \boldsymbol{\phi}_m}$. This difference is seen from the computations above to come from that the change of coordinates one way splits the coordinates into two parts, while the inverse change of coordinates performs the opposite.

There are several reasons why it is smart to express a wavelet transformation in terms of filters. First of all, it enables us to reuse theoretical results from the world of filters in the world of wavelets, and to give useful interpretations of the wavelet transform in terms of frequencies. Secondly, and perhaps most important, it enables us to reuse efficient implementations of filters in order to compute wavelet transformations. A lot of work has been done in order to establish efficient implementations of filters, due to their importance.

In Example 5.21 we argued that the elements in V_{m-1} correspond to frequencies at lower frequencies than those in V_m, since $V_0 = \text{Span}(\phi_{0,n})$ should be interpreted as content of lower frequency than the $\phi_{1,n}$, with $W_0 = \text{Span}(\psi_{0,n}$ the remaining high frequency detail. To elaborate more on this, we have that have that

$$\phi(t) = \sum_{n=0}^{2N-1} (G_0)_{n,0}\phi_{1,n}(t) \tag{7.7}$$

$$\psi(t) = \sum_{n=0}^{2N-1} (G_1)_{n,1}\psi_{1,n}(t)., \tag{7.8}$$

where $(G_k)_{i,j}$ are the entries in the matrix G_k. Similar equations are true for $\phi(t-k), \psi(t-k)$. Due to (7.7), the filter G_0 should have lowpass filter characteristics, since it extracts the information at lower frequencies. G_1 should have highpass filter characteristics du to (7.8). Let us verify this for the different wavelets we have defined up to now.

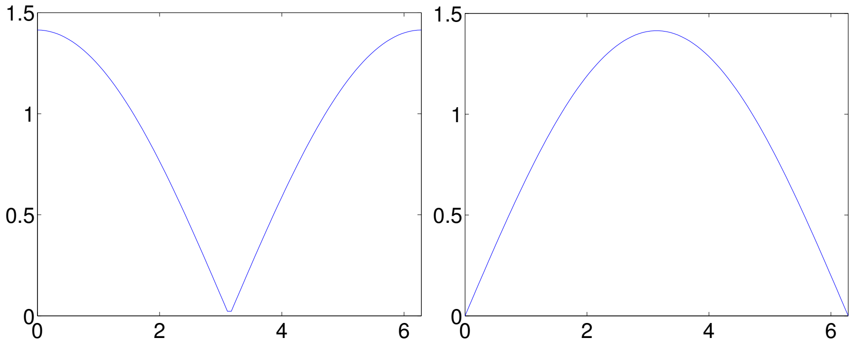
Let us also consider the frequency responses for the filters arising from the wavelets we have looked at. We start with the Haar wavelet.

Example 7.6. For the Haar wavelet we saw that, in $P_{\boldsymbol{\phi}_m \leftarrow \mathcal{C}_m}$, the matrix

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

repeated along the diagonal. From this it is clear that

$$G_0 = \{\underline{1/\sqrt{2}}, 1/\sqrt{2}\}$$
$$G_1 = \{1/\sqrt{2}, \underline{-1/\sqrt{2}}\}.$$



(a) $\lambda_{G_0}(\omega)$

(b) $\lambda_{G_1}(\omega)$

Figure 7.1: The frequency responses for the MRA of piecewise constant functions.

We have seen these filters previously: G_0 is a movinge average filter of two elements (up to multiplication with a constant). This is a lowpass filter. G_1 is a bass-reducing filter, which is a highpass filter. Up to a constant, this is also an approximation to the derivative. Since G_1 is constructed from G_0 by adding an alternating sign to the filter coefficients, we know from before that G_1 is the highpass filter corresponding to the lowpass filter G_0, so that the frequency response of the second is given by a shift of frequency with π in the first. The frequency responses are

$$\begin{aligned}\lambda_{G_0}(\omega) &= \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}e^{-i\omega} = \sqrt{2}e^{-i\omega/2}\cos(\omega/2)\\ \lambda_{G_1}(\omega) &= \frac{1}{\sqrt{2}}e^{i\omega} - \frac{1}{\sqrt{2}} = \sqrt{2}ie^{i\omega/2}\sin(\omega/2).\end{aligned}$$

The magnitude of these are plotted in Figure 7.1, where the lowpass/highpass characteristics are clearly seen. In this case we also have that

$$\begin{aligned}H_0 &= \{\underline{1/\sqrt{2}}, 1/\sqrt{2}\}\\ H_1 &= \{1/\sqrt{2}, \underline{-1/\sqrt{2}}\},\end{aligned}$$

so that the frequency responses for the DWT have the same lowpass/highpass characteristics. ♣

It turns out that this connection between G_0 and G_1 as lowpass and highpass filters corresponding to each other can be found in all orthonormal wavelets. We will prove this in the next chapter.

Example 7.7. For the first wavelet for piecewise linear functions we looked at

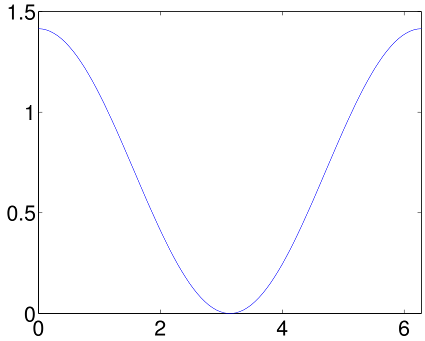


Figure 7.2: The frequency response $\lambda_{G_0}(\omega)$ for the first choice of wavelet for piecewise linear functions

in the previous section, Equation (7.3) gives that

$$\begin{aligned} G_0 &= \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\} \\ G_1 &= \frac{1}{\sqrt{2}}\{\underline{1}\}. \end{aligned} \tag{7.9}$$

G_0 is again a filter we have seen before: Up to multiplication with a constant, it is the treble-reducing filter with values from row 2 of Pascal's triangle. We see something different here when compared to the Haar wavelet, in that the filter G_1 is not the highpass filter corresponding to G_0. The frequency responses are now

$$\begin{aligned} \lambda_{G_0}(\omega) &= \frac{1}{2\sqrt{2}}e^{i\omega} + \frac{1}{\sqrt{2}} + \frac{1}{2\sqrt{2}}e^{-i\omega} = \frac{1}{\sqrt{2}}(\cos\omega + 1) \\ \lambda_{G_1}(\omega) &= \frac{1}{\sqrt{2}}. \end{aligned}$$

$\lambda_{G_1}(\omega)$ thus has magnitude $\frac{1}{\sqrt{2}}$ at all points. The magnitude of $\lambda_{G_0}(\omega)$ is plotted in Figure 7.2. Comparing with Figure 7.1 we see that here also the frequency response has a zero at π. The frequency response seems also to be flatter around π. For the DWT, Equation (7.4) gives us

$$\begin{aligned} H_0 &= \sqrt{2}\{\underline{1}\} \\ H_1 &= \sqrt{2}\{-1/2, \underline{1}, -1/2\}. \end{aligned} \tag{7.10}$$

Even though G_1 was not the highpass filter corresponding to G_0, we see that, up to a constant, H_1 is (it is a bass-reducing filter with values taken from row 2 of Pascals triangle). ♣

Note that the role of H_1 as the highpass filter corresponding to G_0 is the case in both previous examples. We will prove in the next chapter that this is a much more general result which holds for all wavelets, not only for the orthonormal ones.

Exercises for Section 7.1

1. Write down the corresponding filters G_0 og G_1 for Exercise 6.3.3. Plot their frequency responses, and characterize the filters as lowpass- or highpass filters.

2. Find two symmetric filters, so that the corresponding MRA-matrix, constructed with alternating rows from these two filters, is not a symmetric matrix.

3. Assume that an MRA-matrix is symmetric. Are the corresponding filters H_0, H_1, G_0, G_1 also symmetric? If not, find a counterexample.

7.2 Perfect reconstruction systems

If we choose a good wavelet, the output of the DWT is more suitable for compression purposes. The DWT and IDWT are inverse operations, and this pair therefore fits into the following framework:

Definition 7.8 (Perfect reconstruction system)**.** By a *perfect reconstruction system* we mean a pair of $N \times N$-matrices (G, H) so that $GH = I$. For a vector $\boldsymbol{x}$ we refer to $\boldsymbol{z} = H\boldsymbol{x}$ as the *transformed vector.* For a vector $\overline{\boldsymbol{z}}$ we refer to $\overline{\boldsymbol{x}} = G\overline{\boldsymbol{z}}$ as the *reconstructed vector.*

The names perfect reconstruction, transformation, and reconstruction come from signal processing, where one thinks of H as a transform, and G as the inverse transform which reconstructs the input to the first transform from its output. In practice, we are interested in finding perfect reconstruction systems where the transformed $H\boldsymbol{x}$ is so that it is more suitable for further processing, such as compression, or playback in an audio system. Clearly $((F_N)^H, F_N)$ is a perfect reconstruction system for any N, and similarly for the DCT. One problem with these systems is that the matrices are not sparse. Although efficient algorithms exist for the DFT, one may find systems of sparse matrices or other types of matrices which are quicker to compute in the transform and reconstruction steps. In practice we are interested in establishing perfect reconstruction systems, where the involved matrices have particular forms.

Digital filters is one such form. Many filters are invertible, and both the filter and the inverse filter are quick to compute as long as one one of them has few nonzero filter coefficients.

Another form is a perfect reconstruction system constructed from a wavelet. This form offers increased flexibility when compared to using only one filter, since the two filters can concentrate on different frequencies: Low frequencies, and high frequencies.

7.3 Wavelets with symmetric filters

As mentioned, it is desirable to apply the DWT to the symmetric extension of the input when the wavelet is symmetric. This really corresponds to considering the symmetric extension $\breve{f}$ instead of f, where f is our model. If the input $\boldsymbol{x}$ to the DWT is the samples $(f(n/2^m))_{n=0}^{2^m N-1}$, we create a vector $\breve{\boldsymbol{x}}$ representing the samples of $\breve{f}$. It is clear that this vector should be

$$\breve{\boldsymbol{x}} = \left((f(n/2^m))_{n=0}^{2^m N-1}, \lim_{t \to N_-} f(t), (f((2^m N - n)/2^m))_{n=1}^{2^m N-1} \right).$$

In this vector there is symmetry around entry $2^m N$, so that the vector is determined from the $N' = 2^m N + 1$ first elements. Also the boundary is not duplicated, contrary to previous symmetric extensions. Instead using N' as the length of the input vector, we thus need to define a symmetric extension slightly different from before:

Definition 7.9 (Symmetric extension of a vector)**.** By the symmetric extension of $\boldsymbol{x} \in \mathbb{R}^N$, we mean $\breve{\boldsymbol{x}} \in \mathbb{R}^{2N-2}$ defined by

$$\breve{\boldsymbol{x}}_k = \begin{cases} x_k & 0 \leq k < N \\ x_{2N-2-k} & N \leq k < 2N-3 \end{cases} \tag{7.11}$$

With this notation, $N-1$ is the symmetry point in all symmetric extensions. Clearly symmetric filters preserve symmetry around $N-1$. Applying a filter to this kind of symmetric extension in $\mathbb{R}^{2N-2}$ can therefore be viewed as a mapping from $\mathbb{R}^N$ to $\mathbb{R}^N$. If S is a symmetric filter, we will as before write S_r for this operation. We have the following analog to Theorem 4.9 for these new mappings, which is proved in very much the same way.

Theorem 7.10. With $S = \begin{pmatrix} S_1 & S_2 \\ S_3 & S_4 \end{pmatrix}$ a symmetric filter, we have that $S_r = S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix}$.

Proof. We write

$$\begin{pmatrix} y_0 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \\ \hline x_N \\ \vdots \\ x_{2N-3} \end{pmatrix} = S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_N \\ \vdots \\ x_{2N-3} \end{pmatrix}.$$

When $\boldsymbol{x}$ is a symmetric vector we can rewrite this as

$$
\begin{aligned}
& S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + S_2 \begin{pmatrix} x_{N-2} \\ \vdots \\ x_1 \end{pmatrix} \\
&= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + (S_2)^f \begin{pmatrix} x_1 \\ \vdots \\ x_{N-2} \end{pmatrix} \\
&= S_1 \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} \\
&= \begin{pmatrix} S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{N/2-1} \end{pmatrix}.
\end{aligned}
$$

This shows that $S_r = S_1 + \begin{pmatrix} 0 & (S_2)^f & 0 \end{pmatrix}$. □

In practice we want to apply the wavelet transform to a symmetric extension, since then symmetric filters can give a better approximation to the underlying analog filters. In order to achieve this, the following result says that we only need to replace the filters H_0, H_1, G_0, and G_1 in the wavelet transform with $(H_0)_r$, $(H_1)_r$, $(G_0)_r$, and $(G_1)_r$.

Theorem 7.11. If the filters H_0, H_1, G_0, and G_1 in a wavelet transform are symmetric, then the corresponding MRA matrices preserve symmetric extensions (as defined above). Also, applying the filters H_0, H_1, G_0, and G_1 to $\breve{\boldsymbol{x}} \in \mathbb{R}^{2N-2}$ in the DWT/IDWT is equivalent to applying $(H_0)_r$, $(H_1)_r$, $(G_0)_r$, and $(G_1)_r$ to $\boldsymbol{x} \in \mathbb{R}^N$ in the same way.

Proof. Since H_0 and H_1 are symmetric, their output from $\breve{\boldsymbol{x}}$ is also a symmetric vector, and by assembling their outputs as the even- and odd-indexed entries, we see that the output $(c_0, w_0, c_1, w_1, \ldots)$ of the MRA-matrix H also is a symmetric vector. The same then applies for the matrix G, since it inverts the first. This proves the first part.

Now, assume that $\boldsymbol{x} \in \mathbb{R}^N$. By definition of $(H_i)_r$, $(H_i\breve{\boldsymbol{x}})_n = ((H_i)_r\boldsymbol{x})_n$ for $0 \leq n \leq N-1$. This means that we get the same first N output elements in a wavelet transform if we repace H_0, H_1 with $(H_0)_r, (H_1)_r$. Since the vectors $(c_0, 0, c_1, 0, \ldots)$ and $(0, w_0, 0, w_1, \ldots)$ also are symmetric vectors when $(c_0, w_0, c_1, w_1, \ldots)$ is, it follows that $(G_0)_r, (G_1)_r$ will reproduce the same first N elements as G_0, G_1 also. In conclusion, for symmetric vectors, the wavelet transform restricted to the first N elements produces the same result when we replace H_0, H_1, G_0, and G_1 with $(H_0)_r$, $(H_1)_r$, $(G_0)_r$, and $(G_1)_r$. This proves the result. □

Exercises for Section 7.3

1. In the literature it is not customary to write down an orthonormal basis for the eigenvectors of S_r when symmetric extensions is defined as in this section. Show that $\{\cos\left(2\pi\frac{n}{2N-2}k\right)\}_{n=0}^{N-1}$ is an orthogonal basis of eigenvectors for S_r.

7.4 Filter-based algorithm for the DWT and the IDWT

When we apply wavelets in practice, they are often defined in terms of the filters H_0, H_1, G_0, G_1. We therefore need an implementation of the DWT and the IDWT which takes as input the filter coefficients of these filters. All wavelets we look at, except for the Haar wavelet (which we have already implemented), will have symmetric filters, so we will restrict our implementations to symmetric filters. You will be spared writing these implementations: you can assume that the function

```
xnew=DWTImpl(h0,h1,x,m)
```

takes as input symmetric filters `h0` and `h1`, a vector `x`, and `m`, and returns the result of the m-level DWT for us, i.e. it returns the coordinates of `x` in the basis $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$. You can also assume that the function

```
x=IDWTImpl(g0,g1,xnew,m)
```

performs the m-level IDWT in a similar way, where `g0` and `g1` also are symmetric filters.

Although you are spared writing these implementations, we will comment on some of the issues in writing them. Previously we mentioned that we could use Matlab's `conv` function to implement filters. Since the DWT and the IDWT now have been expressed in terms of filters, the `conv`-function can be used to implement the DWT and the IDWT as well. More precisely, they can be implemented by taking the convolution with the rows (since the entries in the first column equal the enctries in the first row when the filter is symmetric) of the MRA-matrix. With the DWT we have these rows, since the rows in $P_{\mathcal{C}_m \leftarrow \phi_m}$ are given by the rows of H_0, H_1. With the IDWT we need a translation from the column representation of the MRA-matrix to the row representation of the matrix. This is a straightforward task, however a bit tedious. In the function `IDWTImpl` you will see some additional code at the beginning taking care of this.

Symmetric filters are very common in practice, since MRA-matrices with symmetric filters also preserve symmetric vectors. Due to this they share some of the desirable properties of symmetric filters (which lead us to the definition of the DCT). `DWTImpl` and `IDWTImpl` are written so that they are applied to the symmetric extension of the sound. Note also that, since these implementations call the `conv`-function, they compute all output elements of the filter. But Theorem 7.3 states that only half of the output entries for each filter need to be computed. The current implementation is therefore not optimal. In Chapter 9 we will describe a method which solves this issue, and which also provides further computational savings when compared to a direct implementation of the filters.

Finally, let us provide an example of how we write down the parameters `h0, h1, g0, g1` to the implementations. They represent the nonzero filter coefficients of H_0, H_1, G_0, G_1, respectively. If

$$\begin{aligned} H_0 &= \{h_{0,-k_0}, \ldots, h_{0,-1}\underline{h_{0,0}}, h_{0,1}, \ldots, h_{0,k_0}\} \\ H_1 &= \{h_{1,-k_1}, \ldots, h_{1,-1}\underline{h_{1,0}}, h_{1,1}, \ldots, h_{1,k_1}\}, \end{aligned}$$

then the vectors

$$\begin{aligned} \texttt{h0} &= (h_{0,-k_0}, \ldots, h_{0,-1}, h_{0,0}, h_{0,1}, \ldots, h_{0,k_0}) \\ \texttt{h1} &= (h_{1,-k_1}, \ldots, h_{1,-1}, h_{1,0}, h_{1,1}, \ldots, h_{1,k_1}) \end{aligned}$$

should be input to the `DWTImpl` function.

Example 7.12. In Exercise 3 you will be asked to implement a function `playDWTfilterslower` which plays the low-resolution approximations of our audio test file, for any type of wavelet, using the functions we have described. With this function we can play the result for all the wavelets we have considered up to now, in succession, and at a given resolution, with the following code:

```
function playDWTall(m)
disp('Haar wavelet');
playDWTlower(m);
disp('Wavelet for piecewise linear functions');
playDWTfilterslower(m,sqrt(2)*[1],...
    sqrt(2)*[-1/2 1 -1/2],...
    [1/2 1 1/2]/sqrt(2),...
    [1]/sqrt(2));
disp('Wavelet for piecewise linear functions, alternative  version');
playDWTfilterslower(m,...
   sqrt(2)*[-1/8 1/4 3/4 1/4 -1/8],...
   sqrt(2)*[-1/2 1 -1/2],...
   [1/2 1 1/2]/sqrt(2),...
   [-1/8 -1/4 3/4 -1/4 -1/8]/sqrt(2));
```

The call to `playDWTlower` first plays the result, using the Haar wavelet. The code then moves on to the piecewise linear wavelet. From Equation (7.10) we first see that

$$\texttt{h0} = (h_{0,-k_0}, \ldots, h_{0,-1}, h_{0,0}, h_{0,1}, \ldots, h_{0,k_0}) = \sqrt{2}(1) \tag{7.12}$$

$$\texttt{h1} = (h_{1,-k_1}, \ldots, h_{1,-1}, h_{1,0}, h_{1,1}, \ldots, h_{1,k_1}) = \sqrt{2}(-1/2, 1, -1/2), \tag{7.13}$$

and from Equation (7.9) we see that

$$\texttt{g0} = (g_{0,0}, g_{0,1}, \ldots, g_{0,l_0}) = \frac{1}{\sqrt{2}}(1/2, 1, 1/2) \tag{7.14}$$

$$\texttt{g1} = (g_{1,0}, g_{1,1}, \ldots, g_{1,l_1}) = \frac{1}{\sqrt{2}}(1). \tag{7.15}$$

These explain the parameters to the call to `playDWTfilterslower` for the piecewise linear wavelet. The code then moves to the alternative piecewise linear wavelet. For this wavelet we have not computed all filter coefficicents yet. This is delayed till Example 8.4, since we need some more techniques in order to compute these. From Equation (8.12) in that example we see that

$$\begin{aligned}
\texttt{h0} &= (h_{0,-k_0}, \ldots, h_{0,-1}, h_{0,0}, h_{0,1}, \ldots, h_{0,k_0}) = \sqrt{2}(-1/8, 1/4, 3/4, 1/4, -1/8) \\
\texttt{h1} &= (h_{1,-k_1}, \ldots, h_{1,-1}, h_{1,0}, h_{1,1}, \ldots, h_{1,k_1}) = \sqrt{2}(-1/2, 1, -1/2),
\end{aligned}$$

and from Equation (8.10) we see that

$$\begin{aligned}
\texttt{g0} &= (g_{0,-l_0}, \ldots, g_{0,-1}, g_{0,0}, g_{0,1}, \ldots, g_{0,l_0}) = \frac{1}{\sqrt{2}}(1/2, 1, 1/2) \\
\texttt{g1} &= (g_{1,-l_1}, \ldots, g_{1,-1}, g_{1,0}, g_{1,1}, \ldots, g_{1,l_1}) = \frac{1}{\sqrt{2}}(-1/8, -1/4, 3/4, -1/4, -1/8).
\end{aligned}$$

These explain the parameters to the call to `playDWTfilterslower` for the alternative piecewise linear wavelet. ♣

Exercises for Section 7.4

1. In this exercise we will practice setting up the parameters `h0,h1,g0,g1` which are used in the calls to `DWTImpl` and `IDWTImpl`.

a. Assume that one stage in a DWT is given by the MRA-matrix

$$P_{\mathcal{C}_1 \leftarrow \phi_1} = \begin{pmatrix}
1/5 & 1/5 & 1/5 & 0 & 0 & 0 & \cdots & 0 & 1/5 & 1/5 \\
-1/3 & 1/3 & -1/3 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
1/5 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & -1/3 & 1/3 & -1/3 & 0 & \cdots 0 & 0 & & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}$$

Write down the compact form for the corresponding filters H_0, H_1, and compute and plot the frequency responses. Are the filters symmetric? If so, also write down the parameters `h0,h1` you would use for this matrix in a call to `DWTImpl`.

b. Assume that one stage in the IDWT is given by the MRA-matrix

$$P_{\phi_1 \leftarrow \mathcal{C}_1} = \begin{pmatrix} 1/2 & -1/4 & 0 & 0 & \cdots \\ 1/4 & 3/8 & 1/4 & 1/16 & \cdots \\ 0 & -1/4 & 1/2 & -1/4 & \cdots \\ 0 & 1/16 & 1/4 & 3/8 & \cdots \\ 0 & 0 & 0 & -1/4 & \cdots \\ 0 & 0 & 0 & 1/16 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots \\ 1/4 & 1/16 & 0 & 0 & \cdots \end{pmatrix}$$

Write down the compact form for the filters G_0, G_1, and compute and plot the frequency responses. Are the filters symmetric? If so, also write down the parameters `g0,g1` you would use for this matrix in a call to `IDWTImpl`.

2. Let us also practice on writing down the change of coordinate matrices from the parameters `h0,h1,g0,g1`.

a. Assume that `h0=[1/16 1/4 3/8 1/4 1/16]` and `h1=[-1/4 1/2 -1/4]`. Write down the compact form for the filters H_0, H_1. Plot the frequency responses and verify that H_0 is a lowpass filter, and that H_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\mathcal{C}_1 \leftarrow \phi_1}$ for the wavelet corresponding to these filters.

b. Assume that `g0=[1/3 1/3 1/3]` and `g1=[1/5 -1/5 1/5 -1/5 1/5]`. Write down the compact form for the filters G_0, G_1. Plot the frequency responses and verify that G_0 is a lowpass filter, and that G_1 is a highpass filter. Also write down the change of coordinate matrix $P_{\phi_1 \leftarrow \mathcal{C}_1}$ for the wavelet corresponding to these filters.

3. Write a function

```
function playDWTfilterslower(m,h0,h1,g0,g1)
```

which reimplements the function `playDWTlower` from Exercise 5.3.9 so that it takes as input the coefficients of the four different filters as in Example 7.12. Listen to the result using the different wavelets we have encountered and for different m, using the code from Example 7.12. Can you hear any difference from the Haar wavelet? If so, which wavelet gives the best sound quality?

4. In this exercise we will change the code in Example 7.12 so that it instead only plays the contribution from the detail spaces (i.e. $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$).

a. Reimplement the function you made in Exercise 3 so that it instead plays the contribution from the detail spaces. Call the new function `playDWTfilterslowerdifference`.

b. In Exercise 5.3.11 we implemented a function `playDWTlowerdifference` for listening to the detail/error when the Haar wavelet is used. In the function `playDWTall` from Example 7.12, replace `playDWTlower` and `playDWTfilterslower` with `playDWTlowerdifference` and `playDWTfilterslowerdifference`. Describe the sounds you hear for different m. Try to explain why the sound seems to get louder when you increase m.

5. Let us return to the piecewise linear wavelet from Exercise 6.3.2.

a. With $\hat{\psi}$ as defined as in Exercise 6.3.2 b., compute the coordinates of $\hat{\psi}$ in the basis $\boldsymbol{\phi}_1$ (i.e. $[\hat{\psi}]_{\boldsymbol{\phi}_1}$) with $N = 8$, i.e. compute the IDWT of

$$[\hat{\psi}]_{\boldsymbol{\phi}_0 \oplus \boldsymbol{\psi}_0} = (-\alpha, -\beta, -\delta, 0, 0, 0, 0, -\gamma) \oplus (1, 0, 0, 0, 0, 0, 0, 0),$$

which is the coordinate vector you computed in Exercise 6.3.2 d.. For this, you should use the function `IDWTImpl`, with parameters being the filters G_0, G_1, given as described by `g0,g1` by equations (7.14)-(7.15) in Example 7.12.

b. If we redefine the basis $\mathcal{C}_1$ from $\{\phi_{0,0}, \psi_{0,0}, \phi_{0,1}, \psi_{0,1}, \ldots\}$, to $\{\phi_{0,0}, \hat{\psi}_{0,0}, \phi_{0,1}, \hat{\psi}_{0,1}, \ldots\}$, the vector you obtained in a. gives us an expression for the second column in $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$. After redefining the basis like this, the corresponding filter G_1 has changed from that of the piecewise linear wavelet we started with. Use Matlab to so state the new filter G_1 with our compact filter notation. Also, plot its frequency response.
Hint: Here you are asked to find the unique filter with the same second column as $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$, i.e. the vector from a..

c. Write code which uses Equation (8.11) to find H_0, H_1 from G_0, G_1, and state these filters with our compact filter notation. Also, state the forms `h0,h1`, which should be used in calls to `DWTImpl` for our new wavelet. These replace the forms from equations (7.12)-(7.13) in Example 7.12, which we found for the first piecewise linear wavelet.
Hint: Note that the filter G_0 is unchanged from that of the first piecewise linear wavelet (since ϕ is unchanged when compared to the other wavelets for piecewise linear functions).

d. The filters you have found above should be symmetric, so that we can follow the procedure from Example 7.12 to listen to sound which has been wavelet-transformed by this wavelet. Write a program which plays our audio test file as in Example 7.12 for $m = 1, 2, 3, 4$ (i.e. plays the part in V_0), as well as the difference as in Exercise 4 (i.e. play the part from $W_0 \oplus W_1 \oplus \cdots \oplus W_{m-1}$), where the new filters you have found are used. Listen to the sounds.

6. Repeat the previous exercise for the Haar wavelet as in exercise 4, and plot the corresponding frequency responses for $k = 2, 4, 6$.

7. (Trial Exam UIO V2012) Suppose that we run the following algorithm on the sound represented by the vector `S`:

```
l=length(S);
c=(S(1:2:(l-1))+S(2:2:l))/sqrt(2);
w=(S(1:2:(l-1))-S(2:2:l))/sqrt(2);

newS=[c w];
newS=newS/max(abs(newS));
playerobj=audioplayer(newS,44100);
playblocking(playerobj)
```

a. Comment the code and explain what happens. Which wavelet is used? What do the vectors `c` and `w` represent? Describe the sound you believe you will hear.

b. Assume that we add lines in the code above which sets the elements in the vector `w` to 0 before we compute an inverse wavelet transform What will you hear if you play the new sound you then get?

7.5 Connections with the MP3 standard

There is a very close connection between the way filters are used in the MP3 standard, and the way filters are applied in the context of wavelets. Much literature fails to mention this:

Fact 7.13 (Similarities between the DWT and the MP3 standard)**.** Let $\boldsymbol{x}$ be a vector representing a sound of length $32N$. In the MP3-standard, $\boldsymbol{x}$ is transformed with the help of 32 filters H_0, H_1,. . . , H_{31} in the following way:

1. Compute $H_0\boldsymbol{x}$. The entries with indices from $(0, 32, \ldots, (N-1)32)$ in the result is called $\boldsymbol{z}_0$, and represent the output of the transform at the same indices.

2. Compute $H_1\boldsymbol{x}$. The entries with indices from $(1, 32+1, \ldots, (N-1)32+1)$ in the result is called $\boldsymbol{z}_1$, and represent the output of the transform at the same indices.

3. ...

4. Compute $H_{31}\boldsymbol{x}$. The entries with indices from $(32-1, 64-1, \ldots, N32-1$ in the result is called $\boldsymbol{z}_{31}$, and represent the output of the transform at the same indices.

In the MP3 standard, the vectors $\boldsymbol{z}_i$ are processed further separately after this transformation.

This has a compelling similarity with how we defined the DWT in terms of filters. The difference is that, instead of two filters which produce the output in alternating order, we now have 32 filters which produce the output in alternating order. While the DWT produces $\begin{pmatrix} \boldsymbol{c}_{m-1} \\ \boldsymbol{w}_{m-1} \end{pmatrix}$ as output, one can group the outputs of the transformation in the MP3 standard as $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{31})$. In both cases the filters can be interpreted as concentrating on specific frequency ranges: The filters for a wavelet concentrate on low and high frequencies, while the filters in the MP3 standard concentrate each on a smaller frequency band. The filters were shown in Example 3.36 as examples of filters which concentrate on specific frequency ranges. These filters shared the property of the Haar wavelet that they were just shifted in frequency with respect to oneanother. Since $\boldsymbol{z}_i$ are samples from a filter concentrating on a specific frequency band, its values are also called *subband samples*.

The similarities are also very clear for the IDWT:

Fact 7.14 (Similarities between the IDWT and the MP3 standard)**.** In the MP3 standard, sound is reconstructed from the transform of Fact 7.13 with the help of 32 filters G_0, G_1,..., G_{31} by computing

$$\boldsymbol{x} = G_0\boldsymbol{w}_1 + G_1\boldsymbol{w}_2 + \ldots + G_{M-1}\boldsymbol{w}_{M-1}. \tag{7.16}$$

where $\boldsymbol{w}_i$ is the vector with the components of $\boldsymbol{z}_i$ at indices $(i, 32+i, \ldots, (N-1)32+i)$, and 0 elsewhere.

One difference from the DWT is that the MP3 standard does not apply this transform in stages. The filters used by the MP3 standard work in such a way that the inverse transform of Fact 7.14 exactly inverts the transform of Fact 7.13. The filters used in the MP3 standard thus provide another type of perfect reconstruction system. It is hard to construct such filters which also are good bandpass filters, and we will therefore not go into the details of this.

7.6 Summary

We started this chapter by noting that the change of coordinate matrices in a DWT/IDWT took a particular form. From this form we realized that the DWT could be realized in terms of two digital filters H_0 and H_1, and that the IDWT could be realized in terms of two other digital filters G_0 and G_1. We looked at the frequency responses of these filters for the wavelets we have encountered upto now. From these we saw that G_0, H_0 were lowpass filters, and that G_1, H_1 were highpass filters, and we argued that this would be the case in any wavelet transformation. We also added the remaining parts we needed to make an implementation of these filters and the DWT, such as symmetric extension. We will use this implementation in the coming sections, in order to analyze images.

The filter characterization gives an entirely different view on wavelets: instead of constructing function spaces with certain properties and consider the corresponding filters, we may instead construct filters with given properties (which is a discrete problem), and construct from these the corresponding mother wavelet, scaling function, and function spaces. In practice this is what is done, as we will see in the next chapter.

Chapter 8

Constructing interesting wavelets

In the previous chapter we saw that a wavelet corresponded in linear algebra to perfect reconstruction systems of MRA matrices, and we examplified this in terms of the two simplest wavelets we have looked at. In this chapter we will start by stating a characterization of perfect reconstruction systems of MRA matrices in terms of the involved filters. We will then translate the problem of finding interesting wavelets to that of finding filters with certain properties that satisfy a certain joint relation. We will then solve this relation in the orthonormal and the non-orthonormal case in order to find the most interesting wavelets, and take a closer look at these in examples. Some of these wavelets are heavily used in applications, in particular in image compression.

8.1 Characterization of perfect reconstruction systems of MRA matrices

Let us first state the general theorem for characterizing perfect reconstruction systems of MRA-matrices:

Theorem 8.1. Assume that G and H are MRA-matrices. The following statements are equivalent:

1. (G, H) is a perfect reconstruction system,

2. there exist $\alpha \in \mathbb{R}$ and $d \in \mathbb{Z}$ so that

3.

$$(H_1)_n = (-1)^n \alpha (G_0)_{-n-2d} \tag{8.1}$$

$$(G_1)_n = (-1)^n \alpha^{-1} (H_0)_{-n-2d} \tag{8.2}$$

$$2 = \lambda_{G_0,n}\overline{\lambda_{H_0,n}} + \lambda_{G_0,n+N/2}\overline{\lambda_{H_0,n+N/2}} \tag{8.3}$$

The proof of Theorem 8.1 will wait till section 9.2, since it uses some techniques we still haven't considered. Theorem 8.1 can be used in the following way to construct perfect reconstruction systems: First we find the filters G_0, H_0. These can be found by first finding $\lambda_{G_0,n}$, $\lambda_{H_0,n}$ by solving (8.3), which is a set of linear equations if some of the coefficients are already known, and then using the IDFT. The filters G_1, H_1 are then found by using (8.1) and (8.2).

It is also common to express Theorem 8.1 in terms of the continuous frequency responses of the filters. Equation (8.3) thus takes the following form:

$$2 = \lambda_{G_0}(\omega)\lambda_{H_0}(-\omega) + \lambda_{G_0}(\omega+\pi)\lambda_{H_0}(-(\omega+\pi)). \tag{8.4}$$

This leaves us with an equation with the unknown functions λ_{G_0}, λ_{H_0}. (8.1) and (8.2) can equivalently be expressed in terms of the continuous frequency responses as

$$\begin{aligned}\lambda_{H_1}(\omega) &= \sum_k (H_1)_k e^{-ik\omega} = \alpha \sum_k (-1)^k (G_0)_{-k-2d} e^{-ik\omega}\\ &= \alpha \sum_k (-1)^k (G_0)_k e^{ik\omega+2id\omega} = \alpha e^{2id\omega} \sum_k (G_0)_k e^{ik(\omega+\pi)}\\ &= \alpha e^{2id\omega} \lambda_{G_0}(-(\omega+\pi)).\end{aligned}$$

A similar computation can be done for $\lambda_{G_1}(\omega)$, so that we get the following result:

Theorem 8.2. We have that

$$\lambda_{H_1}(\omega) = \alpha e^{2id\omega} \lambda_{G_0}(-(\omega+\pi)) \tag{8.5}$$

$$\lambda_{G_1}(\omega) = \alpha^{-1} e^{2id\omega} \lambda_{H_0}(-(\omega+\pi)). \tag{8.6}$$

In other words, when G_0 and H_0 are lowpass filters, when we only consider the magnitude of the frequency response, up to a constant H_1 and G_1 are the corresponding highpass filters, as we previously have considered.

We will be particularly concerned with perfect reconstruction systems of MRA-matrices with symmetric filters, for which we can state the following corollary.

Corollary 8.3. (G, H) is a perfect reconstruction system of MRA-matrices with symmetric filters if and only if

$$\lambda_{H_1}(\omega) = \alpha\lambda_{G_0}(\omega+\pi) \tag{8.7}$$

$$\lambda_{G_1}(\omega) = \alpha^{-1}\lambda_{H_0}(\omega+\pi) \tag{8.8}$$

$$2 = \lambda_{G_0}(\omega)\lambda_{H_0}(\omega) + \lambda_{G_0}(\omega+\pi)\lambda_{H_0}(\omega+\pi). \tag{8.9}$$

Proof. Since H_0 is symmetric, $(H_0)_n = (H_0)_{-n}$, and from (8.1) and (8.2) it follows that

$$\begin{aligned}(G_1)_{n-2d} &= (-1)^{n-2d}\alpha^{-1}(H_0)_{-(n-2d)-2d} = (-1)^n\alpha^{-1}(H_0)_{-n}\\ &= (-1)^n\alpha^{-1}(H_0)_n = (-1)^{-n-2d}\alpha^{-1}(H_0)_{-(-n-2d)-2d}\\ &= (G_1)_{-n-2d}.\end{aligned}$$

This shows that G_1 is symmetric about both $2d$, in addition to being symmetric about 0 (by assumption). We must thus have that $d = 0$, so that $(H_1)_n = (-1)^n\alpha(G_0)_{-n}$ and $(G_1)_n = (-1)^n\alpha^{-1}(H_0)_{-n}$. We now get that

$$\begin{aligned}\lambda_{H_1}(\omega) &= \sum_k (H_1)_k e^{-ik\omega} = \alpha\sum_k (-1)^k(G_0)_{-k}e^{-ik\omega}\\ &= \alpha\sum_k e^{-ik\pi}(G_0)_k e^{-ik\omega} = \alpha\sum_k (G_0)_k e^{-ik(\omega+\pi)}\\ &= \alpha\lambda_{G_0}(\omega+\pi),\end{aligned}$$

which proves Equation (8.7). Equation (8.7) follows similarly. Equation (8.9) follows from Equation (8.4) by using that $\lambda_{G_0}(\omega) = \lambda_{G_0}(-\omega)$ due to symmetry, and similarly for G_1, H_0, and H_1. □

When constructing a perfect reconstruction system it may be that we know one of the two pairs (G_0, G_1), (H_0, H_1), and that we would like to construct the other two. This can be achieved if we can find the constants d and α. If the filters are symmetric we saw above that $d = 0$. If (G_0, G_1) is known, in order for a perfect reconstruction system we must have that

$$1 = \sum_n (G_1)_n(H_1)_n = \sum_n (G_1)_n\alpha(-1)^n(G_0)_n = \alpha\sum_n (-1)^n(G_0)_n(G_1)_n,$$

so that $\alpha = \frac{1}{\sum_n (-1)^n(G_0)_n(G_1)_n}$. Similarly, if (H_0, H_1) is known, we must have that

$$1 = \sum_n (G_1)_n(H_1)_n = \sum_n \alpha^{-1}(-1)^n(H_0)_n(H_1)_n = \alpha^{-1}\sum_n (-1)^n(H_0)_n(H_1)_n,$$

so that $\alpha = \sum_n (-1)^n(H_0)_n(H_1)_n$.

Example 8.4. One wavelet remains for which we have not computed the filters and the frequency response, namely the alternative wavelet of piecewise linear functions. In Equation (7.5) we wrote down the first two columns in $P_{\phi_m \leftarrow \mathcal{C}_m}$. This gives us that the filters G_0 ans G_1 are

$$\begin{aligned} G_0 &= \frac{1}{\sqrt{2}}\{1/2, \underline{1}, 1/2\} \\ G_1 &= \frac{1}{\sqrt{2}}\{-1/8, -1/4, \underline{3/4}, -1/4, -1/8\}. \end{aligned} \tag{8.10}$$

Here G_0 was as for the wavelet of piecewise linear functions since we use the same scaling function. G_1 was changed, however. Let us use Theorem 8.1 and the remark above to compute the two remaining filters H_0 and H_1. These filters are also symmetric, since G_0, G_1 were. We get that

$$\alpha = \frac{1}{\sum_n (-1)^n (G_0)_n (G_1)_n} = \frac{1}{\frac{1}{2}\left(-\frac{1}{2}\left(-\frac{1}{4}\right) + 1 \cdot \frac{3}{4} - \frac{1}{2}\left(-\frac{1}{4}\right)\right)} = 2.$$

Theorem 8.1 now gives

$$\begin{aligned} (H_0)_n &= \alpha(-1)^n (G_1)_n = 2(-1)^n (G_1)_n \\ (H_1)_n &= \alpha(-1)^n (G_0)_n = 2(-1)^n (G_0)_n, \end{aligned} \tag{8.11}$$

so that

$$\begin{aligned} H_0 &= \sqrt{2}\{-1/8, 1/4, \underline{3/4}, 1/4, -1/8\} \\ H_1 &= \sqrt{2}\{-1/2, \underline{1}, -1/2\}. \end{aligned} \tag{8.12}$$

We now have that

$$\begin{aligned} \lambda_{G_1}(\omega) &= -1/(8\sqrt{2})e^{2i\omega} - 1/(4\sqrt{2})e^{i\omega} + 3/(4\sqrt{2}) - 1/(4\sqrt{2})e^{-i\omega} - 1/(8\sqrt{2})e^{-2i\omega} \\ &= -\frac{1}{4\sqrt{2}}\cos(2\omega) - \frac{1}{2\sqrt{2}}\cos\omega + \frac{3}{4\sqrt{2}}. \end{aligned}$$

The magnitude of $\lambda_{G_1}(\omega)$ is plotted in Figure 8.1. Clearly, G_1 now has highpass characteristics, while the lowpass characteristic of G_0 has been preserved. The filters G_0, G_1, H_0, H_1 are particularly important in applications: Apart from the scaling factors $1/\sqrt{2}$, $\sqrt{2}$ in front, we see that the filter coefficients are all dyadic fractions, i.e. they are on the form $\beta/2^j$. Arithmetic operations with dyadic fractions can be carried out exactly on a computer, due to representations as binary numbers in computers. These filters are thus important in applications, since they can be used as transformations for lossless coding. The same argument can be made for the Haar wavelet, but this wavelet had one less vanishing moment.

Previously we have plotted the scaling function and the mother wavelet for this wavelet. But how do the dual functions look? And can we obtain an expression for them? It turns out that one rarely can obtain an analytical expression

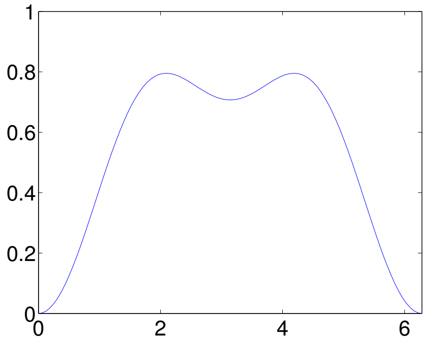
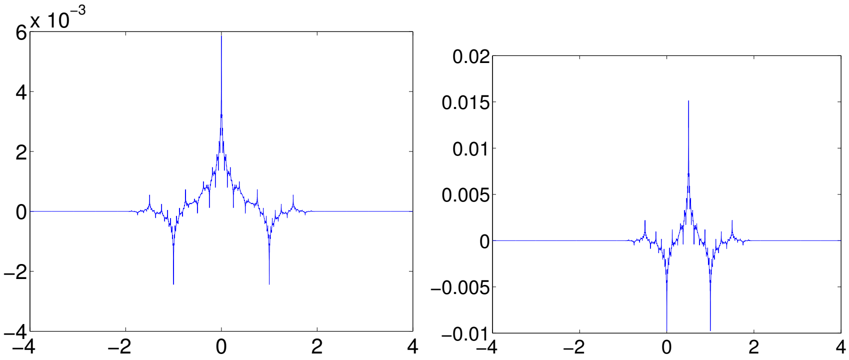


Figure 8.1: The frequency response $\lambda_{G_1}(\omega)$ for the alternative wavelet for piecewise linear functions.

for these functions (as is the case for the scaling function and the mother wavelet itself in most cases). However, it is easy to obtain good approximations for these functions. The reason is that these functions are basis functions in V_0 and W_0, and by performing an IDWT over m levels, we can easily find their coordinates in V_m. If m is high enough, we have previously learnt that these coordinates give a good approximation to the samples of the function. To be precise, the coordinates of $\tilde{\phi}$ in $\tilde{V}_0 \oplus \tilde{W}_0 \oplus \tilde{W}_1 \ldots$ is the vector with 1 first, followed by only zeros. Similarly, the coordinates of $\tilde{\psi}$ in $\tilde{V}_0 \oplus \tilde{W}_0 \oplus \tilde{W}_1 \ldots$ is the vector with N zeros first, then a 1, and then only zeros. The length of the vector is $N2^m$. The coordinates in $\tilde{V}_m$ for $\tilde{\phi}$ and $\tilde{\psi}$ are obtained by performing an m-level IDWT (where the G and H filters have changed placse) on these vectors. In Figure 8.2 we have plotted the coordinates in $\tilde{V}_{10}$, and thus a good approximation to $\tilde{\phi}$ and $\tilde{\psi}$. We see that these functions look very irregular. Also, they are very different from the original scaling function and mother wavelet. We will later argue that this is bad, it would be much better if $\phi \approx \tilde{\phi}$ and $\psi \approx \tilde{\psi}$. The algorithm we employed here in order to plot the dual scaling function and mother wavelet is also called the *cascade algorithm*. ♣

8.2 Construction of useful wavelets: zeros at π in λ_{H_0} and λ_{G_0}

We would like a wavelet to have many vanishing moments, so that they approximate functions well. This in itself is not enough, however, since we also must ensure that the corresponding filters have efficient implementations, such as having few nonzero filter coefficients. The following result shows that this problem can be solved within the world of filters, so that we in fact do not even need to write down the wavelet bases.



(a) ϕ

(b) ψ

Figure 8.2: Dual scaling function and dual mother wavelet for the alternative piecewise linear wavelet.

Theorem 8.5. The number of vanishing moments of ψ, $\tilde{\psi}$ equal the multiplicities of the zeros of the frequency responses $\lambda_{H_0}(\omega)$, $\lambda_{G_0}(\omega)$, respectively, at $\omega = \pi$.

In other words, the flatter the frequency responses $\lambda_{H_0}(\omega)$ and $\lambda_{G_0}(\omega)$ are near high frequencies ($\omega = \pi$), the better the wavelet is for representation. This is analogous to the smoothing filters we constructed previously, where the use of values from Pascals triangle resulted in filters which behaved like the constant function one at low frequencies. The frequency response for the Haar wavelet had just a simple zero at π, so that it cannot represent functions efficiently. The result also proves why we should consider G_0, H_0 as lowpass filters, G_1, H_1 as highpass filters.

Proof. The mathematics behind Theorem 8.5 is quite involved. Theorem 6.19 says that, if $f \in V_r$ is r times differentiable, and $\tilde{\psi}$ has r vanishing moments, then f has a very good approximation from V_0. Denote by s the analog filter corresponding to $\tilde{\psi}$, i.e. $s(f_1) = \int \tilde{\psi}(s) f_1(t-s)ds$. The frequency response of this filter is

$$\lambda_s(f) = \int_{-\infty}^{\infty} \tilde{\psi}(t) e^{-2\pi i f t} dt. \tag{8.13}$$

By differentiating this expression many times w.r.t. f (we can differentiate under the integral sign) we get

$$(\lambda_s)^{(k)}(f) = \int (-2\pi i t)^k \tilde{\psi}(t) e^{-it\omega} dt. \tag{8.14}$$

Evaluating this at 0 gives

$$(\lambda_s)^{(k)}(0) = \int (-2\pi i t)^k \tilde{\psi}(t) dt = 0 \tag{8.15}$$

when $k \leq r$, where we have used that $\tilde{\psi}$ has r vanishing moments. Let us summarize this as follows:

Lemma 8.6. $\tilde{\psi}$ has r vanishing moments if and only if $\lambda_s(f))$ has a zero of multiplicity r at 0, i.e. the first $r+1$ terms in its Taylor series around 0 vanish.

We now can write

$$\begin{aligned}
\lambda_s(f) &= \int_{-\infty}^{\infty} \tilde{\psi}(t) e^{-2\pi i f t} dt \\
&= \int_{-\infty}^{\infty} \left(\sum_n (H_1)_n \sqrt{2} \tilde{\phi}(2t-n) \right) e^{-2\pi i f t} dt \\
&= \sqrt{2} \sum_n \int_{-\infty}^{\infty} (H_1)_n \tilde{\phi}(2t-n) e^{-2\pi i f t} dt \\
&= \frac{1}{\sqrt{2}} \sum_n \int_{-\infty}^{\infty} (H_1)_n \tilde{\phi}(t) e^{-2\pi i f (n+t)/2} dt \\
&= \frac{1}{\sqrt{2}} \sum_n \int_{-\infty}^{\infty} (H_1)_n e^{-2\pi i f n/2} \tilde{\phi}(t) e^{-2\pi i (f/2) t)} dt \\
&= \frac{1}{\sqrt{2}} \left(\sum_n (H_1)_n e^{-2\pi i f n/2} \right) \int_{-\infty}^{\infty} \tilde{\phi}(t) e^{-2\pi i (f/2) t)} dt \\
&= \frac{\lambda_{H_1}(2\pi f/2)}{\sqrt{2}} \lambda_{s_2}(f/2),
\end{aligned}$$

where s_2 is the analog filter corresponding to $\tilde{\phi}$. In a similar way we get that $\lambda_{s_2}(f) = \frac{\lambda_{H_0}(2\pi f/2)}{\sqrt{2}} \lambda_{s_2}(f/2)$. This procedure can also be continued, so that we get

$$\begin{aligned}
\lambda_s(f) &= \frac{\lambda_{H_1}(2\pi f/2)}{\sqrt{2}} \lambda_{s_2}(f/2) = \frac{\lambda_{H_1}(2\pi f/2)}{\sqrt{2}} \frac{\lambda_{H_0}(2\pi f/4)}{\sqrt{2}} \lambda_{s_2}(f/4) \\
&\vdots \\
&= \frac{\lambda_{H_1}(2\pi f/2)}{\sqrt{2}} \prod_{s=2}^{k} \frac{\lambda_{H_0}(2\pi f/2^s)}{\sqrt{2}} \lambda_{s_2}(f/2^k).
\end{aligned} \tag{8.16}$$

$\prod_{s=2}^{k} \frac{\lambda_{H_0}(2\pi f/2^s)}{\sqrt{2}}$ is nonzero at 0, since by our construction H_0 is a lowpass filter. Moreover, $\lambda_{s_2}(f/2^k)$ is nonzero close to 0 by our assumptions. The only possibility left is that $\lambda_{H_1}(2\pi f/2)$ holds the zeros of λ_s at 0, so that $\lambda_{H_1}(\omega)$

does the same. But this also means that $\lambda_{G_0}(\omega)$ has the same number of zeros at π, due to Theorem 8.1. All this means that the number of zeros for $\lambda_s(f)$ at $f = 0$ equals the number of zeros for $\lambda_{G_0}(\omega)$ at $\omega = \pi$. From Lemma 8.6 it follows that the number of zeros for $\lambda_{G_0}(\omega)$ at $\omega = \pi$ equals the number of vanishing moments of $\tilde{\psi}$. This proves one way of the result. The other way follows by substituting with ϕ, ψ instead. □

We will use this result in the next chapter to construct interesting wavelets.

8.3 Characterization of wavelets w.r.t. number of vanishing moments

We have seen that wavelets are particularly suitable for approximation of functions when the mother wavelet or the dual mother wavelet have vanishing moments. The more vanishing moments they have, the more attractive they are. In this section we will attempt to characterize wavelets which have a given number of vanishing moments. In particular we will characterize the simplest such, those where the filters have few filters coefficients.

There are two particular cases we will look at. First we will consider the case when all filters are symmetric. Then we will look at the case of orthonormal wavelets. It turns out that these two cases are mutually disjoint (except for trivial examples), but that there is a common result which can be used to characterize the solutions to both problems. We state the result in terms of the multiplicities of the zeros of λ_{H_0}, λ_{G_0} at π, which we proved are the same as the number of vanishing moments. The main result when the filters are symmetric looks as follows.

Theorem 8.7. Assume that (H, G) is a perfect reconstruction system of MRA-matrices where all filters are symmetric, and assume also that λ_{H_0} has a zero of multiplicity N_1 at π, and that λ_{G_0} has a zero of multiplicity N_2 at π. Then N_1 and N_2 are even, and

$$\lambda_{H_0}(\omega) = \left(\frac{1}{2}(1+\cos\omega)\right)^{N_1/2} Q_1\left(\frac{1}{2}(1-\cos\omega)\right) \tag{8.17}$$

$$\lambda_{G_0}(\omega) = \left(\frac{1}{2}(1+\cos\omega)\right)^{N_2/2} Q_2\left(\frac{1}{2}(1-\cos\omega)\right), \tag{8.18}$$

where $Q = Q_1 Q_2$ satisfies the equation

$$u^{(N_1+N_2)/2} Q(1-u) + (1-u)^{(N_1+N_2)/2} Q(u) = 2. \tag{8.19}$$

Proof. Since the filters are symmetric, $\lambda_{H_0}(\omega) = \lambda_{H_0}(-\omega)$ and $\lambda_{G_0}(\omega) = \lambda_{G_0}(-\omega)$. Since $e^{in\omega} + e^{-in\omega} = 2\cos(n\omega)$, and since $\cos(n\omega)$ is the real part of $(\cos\omega +$

$i\sin\omega)^n$, which is a polynomial in $\cos^k\omega\sin^l\omega$ with l even, and since $\sin^2\omega = 1-\cos^2\omega$, λ_{H_0} and λ_{G_0} can both be written on the form $P(\cos\omega)$, with P a real polynomial.

Note that a zero at π in $\lambda_{H_0}, \lambda_{G_0}$ corresponds to a factor of the form $1+e^{-i\omega}$, so that we can write

$$\lambda_{H_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^{N_1} f(e^{i\omega}) = e^{-iN_1\omega/2}\cos^{N_1}(\omega/2)f(e^{i\omega}),$$

where f is a polynomial. In order for this to be real, we must have that $f(e^{i\omega}) = e^{iN_1\omega/2}g(e^{i\omega})$ where g is real-valued, and then we can write $g(e^{i\omega})$ as a real polynomial in $\cos\omega$. This means that $\lambda_{H_0}(\omega) = \cos^{N_1}(\omega/2)P_1(\cos\omega)$, and similarly for $\lambda_{G_0}(\omega)$. Clearly this can be a polynomial in $e^{i\omega}$ only if N_1 is even. Both N_1 and N_2 must then be even, and we can write

$$\begin{aligned}\lambda_{H_0}(\omega) &= \cos^{N_1}(\omega/2)P_1(\cos\omega) = (\cos^2(\omega/2))^{N_1/2}P_1(1-2\sin^2(\omega/2))\\ &= (\cos^2(\omega/2))^{N_1/2}Q_1(\sin^2(\omega/2)),\end{aligned}$$

where we have used that $\cos\omega = 1-2\sin^2(\omega/2)$, and defined Q_1 by the relation $Q_1(x) = P_1(1-2x)$. Similarly we can write $\lambda_{G_0}(\omega) = (\cos^2(\omega/2))^{N_2/2}Q_2(\sin^2(\omega/2))$ for another polynomial Q_2. Using the identities

$$\cos^2\frac{\omega}{2} = \frac{1}{2}(1+\cos\omega) \qquad\qquad \sin^2\frac{\omega}{2} = \frac{1}{2}(1-\cos\omega),$$

we see that λ_{H_0} and λ_{G_0} satisfy (8.17) and (8.18). With $Q = Q_1Q_2$, (8.3) can now be rewritten as

$$\begin{aligned}2 &= \lambda_{G_0}(\omega)\lambda_{H_0}(-\omega) + \lambda_{G_0}(\omega+\pi)\lambda_{H_0}(-(\omega+\pi))\\ &= \left(\cos^2\frac{\omega}{2}\right)^{(N_1+N_2)/2} Q(\sin^2(\omega/2)) + \left(\cos^2\frac{\omega+\pi}{2}\right)^{(N_1+N_2)/2} Q(\sin^2((\omega+\pi)/2))\\ &= (\cos^2(\omega/2))^{(N_1+N_2)/2}Q(\sin^2(\omega/2)) + (\sin^2(\omega/2))^{(N_1+N_2)/2}Q(\cos^2(\omega/2))\\ &= (\cos^2(\omega/2))^{(N_1+N_2)/2}Q(1-\cos^2(\omega/2)) + (1-\cos^2(\omega/2))^{(N_1+N_2)/2}Q(\cos^2(\omega/2))\end{aligned}$$

Setting $u = \cos^2(\omega/2)$ we see that Q must fulfill the equation

$$u^{(N_1+N_2)/2}Q(1-u) + (1-u)^{(N_1+N_2)/2}Q(u) = 2,$$

which is (8.19). This completes the proof. □

While this result characterizes all wavelets with a given number of vanishing moments, it does not say which of these have fewest filter coefficients. The polynomial Q decides the length of the filters H_0, G_0, however, so that what we need to do is to find the polynomial Q of smallest degree. In this direction, note first that the polynomials $u^{N_1+N_2}$ and $(1-u)^{N_1+N_2}$ have no zeros in common. Bezouts theorem, proved in Appendix 8.4, states that $u^{(N_1+N_2)/2}q_1(u) + (1-u)^{(N_1+N_2)/2}q_2(u) = 1$ has unique solutions with $\deg(q_1), \deg(q_2) < (N_1+N_2)/2$, when q_1 and q_2 have no zeros in common. To find these solutions, substituting $1-u$ for u gives the following equations:

$$\begin{aligned} u^{(N_1+N_2)/2}q_1(u) + (1-u)^{(N_1+N_2)/2}q_2(u) &= 1 \\ u^{(N_1+N_2)/2}q_2(1-u) + (1-u)^{(N_1+N_2)/2}q_1(1-u) &= 1, \end{aligned}$$

and uniqueness in Bezouts theorem gives that $q_1(u) = q_2(1-u)$, and $q_2(u) = q_1(1-u)$. $u^{(N_1+N_2)/2}q_1(u) + (1-u)^{(N_1+N_2)/2}q_2(u) = 1$ now gives

$$\begin{aligned} q_2(u) &= (1-u)^{-(N_1+N_2)/2}(1-u^{(N_1+N_2)/2}q_1(u)) \\ &= (1-u)^{-(N_1+N_2)/2}(1-u^{(N_1+N_2)/2}q_2(1-u)) \\ &= \left(\sum_{k=0}^{(N_1+N_2)/2-1} \binom{(N_1+N_2)/2+k-1}{k} u^k + O(u^{(N_1+N_2)/2}) \right) (1-u^{(N_1+N_2)/2}q_2(1-u)) \\ &= \sum_{k=0}^{(N_1+N_2)/2-1} \binom{(N_1+N_2)/2+k-1}{k} u^k + O(u^{(N_1+N_2)/2}), \end{aligned}$$

where we have used the first $(N_1+N_2)/2$ terms in the Taylor series expansion of $(1-u)^{-(N_1+N_2)/2}$ around 0. Since q_2 is a polynomial of degree $(N_1+N_2)/2-1$, we must have that

$$Q(u) = 2q_2(u) = 2 \sum_{k=0}^{(N_1+N_2)/2-1} \binom{(N_1+N_2)/2+k-1}{k} u^k. \tag{8.20}$$

The first such Q are as follows:

$$\begin{aligned} Q^{(0)}(u) &= 2 \\ Q^{(1)}(u) &= 2+4u \\ Q^{(2)}(u) &= 2+6u+12u^2 \\ Q^{(3)}(u) &= 2+8u+20u^2+40u^3, \end{aligned}$$

for which we compute

$$\begin{aligned} Q^{(0)}\left(\frac{1}{2}(1-\cos\omega)\right) &= 2 \\ Q^{(1)}\left(\frac{1}{2}(1-\cos\omega)\right) &= -e^{-i\omega} + 4 - e^{i\omega} \\ Q^{(2)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{3}{4}e^{-2i\omega} - \frac{9}{2}e^{-i\omega} + \frac{19}{2} - \frac{9}{2}e^{i\omega} + \frac{3}{4}e^{2i\omega} \\ Q^{(3)}\left(\frac{1}{2}(1-\cos\omega)\right) &= -\frac{5}{8}e^{-3i\omega} + 5e^{-2i\omega} - \frac{131}{8}e^{-i\omega} + 26 - \frac{131}{8}e^{i\omega} + 5e^{2i\omega} - \frac{5}{8}e^{3i\omega}, \end{aligned}$$

Thus in order to construct wavelets where $\lambda_{H_0}, \lambda_{G_0}$ have as many zeros at π as possible, and where there are as few filter coefficients as possible, we need to compute the polynomials above, factorize them, and distribute the factors among λ_{H_0} and λ_{G_0}. Since we need real factorizations, we must in any case pair complex roots. If we do this we obtain the factorizations

$$\begin{aligned}
Q^{(0)}\left(\frac{1}{2}(1-\cos\omega)\right) &= 2 \\
Q^{(1)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{1}{3.7321}(e^{i\omega}-3.7321)(e^{-i\omega}-3.7321) \\
Q^{(2)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{3}{4}\frac{1}{9.4438}(e^{2i\omega}-5.4255e^{i\omega}+9.4438) \\
&\times(e^{-2i\omega}-5.4255e^{-i\omega}+9.4438) \\
Q^{(3)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}(e^{i\omega}-3.0407)(e^{2i\omega}-4.0623e^{i\omega}+7.1495) \\
&\times(e^{-i\omega}-3.0407)(e^{-2i\omega}-4.0623e^{-i\omega}+7.1495), \quad (8.21)
\end{aligned}$$

One possibility is to let one of these frequency responses absorb all the factors, another possibility is to split the factors as evenly as possible across the two. That a frequency response absorbs more factors means that it gets more filter coefficients. In the following examples, bith factor distribution staregies will be encountered.

Now we turn to the other case where the filters are equal, i.e. where the MRA-matrix is orthogonal. We have the following result:

Theorem 8.8. We get an orthogonal MRA-matrix (i.e. $G_0 = H_0$) with a zero of multiplicity N at π if we choose

$$\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{i\omega}), \tag{8.22}$$

with f satisfying $f(e^{i\omega})f(e^{-i\omega}) = Q\left(\frac{1}{2}(1-\cos\omega)\right)$, and with Q satisfying Equation (8.19).

Proof. N vanishing moments means that we can write

$$\lambda_{G_0}(\omega) = \left(\frac{1+e^{-i\omega}}{2}\right)^N f(e^{i\omega}) = (\cos(\omega/2))^N e^{-iN\omega/2} f(e^{i\omega}),$$

where f is a polynomial. The condition for perfect reconstruction now says that

$$\begin{aligned}
2 &= \lambda_{G_0}(\omega)\lambda_{G_0}(-\omega) + \lambda_{G_0}(\omega+\pi)\lambda_{G_0}(-(\omega+\pi)) \\
&= (\cos^2(\omega/2))^N f(e^{i\omega})f(e^{-i\omega}) + (\sin^2(\omega/2))^N f(e^{i(\omega+\pi)})f(e^{-i(\omega+\pi)}).
\end{aligned}$$

Now, the function $f(e^{i\omega})f(e^{-i\omega})$ is symmetric around 0, so that it can be written on the form $P(\cos\omega)$ with P a polynomial, so that

$$(\cos^2(\omega/2))^N P(\cos\omega) + (\sin^2(\omega/2))^N P(\cos(\omega+\pi)) = 2.$$

If we as in the proof of Theorem 8.7 define Q by $Q(x) = P(1-2x)$, we can write this as

$$(\cos^2(\omega/2))^N Q(\sin^2(\omega/2)) + (\sin^2(\omega/2))^N Q(\cos^2(\omega/2)) = 2,$$

which again gives Equation (8.19) for finding Q. What we thus need to do is to compute the polynomial $Q\left(\frac{1}{2}(1-\cos\omega)\right)$ as before, and consider the different factorizations of this on the form $f(e^{i\omega})f(e^{-i\omega})$. Since this polynomial is symmetric, a is a root if and only $1/a$ is, and if and only if $\bar{a}$ is. If the real roots are $b_1, \ldots, b_m, 1/b_1, \ldots, 1/b_m$, and the complex roots are $a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}$ and $1/a_1, \ldots, 1/a_n, \overline{1/a_1}, \ldots, \overline{1/a_n}$, we can write

$$\begin{aligned}
&Q\left(\frac{1}{2}(1-\cos\omega)\right)\\
&= K(e^{-i\omega} - b_1)\ldots(e^{-i\omega} - b_m)\\
&\times (e^{-i\omega} - a_1)(e^{-i\omega} - \overline{a_1})(e^{-i\omega} - a_2)(e^{-i\omega} - \overline{a_2})\cdots(e^{-i\omega} - a_n)(e^{-i\omega} - \overline{a_n})\\
&\times (e^{i\omega} - b_1)\ldots(e^{i\omega} - b_m)\\
&\times (e^{i\omega} - a_1)(e^{i\omega} - \overline{a_1})(e^{i\omega} - a_2)(e^{i\omega} - \overline{a_2})\cdots(e^{i\omega} - a_n)(e^{i\omega} - \overline{a_n})
\end{aligned}$$

where K is a constant. We now can define

$$\begin{aligned}
f(e^{i\omega}) =& \sqrt{K}(e^{-i\omega} - b_1)\ldots(e^{-i\omega} - b_m)\\
&\times (e^{-i\omega} - a_1)(e^{-i\omega} - \overline{a_1})(e^{-i\omega} - a_2)(e^{-i\omega} - \overline{a_2})\cdots(e^{-i\omega} - a_n)(e^{-i\omega} - \overline{a_n})
\end{aligned}$$

in order to obtain a factorization $Q\left(\frac{1}{2}(1-\cos\omega)\right) = f(e^{i\omega})f(e^{-i\omega})$. This concludes the proof. □

In the previous proof we note that the polynomial f is not unique - we could choose the roots to pair in many different ways. The new algorithm is thus as follows:

1. As before, write $Q\left(\frac{1}{2}(1-\cos\omega)\right)$ as a polynomial in $e^{i\omega}$, and find the roots.
2. Split the roots into the two classes $\{b_1, \ldots, b_m, a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}$ and $\{1/b_1, \ldots, 1/b_m, 1/a_1, \ldots, 1/a_n, \overline{1/a_1}, \ldots, \overline{1/a_n}\}$, and form the polynomial f.
3. Compute $\left(\frac{1+e^{i\omega}}{2}\right)^N f(e^{i\omega})$.

Clearly the filters obtained with this strategy are not symmetric since f is not symmetric. In Section 8.7 we will take a closer look at the wavelets constructed in this way.

Exercises for Section 8.3

1. Calculate the number of vanishing moment for the ψ as defined in Lemma **??**.

8.4 *The proof of Bezouts theorem

Theorem 8.9. If p_1 and p_2 are two polynomials, of degrees n_1 and n_2 respectively, with no common zeros, then there exist unique polynomials q_1, q_2, of degree less than n_2, n_1, respectively, so that

$$p_1(x)q_1(x) + p_2(x)q_2(x) = 1. \tag{8.23}$$

Proof. We first establish the existence of q_1, q_2 satisfying (8.23). Denote by $\deg(P)$ the degree of the polynomial P. Renumber the polynomials if necessary, so that $n_1 \geq n_2$. By polynomial division, we can now write

$$p_1(x) = a_2(x)p_2(x) + b_2(x),$$

where $\deg(a_2) = \deg(p_1) - \deg(p_2)$, $\deg(b_2) < deg(p_2)$. Similarly, we can write

$$p_2(x) = a_3(x)b_2(x) + b_3(x),$$

where $\deg(a_3) = \deg(p_2) - \deg(b_2)$, $\deg(b_3) < \deg(b_2)$. We can repeat this procedure, so that we obtain a sequence of polynomials $a_n(x), b_n(x)$ so that

$$b_{n-1}(x) = a_{n+1}(x)b_n(x) + b_{n+1}(x), \tag{8.24}$$

where $\deg a_{n+1} = \deg(b_{n-1}) - \deg(b_n)$, $deg(b_{n+1} < \deg(b_n)$. Since $\deg(b_n)$ is strictly decreasing, we must have that $b_{N+1} = 0$ and $b_N \neq 0$ for some N, i.e. $b_{N-1}(x) = a_{N+1}(x)b_N(x)$. Since $b_{N-2} = a_N b_{N-1} + b_N$, it follows that b_{N-2} can be divided by b_N, and by induction that all b_n can be divided by b_N, in particlar p_1 and p_2 can be divided by b_N. Since p_1 and p_2 have no common zeros, b_N must be a nonzero constant.

Using (8.24), we can write recursively

$$\begin{aligned} b_N &= b_{N-2} - a_N b_{N-1} \\ &= b_{N-2} - a_N(b_{N-3} - a_{N-1}b_{N-2}) \\ &= (1 + a_N a_{N-1})b_{N-2} - a_N b_{N-3}. \end{aligned}$$

By induction we can write

$$b_N = a^{(1)}_{N,k} b_{N-k} + a^{(2)}_{N,k} b_{N-k-1}.$$

We see that the leading order term for $a^{(1)}_{N,k}$ is $a_N \cdots a_{N-k+1}$, which has degree

$$(\deg(b_{N-2}) - \deg(b_{N-1}) + \cdots + (\deg(b_{N-k-1}) - \deg(b_{N-k}) = \deg(b_{N-k-1}) - \deg(b_{N-1}),$$

while the leading order term for $a^{(2)}_{N,k}$ is $a_N \cdots a_{N-k+2}$, which similarly has order $\deg(b_{N-k}) - \deg(b_{N-1})$. For $k = N-1$ we find

$$b_N = a^{(1)}_{N,N-1} b_1 + a^{(2)}_{N,N-1} b_0 = a^{(1)}_{N,N-1} p_2 + a^{(2)}_{N,N-1} p_1, \tag{8.25}$$

with $\deg(a^{(1)}_{N,N-1}) = \deg(p_1) - \deg(b_{N-1}) < \deg(p_1)$ (since by construction $\deg(b_{N-1}) > 0$), and $\deg(a^{(2)}_{N,N-1}) = \deg(p_2) - \deg(b_{N-1}) < \deg(p_2)$. From (8.25) it follows that $q_1 = a^{(2)}_{N,N-1}/b_N$ and $q_2 a^{(1)}_{N,N-1}/b_N$ satisfies (8.23), and that they satisfy the required degree constraints.

Now we turn to uniquness of solutions q_1, q_2. Assume that r_1, r_2 are two other solutions to (8.23). Then

$$p_1(q_1 - r_1) + p_2(q_2 - r_2) = 0.$$

Since p_1 and p_2 have no zeros in common this means that every zero of p_2 is a zero of $q_1 - r_1$, with at least the same multiplicity. If $q_1 \neq r_1$, this means that $\deg(q_1 - r_1) \geq \deg(p_2)$, which is impossible since $\deg(q_1) < \deg(p_2)$, $\deg(r_1) < \deg(p_2)$. Hence $q_1 = r_1$. Similarly $q_2 = r_2$, establishing uniqueness. □

8.5 A design strategy suitable for lossless compression

We choose $Q_1 = Q$, $Q_2 = 1$. In this case there is no need to find factors in Q. The filters in the filter factorization take the form

$$\begin{aligned}\lambda_{H_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^{N_1/2} Q\left(\frac{1}{2}(1-\cos\omega)\right)\\ \lambda_{G_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^{N_2/2}.\end{aligned}$$

Since Q has degree $\frac{N_1+N_2}{2} - 1$, λ_{H_0} has degree $N_1 + N_1 + N_2 - 2 = 2N_1 + N_2 - 2$, and λ_{G_0} has degree N_2. These are both even numbers, so that the filters have odd length. The names of these filters are indexed by the filter lengths, and are called *Spline wavelets*, since, as we now now will show, the scaling function for this design strategy is the B-spline of order N_2: we have that

$$\lambda_{G_0}(\omega) = \frac{1}{2^{N_2/2}}(1+\cos\omega)^{N_2/2} = \cos(\omega/2)^{N_2}.$$

Letting s be the analog filter with convolution kernel ϕ we can as in Equation (8.16) write

$$\begin{aligned}
\lambda_s(f) &= \lambda_s(f/2^k)\prod_{i=1}^{k}\frac{\lambda_{G_0}(2\pi f/2^i)}{2} \\
&= \lambda_s(f/2^k)\prod_{i=1}^{k}\frac{\cos^{N_2}(\pi f/2^i)}{2} \\
&= \lambda_s(f/2^k)\prod_{i=1}^{k}\left(\frac{\sin(2\pi f/2^i)}{2\sin(\pi f/2^i)}\right)^{N_2} \\
&= \lambda_s(f/2^k)\left(\frac{\sin(\pi f)}{2^k\sin\pi f/2^k}\right)^{N_2},
\end{aligned}$$

where we have used the identity $\cos\omega = \frac{\sin(2\omega)}{2\sin\omega}$. If we here let $k \to \infty$, and use the identity $\lim_{f\to 0}\frac{\sin f}{f} = 1$, we get that

$$\lambda_s(f) = \lambda_s(0)\left(\frac{\sin(\pi f)}{\pi f}\right)^{N_2}.$$

On the other hand, the frequency response of $\chi_{[-1/2,1/2)}(t)$

$$\begin{aligned}
&= \int_{-1/2}^{1/2} e^{-2\pi i f t}dt = \left[\frac{1}{-2\pi i f}e^{-2\pi i f t}\right]_{-1/2}^{1/2} \\
&= \frac{1}{-2\pi i f}(e^{-\pi i f} - e^{\pi i f}) = \frac{1}{-2\pi i f}2i\sin(-\pi f) \\
&= \frac{\sin(\pi f)}{\pi f}.
\end{aligned}$$

Due to this $\left(\frac{\sin(\pi f)}{\pi f}\right)^{N_2}$ is the frequency response of $*_{k=1}^{N_2}\chi_{[-1/2,1/2)}(t)$. By the uniqueness of the frequency response we have that $\phi(t) = \hat{\phi}(0)*_{k=1}^{N_2}\chi_{[-1/2,1/2)}(t)$. In Exercise 2 you will be asked to show that this scaling function gives rise to the multiresolution analysis of functions which are piecewise polynomials which are differentiable at the borders, also called *splines*. This explains why this type of wavelet is called a spline wavelet. To be more precise, the resolution spaces are as follows

Definition 8.10 (Resolution spaces of piecewise polynomials). We define V_m as the subspace of functions which are $r-1$ times continuously differentiable and equal to a polynomial of degree r on any interval of the form $[n2^{-m}, (n+1)2^{-m}]$.

Note that the piecewise linear wavelet can be considered as the first Spline wavelet. This is further considered in the following example.

Example 8.11. For the case of $N_1 = N_2 = 2$ when the first design strategy is used, equations (8.17) and (8.18) take the form

$$\begin{aligned}
\lambda_{G_0}(\omega) &= \frac{1}{2}(1+\cos\omega) = \frac{1}{4}e^{i\omega} + \frac{1}{2} + \frac{1}{4}e^{-i\omega} \\
\lambda_{H_0}(\omega) &= \frac{1}{2}(1+\cos\omega)Q^{(1)}\left(\frac{1}{2}(1-\cos\omega)\right) \\
&= \frac{1}{4}(2+e^{i\omega}+e^{-i\omega})(4-e^{i\omega}-e^{-i\omega}) \\
&= -\frac{1}{4}e^{2i\omega} + \frac{1}{2}e^{i\omega} + \frac{3}{2} + \frac{1}{2}e^{-i\omega} - \frac{1}{4}e^{-2i\omega}.
\end{aligned}$$

The filters G_0, H_0 are thus

$$\begin{aligned}
G_0 &= \{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\} \\
H_0 &= \{-\frac{1}{4}, \frac{1}{2}, \underline{\frac{3}{2}}, \frac{1}{2}, -\frac{1}{4}\}
\end{aligned}$$

The length of the filters are 3 and 5 in this case, so that this wavelet is called the *Spline 5/3 wavelet*. Up to a constant, the filters are seen to be the same as those of the alternative piecewise linear wavelet, see Example 8.4. Now, how do we find the filters (G_1, H_1)? Previously we saw how to find the constant α in Theorem 8.1 when we knew one of the two pairs (G_0, G_1), (H_0, H_1). This was the last part of information we needed in order to construct the other two filters. Here we know (G_0, H_0) instead. In this case it is even easier to find (G_1, H_1) since we can set $\alpha = 1$. This means that (G_1, H_1) can be obtained simply by adding alternating signs to (G_0, H_0), i.e. they are the corresponding highpass filters. We thus can set

$$\begin{aligned}
G_1 &= \{-\frac{1}{4}, -\frac{1}{2}, \underline{\frac{3}{2}}, -\frac{1}{2}, -\frac{1}{4}\} \\
H_1 &= \{-\frac{1}{4}, \underline{\frac{1}{2}}, -\frac{1}{4}\}.
\end{aligned}$$

We have now found all the filters. It is clear that the scaling function and mother wavelet of this wavelet equals those of the alternative piecewise linear wavelet, up to a constant. ♣

The coefficients for the Spline wavelets are always dyadic fractions, and are therefore suitable for lossless compression, as they can be computed using low precision arithmetic and bitshift operations. Due to this, this wavelet is used for lossless compression with JPEG2000.

Exercises for Section 8.5

1. Write code which computes the coefficents in $\left(\frac{1}{2}(1 \pm \cos\omega)\right)^k$ as a polynomial in $e^{i\omega}$. Since $1 \pm \cos\omega = 1 \pm \frac{1}{2}e^{i\omega} \pm \frac{1}{2}e^{-i\omega}$ you should think of it as the sequence $\{\pm\frac{1}{2}, \underline{1}, \pm\frac{1}{2}\}$, and then use the `conv`-function to compute the coefficients in the k'th power.

2. Show that $B_r(t) = *_{k=1}^r \chi_{[-1/2,1/2)}(t)$ is $r-2$ times differentiable, and equals a polynomial of degree $r-1$ on subintervals of the form $[n, n+1]$. Explain why these functions can be used as basis for the spaces V_j of functions which are piecewise polynomials of degree $r-1$ on intervals of the form $[n2^{-m}, (n+1)2^{-m}]$, and $r-2$ times differentiable. B_r is also called the B-spline of order r.

8.6 A design strategy suitable for lossy compression

The factors of Q are split evenly among Q_1 and Q_2. In this case we need to factorize Q into a product of real polynomials. This can be done by finding all roots, and pairing the complex conjugate roots into real second degree polynomials (if Q is real, its roots come in conjugate pairs), and then distribute these as evenly as possible among Q_1 and Q_2. These filters are called the CDF-wavelets, after Cohen, Daubechies, and Feauveau, who discovered them.

Example 8.12. We choose $N_1 = N_2 = 4$. In Equation (8.21) we pair inverse terms to obtain

$$\begin{aligned}
Q^{(3)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}(e^{i\omega} - 3.0407)(e^{-i\omega} - 3.0407) \\
&\quad \times (e^{2i\omega} - 4.0623e^{i\omega} + 7.1495)(e^{-2i\omega} - 4.0623e^{-i\omega} + 7.1495) \\
&= \frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}(-3.0407e^{i\omega} + 10.2456 - 3.0407e^{-i\omega}) \\
&\quad \times (7.1495e^{2i\omega} - 33.1053e^{i\omega} + 68.6168 - 33.1053e^{-i\omega} + 7.1495e^{-2i\omega}).
\end{aligned}$$

We can write this as Q_1Q_2 with $Q_1(0) = Q_2(0)$ when

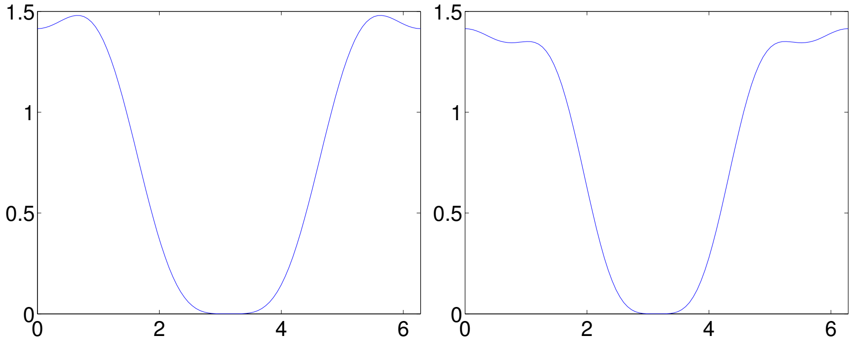
$$\begin{aligned}
Q_1(\omega) &= -1.0326e^{i\omega} + 3.4795 - 1.0326e^{-i\omega} \\
Q_2(\omega) &= 0.6053e^{2i\omega} - 2.8026e^{i\omega} + 5.8089 - 2.8026e^{-i\omega} + 0.6053e^{-2i\omega},
\end{aligned}$$

from which we obtain

$$\begin{aligned}
\lambda_{G_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^2 Q_1(\omega) \\
&= -0.0645e^{3i\omega} - 0.0407e^{2i\omega} + 0.4181e^{i\omega} + 0.7885 \\
&\quad + 0.4181e^{-i\omega} - 0.0407e^{-2i\omega} - 0.0645e^{-3i\omega} \\
\lambda_{H_0}(\omega) &= \left(\frac{1}{2}(1+\cos\omega)\right)^2 40Q_2(\omega) \\
&= 0.0378e^{4i\omega} - 0.0238e^{3i\omega} - 0.1106e^{2i\omega} + 0.3774e^{i\omega} + 0.8527 \\
&\quad + 0.3774e^{-i\omega} - 0.1106e^{-2i\omega} - 0.0238e^{-3i\omega} + 0.0378e^{-4i\omega}.
\end{aligned}$$

The filters G_0, H_0 are thus

$$\begin{aligned}
G_0 &= \{-0.0645, -0.0407, 0.4181, \underline{0.7885}, 0.4181, -0.0407, -0.0645\} \\
H_0 &= \{0.0378, -0.0238, -0.1106, 0.3774, \underline{0.8527}, 0.3774, -0.1106, -0.0238, 0.0378\}.
\end{aligned}$$



(a) $\lambda_{G_0}(\omega)$

(b) $\lambda_{H_0}(\omega)$

Figure 8.3: The frequency responses for the filters of Example 8.12.

The corresponding frequency responses are plotted in Figure 8.3. It is seen that both filters are lowpass filters also here, and that the are closer to an ideal bandpass filter. Here, the frequency response acts even more like the constant zero function close to π, proving that our construction has worked. We also get

$$G_1 = \{0.0378, 0.0238, -0.1106, -0.3774, \underline{0.8527}, -0.3774, -0.1106, 0.0238, 0.0378\}$$
$$H_1 = \{0.0645, -0.0407, -0.4181, \underline{0.7885}, -0.4181, -0.0407, 0.0645\}.$$

The length of the filters are 9 and 7 in this case, so that this wavelet is called the *CDF 9/7 wavelet.* This wavelet is for instance used for lossy compression with JPEG2000 since it gives a good tradeoff between complexity and compression.

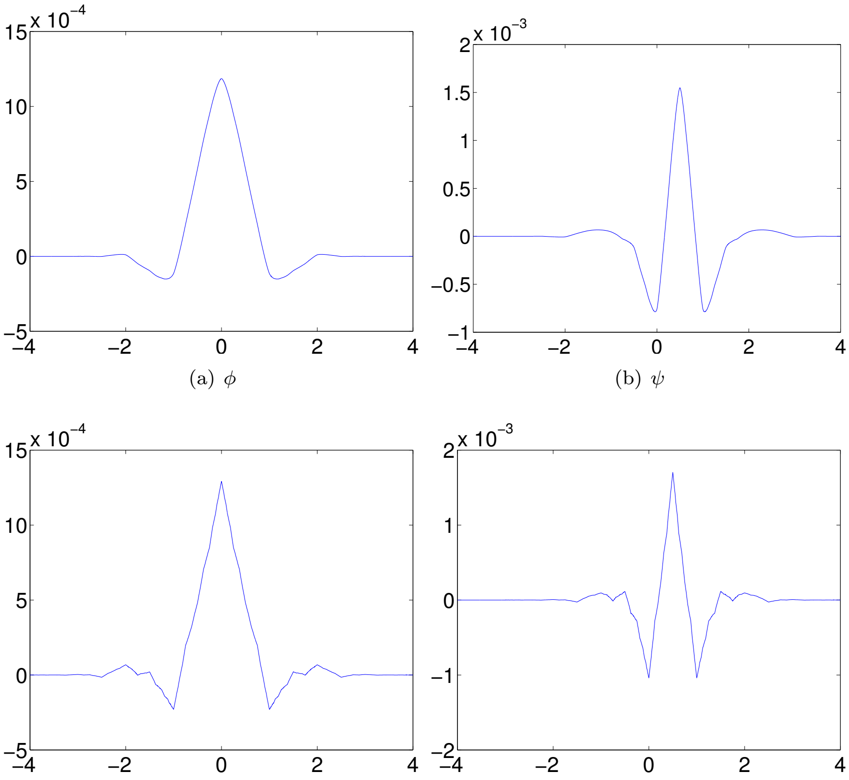
In Example 8.4 we saw that we had analytical expressions for the scaling functions and the mother wavelet, but that we could not obtain this for the dual functions. For the CDF 9/7 wavelet it turns out that none of the four functions have analytical expressions. Let us therefore use the strategy from Example 8.4 to plot these functions. The results are shown in Figure 8.4. Again they have irregular shapes, but now at least the functions and dual functions more resemple each other. ♣

In the above example there was a unique way of factoring Q into a product of real polynomials. For higher degree polynomials there is no unique way to form to distribute the factors, and we will not go into what strategy can be used for this. In general, the steps we must go through are as follows:

1. Compute the polynomial Q, and find its roots.

2. Pair complex conjugate roots into real second degree polynomials, and form polynomials Q_1, Q_2.

3. Compute the coefficients in (8.17) and (8.18).



(a) ϕ

(b) ψ

(c) $\tilde{\phi}$

(d) $\tilde{\psi}$

Figure 8.4: Scaling functions and mother wavelets for the CDF 9/7 wavelet.

Exercises for Section 8.6

1. As in Exercise 8.5.**??** write a function

```
function [g0,g1,h0,h1]=filters97()
```

which verifies the filter coefficients of Example 8.12.

8.7 Orthonormal wavelets

Since the filters here are not symmetric, the method of symmetric extension does not work in the same simple way as before. This partially explains why symmetric filters are used more often: They may not be as efficient in representing functions, since the corresponding basis is not orthogonal, but their simple implementation still makes them attractive.

The polynomials $Q^{(0)}$, $Q^{(1)}$, and $Q^{(2)}$ require no further action to obtain the factorization $f(e^{i\omega})f(e^{-i\omega}) = Q\left(\frac{1}{2}(1-\cos\omega)\right)$. The polynomial $Q^{(3)}$ in Equation (8.21) can be factored further as

$$
\begin{aligned}
Q^{(3)}\left(\frac{1}{2}(1-\cos\omega)\right) &= \frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}(e^{-3i\omega} - 7.1029e^{-2i\omega} + 19.5014^{-i\omega} - 21.7391)\\
&\times (e^{3i\omega} - 7.1029e^{-i\omega} + 19.5014^{i\omega} - 21.7391),
\end{aligned}
$$

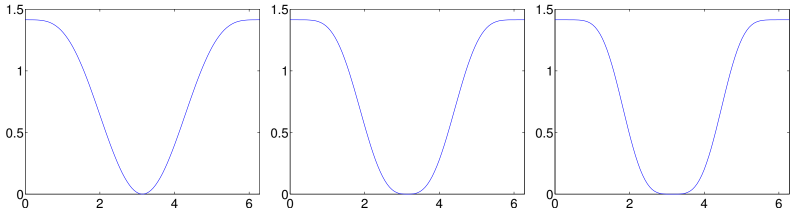
Note, however, that this factorization is not unique. This gives the frequency responses

$$
\begin{aligned}
&\frac{1}{2}(e^{-i\omega}+1)\sqrt{2}\\
&\frac{1}{4}(e^{-i\omega}+1)^2\sqrt{\frac{1}{3.7321}}(e^{-i\omega}-3.7321)\\
&\frac{1}{8}(e^{-i\omega}+1)^3\sqrt{\frac{3}{4}\frac{1}{9.4438}}(e^{-2i\omega}-5.4255e^{-i\omega}+9.4438)\\
&\frac{1}{16}(e^{-i\omega}+1)^4\sqrt{\frac{5}{8}\frac{1}{3.0407}\frac{1}{7.1495}}(e^{-3i\omega}-7.1029e^{-2i\omega}+19.5014^{-i\omega}-21.7391),
\end{aligned}
$$

which gives the filters

$$
\begin{aligned}
G_0 = H_0 =&(\underline{\sqrt{2}/2}, \sqrt{2}/2)\\
G_0 = H_0 =&(\underline{-0.4830}, -0.8365, -0.2241, 0.1294)\\
G_0 = H_0 =&(\underline{0.3327}, 0.8069, 0.4599, -0.1350, -0.0854, 0.0352)\\
G_0 = H_0 =&(\underline{-0.2304}, -0.7148, -0.6309, 0.0280, 0.1870, -0.0308, -0.0329, 0.0106)
\end{aligned}
$$

so that we get 2, 4, 6 and 8 filter coefficients in $G_0 = H_0$. We see that the filter coefficients when $N = 1$ are those of the Haar wavelet. The three next filters we



(a) $N = 2$ (b) $N = 3$ (c) $N = 4$

Figure 8.5: The magnitude of the frequency responses $\lambda_{G_0}(\omega) = \lambda_{H_0}(\omega)$ for the first orthonormal wavelets with vanishing moments.

have not seen before. Using Theorem 8.1, the filter $H_1 = G_1$ can be obtained by reversing the elements and adding an alternating sign, so that

$$\begin{aligned}
G_1 = H_1 =&(\sqrt{2}/2, \underline{-\sqrt{2}/2})\\
G_1 = H_1 =&(0.1294, \underline{0.2241}, -0.8365, 0.4830)\\
G_1 = H_1 =&(0.0352, \underline{0.0854}, -0.1350, -0.4599, 0.8069, -0.3327)\\
G_1 = H_1 =&(0.0106, \underline{0.0329}, -0.0308, -0.1870, 0.0280, 0.6309, -0.7148, 0.2304).
\end{aligned}$$

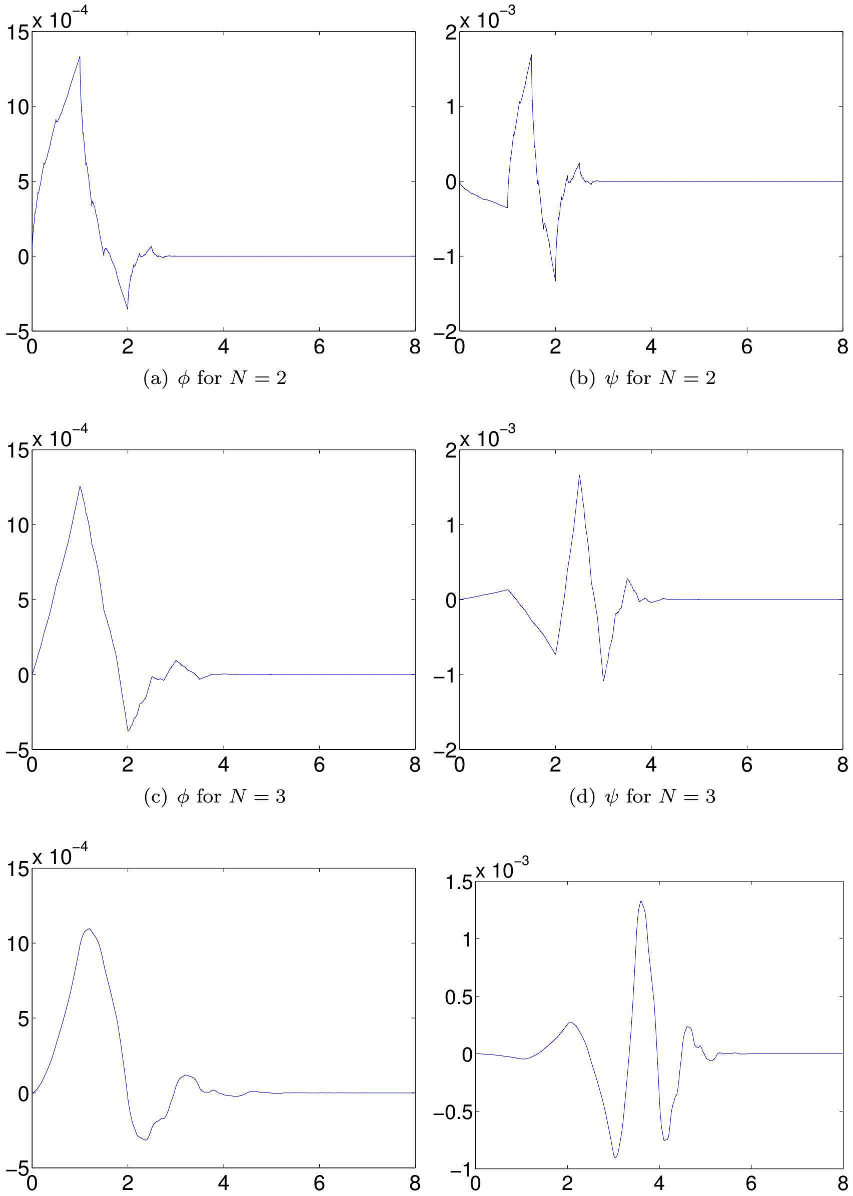
Frequency responses are shown in Figure 8.5. The frequency responses are now complex, so their magnitudes are plotted. The lowpass characteristic of these filters is clearly visible. We also see the highpass characteristics resemble the lowpass characteristics.

Also in this case we can use the cascade algorithm to visualize the scaling functions and mother wavelets. The way we have defined the filters, the supports for both the scaling function and the mother wavelet must be $[0, 3]$ when $N = 2$, $[0, 5]$ when $N = 3$, and $[0, 7]$ when $N = 4$. These are shown in Figure 8.6.

8.8 Summary

We started the section with giving a general characterization of perfect reconstruction systems of MRA-matrices. In this characterization, the frequency responses of the lowpass filters needed to satisfy a certain equation, and once this is satsified the highpass filters can be easily obtained in the way we previously have obtained highpass filters from lowpass filters. We then specialized our search for wavelets to cases with vanishing moments, and where we have few filter coefficients. This search could also be done in terms of the frequency responses of the involved filters. Finally we studied some examples, which have applications to image compression.

For the wavelets we constructed in this chapter, we also plotted the corresponding scaling functions and mother wavelets (see figures 8.2, 8.4, 8.6). The importance of these functions are that they are particularly suited for approximation of regular functions, and providing a compact representation of these



(a) ϕ for $N = 2$

(b) ψ for $N = 2$

(c) ϕ for $N = 3$

(d) ψ for $N = 3$

(e) ϕ for $N = 4$

(f) ψ for $N = 4$

Figure 8.6: The scaling functions and mother wavelets for the first orthonormal wavelets with vanishing moments.

functions which is localized in time. It seems difficult to guess that these strange shapes are connected to such approximation. Moreover, it may seem strange that, although these functions are useful, we can't write down exact expressions for them, and they are only approximated in terms of the Cascade Algorithm.

Chapter 9

Lifting

Previously we saw that a filter could be factored into smaller filters, which allows for efficient hardware implementations. Since a DWT is equivalent to applying two filters, the same factorization can be applied for the DWT. However, we need not take into account that only every second component of the filter needs to be evaluated. Therefore we need to make some small adjustements to the factorization into filters. After these adjustments the new factorization is called the *lifting factorization*. It turns out that this factorization provides some reductions in the number of arithmetic operations as well. This chapter is devoted to establishing the lifting factorization, and proving its properties.

9.1 Motivation

Let us again consider the piecewise linear wavelet, for which we found that the change of coordinates matrix $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ was given by Equation (7.3). Let us instead consider the change of coordinates from $\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1$ to

$$\mathcal{D}_1 = \{\phi_{1,0}, \phi_{1,2}, \phi_{1,4} \ldots, \phi_{1,1}, \phi_{1,3}, \phi_{1,5}, \ldots\},$$

i.e. we reorder the basis vectors in $\boldsymbol{\phi}_1$ so that the even-indexed ones come first. Clearly $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is formed from $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ by taking even- and odd-indexed elements in all possible ways from $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$, and storing these as the four different blocks in the resulting matrix. Clearly these blocks must be filters, and we can easily deduce them from the first two columns. If we underline the corresponding elements in $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ we get

$$\frac{1}{\sqrt{2}}\begin{pmatrix} \underline{1} & 0 \\ 1/2 & 1 \\ \underline{0} & 0 \\ \vdots & \vdots \\ \underline{0} & 0 \\ 1/2 & 0 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & \underline{0} \\ 1/2 & 1 \\ 0 & \underline{0} \\ \vdots & \vdots \\ 0 & \underline{0} \\ 1/2 & 0 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ \underline{1/2} & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \underline{1/2} & 0 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 1/2 & \underline{1} \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1/2 & \underline{0} \end{pmatrix}. \tag{9.1}$$

We therefore get the following:

- The upper left corner of $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is $\frac{1}{\sqrt{2}} I$.
- The upper right corner of $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is $\mathbf{0}$.
- The lower left corner of $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is $\frac{1}{\sqrt{2}} T_1$, where $T_1 = \{1/2, \underline{1/2}\}$.
- The lower right corner of $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is $\frac{1}{\sqrt{2}} I$.

In other words,

$$P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1} = \frac{1}{\sqrt{2}} \begin{pmatrix} I & \mathbf{0} \\ T_1 & I \end{pmatrix}. \tag{9.2}$$

Note also that this matrix is easily inverted, since

$$\begin{pmatrix} I & \mathbf{0} \\ T_1 & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -T_1 & I \end{pmatrix} = \begin{pmatrix} I & \mathbf{0} \\ T_1 - T_1 & I \end{pmatrix} = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & I \end{pmatrix} = I.$$

This means that

$$P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \mathcal{D}_1} = \sqrt{2} \begin{pmatrix} I & \mathbf{0} \\ -T_1 & I \end{pmatrix}.$$

We have thus seen two advantages of reordering the wavelet bases as we did here: First of all, we obtain block matrices where all blocks are filters. Secondly, these block matrices are easily inverted.

Clearly, $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ for a general wavelet is a also a block matrix where the blocks are filters:

Theorem 9.1. For any wavelet, $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ and $P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \mathcal{D}_1}$ are on the form $\begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix}$, where the matrices $S^{(i,j)}$ are filters. $S^{(i,j)}$ are also called the *DWT polyphase components* and *the IDWT polyphase components*, and can be found as the submatrices of $P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ and $P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ where we extract even- and odd-indexed elements from the rows/columns in all possible ways.

In the following we will interchangeably use this polyphase representation and the MRA-matrix when we represent the wavelet, We will write $A \leftrightarrow B$ when $A = P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \mathcal{D}_1}$ or $A = P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is the polyphase representation of the wavelet, and $B = P_{\mathcal{C}_1 \leftarrow \boldsymbol{\phi}_1}$ or $B = P_{\boldsymbol{\phi}_1 \leftarrow \mathcal{C}_1}$ is the MRA-matrix of the wavelet. We have the following result on the polyphase components. This result is easily proved from manipulation with block matrices, and is therefore left to the reader.

Theorem 9.2. Let A and B be MRA-matrices with polyphase components $A^{(i,j)}$, $B^{(i,j)}$. The following hold

1. $C = AB$ is also an MRA-matrix, and with polyphase components $C^{(i,j)} = A^{(i,0)} B^{(0,j)} + A^{(i,1)} B^{(1,j)}$.

2. A^T is also an MRA-matrix, and with polyphase components of A^T are $((A^T)^{(i,j)})_{kl} = (A^{(j,i)})_{lk}$.

Also, the polyphase from of the identity matrix is $\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}$.

Example 9.3. Polyphase components can be defined for any MRA-matrix, regardless of whether it arises from a wavelet or not. As an example, consider the 6×6 MRA-matrix

$$S = \begin{pmatrix} 2 & 3 & 0 & 0 & 0 & 1 \\ 4 & 5 & 6 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \\ 6 & 0 & 0 & 0 & 4 & 5 \end{pmatrix}. \tag{9.3}$$

The polyphase components of S are

$$S^{(0,0)} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \qquad S^{(0,1)} = \begin{pmatrix} 3 & 0 & 1 \\ 1 & 3 & 0 \\ 0 & 1 & 3 \end{pmatrix}$$

$$S^{(1,0)} = \begin{pmatrix} 4 & 6 & 0 \\ 0 & 4 & 6 \\ 6 & 0 & 4 \end{pmatrix} \qquad S^{(1,1)} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

♣

Even though the DWT and the IDWT polyphase components are filters, they are not the same as the filters G_0, G_1, H_0, H_1, since they are formed by taking every second element from these in all possible ways.

$P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is different from Equation (9.2) in general, so that it is not so easily inverted in general. To look further into this, let us consider the alternative piecewise linear wavelet. In this case, Equation (7.5) shows that $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1}$ is not on the form $\begin{pmatrix} I & \mathbf{0} \\ T_1 & I \end{pmatrix}$ for some T_1, since there is more than one element in every column. However we can write

$$P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1} = P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1} P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1}.$$

where $\hat{\psi}$ is defined in Section 6.3 by Equation 6.15. From this equation it is clear that

$$P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1} = \begin{pmatrix} I & T_2 \\ 0 & I \end{pmatrix},$$

where

$$T_2 = -\frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix} = -\frac{1}{4}\{\underline{1},1\}. \tag{9.4}$$

Since we already have computed $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$, this means that

$$P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1} = \frac{1}{\sqrt{2}}\begin{pmatrix} I & \mathbf{0} \\ T_1 & I \end{pmatrix}\begin{pmatrix} I & T_2 \\ \mathbf{0} & I \end{pmatrix}.$$

In other words, also here the same type of matrix could be used to express the change of coordinates. This matrix is also easily invertible, and

$$P_{\boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1 \leftarrow \mathcal{D}_1} = \sqrt{2}\begin{pmatrix} I & -T_2 \\ \mathbf{0} & I \end{pmatrix}\begin{pmatrix} I & \mathbf{0} \\ -T_1 & I \end{pmatrix}.$$

It is also clear how the DWT for the Haar wavelet can be written in the same way: If we first defined $\hat{\psi} = -\sqrt{2}\phi_{1,1}$ for the Haar wavelet, and then continued by defining $\psi = \hat{\psi} + \phi$, ψ is the same function as we originally defined for the Haar wavelet. This gives us that $P_{\boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \hat{\boldsymbol{\psi}}_1} = \begin{pmatrix} I & T_2 \\ 0 & I \end{pmatrix}$ for some T_2 also here. As above, it also follows that $P_{\mathcal{D}_1 \leftarrow \boldsymbol{\phi}_1 \oplus \boldsymbol{\psi}_1}$ is on the form $\begin{pmatrix} I & \mathbf{0} \\ T_1 & I \end{pmatrix}$ for some T_1, so that the same type of factorization can be written down for the Haar wavelet as well.

9.2 The lifting factorization

For the wavelets we have considered we saw above that their matrices could be factored into a product of matrices on the form $\begin{pmatrix} I & T \\ \mathbf{0} & I \end{pmatrix}$ or $\begin{pmatrix} I & \mathbf{0} \\ T & I \end{pmatrix}$. These matrices will be the building blocks in the lifting factorization, so let us make the following definition.

Definition 9.4. A matrix on the form $\left(\begin{array}{cc} I & T \\ 0 & I \end{array} \right)$ where S is a filter is called an *elementary lifting matrix of even type*. A matrix on the form $\left(\begin{array}{cc} I & 0 \\ T & I \end{array} \right)$ is called an *elementary lifting matrix of odd type*.

The following are the most useful properties of elementary lifting matrices:

Lemma 9.5. The following hold:

1. $\begin{pmatrix} I & T \\ 0 & I \end{pmatrix}^T = \begin{pmatrix} I & 0 \\ T^T & I \end{pmatrix}$, and $\begin{pmatrix} I & 0 \\ T & I \end{pmatrix}^T = \begin{pmatrix} I & T^T \\ 0 & I \end{pmatrix}$,
2. $\begin{pmatrix} I & T_1 \\ 0 & I \end{pmatrix}\begin{pmatrix} I & T_2 \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & T_1 + T_2 \\ 0 & I \end{pmatrix}$,
3. $\begin{pmatrix} I & 0 \\ T_1 & I \end{pmatrix}\begin{pmatrix} I & 0 \\ T_2 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ T_1 + T_2 & I \end{pmatrix}$,
4. $\begin{pmatrix} I & T \\ 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -T \\ 0 & I \end{pmatrix}^{-1}$
5. $\begin{pmatrix} I & 0 \\ T & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ -T & I \end{pmatrix}^{-1}$

These statements follow directly from Theorem 9.2. Due to Property 2, one can assume that odd and even types of lifting matrices appear in alternating order, since matrices of the same type can be grouped together. The following result says that the elementary lifting matrices can be used as general building blocks:

Theorem 9.6. Any invertible matrix on the form $S = \begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix}$ where the $S^{(i,j)}$ are filters can be written on the form

$$\Lambda_1 \cdots \Lambda_n \begin{pmatrix} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{pmatrix}, \tag{9.5}$$

where Λ_i are elementary lifting matrices, p, q are integers, α_0, α_1 are nonzero scalars, and E_p, E_q are time delay filters. The inverse is given by

$$\begin{pmatrix} \alpha_0^{-1} E_{-p} & 0 \\ 0 & \alpha_1^{-1} E_{-q} \end{pmatrix} (\Lambda_n)^{-1} \cdots (\Lambda_1)^{-1}. \tag{9.6}$$

Note that $(\Lambda_i)^{-1}$ can be computed with the help of properties 4 and 5 of Lemma 9.5.

Proof. The proof will use the concept of the length of a filter, as defined in Definition 3.24. Let $S = \begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix}$ be an arbitrary invertible matrix. We will incrementally find an elementary lifting matrix Λ_i with filter T_i in the lower left or upper right corner so that $\Lambda_i S$ has filters of lower length in the first column. Assume first that $l(S^{(0,0)}) \geq l(S^{(1,0)})$, where $l(S)$ is the length of a filter as given by Definition 3.24. If Λ_i is of even type, then the first column in $\Lambda_i S$ is

$$\begin{pmatrix} I & T_i \\ 0 & I \end{pmatrix} \begin{pmatrix} S^{(0,0)} \\ S^{(1,0)} \end{pmatrix} = \left(\begin{array}{c} S^{(0,0)} + T_i S^{(1,0)} \\ S^{(1,0)} \end{array} \right). \tag{9.7}$$

T_i can now be chosen so that $l(S^{(0,0)} + T_i S^{(1,0)}) < l(S^{(1,0)})$. To see how, recall that we in Section 3.4 stated that multiplying filters corresponds to multiplying polynomials. T_i can thus be found from polynomial division with remainder: when we divide $S^{(0,0)}$ by $S^{(1,0)}$, we actually find polynomials T_i and P with $l(P) < l(S^{(1,0)})$ so that $S^{(0,0)} = T_i S^{(1,0)} + P$, so that the length of $P = S^{(0,0)} - T_i S^{(1,0)}$ is less than $l(S^{(1,0)})$. The same can be said if Λ_i is of odd type, in which case the first and second components are simply swapped. This procedure can be continued until we arrive at a product

$$\Lambda_n \cdots \Lambda_1 S$$

where either the first or the second component in the first column is 0. If the first component in the first column is 0, the identity

$$\left(\begin{array}{cc} I & 0 \\ -I & I \end{array} \right) \left(\begin{array}{cc} I & I \\ 0 & I \end{array} \right) \left(\begin{array}{cc} 0 & X \\ Y & Z \end{array} \right) = \left(\begin{array}{cc} Y & X+Z \\ 0 & -X \end{array} \right)$$

explains that we can bring the matrix to a form where the second element in the first column is zero instead, with the help of the additional lifting matrices $\Lambda_{n+1} = \left(\begin{array}{cc} I & I \\ 0 & I \end{array} \right)$, and $\Lambda_{n+2} = \left(\begin{array}{cc} I & 0 \\ -I & I \end{array} \right)$, so that we always can assume that the second element in the first column is 0, i.e.

$$\Lambda_n \cdots \Lambda_1 S = \left(\begin{array}{cc} P & Q \\ 0 & R \end{array} \right),$$

for some matrices P, Q, R. From the proof of Theorem 8.1 we will see that in order for S to be invertible, we must have that $S^{(0,0)} S^{(1,1)} - S^{(0,1)} S^{(1,0)} = -\alpha^{-1} E_d$ for some nonzero scalar α and integer d. Since $\left(\begin{array}{cc} P & Q \\ 0 & R \end{array} \right)$ is also invertible, we must thus have that PR must be on the form αE_n, and for this we must have that $P = \alpha_0 E_p$ and $R = \alpha_1 E_q$ for some p, q, α_0, α_1. Using this, and also isolating S on one side, we obtain that

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \left(\begin{array}{cc} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{array} \right), \tag{9.8}$$

Noting that

$$\left(\begin{array}{cc} \alpha_0 E_p & Q \\ 0 & \alpha_1 E_q \end{array} \right) = \left(\begin{array}{cc} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{array} \right) \left(\begin{array}{cc} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{array} \right),$$

we can rewrite (9.8) as

$$S = (\Lambda_1)^{-1} \cdots (\Lambda_n)^{-1} \left(\begin{array}{cc} 1 & \frac{1}{\alpha_1} E_{-q} Q \\ 0 & 1 \end{array} \right) \left(\begin{array}{cc} \alpha_0 E_p & 0 \\ 0 & \alpha_1 E_q \end{array} \right),$$

which is a lifting factorization of the form we wanted to arrive at. The last matrix in the lifting factorization is not really a lifting matrix, but it too can easily be inverted, so that we arrive at (9.6). This completes the proof. □

The factorization 9.5 is what we call the *lifting factorization* of S. In practice, one starts with a given wavelet with certain proved properties such as the ones from the preceding chapter, and applies an algorithm to obtain a lifting factorization of it. The algorihtm can easily be written down from the proof of Theorem 9.6. The lifting factorization is far from unique, and the algorithm only gives one of them.

It is desirable for an implementation to obtain a lifting factorization where the lifting steps are as simple as possible. Let us restrict to the case of wavelets with symmetric filters, since the wavelets used in most applications are symmetric.

Assume that we in the proof of Theorem 9.6 add an elementary lifting of even type. At this step we then compute $S^{(0,0)}+T_iS^{(1,0)}$ in the first entry of the first column. Since $S^{(0,0)}$ is symmetric, $T_iS^{(1,0)}$ must also be symmetric in order for the length to be reduced. And since the filter coefficients of $S^{(1,0)}$ are symmetric about $-1/2$ when S is symmetric, T_i must have coefficients symmetric around $1/2$. If the difference in the lengths of the filters in the first column is 1, we can choose a filter of length 2 to reduce the lengths by 2, so that the T_i in the even lifting steps take the form $T_i = \lambda_i\{\underline{1},1\}$. Similarly, for odd lifting steps one shows that $T_i = \lambda_i\{1,\underline{1}\}$. Let us summarize this as follows:

Theorem 9.7. When the filters in a wavelet are symmetric and the lengths of the filters in the first column differ by 1 at all steps in the lifting factorization, the lifting steps of even type take the simplified form $\begin{pmatrix} I & \lambda_i\{\underline{1},1\} \\ 0 & I \end{pmatrix}$, and the lifting steps of odd type take the simplified form $\begin{pmatrix} I & 0 \\ \lambda_i\{1,\underline{1}\} & I \end{pmatrix}$.

For the wavelets we will consider in the following examples it will turn out the filters in the first column differ by 1 at all steps in the lifting factorization, so that this theorem applies. Such lifting steps are quickly computed due to their simple structure. Writing these elementary lifting steps as MRA-matrices we get

$$\begin{pmatrix} I & \lambda_i\{\underline{1},1\} \\ 0 & I \end{pmatrix} \leftrightarrow \begin{pmatrix} 1 & \lambda & 0 & 0 & \cdots & 0 & 0 & \lambda \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} I & 0 \\ \lambda_i\{1,\underline{1}\} & I \end{pmatrix} \leftrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \lambda & 0 & 0 & 0 & \cdots & 0 & \lambda & 1 \end{pmatrix}.$$

We see that these lifting steps are also MRA-matrices with symmetric filters. In other words, when the lifting factorization is applied as above, it means that an MRA-matrix with symmetric filters is factored into simpler MRA-matrices which also have symmetric filters, i.e. $S = A_1 \cdots A_n$ where all S, A_i are MRA-matrices with symmetric filters. If we apply the DWT to symmetric extensions, we clearly also have that $S_r = (A_1)_r \cdots (A_n)_r$, where S_r is given by Theorem 7.10. We also have that

$$\begin{pmatrix} I & \lambda_i\{\underline{1},1\} \\ 0 & I \end{pmatrix}_r \leftrightarrow \begin{pmatrix} 1 & 2\lambda & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \lambda & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \lambda & 1 & \lambda \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \tag{9.9}$$

$$\begin{pmatrix} I & 0 \\ \lambda_i\{1,\underline{1}\} & I \end{pmatrix}_r \leftrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \lambda & 1 & \lambda & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2\lambda & 1 \end{pmatrix}. \tag{9.10}$$

These matrices can thus be used to implement the wavelets we look at. These have very simple implementations: everey second element is left unchanged, while for the remaining elements we simply add the two neighbours.

In the following examples we consider how the lifting factorization looks for two of the wavelets we have encountered. The examples also explain how lifting helps us to obtain some computational improvements in the number of aritmetic operations.

Example 9.8 (Lifting of the Spline 5/3 wavelet)**.** Let us consider the Spline 5/3 wavelet, which we defined in Examplee 8.11. Let us start by looking at the matrix where G^T. This has the filter coefficients of G_0 and G_1 in the first two rows, and we recall that

$$\begin{aligned} G_0 &= \{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\} \\ G_1 &= \{-\frac{1}{4}, -\frac{1}{2}, \underline{\frac{3}{2}}, -\frac{1}{2}, -\frac{1}{4}\}, \end{aligned}$$

from which we see that the polyphase components are

$$\begin{pmatrix} S^{(0,0)} & S^{(0,1)} \\ S^{(1,0)} & S^{(1,1)} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}I & \{\underline{\tfrac{1}{4}}, \tfrac{1}{4}\} \\ \{-\tfrac{1}{2}, \underline{-\tfrac{1}{2}}\} & \{-\tfrac{1}{4}, \underline{\tfrac{3}{2}}, -\tfrac{1}{4}\} \end{pmatrix}$$

We see here that the lower filter has biggest length in the first column, so that we must start with an elementary lifting of odd type. We must then find a T_1 so that $T_i \frac{1}{2} I + \{-\frac{1}{2}, \underline{-\frac{1}{2}}\}$ has lower length than $\frac{1}{2}I$. It is clear that we can choose $T_i = \{1, \underline{1}\}$, and that we then get $\mathbf{0}$. The first lifting step thus gives

$$\begin{aligned} \Lambda_1 G^T &= \begin{pmatrix} I & \mathbf{0} \\ \{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} \frac{1}{2}I & \{\underline{\tfrac{1}{4}}, \tfrac{1}{4}\} \\ \{-\tfrac{1}{2}, \underline{-\tfrac{1}{2}}\} & \{-\tfrac{1}{4}, \underline{\tfrac{3}{2}}, -\tfrac{1}{4}\} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}I & \{\underline{\tfrac{1}{4}}, \tfrac{1}{4}\} \\ \mathbf{0} & 2I \end{pmatrix} \\ &= \begin{pmatrix} I & \frac{1}{8}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} \frac{1}{2}I & \mathbf{0} \\ \mathbf{0} & 2I \end{pmatrix}, \end{aligned}$$

where we also used the same rewriting as in the proof of Theorem 9.6. This gives

$$G^T = \begin{pmatrix} I & \mathbf{0} \\ \{-1, \underline{-1}\} & I \end{pmatrix} \begin{pmatrix} I & \frac{1}{8}\{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} \frac{1}{2}I & \mathbf{0} \\ \mathbf{0} & 2I \end{pmatrix}.$$

Transposing this expression gives

$$G = \begin{pmatrix} \frac{1}{2}I & \mathbf{0} \\ \mathbf{0} & 2I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ \frac{1}{8}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & \{\underline{-1}, -1\} \\ \mathbf{0} & I \end{pmatrix}$$

Taking inverses of this expression gives

$$H = \begin{pmatrix} I & \{\underline{1}, 1\} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ -\frac{1}{8}\{1, \underline{1}\} & I \end{pmatrix} \begin{pmatrix} 2I & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}I \end{pmatrix}$$

We now have obtained the lifting factorization. Only two lifting steps were required. We also see that the lifting steps involve only dyadic fractions, just as the filter coefficients did. This means that both the lifting factorization also can be used for lossless operations.

Let us also compute the number of additions and multiplications performed when we apply the lifting factorization. For a signal of length N, each lifting step uses a number of N additions and $N/2$ multiplications. Since there are two lifting steps, the number of additions and multiplications are $2N$ and $2N$, respectively, where we have taken into account multiplication with the diagonal matrix as well. A direct implementation of DWT in terms of filters would need a total of $3N$ additions and $2.5N$ multiplications, where we have taken the symmetry of the filters into account. We thus see that the lifting factorization gives a small decrease in the number of operations. ♣

Example 9.9 (Lifting of the CDF 9/7 wavelet)**.** For the wavelet we considered in Example 8.12, it is more cumbersome to compute the lifting factorization by hand. It is however, straightforward to write an algorithm which computes the lifting steps, as these are performed in the proof of Theorem 9.6. You will be spared the details of this algorithm. Also, when we use these wavelets in implementations later they will use precomputed values of these lifting steps, and you can take these implementations for granted too. If we run the algorithm for computing the lifting factorization we obtain

$$\begin{aligned}
H &= \begin{pmatrix} I & 0.5861\{\underline{1},1\} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0.6681\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.0700\{\underline{1},1\} \\ 0 & I \end{pmatrix} \\
&\quad \times \begin{pmatrix} I & 0 \\ -1.2002\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} 1/0.8699 & 0 \\ 0 & 1/1.1496 \end{pmatrix} \\
G &= \begin{pmatrix} 0.8699 & 0 \\ 0 & 1.1496 \end{pmatrix} \begin{pmatrix} I & 0 \\ 1.2002\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & 0.0700\{\underline{1},1\} \\ 0 & I \end{pmatrix} \\
&\quad \times \begin{pmatrix} I & 0 \\ -0.6681\{1,\underline{1}\} & I \end{pmatrix} \begin{pmatrix} I & -0.5861\{\underline{1},1\} \\ 0 & I \end{pmatrix}.
\end{aligned}$$

In this case four lifting steps were required. Let us also here compute the number of additions and multiplications performed. Since there now are four lifting steps, the number of additions and multiplications are $4N$ and $3N$, respectively. A direct implementation of DWT in terms of filters would need a total of $7N$ additions and $4.5N$ multiplications, where we again have taken the symmetry of the filters into account. Still the decrease in the number of operations is not very big, but at least the decrease is clearer than in the previous example. ♣

It is not very hard to show that, as the number of lifting steps increase, the lifting factorization halves the number of additions and multiplications when compared to the direct implementation. Perhaps more important than this is the fact that the lifting factorization splits the DWT and IDWT into simpler components, each very attractive for hardware implementations. Lifting provides us with a complete implementation strategy for the DWT and IDWT. What remains now is to extend to two-dimensional objects such as images. We will start with this in the next chapter.

Exercises for Section 9.2

1. Write functions

```
function x=liftingstepapplyA(lambda,x)
function x=liftingstepapplyB(lambda,x)
```

which applies the elementary lifting matrices (9.9) and (9.10), respectively, to the vector `x`. You can assume that N has even length. The function should not perform matrix multiplication to achieve this, and the result should be returned in the same vector as the input. The function should also perform as few multiplications as possible. How can you achieve this?

2. Write functions `DWTImpl53` and `IDWTImpl53` which implements the DWT and the IDWT for the Spline 5/3 wavelet, using the lifting factorization obtained in Example 9.8. You should use the functions you implemented in Exercise 1.

3. Write functions `DWTImpl97` and `IDWTImpl97` which implements the DWT and the IDWT for the CDF 9/7 wavelet, using the lifting factorization obtained in Example 9.9.

The proof of Theorem 8.1

Since the polyphase form of H^T is $\begin{pmatrix} (H^{(0,0)})^T & (H^{(1,0)})^T \\ (H^{(0,1)})^T & (H^{(1,1)})^T \end{pmatrix}$, The polyphase form for the equation $GH^T = I$ is thus

$$\begin{pmatrix} G^{(0,0)} & G^{(0,1)} \\ G^{(1,0)} & G^{(1,1)} \end{pmatrix} \begin{pmatrix} (H^{(0,0)})^T & (H^{(1,0)})^T \\ (H^{(0,1)})^T & (H^{(1,1)})^T \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix},$$

which can be split into the four equations

$$\begin{aligned}
G^{(0,0)}(H^{(0,0)})^T + G^{(0,1)}(H^{(0,1)})^T &= I && (I)\\
G^{(0,0)}(H^{(1,0)})^T + G^{(0,1)}(H^{(1,1)})^T &= 0 && (II)\\
G^{(1,0)}(H^{(0,0)})^T + G^{(1,1)}(H^{(0,1)})^T &= 0 && (III)\\
G^{(1,0)}(H^{(1,0)})^T + G^{(1,1)}(H^{(1,1)})^T &= I && (IV)
\end{aligned}$$

Combining these and substracting, this can be written

$$\begin{aligned}
(G^{(0,0)}G^{(1,1)} - G^{(0,1)}G^{(1,0)})(H^{(0,1)})^T &= -G^{(1,0)}\\
((H^{(0,0)})^T(H^{(1,1)})^T - (H^{(0,1)})^T(H^{(1,0)})^T)G^{(1,0)} &= -(H^{(0,1)})^T
\end{aligned}$$

where we in the first formula performed $G^{(0,0)}(III) - G^{(1,0)}(I)$, and in the second formula performed $(H^{(1,1)})^T(III) - (H^{(0,1)})^T(IV)$. Since $G^{(0,0)}G^{(1,1)} - G^{(0,1)}G^{(1,0)}$ and $(H^{(0,0)})^T(H^{(1,1)})^T - (H^{(0,1)})^T(H^{(1,0)})^T$ also are circulant Toeplitz matrices the number of filter coefficients satisfy $l((H^{(0,1)})^T) \leq l(G^{(1,0)})$, and $l((H^{(0,1)})^T) \geq l(G^{(1,0)})$, respectively, so that $l(H^{(0,1)}) = l(G^{(1,0)})$, and both matrices

$$\begin{aligned}
&G^{(0,0)}G^{(1,1)} - G^{(0,1)}G^{(1,0)}\\
&(H^{(0,0)})^T(H^{(1,1)})^T - (H^{(0,1)})^T(H^{(1,0)})^T
\end{aligned}$$

must have only one nonzero diagonal. By combining equations (I)-(IV) in other ways as well, and defining the diagonal matrix $D = G^{(0,0)}G^{(1,1)} - G^{(0,1)}G^{(1,0)} = -\alpha^{-1}E_d$ for some α, d, we obtain

$$\begin{aligned} G^{(1,0)} &= -D(H^{(0,1)})^T = \alpha^{-1}E_d(H^{(0,1)})^T \qquad (9.11)\\ G^{(1,1)} &= D(H^{(0,0)})^T = -\alpha^{-1}E_d(H^{(0,0)})^T \qquad (9.12)\\ (H^{(1,0)})^T &= -D^{-1}G^{(0,1)} = \alpha E_{-d}G^{(0,1)}\\ (H^{(1,1)})^T &= D^{-1}G^{(0,0)} = -\alpha E_{-d}G^{(0,0)}, \end{aligned}$$

where the last two equations also can be written

$$\begin{aligned} H^{(1,0)} &= \alpha(G^{(0,1)})^T E_d = \alpha E_d(G^{(0,1)})^T \qquad (9.13)\\ H^{(1,1)} &= -\alpha(G^{(0,0)})^T E_d = -E_d\alpha(G^{(0,0)})^T. \qquad (9.14) \end{aligned}$$

The equations can thus be written

$$\begin{aligned} (G^{(1,0)})_n &= \alpha^{-1}(H^{(0,1)})_{-n-d}\\ (G^{(1,1)})_n &= -\alpha^{-1}(H^{(0,0)})_{-n-d}\\ (H^{(1,0)})_n &= \alpha(G^{(0,1)})_{-n-d}\\ (H^{(1,1)})_n &= -\alpha(G^{(0,0)})_{-n-d} \end{aligned}$$

Using the two first equations we get

$$\begin{aligned} (G_1)_{2n-1} &= (G^{(1,0)})_n = \alpha^{-1}(H^{(0,1)})_{-n-d} = \alpha^{-1}(H_0)_{2(-n-d)+1}\\ &= (-1)^{2n}\alpha^{-1}(H_0)_{-(2n-1)-2d}\\ (G_1)_{2n} &= (G^{(1,1)})_n = -\alpha^{-1}(H^{(0,0)})_{-n-d} = -\alpha^{-1}(H_0)_{2(-n-d)}\\ &= (-1)^{2n+1}\alpha^{-1}(H_0)_{-2n-2d}. \end{aligned}$$

Using the two last equations we get

$$\begin{aligned} (H_1)_{2n-1} &= (H^{(1,0)})_n = \alpha(G^{(0,1)})_{-n-d} = \alpha(G_0)_{2(-n-d)+1}\\ &= (-1)^{2n}\alpha(G_0)_{-(2n-1)-2d}\\ (H_1)_{2n} &= (H^{(1,1)})_n = -\alpha(G^{(0,0)})_{-n-d} = -\alpha(G_0)_{2(-n-d)}\\ &= (-1)^{2n+1}\alpha(G_0)_{-2n-2d}. \end{aligned}$$

We thus see that

$$\begin{aligned} (G_1)_n &= (-1)^n\alpha^{-1}(H_0)_{-n-2d}\\ (H_1)_n &= (-1)^n\alpha(G_0)_{-n-2d}, \end{aligned}$$

which proves (8.1) and (8.2). To also prove (8.3), first substitute (9.11)-(9.14) in (II)-(III). Then we get

$$\begin{aligned} G^{(0,0)}G^{(0,1)} - G^{(0,1)}G^{(0,0)} &= 0\\ (H^{(0,1)})^T(H^{(0,0)})^T - (H^{(0,0)})^T(H^{(0,1)})^T &= 0. \end{aligned}$$

This is always the case since filters commute, so we are left with considering (I) and (IV). These become

$$G^{(0,0)}(H^{(0,0)})^T + G^{(0,1)}(H^{(0,1)})^T = I \tag{9.15}$$

$$(H^{(0,1)})^T G^{(0,1)} + (H^{(0,0)})^T G^{(0,0)} = I \tag{9.16}$$

These two are equivalent for the same reason, so we are left with Equation 9.15. Using Theorem 3.10 and Theorem 3.20, this can be written as

$$\lambda_{G^{(0,0)},n}\overline{\lambda_{H^{(0,0)},n}} + \lambda_{G^{(0,1)},n}\overline{\lambda_{H^{(0,1)},n}} = 1. \tag{9.17}$$

The rest of the proof follows by rewriting (8.3), till we arrive at (9.17). We first write

$$\begin{aligned}
&\lambda_{G_0,n}\overline{\lambda_{H_0,n}} + \lambda_{G_0,n+N/2}\overline{\lambda_{H_0,n+N/2}} \\
&= \lambda_{G_0,n}\overline{\lambda_{H_0,n}} + \overline{\lambda_{G_0,N/2-n}}\lambda_{H_0,N/2-n},
\end{aligned}$$

where we have used the symmetry property of the DFT coefficients (Theorem 2.21 1.). Using Theorem **??** we have

$$\begin{aligned}
\lambda_{G_0,q} &= \lambda_{G^{(0,0)},q} + e^{-2\pi iq/N}\lambda_{G^{(0,1)},q} \\
\lambda_{H_0,q} &= \lambda_{H^{(0,0)},q} + e^{-2\pi iq/N}\lambda_{H^{(0,1)},q},
\end{aligned}$$

and substituting these expressions in the above we get

$$\begin{aligned}
&= \left(\lambda_{G^{(0,0)},n} + e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\right)\left(\overline{\lambda_{H^{(0,0)},n} + e^{-2\pi in/N}\lambda H^{(0,1)},n}\right) + \\
&\left(\overline{\lambda_{G^{(0,0)},N/2-n} + e^{-2\pi i(N/2-n)/N}\lambda_{G^{(0,1)},N/2-n}}\right)\left(\lambda_{H^{(0,0)},N/2-n} + e^{-2\pi i(N/2-n)/N}\lambda_{H^{(0,1)},N/2-n}\right) \\
&= \left(\lambda_{G^{(0,0)},n} + e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\right)\left(\overline{\lambda_{H^{(0,0)},n} + e^{-2\pi in/N}\lambda_{H^{(0,1)},n}}\right) \\
&\quad + \left(\overline{\lambda_{G^{(0,0)},N/2-n} - e^{2\pi in/N}\lambda_{G^{(0,1)},N/2-n}}\right)\left(\lambda_{H^{(0,0)},N/2-n} - e^{2\pi in/N}\lambda_{H^{(0,1)},N/2-n}\right).
\end{aligned}$$

Using the symmetry property of the DFT coefficients again, and multiplying out we get

$$\begin{aligned}
&= \left(\lambda_{G^{(0,0)},n} + e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\right)\left(\overline{\lambda_{H^{(0,0)},n}} + e^{2\pi in/N}\overline{\lambda_{H^{(0,1)},n}}\right) \\
&\quad + \left(\lambda_{G^{(0,0)},n} - e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\right)\left(\overline{\lambda_{H^{(0,0)},n}} - e^{2\pi in/N}\overline{\lambda_{H^{(0,1)},n}}\right) \\
&=2\lambda_{G^{(0,0)},n}\overline{\lambda_{H^{(0,0)},n}} + 2\lambda_{G^{(0,1)},n}\overline{\lambda_{H^{(0,1)},n}} \\
&\quad + e^{2\pi in/N}\lambda_{G^{(0,0)},n}\overline{\lambda_{H^{(0,1)},n}} + e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\overline{\lambda_{H^{(0,0)},n}} \\
&\quad - e^{2\pi in/N}\lambda_{G^{(0,0)},n}\overline{\lambda_{H^{(0,1)},n}} - e^{-2\pi in/N}\lambda_{G^{(0,1)},n}\overline{\lambda_{H^{(0,0)},n}} \\
&=2\lambda_{G^{(0,0)},n}\overline{\lambda_{H^{(0,0)},n}} + 2\lambda_{G^{(0,1)},n}\overline{\lambda_{H^{(0,1)},n}}
\end{aligned}$$

Combining this with (9.17) we get

$$\lambda_{G_0,n}\overline{\lambda_{H_0,n}} + \lambda_{G_0,n+N/2}\overline{\lambda_{H_0,n+N/2}} = 2,$$

which is (8.3).

9.3 Summary

We found a way to factorize a wavelet into simpler components, and called this the lifting factorization. The lifting factorization has a similar role as the FFT for the DFT. We applied the lifting factorization to the wavelets we constructed in the previous chapter. We now have made most developments for wavelets, except to extend the analysis so that it can be applied to two-dimensional functions such as images.

Chapter 10

Digital images

The theory on wavelets has been presented as a one-dimensional theory upto now. Images, however, are two-dimensional by nature. This poses another challenge, contrary to the case for sound. In the next chapter we will establish the mathematics to handle this, but first we will present some basics on images. Images are a very important type of digital media, and we will go through how they can be represented and manipulated with simple mathematics. This is useful general knowledge for anyone who has a digital camera and a computer, but for many scientists, it is an essential tool. In astrophysics, data from both satellites and distant stars and galaxies is collected in the form of images, and information extracted from the images with advanced image processing techniques. Medical imaging makes it possible to gather different kinds of information in the form of images, even from the inside of the body. By analysing these images it is possible to discover tumours and other disorders.

10.1 What is an image?

Before we do computations with images, it is helpful to be clear about what an image really is. Images cannot be perceived unless there is some light present, so we first review superficially what light is.

10.1.1 Light

Fact 10.1 (What is light?)**.** Light is electromagnetic radiation with wavelengths in the range 400–700 nm (1 nm is 10^{-9} m): Violet has wavelength 400 nm and red has wavelength 700 nm. White light contains roughly equal amounts of all wave lengths.

Other examples of electromagnetic radiation are gamma radiation, ultraviolet and infrared radiation and radio waves, and all electromagnetic radiation travel at the speed of light (3×10^8 m/s). Electromagnetic radiation consists of waves and may be reflected and refracted, just like sound waves (but sound waves are not electromagnetic waves).

We can only see objects that emit light, and there are two ways that this can happen. The object can emit light itself, like a lamp or a computer monitor, or it reflects light that falls on it. An object that reflects light usually absorbs light as well. If we perceive the object as red it means that the object absorbs all light except red, which is reflected. An object that emits light is different; if it is to be perceived as being red it must emit only red light.

10.1.2 Digital output media

Our focus will be on objects that emit light, for example a computer display. A computer monitor consists of a rectangular array of small dots which emit light. In most technologies, each dot is really three smaller dots, and each of these smaller dots emit red, green and blue light. If the amounts of red, green and blue is varied, our brain merges the light from the three small light sources and perceives light of different colours. In this way the colour at each set of three dots can be controlled, and a colour image can be built from the total number of dots.

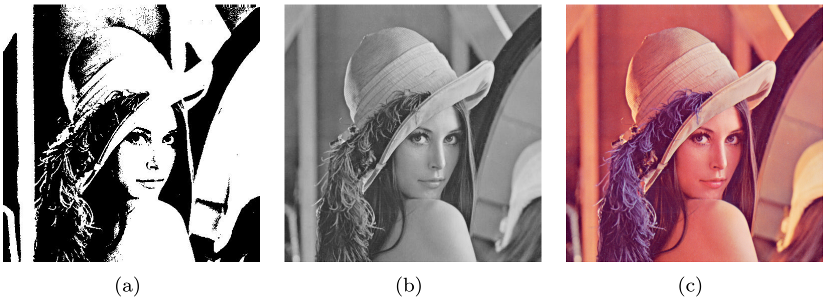
It is important to realise that it is possible to generate most, but not all, colours by mixing red, green and blue. In addition, different computer monitors use slightly different red, green and blue colours, and unless this is taken into consideration, colours will look different on the two monitors. This also means that some colours that can be displayed on one monitor may not be displayable on a different monitor.

Printers use the same principle of building an image from small dots. On most printers however, the small dots do not consist of smaller dots of different colours. Instead as many as 7–8 different inks (or similar substances) are mixed to the right colour. This makes it possible to produce a wide range of colours, but not all, and the problem of matching a colour from another device like a monitor is at least as difficult as matching different colours across different monitors.

Video projectors builds an image that is projected onto a wall. The final image is therefore a reflected image and it is important that the surface is white so that it reflects all colours equally.

The quality of a device is closely linked to the density of the dots.

Fact 10.2 (Resolution)**.** The resolution of a medium is the number of dots per inch (dpi). The number of dots per inch for monitors is usually in the range 70–120, while for printers it is in the range 150–4800 dpi. The horizontal and vertical densities may be different. On a monitor the dots are usually referred to as *pixels* (picture elements).



(a)

(b)

(c)

Figure 10.1: Different version of the same image; black and white (a), grey-level (b), and colour (c).

10.1.3 Digital input media

The two most common ways to acquire digital images is with a digital camera or a scanner. A scanner essentially takes a photo of a document in the form of a rectangular array of (possibly coloured) dots. As for printers, an important measure of quality is the number of dots per inch.

Fact 10.3. The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the colour is represented with up to 48 bits per dot.

For digital cameras it does not make sense to measure the resolution in dots per inch, as this depends on how the image is printed (its size). Instead the resolution is measured in the number of dots recorded.

Fact 10.4. The number of pixels recorded by a digital camera usually varies in the range 320×240 to 6000×4000 with 24 bits of colour information per pixel. The total number of pixels varies in the range 76 800 to 24 000 000 (0.077 megapixels to 24 megapixels).

For scanners and cameras it is easy to think that the more dots (pixels), the better the quality. Although there is some truth to this, there are many other factors that influence the quality. The main problem is that the measured colour information is very easily polluted by noise. And of course high resolution also means that the resulting files become very big; an uncompressed 6000×4000 image produces a 72 MB file. The advantage of high resolution is that you can magnify the image considerably and still maintain reasonable quality.

10.1.4 Definition of digital image

We have already talked about digital images, but we have not yet been precise about what it is. From a mathematical point of view, an image is quite simple.

Fact 10.5 (Digital image)**.** A digital image P is a rectangular array of *intensity values* $\{p_{i,j}\}_{i,j=1}^{m,n}$. For grey-level images, the value $p_{i,j}$ is a single number, while for colour images each $p_{i,j}$ is a vector of three or more values. If the image is recorded in the rgb-model, each $p_{i,j}$ is a vector of three values,

$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$

that denote the amount of red, green and blue at the point (i, j).

Note that, when refering to the coordinates (i, j) in an image, i will refer to row index, j to column index, in the same was as for matrices. In particular, the top row in the image have coordinates $\{(0, j)\}_{j=0}^{N-1}$, while the left column in the image has coordinates $\{(i, 0)\}_{i=0}^{M-1}$. With this notation, the dimension of the image is $M \times N$. The value $p_{i,j}$ gives the colour information at the point (i, j). It is important to remember that there are many formats for this. The simplest case is plain black and white images in which case $p_{i,j}$ is either 0 or 1. For grey-level images the intensities are usually integers in the range 0–255. However, we will assume that the intensities vary in the interval $[0, 1]$, as this sometimes simplifies the form of some mathematical functions. For colour images there are many different formats, but we will just consider the rgb-format mentioned in the fact box. Usually the three components are given as integers in the range 0–255, but as for grey-level images, we will assume that they are real numbers in the interval $[0, 1]$ (the conversion between the two ranges is straightforward, see section 10.11 below). Figure 10.1 shows an image in different formats. We will use this image as our test image in most parts of this book.

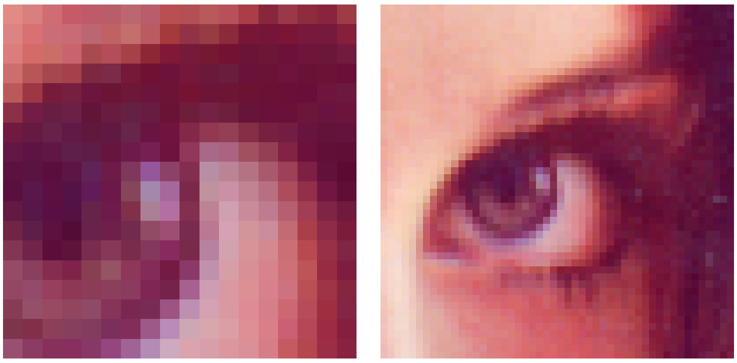
Fact 10.6. In these notes the intensity values $p_{i,j}$ are assumed to be real numbers in the interval $[0, 1]$. For colour images, each of the red, green, and blue intensity values are assumed to be real numbers in $[0, 1]$.

If we magnify the part of the colour image in figure 10.1 around one of the eyes, we obtain the images in figure 10.2. As we can see, the pixels have been magnified to big squares. This is a standard representation used by many programs — the actual shape of the pixels will depend on the output medium. Nevertheless, we will consider the pixels to be square, with integer coordinates at their centres, as indicated by the grids in figure 10.2.

Fact 10.7 (Shape of pixel)**.** The pixels of an image are assumed to be square with sides of length one, with the pixel with value $p_{i,j}$ centred at the point (i, j).

10.1.5 Images as surfaces

Recall from your previous calculus courses that a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$ can be visualised as a surface in space. A grey-level image is almost on this form. If



(a) 18 × 18 pixels

(b) 50 × 50 pixels

Figure 10.2: Two excerpts of the colour image in figure 10.1. The grid indicates the borders between the pixels.

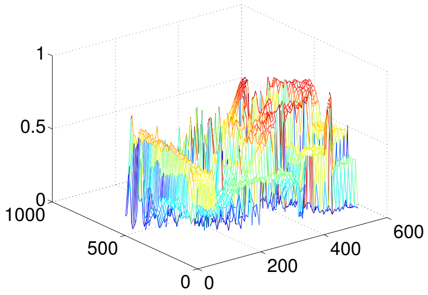


Figure 10.3: A grey-level image viewed as a parametric surface in space.

we define the set of integer pairs by

$$\mathbb{Z}_{m,n} = \big\{(i,j) \mid 1 \leq i \leq m \text{ and } 1 \leq j \leq n\big\},$$

we can consider a grey-level image as a function $P : \mathbb{Z}_{m,n} \mapsto [0,1]$. In other words, we may consider an image to be a sampled version of a surface with the intensity value denoting the height above the (x,y)-plane. In Figure 10.3 we have plotted the surface obtained from the grey-level version of our sample image.

Fact 10.8 (Grey-level image as a surface)**.** Let $P = (p)_{i,j=1}^{m,n}$ be a grey-level image. Then P can be considered a sampled version of the piecewise constant surface

$$F_P : [1/2, m+1/2] \times [1/2, n+1/2] \mapsto [0,1]$$

which has the constant value $p_{i,j}$ in the square (pixel)

$$[i - 1/2, i + 1/2] \times [j - 1/2, j + 1/2]$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

10.2 Operations on images

Images are two-dimensional arrays of numbers, contrary to the sound signals we considered in the previous section. In this respect it is quite obvious that we can manipulate an image by performing mathematical operations on the numbers. In this section we will consider some of the simpler operations. In later sections we will go through more advanced operations, and explain how the theory for these can be generalized from the corresponding theory for one-dimensional (sound) signals (which we wil go through first).

In order to perform these operations, we need to be able to use images with a programming environment such as MATLAB.

10.2.1 Images and MATLAB

An image can also be thought of as a matrix, by associating each pixel with an element in a matrix. The matrix indices thus correspond to positions in the pixel grid. Black and white images correspond to matrices where the elements are natural numbers between 0 and 255. To store a colour image, we need 3 matrices, one for each colour component. This enables us to use linear algebra packages, such as MATLAB, on order to work with images. After we now have made the connection with matrices, we can create images from mathematical formulas, just as we could with sound in the previuos sections. But what we also need before we go through operations on images, is, as in the sections on sound, means of reading an image from a file so that its contents are accessible as a matrix, and write images represented by a matrix which we have constructed ourself to file. Reading a function from file can be done with help of the function `imread`. If we write

```
A = double(imread('filename.fmt','fmt'));
```

the image with the given path and format is read, and stored in the matrix `A`. 'fmt' can be 'jpg','tif', 'gif', 'png',... You should consult the MATLAB help pages to see which formats are supported. After the call to `imread`, we have a matrix where the entries represent the pixel values, and of integer data type (more precisely, the data type `uint8` in Matlab). To perform operations on the image, we must first convert the entries to the data type `double`. This is done with a call to the Matlab function `double`. Similarly, the function `imwrite` can be used to write the image represented by a matrix to file. If we write

```
imwrite(uint8(A), 'filename.fmt','fmt')
```

the image represented by the matrix A is written to the given path, in the given format. Before the image is written to file, you see that we have converted the matrix values back to the integer data type with the help of the function uint8. In other words: imread and imwrite both assume integer matrix entries, while operations on matrices assume double matrix entries. If you want to print images you have created yourself, you can use this function first to write the image to a file, and then send that file to the printer using another program. Finally, we need an alternative to playing a sound, namely displaying an image. The function imshow(A) displays the matrix A as an image in a separate window. Also here you need to convert the samples using the function uint8 prior to this call.

The following examples go through some much used operations on images.

Example 10.9 (Normalising the intensities)**.** We have assumed that the intensities all lie in the interval $[0,1]$, but as we noted, many formats in fact use integer values in the range [0,255]. And as we perform computations with the intensities, we quickly end up with intensities outside $[0,1]$ even if we start out with intensities within this interval. We therefore need to be able to *normalise* the intensities. This we can do with the simple linear function

$$g(x) = \frac{x-a}{b-a}, \quad a < b,$$

which maps the interval $[a,b]$ to $[0,1]$. A simple case is mapping $[0,255]$ to $[0,1]$ which we accomplish with the scaling $g(x) = x/255$. More generally, we typically perform computations that result in intensities outside the interval $[0,1]$. We can then compute the minimum and maximum intensities $p_{\min}$ and $p_{\max}$ and map the interval $[p_{\min}, p_{\max}]$ back to $[0,1]$. Below we have shown a function mapto01 which achieves this task.

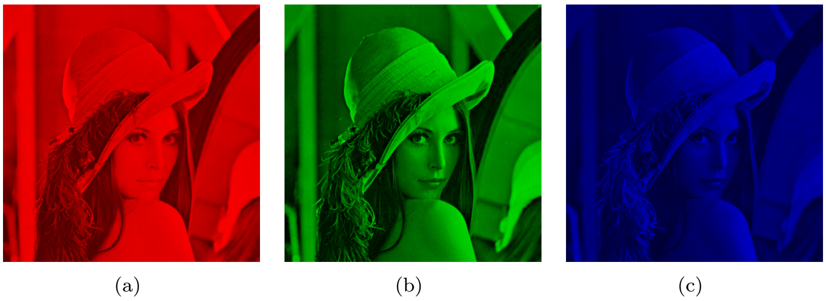
```
function newimg=mapto01(img)
  minval = min(min(min(img)));
  maxval = max(max(max(img)));
  newimg = (img - minval)/(maxval-minval);
```

Several examples of using this function will be shown below. ♣

Example 10.10 (Extracting the different colours)**.** If we have a colour image $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ it is often useful to manipulate the three colour components separately as the three images

$$P_r = (r_{i,j})_{i,j=1}^{m,n}, \quad P_r = (g_{i,j})_{i,j=1}^{m,n}, \quad P_r = (b_{i,j})_{i,j=1}^{m,n}.$$

As an example, let us first see how we can produce three separate images, showing the R,G, and B colour components, respectively. Let us take the image lena.png used in Figure 10.1. When the image is read, three matrices are returned, one for each colour component, and we can generate new files for the different colour components with the following code:



(a) (b) (c)

Figure 10.4: The red (a), green (b), and blue (c) components of the colour image in Figure 10.1.

```
img=double(imread('lena.png','png'));
newimg=zeros(size(img));
newimg(:,:,1)=img(:,:,1);
imwrite(uint8(newimg),'gr.png','png');
newimg=zeros(size(img));
newimg(:,:,2)=img(:,:,2);
imwrite(uint8(newimg),'ggg.png','png');
newimg=zeros(size(img));
newimg(:,:,3)=img(:,:,3);
imwrite(uint8(newimg),'gb.png','png');
```

The resulting image files are shown in Figure 10.4. ♣

Example 10.11 (Converting from colour to grey-level)**.** If we have a colour image we can convert it to a grey-level image. This means that at each point in the image we have to replace the three colour values (r, g, b) by a single value p that will represent the grey level. If we want the grey-level image to be a reasonable representation of the colour image, the value p should somehow reflect the intensity of the image at the point. There are several ways to do this.

It is not unreasonable to use the largest of the three colour components as a measure of the intensity, i.e, to set $p = \max(r, g, b)$. The result of this can be seen in Figure 10.5(a).

An alternative is to use the sum of the three values as a measure of the total intensity at the point. This corresponds to setting $p = r + g + b$. Here we have to be a bit careful with a subtle point. We have required each of the r, g and b values to lie in the range $[0, 1]$, but their sum may of course become as large as 3. We also require our grey-level values to lie in the range $[0, 1]$ so after having computed all the sums we must normalise as explained above. The result can be seen in Figure 10.5(b).

A third possibility is to think of the intensity of (r, g, b) as the length of the colour vector, in analogy with points in space, and set $p = \sqrt{r^2 + g^2 + b^2}$.

A colour image $P = (r_{i,j}, g_{i,j}, b_{i,j})_{i,j=1}^{m,n}$ can be converted to a grey level image $Q = (q_{i,j})_{i,j=1}^{m,n}$ by one of the following three operations:

1. Set $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$ for all i and j.
2. (a) Compute $\hat{q}_{i,j} = r_{i,j} + g_{i,j} + b_{i,j}$ for all i and j.

   (b) Transform all the values to the interval $[0, 1]$ by setting
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$
3. (a) Compute $\hat{q}_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$ for all i and j.

   (b) Transform all the values to the interval $[0, 1]$ by setting
   $$q_{i,j} = \frac{\hat{q}_{i,j}}{\max_{k,l} \hat{q}_{k,l}}.$$

Again, we may end up with values in the range $[0, \sqrt{3}]$ so we have to normalise like we did in the second case. The result is shown in Figure 10.5(c).

Let us sum this up as an algorithm.

This can be implemented by using most of the code from the previous example, and replacing with the lines

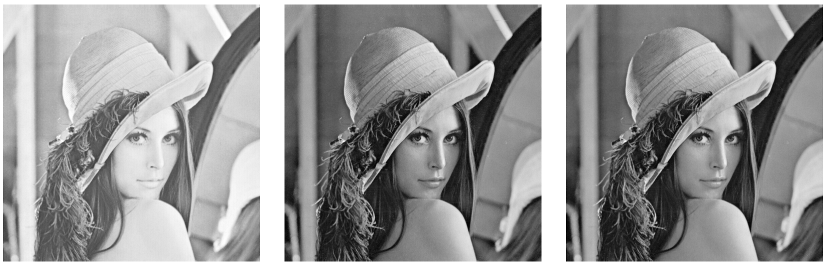
```
newimg1=max(img,[],3);
newvals=img(:,:,1)+img(:,:,2)+img(:,:,3);
newimg2=newvals/max(max(newvals))*255;
newvals=sqrt(img(:,:,1).^2+img(:,:,2).^2+img(:,:,3).^2);
newimg3=newvals/max(max(newvals))*255;
```

respectively. In practice one of the last two methods are usually preferred, perhaps with a preference for the last method, but the actual choice depends on the application. These resuliting images are visualised as grey-level images in Figure 10.4.

♣

Example 10.12 (Computing the negative image)**.** In film-based photography a negative image was obtained when the film was developed, and then a positive image was created from the negative. We can easily simulate this and compute a negative digital image.

Suppose we have a grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ with intensity values in the interval $[0, 1]$. Here intensity value 0 corresponds to black and 1 corresponds to white. To obtain the negative image we just have to replace an intensity p by its 'mirror value' $1 - p$.

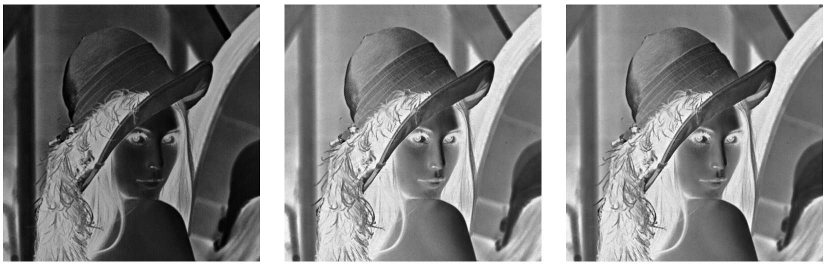


(a) Each colour triple has been replaced by its maximum

(b) Each colour triple has been replaced by its sum and the result mapped to $(0, 1)$

(c) Each triple has been replaced by its length and the result mapped to $(0, 1)$

Figure 10.5: Alternative ways to convert the colour image in Figure 10.1 to a grey level image.



(a)

(b)

(c)

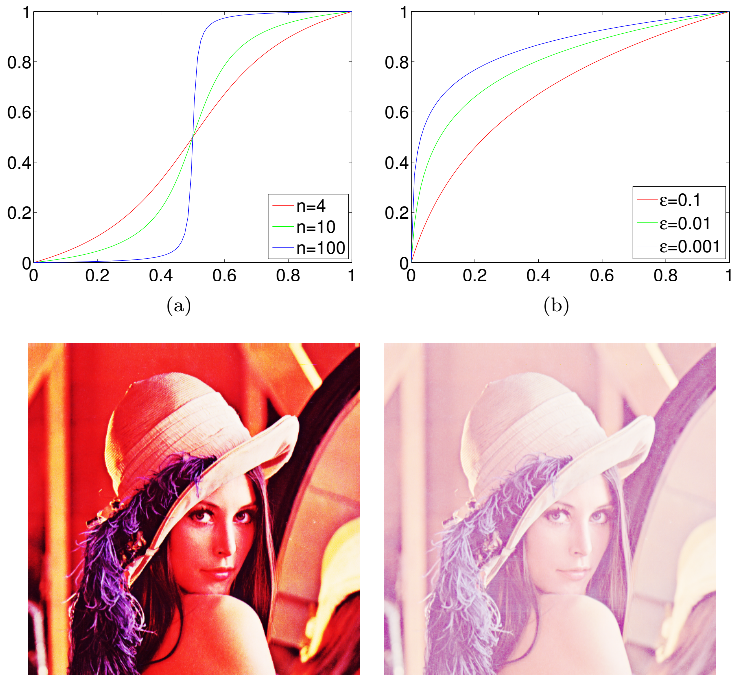
Figure 10.6: The negative versions of the corresponding images in figure 10.5.

Fact 10.13 (Negative image)**.** Suppose the grey-level image $P = (p_{i,j})_{i,j=1}^{m,n}$ is given, with intensity values in the interval $[0, 1]$. The negative image $Q = (q_{i,j})_{i,j=1}^{m,n}$ has intensity values given by $q_{i,j} = 1 - p_{i,j}$ for all i and j.

This is also easily translated to code as above. The resulting image is shown in Figure 10.6. ♣

Example 10.14 (Increasing the contrast)**.** A common problem with images is that the contrast often is not good enough. This typically means that a large proportion of the grey values are concentrated in a rather small subinterval of $[0, 1]$. The obvious solution to this problem is to somehow spread out the values. This can be accomplished by applying a function f to the intensity values, i.e., new intensity values are computed by the formula

$$\hat{p}_{i,j} = f(p_{i,j})$$



(a)

(b)

(c) The middle function in (a) has been applied to the intensity values of the image in figure 10.5

(d) The middle function in (b) has been applied to the same image

Figure 10.7: The plots in (a) and (b) show some functions that can be used to improve the contrast of an image.

for all i and j. If we choose f so that its derivative is large in the area where many intensity values are concentrated, we obtain the desired effect.

Figure 10.7 shows some examples. The functions in the left plot have quite large derivatives near $x = 0.5$ and will therefore increase the contrast in images with a concentration of intensities with value around 0.5. The functions are all on the form

$$f_n(x) = \frac{\arctan\bigl(n(x-1/2)\bigr)}{2\arctan(n/2)} + \frac{1}{2}. \tag{10.1}$$

For any $n \neq 0$ these functions satisfy the conditions $f_n(0) = 0$ and $f_n(1) = 1$. The three functions in figure 10.7(a) correspond to $n = 4$, 10, and 100.

Functions of the kind shown in figure 10.7(b) have a large derivative near $x = 0$ and will therefore increase the contrast in an image with a large proportion of small intensity values, i.e., very dark images. The functions are given by

$$g_\epsilon(x) = \frac{\ln(x+\epsilon) - \ln\epsilon}{\ln(1+\epsilon) - \ln\epsilon}, \tag{10.2}$$

and the ones shown in the plot correspond to $\epsilon = 0.1$, 0.01, and 0.001.

In figure 10.7c the middle function in (a) has been applied to the image in figure 10.5c. Since the image was quite well balanced, this has made the dark areas too dark and the bright areas too bright. In figure 10.7d the function in (b) has been applied to the same image. This has made the image as a whole too bright, but has brought out the details of the road which was very dark in the original.

Observation 10.15. Suppose a large proportion of the intensity values $p_{i,j}$ of a grey-level image P lie in a subinterval I of $[0,1]$. Then the contrast of the image can be improved by computing new intensities $\hat{p}_{i,j} = f(p_{,j})$ where f is a function with a large derivative in the interval I.

Increasing the contrast is easy to implement. The following function has been used to generate the image in Figure 10.7(d). The function takes ϵ in Equation (10.2) as parameter:

```
function newimg=contrastadjust(img,epsilon)
  newimg = img/255; % Maps the pixel values to [0,1]
  newimg = (log(newimg+epsilon) - log(epsilon))/...
           (log(1+epsilon)-log(epsilon));
  newimg = newimg*255; % Maps the values back to [0,255]
```

We will see more examples of how the contrast in an image can be enhanced when we try to detect edges below. ♣

The next examples fall within the same category of operations, those given by computational molecules, defined in the following way:

Definition 10.16. We say that an operation T on an image X is given by the computational molecule

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & a_{-1,-1} & a_{-1,0} & a_{-1,1} & \cdots \\ \cdots & a_{0,-1} & \underline{a_{0,0}} & a_{0,1} & \cdots \\ \cdots & a_{1,-1} & a_{1,0} & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

(where we underline the element with index $(0,0)$, and where only the nonzero $a_{i,j}$ are listed, similarly to how we do this with compact filter notation). if we have that

$$(TX)_{i,j} = \sum_{k_1,k_2} a_{k_1,k_2} X_{i+k_1,j+k_2} \tag{10.3}$$

Here the indices are allowed to be both positive and negative.

The interpretation of a computation molecule is that we place the centre of the molecule on a pixel, multiply the pixel and its neighbours by the corresponding weights $a_{i,j}$, and finally summing up in order to produce the new value.

Example 10.17 (Smoothing an image)**.** When we considered filtering of digital sound, we observed that replacing each sample of a sound by an average of the sample and its neighbours dampened the high frequencies of the sound. We can do a similar operation on images.

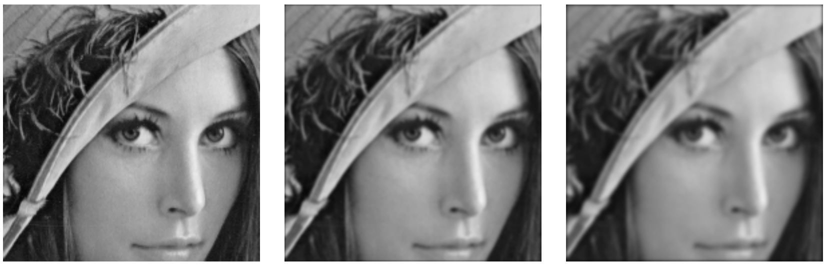
Consider the computational molecule

$$A = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{10.4}$$

Applying this computational molecule to an image corresponds to smoothing the image. More precisely, we would compute the new pixels by

$$\begin{aligned}\hat{p}_{i,j} = \frac{1}{16}\big(4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) \\ + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}\big).\end{aligned}$$

Since the weights sum to one, the new intensity value $\hat{p}_{i,j}$ is a weighted average of the intensity values on the right. As in the section on sound, we could have used equal weights for all pixels, but it seems reasonable that the weight of a pixel should be larger the closer it is to the centre pixel. For the onedimensional case on sound, we used the values of Pascal's triangle here, since these weights are known to give a very good smoothing effect. We will return to how we can generalize the use of Pascal's triangle to obtain computational molecules for use in images.



(a) (b) (c)

Figure 10.8: The images in (b) and (c) show the effect of smoothing the image in (a).

A larger filter is given by the array

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}. \tag{10.5}$$

These numbers are taken from row six of Pascal's triangle. More precisely, the value in row k and column l is given by the product $\binom{6}{k}\binom{6}{l}$. The scaling $1/4096$ comes from the fact that the sum of all the numbers in the table is $2^{6+6} = 4096$.

The result of applying the two filters in (10.4) and (10.5) to our greyscale-image is shown in Figure 10.8(b) and -(c) respectively. To make the smoothing effect visible, we have zoomed in on the face in the image. The smoothing effect is most visible in the second image, since this molecule is bigger.

Observation 10.18. An image P can be smoothed out by replacing the intensity value at each pixel by a weighted average of the intensity at the pixel and the intensity of its neighbours.

It is straightforward to write a function which performs smoothing. Assume that the image is stored as the matrix `img`, and the computational molecule is stored as the matrix `compmolecule`. The following function will return the smoothed image:

```
function newimg=smooth(img,compmolecule)
  [m,n]=size(img);
  [k,k1] = size(compmolecule); % We need k==k1, and odd
  sc = (k+1)/2;
  for m1=1:m
```

```
    for n1=1:n
      slidingwdw = zeros(k,k);
        % slidingwdw is the part of the picture which
        % compmolecule is applied to pixel (m1,n1)
      slidingwdw(max(sc+1-m1,1):min(sc+m-m1,2*sc-1) , ...
                 max(sc+1-n1,1):min(sc+n-n1,2*sc-1)) = ...
      img(max(1,m1-(sc-1)):min(m,m1+(sc-1)) , ...
          max(1,n1-(sc-1)):min(n,n1+(sc-1)));
      newimg(m1,n1) = sum(sum(compmolecule .* slidingwdw));
    end
  end
```

What makes this code difficult to write is the fact that the computational molecule may extend outside the borders of the image, when we are close to these borders. With this function, the first smoothing above can be performed by writing

```
smooth(img,(1/16)*[ 1 2 1; 2 4 2; 1 2 1]);
```

♣

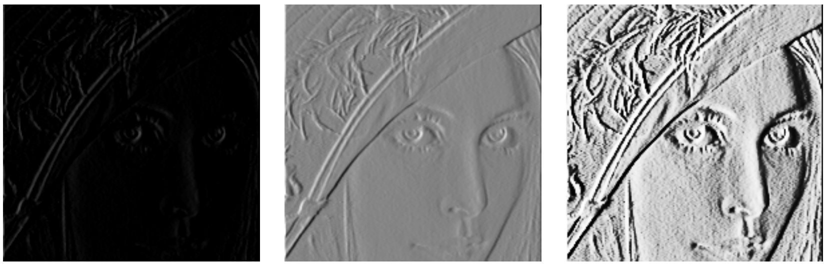
Example 10.19 (Detecting edges)**.** The final operation on images we are going to consider is edge detection. An edge in an image is characterised by a large change in intensity values over a small distance in the image. For a continuous function this corresponds to a large derivative. An image is only defined at isolated points, so we cannot compute derivatives, but we have a perfect situation for applying numerical differentiation. Since a grey-level image is a scalar function of two variables, numerical differentiation techniques can be applied.

Partial derivative in x-direction. Let us first consider computation of the partial derivative $\partial P/\partial x$ at all points in the image. Note first that it is the second coordinate in an image which refers to the x-direction we are used to from plotting functions. This means that the familiar symmetric Newton quotient approximation for the partial derivative takes the form

$$\frac{\partial P}{\partial x}(i,j) = \frac{p_{i,j+1} - p_{i,j-1}}{2}, \tag{10.6}$$

where we have used the convention $h = 1$ which means that the derivative is measured in terms of 'intensity per pixel'. We can run through all the pixels in the image and compute this partial derivative, but have to be careful for $j = 1$ and $j = m$ where the formula refers to non-existing pixels. We will adapt the simple convention of assuming that all pixels outside the image have intensity 0. If we apply this to the same excerpt of the Lena image, we obtain figure 10.9(a).

This image is not very helpful since it is almost completely black. The reason for this is that many of the intensities are in fact negative, and these are just displayed as black. More specifically, the intensities turn out to vary in the interval $[-0.424, 0.418]$. We therefore normalise and map all intensities



(a) The partial derivative in the x-direction

(b) The intensities in (a) have been normalised to $(0, 1)$

(c) The contrast has been enhanced with the function f_{50}, equation 10.1

Figure 10.9: Experimenting with the partial derivative for the image in 10.5.

to $[0, 1]$. The result of this is shown in (b). The predominant colour of this image is an average grey, i.e, an intensity of about 0.5. To get more detail in the image we therefore try to increase the contrast by applying the function f_{50} in equation 10.1 to each intensity value. The result is shown in figure 10.9(c) which does indeed show more detail.

It is important to understand the colours in these images. We have computed the derivative in the x-direction, and we recall that the computed values varied in the interval $[-0.424, 0.418]$. The negative value corresponds to the largest average decrease in intensity from a pixel $p_{i-1,j}$ to a pixel $p_{i+1,j}$. The positive value on the other hand corresponds to the largest average increase in intensity. A value of 0 in figure 10.9a corresponds to no change in intensity between the two pixels.

When the values are mapped to the interval $[0, 1]$ in figure 10.9b, the small values are mapped to something close to 0 (almost black), the maximal values are mapped to something close to 1 (almost white), and the values near 0 are mapped to something close to 0.5 (grey). In figure 10.9c these values have just been emphasised even more.

Figure 10.9c tells us that in large parts of the image there is very little variation in the intensity. However, there are some small areas where the intensity changes quite abruptly, and if you look carefully you will notice that in these areas there is typically both black and white pixels close together, like down the vertical front corner of the bus. This will happen when there is a stripe of bright or dark pixels that cut through an area of otherwise quite uniform intensity.

Since we display the derivative as a new image, the denominator is actually not so important as it just corresponds to a constant scaling of all the pixels; when we normalise the intensities to the interval $[0, 1]$ this factor cancels out.

We sum up the computation of the partial derivative by giving its computational molecule.

Observation 10.20. Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial x$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ -1 & \underline{0} & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{10.7}$$

As we remarked above, the factor 1/2 can usually be ignored. We have included the two rows of 0s just to make it clear how the computational molecule is to be interpreted; it is obviously not necessary to multiply by 0.

Partial derivative in y-direction. The partial derivative $\partial P/\partial y$ can be computed analogously to $\partial P/\partial x$. Note that the positive direction of this axis in an image is opposite to the direction of the y-axis we use when plotting functions.

Observation 10.21. Let $P = (p_{i,j})_{i,j=1}^{m,n}$ be a given image. The partial derivative $\partial P/\partial y$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & -1 & 0 \\ 0 & \underline{0} & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{10.8}$$

The result is shown in figure 10.11(b). The intensities have been normalised and the contrast enhanced by the function f_{50} in (10.1).

The gradient. The gradient of a scalar function is often used as a measure of the size of the first derivative. The gradient is defined by the vector

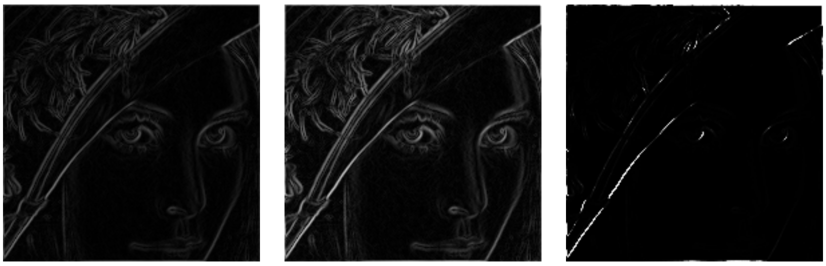
$$\nabla P = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y}\right),$$

so its length is given by

$$|\nabla P| = \sqrt{\left(\frac{\partial P}{\partial x}\right)^2 + \left(\frac{\partial P}{\partial y}\right)^2}.$$

When the two first derivatives have been computed it is a simple matter to compute the gradient vector and its length; the resulting is shown as an image in figure 10.10c.

The image of the gradient looks quite different from the images of the two partial derivatives. The reason is that the numbers that represent the length of the gradient are (square roots of) sums of squares of numbers. This means that

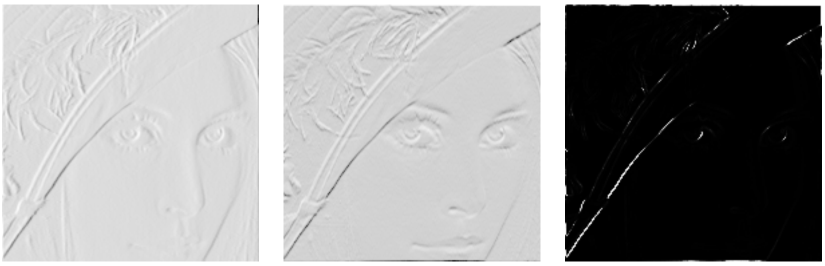


(a) The computed gradient

(b) The intensities in (a) have been normalised to $(0, 1)$

(c) The contrast has been enhanced with the function f_{50}

Figure 10.10: Computing the gradient.



(a) The first-order partial derivatives in the x-direction

(b) The first-order partial derivatives in the y-direction

(c) The length of the gradient

Figure 10.11: In all images, the computed numbers have been normalised and the contrast enhanced.

the parts of the image that have virtually constant intensity (partial derivatives close to 0) are coloured black. In the images of the partial derivatives these values ended up in the middle of the range of intensity values, with a final colour of grey, since there were both positive and negative values.

Figure 10.10a shows the computed values of the gradient. Although it is possible that the length of the gradient could become larger than 1, the maximum value in this case is about 0.876. By normalising the intensities we therefore increase the contrast slightly and obtain the image in figure 10.10b.

To enhance the contrast further we have to do something different from what was done in the other images since we now have a large number of intensities near 0. The solution is to apply a function like the ones shown in figure 10.7(b) to the intensities. If we use the function $g_{0.01}$ defined in equation(10.2) we obtain the image in figure 10.10(c).

♣

10.2.2 Comparing the first derivatives

Figure 10.11 shows the two first-order partial derivatives and the gradient. If we compare the two partial derivatives we see that the x-derivative seems to emphasise vertical edges while the y-derivative seems to emphasise horizontal edges. This is precisely what we must expect. The x-derivative is large when the difference between neighbouring pixels in the x-direction is large, which is the case across a vertical edge. The y-derivative enhances horizontal edges for a similar reason.

The gradient contains information about both derivatives and therefore emphasises edges in all directions. It also gives a simpler image since the sign of the derivatives has been removed.

10.2.3 Second-order derivatives

To compute the three second order derivatives we apply the corresponding computational molecules which we already have described.

Observation 10.22 (Second order derivatives of an image)**.** The second order derivatives of an image P can be computed by applying the computational molecules

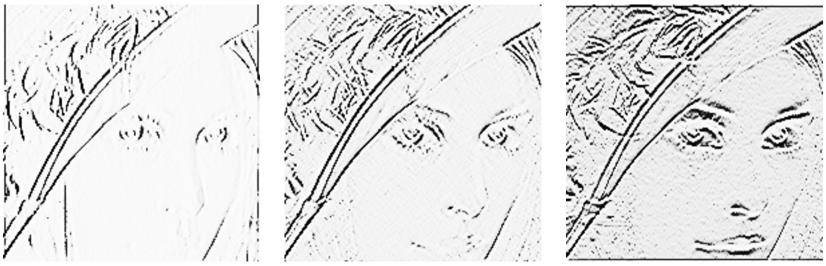
$$\frac{\partial^2 P}{\partial x^2}: \qquad \frac{1}{4}\begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{-2} & 1 \\ 0 & 0 & 0 \end{pmatrix}, \tag{10.9}$$

$$\frac{\partial^2 P}{\partial y \partial x}: \qquad \frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 0 & \underline{0} & 0 \\ -1 & 0 & 1 \end{pmatrix}, \tag{10.10}$$

$$\frac{\partial^2 P}{\partial y^2}: \qquad \frac{1}{4}\begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{-2} & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{10.11}$$

With the information in observation 10.22 it is quite easy to compute the second-order derivatives, and the results are shown in figure 10.12. The computed derivatives were first normalised and then the contrast enhanced with the function f_{100} in each image, see equation 10.1.

As for the first derivatives, the xx-derivative seems to emphasise vertical edges and the yy-derivative horizontal edges. However, we also see that the second derivatives are more sensitive to noise in the image (the areas of grey are less uniform). The mixed derivative behaves a bit differently from the other two, and not surprisingly it seems to pick up both horizontal and vertical edges.



(a) The second-order partial derivatives in the x-direction

(b) The second-order partial derivatives in the xy-direction

(c) The second-order partial derivatives in the y-direction

Figure 10.12: In all images, the computed numbers have been normalised and the contrast enhanced.

Exercises for Section 10.2

1. Black and white images can be generated from greyscale images (with values between 0 and 255) by replacing each pixel value with the one of 0 and 255 which is closest. Use this strategy to generate the black and white image shown in Figure 10.1(a).

2. Generate the image in Figure 10.7(d) on your own by writing code which uses the function `contrastadjust`.

3. Let us also consider the second way we mentioned for increasing the contrast.

a. Write a function `contrastadjust0` which instead uses the function 10.1 to increase the contrast. n should be a parameter to the function.

b. Generate the image in Figure 10.7(c) on your own by using your code from Exercise 2, and instead calling the function `contrastadjust0`.

4. In this exercise we will look at another function for increasing the contrast of a picture.

a. Show that the function $f : \mathbb{R} \to \mathbb{R}$ given by

$$f_n(x) = x^n,$$

for all n maps the interval $[0, 1] \to [0, 1]$, and that $f'(1) \to \infty$ as $n \to \infty$.

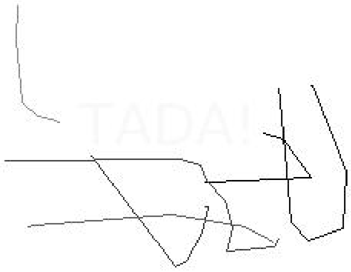


Figure 10.13: Secret message

b. The color image `secret.jpg`,shown in Figure 10.13, contains some information that is nearly invisible to the naked eye on most computer monitors. Use the function $f(x)$, to reveal the secret message.
Hint: You will first need to convert the image to a greyscale image. You can then use the function `contrastadjust` as a starting point for your own program.

5. Generate the image in Figure 10.8(b) and -(c) by writing code which calls the function `smooth` with the appropriate computational molecules.

6. Generate the image in Figure 10.9(c) by writing code in the same way. Also generate the images in figures 10.10, 10.11, and 10.12.

10.3 Adaptations to image processing

There are two particular image standards we will consider in these notes. The first is the JPEG standard. JPEG is short for *Joint Photographic Experts Group*, and is an image format that was approved as an international standard in 1994. JPEG is usually lossy, but may also be lossless and has become a popular format for image representation on the Internet. The standard defines both the algorithms for encoding and decoding and the storage format. JPEG performs a DCT on the image, and neglects DCT-coefficients which are below a given threshold. We will describe this in the next chapter. JPEG codes the remaining DCT-coefficients by a variation of Huffman coding, but it may also use arithmetic coding. The compression level in JPEG images is selected by the user and may result in conspicuous artefacts if set too high. JPEG is especially prone to artefacts in areas where the intensity changes quickly from pixel to pixel. The extension of a JPEG-file is `.jpg` or `.jpeg`.

The second standard we will consider is JPEG2000. It was developed to address some of the shortcomings of JPEG, and is based on wavelets.

There are several extensions, additions and modifications to the theory we will present in the later chapters, which are needed in order for us to have a full image compression system: the wavelet transform, the DCT, and the DFT were addressed because these are linked to relevant mathematics, and because they are important ingredients in many standards for sound and images. In this section we will mention many other extensions which also are important.

10.3.1 Lossless coding

Somewhere in the image processing or sound processing pipeline we need a step which actually achieves compression of the data. We have not mentioned anything about this step, since the output from the wavelet transform or the DCT is simply another set of data of the same size, called the *transformed data*. What we need to do is to apply a coding algorithm to this data to achieve compression. This may be Huffman coding, arithmetic coding, or any other algorithm. These methods are applied to the transformed data, since the effect of the wavelet transform is that it exploits the data so that it can be represented with data with lower entropy, so that it can be compressed more efficiently with these techniques.

10.3.2 Quantization

Coding as we have learnt previously is a lossless operation. As we saw, for certain wavelets the transform can also be performed in a lossless manner. These wavelets are, however, quite restrictive, which is why there is some loss involved with most wavelets used in practical applications. When there is some loss inherent in the transform, a quantization of the transformed data is also performed before the coding takes place. This quantization is typically done with a fixed number of bits, but may also be more advanced.

10.3.3 Preprocessing

Image compression as performed for certain image standards also often preprocess the pixel values before any transform is applied. The preprocessing may be centering the pixel values around a certain value, or extracting the different image components before they are processed separately.

10.3.4 Tiles, blocks, and error resilience

We have presented the wavelet transform as something which transforms the entire image. In practice this is not the case. The image is very often split into smaller parts, often called tiles. The tiles in an image are processed independently, so that errors whih occur within one tile do not affect the appearance of parts in the image which correspond to other tiles. This makes the image compression what we call *error-resilient*, to errors such as transmission errors. The second reason for splitting into tiles has to do with that it may be more efficient to perform many transforms on smaller parts, rather than one big transform for the entire image. Performing one big transform would force us to have a big part of the image in memory during each computation, as well as remove possibilities for parallel computing. Often the algorithm itself also requires more computations when the size is bigger. For some standards, tiles are split into even smaller parts, called blocks, where parts of the processing within each block also is performed independently. This makes the possibilities for parallel computing even bigger. As an example, we mentioned that the JPEG standard performs the two-dimensional DCT on blocks as small as size 8×8.

10.3.5 Metadata

An image standard also defines how to store metadata about an image, and what metadata is accepted, like resolution, time when the image was taken, or where the image was taken (GPS coordinates) and similar information. Metadata can also tell us how the colour in the image are represented. As we have already seen, in most colour images the colour of a pixel is represented in terms of the amount of red, green and blue or (r, g, b). But there are other possibilities as well: Instead of storing all 24 bits of colour information in cases where each of the three colour components needs 8 bits, it is common to create a table of 256 colours with which a given image could be represented quite well. Instead of storing the 24 bits, one then just stores a colour table in the metadata, and at each pixel, the eight bits corresponding to the correct entry in the table. This is usually referred to as eight-bit colour and the table is called a *look-up* table or *palette*. For large photographs, however, 256 colours is far from sufficient to obtain reasonable colour reproduction.

Metadata is usually stored in the beginning of the file, formatted in a very specific way.

10.4 Summary

We started by discussing the basic question what an image is, and took a closer look at digital images. We then went through several operations which give meaning for digital images, and showed how to implement these.

In introductory image processing textbooks, many other techniques for processing images are presented. We limited the techniques prrsented here, since our interest in images is mainly for compression purposes. The tensor product construction, which will be introduced in the next chapter, will help us in this direction.

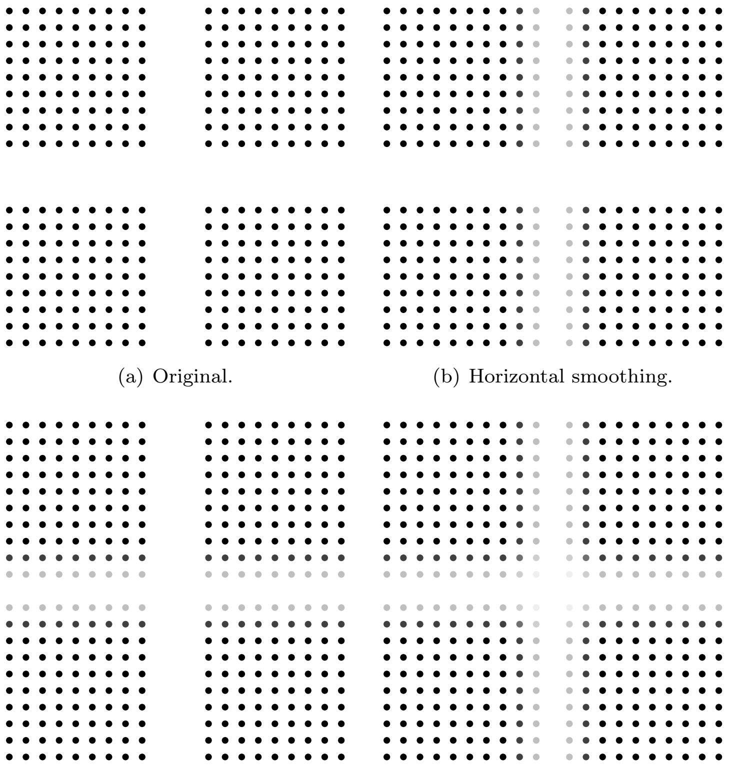
Chapter 11

Definition and properties of tensor products

The DFT, the DCT, and the wavelet transform were all defined as changes of coordinates for vectors or functions of one variable and therefore cannot be directly applied to higher dimensional data like images. In this chapter we will introduce a simple recipe for extending such one-dimensional schemes to two (and higher) dimensions. The basic ingredient is the *tensor product* construction. This is a general tool for constructing two-dimensional functions and filters from one-dimensional counterparts. This will allow us to generalise the filtering and compression techniques for audio to images, and we will also recognise some of the basic operations for images introduced in Chapter 10 as tensor product constructions.

A two-dimensional discrete function on a rectangular domain, like for example an image, is conveniently represented in terms of a matrix X with elements $X_{i,j}$, and with indices in the ranges $0 \leq i \leq M-1$ and $0 \leq j \leq N-1$. One way to apply filters to X would be to rearrange the matrix into a long vector, column by column. We could then apply a one-dimensional filter to this vector and then split the resulting vector into columns that can be reassembled back into a matrix again. This approach may have some undesirable effects near the boundaries between columns. In addition, the resulting computations may be rather ineffective. Consider for example the case where X is an $N \times N$ matrix so that the long vector has length N^2. Then a linear transformation applied to X involves multiplication with an $N^2 \times N^2$-matrix. Each such matrix multiplication may require as many as N^4 multiplications which is substantial when N is large.

The concept of tensor products can be used to address these problems. Using tensor products, one can construct operations on two-dimensional functions which inherit properties of one-dimensional operations. Tensor products also turn out to be computationally efficient.



(a) Original.

(b) Horizontal smoothing.

(c) Vertical smoothing.

(d) Horizontal and vertical smoothing.

Figure 11.1: The results of smoothing an image with the filter $\{1/4, \underline{1/2}, 1/4\}$ horizontally, vertically, and both. The pixels are shown as disks with intensity corresponding to the pixel values.

11.1 The tensor product of vectors

In Chapter 10 we saw examples of several filters applied to images. The filters of special interest to us now are those that determined a new image by combining neighbouring pixels, like the smoothing filter in Example 10.17 and the edge detection filter in Example 10.19. Our aim now is to try and apply filters like these as a repeated application of one-dimensional filters rather than using a computational molecule like in Chapter 10. It will be instructive to do this with an example before we describe the general construction, so let us revisit Example 10.17.

Figure 11.1 (a) shows an example of a simple image. We want to smooth this image X with the one-dimensional filter T given by $y_n = (T(\boldsymbol{x}))_n = (x_{n-1} + 2x_n + x_{n+1})/4$, or equivalently $T = \{1/4, \underline{1/2}, 1/4\}$. There are two obvious one-dimensional operations we can do:

1. Apply the filter to each row in the image.
2. Apply the filter to each column in the image.

The problem is of course that these two operations will only smooth the image either horizontally or vertically as can be seen in the image in (b) and (c) of Figure 11.1.

So what else can we do? We can of course first smooth all the rows of the image and then smooth the columns of the resulting image. The result of this is shown in Figure 11.1 (d). Note that we could have performed the operations in the opposite order: first vertical smoothing and then horizontal smoothing, and currently we do not know if this is the same. We will show that these things actually are the same, and that computational molecules, as we saw in Chapter 10, naturally describe operations which are performed both vertically and horizontally. The main ingredient in this will be the tensor product construction. We start by defining the tensor product of two vectors.

Definition 11.1 (Tensor product of vectors)**.** If $\boldsymbol{x}, \boldsymbol{y}$ are vectors of length M and N, respectively, their tensor product $\boldsymbol{x} \otimes \boldsymbol{y}$ is defined as the $M \times N$-matrix defined by $(\boldsymbol{x} \otimes \boldsymbol{y})_{ij} = x_i y_j$. In other words, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.

In particular $\boldsymbol{x} \otimes \boldsymbol{y}$ is a matrix of rank 1, which means that most matrices cannot be written as tensor products. The special case $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$ is the matrix which is 1 at (i, j) and 0 elsewhere, and the set of all such matrices forms a basis for the set of $M \times N$-matrices.

Observation 11.2 (Interpretation of tensor products for vectors)**.** Let

$$\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1} \quad \text{and} \quad \mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$$

be the standard bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. Then

$$\mathcal{E}_{M,N} = \{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

is a basis for $L_{M,N}(\mathbb{R})$, the set of $M \times N$-matrices. This basis is often referred to as the standard basis for $L_{M,N}(\mathbb{R})$.

An image can simply be thought of as a matrix in $L_{M,N}(\mathbb{R})$. With this definition of tensor products, we can define operations on images by extending the one-dimensional filtering operations defined earlier.

Definition 11.3 (Tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, we define the linear mapping $S \otimes T : L_{M,N}(\mathbb{R}) \to L_{M,N}(\mathbb{R})$ by linear extension of $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. The linear mapping $S \otimes T$ is called the tensor product of the matrices S and T.

A couple of remarks are in order. First, from linear algebra we know that, when T is linear mapping from V and $T(\boldsymbol{v}_i)$ is known for a basis $\{\boldsymbol{v}_i\}_i$ of V, T is uniquely determined. In particular, since the $\{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{i,j}$ form a basis, there exists a unique linear transformation $S \otimes T$ so that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)$. This unique linear transformation is what we call the linear extension from the values in the given basis.

Secondly $S \otimes T$ also satisfies $(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) = (S\boldsymbol{x}) \otimes (T\boldsymbol{y})$. This follows from

$$\begin{aligned}
(S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}) &= (S \otimes T)((\sum_i x_i \boldsymbol{e}_i) \otimes (\sum_j y_j \boldsymbol{e}_j)) = (S \otimes T)(\sum_{i,j} x_i y_j (\boldsymbol{e}_i \otimes \boldsymbol{e}_j)) \\
&= \sum_{i,j} x_i y_j (S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = \sum_{i,j} x_i y_j (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j) \\
&= \sum_{i,j} x_i y_j S\boldsymbol{e}_i((T\boldsymbol{e}_j))^T = S(\sum_i x_i \boldsymbol{e}_i)(T(\sum_j y_j \boldsymbol{e}_j))^T \\
&= S\boldsymbol{x}(T\boldsymbol{y})^T = (S\boldsymbol{x}) \otimes (T\boldsymbol{y}).
\end{aligned}$$

Here we used the result from Exercise 6. Linear extension is necessary anyway, since only rank 1 matrices have the form $\boldsymbol{x} \otimes \boldsymbol{y}$.

Example 11.4 (Smoothing an image)**.** Assume that S and T are both filters, and that $S = T = \{1/4, \underline{1/2}, 1/4\}$. Let us set $M = 5$ and $N = 7$, and let us compute $(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3)$. We have that

$$(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3) = (S\boldsymbol{e}_2)(T\boldsymbol{e}_3)^T = (\mathrm{col}_2 S)(\mathrm{col}_3 T)^T.$$

Since

$$S = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 1 & 2 \end{pmatrix} \qquad T = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix},$$

we get that $(S \otimes T)(\boldsymbol{e}_2 \otimes \boldsymbol{e}_3)$ is

$$\frac{1}{4}\begin{pmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \end{pmatrix} = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 4 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We recognize here the computational molecule from Example 10.17 for smoothing an image. More generally it is not hard to see that $(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)$ is the matrix where the same computational molecule is placed with its centre at

(i,j). Clearly then, the linear extension $S \otimes T$ is obtained by placing the computational molecule over all indices, multiplying by the value at that index, and summing everything together. This is equivalent to the procedure for smoothing we learnt in Example 10.17. ♣

As hinted in the previous example, there is a close connection between tensor products of filters and computational molecules. This is the content of the next theorem.

Theorem 11.5. If S and T are filters with compact filter notation $\boldsymbol{s}$ and $\boldsymbol{t}$, respectively, then $S \otimes T$ is a mapping with computational molecule $\text{rev}(\boldsymbol{s}) \otimes \text{rev}(\boldsymbol{t})$, where $\text{rev}(\boldsymbol{s})$ denotes the vector with the order of the elements reversed.

Proof. We have that

$$\begin{aligned}
&((S \otimes T)(\boldsymbol{x} \otimes \boldsymbol{y}))_{i,j} \\
&= ((S\boldsymbol{x}) \otimes (T\boldsymbol{y}))_{i,j} = (S\boldsymbol{x})(T\boldsymbol{y})^T)_{i,j} = (S\boldsymbol{x})_i (T\boldsymbol{y})_j \\
&= \left(\sum_k s_k x_{i-k}\right)\left(\sum_k t_k y_{j-k}\right) \\
&= \sum_{k_1,k_2} x_{i-k_1} s_{k_1} t_{k_2} y_{j-k_2} = \sum_{k_1,k_2} x_{i+k_1} s_{-k_1} t_{-k_2} y_{j+k_2} \\
&= \sum_{k_1,k_2} x_{i+k_1} (\text{rev}(\boldsymbol{s}))_{k_1} (\text{rev}(\boldsymbol{t}))_{k_2} y_{j+k_2} \\
&= \sum_{k_1,k_2} x_{i+k_1} (\text{rev}(\boldsymbol{s}) \otimes \text{rev}(\boldsymbol{t}))_{k_1,k_2} y_{j+k_2} \\
&= \sum_{k_1,k_2} (\text{rev}(\boldsymbol{s}) \otimes \text{rev}(\boldsymbol{t}))_{k_1,k_2} (\boldsymbol{x} \otimes \boldsymbol{y}))_{i+k_1,j+k_2}.
\end{aligned}$$

By linear extension we have also that

$$((S \otimes T)X)_{i,j} = \sum_{k_1,k_2} (\text{rev}(\boldsymbol{s}) \otimes \text{rev}(\boldsymbol{t}))_{k_1,k_2} X_{i+k_1,j+k_2}$$

for all images X, so that $S \otimes T$ is a mapping with computational molecule $\text{rev}(\boldsymbol{s}) \otimes \text{rev}(\boldsymbol{t})$. □

We have not formally defined the tensor product of compact filter notations. This is a straightforward extension of the usual tensor product of vectors, where we additionally mark the element at index $(0,0)$.

In the previous theorem, note that the filter coefficients need to be reversed when we compute the computational molecule. This may seem strange. The difference lies in that we define filtering differently, compared to how we define applications of computational molecules: a filter is "placed" over the samples in reverse order, contrary to a computational molecule. To be more precise, this has to do with the difference of sign in the equations

$$\begin{aligned}(Sx)_n &= \sum_k s_k x_{n-k}\\ (TX)_{i,j} &= \sum_{k_1,k_2} a_{k_1,k_2} X_{i+k_1,j+k_2}\end{aligned}$$

in how we define applying a filter and a computational molecule, respectively. In most of the examples here the filters are symmetric, and then this difference causes no confusion.

Example 11.6. Returning to Example 11.4, using the previous result we find that the computational molecule of $S \otimes T$ is

$$\boldsymbol{s} \otimes \boldsymbol{t} = \begin{pmatrix} 1/4 \\ \underline{1/2} \\ 1/4 \end{pmatrix} \begin{pmatrix} 1/4 & \underline{1/2} & 1/4 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

This again confirms that the computational molecule given by Equation 10.4 in Example 10.17 is the tensor product of the filter $\{1/4, \underline{1/2}, 1/4\}$ with itself. ♣

While we have seen that the computational molecules from Chapter 10 can be written as tensor products, not all computational molecules can be written as tensor products: we need of course that the molecule is a rank 1 matrix, since matrices which can be written as a tensor product always have rank 1.

The tensor product can be expressed explicitly in terms of matrix products.

Theorem 11.7. If $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, the action of their tensor product on a matrix X is given by $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$.

Proof. We have that

$$\begin{aligned}(S \otimes T)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) &= (S\boldsymbol{e}_i) \otimes (T\boldsymbol{e}_j)\\ &= (\mathrm{col}_i(S)) \otimes (\mathrm{col}_j(T)) = \mathrm{col}_i(S)(\mathrm{col}_j(T))^T\\ &= \mathrm{col}_i(S)\mathrm{row}_j(T^T) = S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)T^T.\end{aligned}$$

This means that $(S \otimes T)X = SXT^T$ for any $X \in L_{M,N}(\mathbb{R})$, since equality holds on the basis vectors $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$. □

This leads to the following implementation for the tensor product of matrices:

Theorem 11.8 (Implementation of a tensor product of matrices)**.** If $S : \mathbb{R}^M \to \mathbb{R}^M$, $T : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, and $X \in L_{M,N}(\mathbb{R})$, we have that $(S \otimes T)X$ can be computed as follows:

1. Apply S to every column of X.
2. Transpose the resulting matrix.
3. Apply T to every column in the resulting matrix.
4. Transpose the resulting matrix.

This recipe for computing $(S \otimes T)X$ is perhaps best seen if we write

$$(S \otimes T)X = SXT^T = (T(SX)^T)^T. \tag{11.1}$$

In the first step above we compute SX, in the second step $(SX)^T$, in the third step $T(SX)^T$, in the fourth step $(T(SX)^T)^T$. The reason for writing the tensor product this way, as an operation column by column, has to do with with that S and T are mostly filters for our purposes, and that we want to reuse efficient implementations instead of performing full matrix multiplications, just as we decided to express a wavelet transformation in terms of filters. The reason for using columns instead of rows has to do with that we have expressed filtering as a matrix by column multiplication. Note that this choice of using columns instead of rows should be influenced by how the computer actually stores values in a matrix. If these values are stored column by column, performing operations columnwise may be a good idea, since then the values from the matrix are read in the same order as they are stored. If matrix values are stored row by row, it may be a good idea to rewrite the procedure above so that operations are performed row by row also (see Exercise 8).

Theorem 11.8 leads to the following algorithm for computing the tensor product of matrices:

```
[M,N]=size(X);
for col=1:N
  X(:,col)=S*X(:,col);
end
X=X'
for col=1:M
  X(:,col)=T*X(:,col);
end
X=X';
```

This algorithm replaces the rows and columns in X at each step. In the following, $S = T$ in most cases. In this case we can replace with the following algorithm, which is even simpler:

```
for k=1:2
  for col=1:size(X,2)
    X(:,col)=S*X(:,col);
  end
  X=X';
end
```

Here `size(X,2)` returns the number of columns of `X` (similarly, `size(X,1)` returns the number of rows of `X`). In an efficient algorithm, we would of course replace the matrix multiplications with S and T with efficient implementations.

If we want to apply a sequence of tensor products of filters to a matrix, the order of the operations does not matter. This will follow from the next result:

Corollary 11.9. If $S_1 \otimes T_1$ and $S_2 \otimes T_2$ are two tensor products of one dimensional filters, then $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1 S_2) \otimes (T_1 T_2)$.

Proof. By Theorem 11.7 we have that

$$(S_1 \otimes T_1)(S_2 \otimes T_2)X = S_1(S_2 X T_2^T)T_1^T = (S_1 S_2)X(T_1 T_2)^T = ((S_1 S_2) \otimes (T_1 T_2))X.$$

for any $X \in L_{M,N}(\mathbb{R})$. This proves the result. □

Suppose that we want to apply the operation $S \otimes T$ to an image. We can write

$$S \otimes T = (S \otimes I)(I \otimes T) = (I \otimes T)(S \otimes I). \tag{11.2}$$

Moreover, from Theorem 11.7 it follows that

$$\begin{aligned}(S \otimes I)X &= SX \\ (I \otimes T)X &= XT^T = (TX^T)^T.\end{aligned}$$

This means that $S \otimes I$ corresponds to applying S to each column in X, and $I \otimes T$ corresponds to applying T to each row in X. When S and T are smoothing filters, this is what we refered to as vertical smoothing and horizontal smoothing, respectively. The relations in Equation (11.2) thus have the following interpretation (alternatively note that the order of left or right multiplication does not matter).

Observation 11.10. The order of vertical and horizontal smoothing does not matter, and any tensor product of filters $S \otimes T$ can be written as a horizontal filtering operation $I \otimes T$ followed by a vertical filtering operation $S \otimes I$.

In fact, the order of any vertical operation $S \otimes I$ and horizontal operation $I \otimes T$ does not matter: it is not required that the operations are filters. For filters we have a stronger result: If S_1, T_1, S_2, T_2 all are filters, we have from Corollary 11.9 that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_2 \otimes T_2)(S_1 \otimes T_1)$, since all filters commute. This does not hold in general since general matrices do not commute.

Example 11.11 (Computing partial derivatives and detecting edges)**.** Consider the bass reducing filter $T = \{1/2, \underline{0}, -1/2\}$, i.e. $(T(\boldsymbol{x}))_n = \frac{1}{2}(x_{n+1} - x_{n-1})$. The computational molecule of the vertical filtering operation $T \otimes I$ is

$$\operatorname{rev}(\boldsymbol{s}) \otimes \operatorname{rev}(\boldsymbol{t}) = \begin{pmatrix} -1/2 \\ \underline{0} \\ 1/2 \end{pmatrix} \begin{pmatrix} \underline{1} \end{pmatrix} = \begin{pmatrix} -1/2 \\ \underline{0} \\ 1/2 \end{pmatrix}.$$

This shows as above that $T \otimes I$ is the transformation where the computational molecule given by Equation 10.8 in Example 10.19 is placed over the image samples. This tensor product thus computes the symmetric Newton quotient approximation to the partial derivative in the y-direction (but note that the y-direction points downwards, as commented in Example 10.19!). Equivalently, it can be used for detecting horizontal edges in images. Similarly, $I \otimes T$ computes an approximation to the partial derivative in the x-direction (see Exercise 2). This can be generalized to second order derivatives, as in Observation 10.22. The computational molecules there can be factored as

$$\begin{aligned}
(I \otimes T)(I \otimes T) &= I \otimes (T^2) \\
(I \otimes T)(T \otimes I) &= T \otimes T \\
(T \otimes I)(T \otimes I) &= (T^2) \otimes I,
\end{aligned}$$

respectively. More generally, the approximation to the partial derivative $\frac{\partial^{k+l} P}{\partial x^k \partial y^l}$ of the function P can be computed by applying the tensor product $(T^l) \otimes (T^k)$ to the sample values of P. ♣

Exercises for Section 11.1

1. Let the filter T be defined by $T = \{\underline{-1}, 1\}$.

a. Let A be a matrix which represents the pixel values in an image. What can you say about how the new images $(T \otimes I)A$ og $(I \otimes T)A$ look? What are the interpretations of these operations?

b. Write down the $4 \otimes 4$-matrix $A = (1, 1, 1, 1) \otimes (0, 0, 1, 1)$. Compute $(T \otimes I)A$ by applying the filters to the corresponding rows/columns of A as we have learnt, and interpret the result. Do the same for $(I \otimes T)A$.

2. With $T = \{1/2, \underline{0}, -1/2\}$, show that $I \otimes T$ is the transformation where the computational molecule is given by Equation 10.7 in Example 10.19. This tensor product can be used for computing partial derivatives in the x-direction, equivalently detecting vertical edges in images.

3. With $T = \{1/2, \underline{0}, -1/2\}$, show that $T \otimes T$ corresponds to the computational molecule given by Equation 10.10 in Example 10.19.

4. Let T be the moving average filter of length $2L+1$, i.e. $T = \frac{1}{L}\{\underbrace{1, \cdots, 1, \underline{1}, 1, \cdots, 1}_{2L+1 \text{ times}}\}$. As in Example 11.4, find the computational molecule of $T \otimes T$.

5. Verify that the computational molecule given by Equation 10.5 in Example 10.19 is the same as that of $T \otimes T$, where $T = \{\frac{1}{64}, \frac{6}{64}, \frac{15}{64}, \underline{\frac{20}{64}}, \frac{15}{64}, \frac{6}{64}, \frac{1}{64}\}$ (the coefficients come from row 6 of Pascals triangle).

6. Show that the mapping $F(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y}$ is bi-linear, i.e. that $F(\alpha\boldsymbol{x}_1 + \beta\boldsymbol{x}_2, \boldsymbol{y}) = \alpha F(\boldsymbol{x}_1, \boldsymbol{y}) + \beta F(\boldsymbol{x}_2, \boldsymbol{y})$, and $F(\boldsymbol{x}, \alpha\boldsymbol{y}_1 + \beta\boldsymbol{y}_2) = \alpha F(\boldsymbol{x}, \boldsymbol{y}_1) + \beta F(\boldsymbol{x}, \boldsymbol{y}_2)$.

7. Attempt to find matrices $S : \mathbb{R}^M \to \mathbb{R}^M$ and $T : \mathbb{R}^N \to \mathbb{R}^N$ so that the following mappings from $L_{M,N}(\mathbb{R})$ to $L_{M,N}(\mathbb{R})$ can be written on the form $X \to SXT^T = (S \otimes T)X$. In all the cases, it may be that no such S, T can be found. If this is the case, prove it.

a. The mapping which reverses the order of the rows in a matrix.

b. The mapping which reverses the order of the columns in a matrix.

c. The mapping which transposes a matrix.

8. Find an alternative form for Equation (11.1) and an accompanying reimplementation of Theorem 11.8 which is adapted to the case when we want all operations to be performed row by row, instead of column by column.

9. (Trial Exam UIO V2012) Let the filter T be defined by $T = \{1, \underline{2}, 1\}$.

a. Write down the computational molecule of $T \otimes T$.

b. Let us define $\boldsymbol{x} = (1, 2, 3)$, $\boldsymbol{y} = (3, 2, 1)$, $\boldsymbol{z} = (2, 2, 2)$, and $\boldsymbol{w} = (1, 4, 2)$. Compute the matrix $A = \boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{z} \otimes \boldsymbol{w}$.

c. Compute $(T \otimes T)A$ by applying the filter T to every row and column in the matrix the way we have learnt. If the matrix A was more generally an image, what can you say about how the new image will look?

11.2 Change of bases in tensor products

In this section we will prove a specialization of our previous result to the case where S and T are change of coordinate matrices. We start by proving the following:

Theorem 11.12. If $\mathcal{B}_1 = \{\boldsymbol{v}_i\}_{i=0}^{M-1}$ is a basis for $\mathbb{R}^M$, and $\mathcal{B}_2 = \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ is a basis for $\mathbb{R}^N$, then $\{\boldsymbol{v}_i \otimes \boldsymbol{w}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $L_{M,N}(\mathbb{R})$. We denote this basis by $\mathcal{B}_1 \otimes \mathcal{B}_2$.

Proof. Suppose that $\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = 0$. Setting $\boldsymbol{h}_i = \sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j$ we get

$$\sum_{j=0}^{N-1} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \boldsymbol{v}_i \otimes (\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j) = \boldsymbol{v}_i \otimes \boldsymbol{h}_i.$$

where we have used the bi-linearity of the tensor product mapping $(\boldsymbol{x}, \boldsymbol{y}) \to \boldsymbol{x} \otimes \boldsymbol{y}$ (Exercise 11.1.6). This means that

$$\mathbf{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \sum_{i=0}^{M-1} \boldsymbol{v}_i \otimes \boldsymbol{h}_i = \sum_{i=0}^{M-1} \boldsymbol{v}_i \boldsymbol{h}_i^T.$$

Column k in this matrix equation says $\mathbf{0} = \sum_{i=0}^{M-1} h_{i,k}\boldsymbol{v}_i$, where $h_{i,k}$ are the components in $\boldsymbol{h}_i$. By linear independence of the $\boldsymbol{v}_i$ we must have that $h_{0,k} = h_{1,k} = \cdots = h_{M-1,k} = 0$. Since this applies for all k, we must have that all $\boldsymbol{h}_i = 0$. This means that $\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j = \mathbf{0}$ for all i, from which it follows by linear independence of the $\boldsymbol{w}_j$ that $\alpha_{i,j} = 0$ for all j, and for all i. This means that $\mathcal{B}_1 \otimes \mathcal{B}_2$ is a basis. □

In particular, as we have already seen, the standard basis for $L_{M,N}(\mathbb{R})$ can be written $\mathcal{E}_{M,N} = \mathcal{E}_M \otimes \mathcal{E}_N$. This is the basis for a useful convention: For a tensor product the bases are most naturally indexed in two dimensions, rather than the usual sequential indexing. This difference translates also to the meaning of coordinate vectors, which now are more naturally thought of as coordinate matrices:

Definition 11.13 (Coordinate matrix)**.** Let $\{\boldsymbol{v}_i\}_{i=0}^{M-1}, \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ be bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. By the coordinate matrix of $\sum_{k,l} \alpha_{k,l}(\boldsymbol{v}_k \otimes \boldsymbol{w}_l)$ we will mean the $M \times N$-matrix X with entries $X_{kl} = \alpha_{k,l}$.

We will have use for the following theorem, which shows how change of coordinates in $\mathbb{R}^M$ and $\mathbb{R}^N$ translate to a change of coordinates in the tensor product:

Theorem 11.14 (Change of coordinates in tensor products)**.** Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for $\mathbb{R}^M$, and $\mathcal{B}_2, \mathcal{C}_2$ are bases for $\mathbb{R}^N$, and that S is the change of coordinates matrix from $\mathcal{C}_1$ to $\mathcal{B}_1$, and that T is the change of coordinates matrix from $\mathcal{C}_2$ to $\mathcal{B}_2$. Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $L_{M,N}(\mathbb{R})$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \tag{11.3}$$

Proof. Let $\boldsymbol{c}_{ki}$ be the i'th basis vector in $\mathcal{C}_k$, $\boldsymbol{b}_{ki}$ the i'th basis vector in $\mathcal{B}_k$, $k = 1, 2$. Since any change of coordinates is linear, it is enough to show that it coincides with $X \to SXT^T$ on the basis $\mathcal{C}_1 \otimes \mathcal{C}_2$. The basis vector $\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j}$ has coordinate vector $X = \boldsymbol{e}_i \otimes \boldsymbol{e}_j$ in $\mathcal{C}_1 \otimes \mathcal{C}_2$. With the mapping $X \to SXT^T$ this is sent to

$$SXT^T = S(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)T^T = \text{col}_i(S)\text{row}_j(T^T).$$

On the other hand, since column i in S is the coordinates of $\boldsymbol{c}_{1i}$ in the basis $\mathcal{B}_1$, and column j in T is the coordinates of $\boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_2$, we can write

$$\begin{aligned}
\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j} &= \left(\sum_k S_{k,i}\boldsymbol{b}_{1k}\right) \otimes \left(\sum_l T_{l,j}\boldsymbol{b}_{2l}\right) = \sum_{k,l} S_{k,i}T_{l,j}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) \\
&= \sum_{k,l} S_{k,i}(T^T)_{j,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l}) = \sum_{k,l} (\text{col}_i(S)\text{row}_j(T^T))_{k,l}(\boldsymbol{b}_{1k} \otimes \boldsymbol{b}_{2l})
\end{aligned}$$

we see that the coordinate vector of $\boldsymbol{c}_{1i} \otimes \boldsymbol{c}_{2j}$ in the basis $\mathcal{B}_1 \otimes \mathcal{B}_2$ is $\text{col}_i(S)\text{row}_j(T^T)$. In other words, change of coordinates coincides with $X \to SXT^T$, and the proof is done. □

In both cases of tensor products of matrices and change of coordinates in tensor products, we see that we need to compute the mapping $X \to SXT^T$. This means that we can restate Theorem 11.8 for change of coordinates as follows:

Theorem 11.15 (Implementation of change of coordinates in tensor products)**.** The change of coordinates from $\mathcal{C}_1 \otimes \mathcal{C}_2$ to $\mathcal{B}_1 \otimes \mathcal{B}_2$ can be implemented as follows:

1. For every column in the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, perform a change of coordinates from $\mathcal{C}_1$ to $\mathcal{B}_1$.
2. Transpose the resulting matrix.
3. For every column in the resulting matrix, perform a change of coordinates from $\mathcal{C}_2$ to $\mathcal{B}_2$.
4. Transpose the resulting matrix.

We can reuse the algorithm from the previous section to implement this. In the following operations on images, we will visualize the pixel values in an image as coordinates in the standard basis, and perform a change of coordinates.

Example 11.16 (Change of coordinates with the DFT)**.** The DFT is one particular change of coordinates which we have considered. The DFT was the change of coordinates from the standard basis to the Fourier basis. The corresponding change of coordinates in a tensor product is obtained by substituting with the DFT as the function implementing change of coordinates in Theorem 11.15. The change of coordinates in the opposite direction is obtained by using the IDFT instead of the DFT.

Modern image standards do typically not apply a change of coordinates to the entire image. Rather one splits the image into smaller squares of appropriate size, called blocks, and perform change of coordinates independently for each block. With the JPEG standard, the blocks are always 8×8. It is of course not a coincidence that a power of 2 is chosen here, since the DFT takes a simplified form in case of powers of 2.

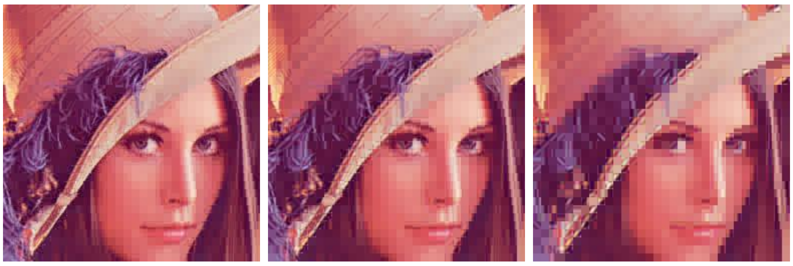
The DFT values express frequency components. The same applies for the two-dimensional DFT and thus for images, but frequencies are now represented in two different directions. The thing which actually provides compression in many image standards is that frequency components which are small are set to 0. This corresponds to neglecting frequencies in the image which have small contributions. This type of lossy compression has little effect on the human perception of the image, if we use a suitable neglection threshold. After we have performed the two-dimensional DFT on an image, we can neglect DFT-coefficients below a threshold on the resulting matrix `X` with the following code:

```
X=X.*(abs(X)>=threshold);
```

Here, the command `abs(X)>=threshold` returns a matrix with 1 and 0 of the same size as `X`. 1 represents a value of `true` in the logical expression which is evaluated, 0 represents false. The value is 1 if and only if the absolute value of the corresponing element is greater than or equal to `threshold`. `abs(X)>=threshold` thus returns what we could call a threshold matrix, and when we multiply with this, elements below a given threshold are neglected.

In Figure 11.2 we have applied the two-dimensional DFT to our test image. We have then neglected DFT coefficients which are below certain thresholds, and transformed the samples back to reconstruct the image. When increasing the threshold, the image becomes more and more unclear, but the image is quite clear in the first case, where as much as more than 90% of the samples have been neglected. The blocking effect at the block boundaries is clearly visible. ♣

Example 11.17 (Change of coordinates with the DCT)**.** Similarly to the DFT, the DCT was the change of coordinates from the standard basis to what we called the DCT basis. The DCT is used more than the DFT in image processing. Change of coordinates in tensor products between the standard basis and the DCT basis is obtained by substituting with the DCT and the IDCT in



(a) Threshold 30. 91.4% of the DFT-values were neglected

(b) Threshold 50. 95.3% of the DFT-values were neglected

(c) Threshold 100. 97.7% of the DFT-values were neglected

Figure 11.2: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold were neglected.

Theorem 11.15. The JPEG standard actually applies a two-dimensional DCT to the blocks of size 8×8, it does not apply the two-dimensional DFT.

If we follow the same strategy for the DCT as for the DFT, so that we neglect DCT-coefficients which are below a given threshold, we get the images shown in Figure 11.3. We see similar effects as with the DFT, but it seems that the latter images are a bit clearer, verifying that the DCT is a better choice than the DFT. It is also interesting to compare with what happens when we drop splitting the image into blocks. Of course, when we neglect many of the DCT-coefficients, we should see some artifacts, but there is no reason to believe that these should be at the old block boundaries. The new artifacts can be seen in Figure 11.4, where the same thresholds as before have been used. Clearly, the new artifacts take a completely different shape. ♣

In the exercises you will be asked to implement functions which generate the images shown in these examples.

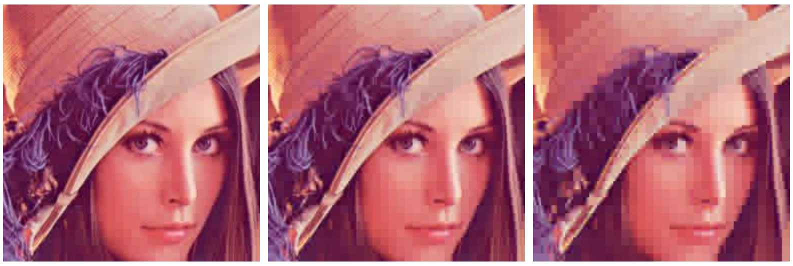
Exercises for Section 11.2

1. Implement functions

```
function newx=FFT2Impl(x)
function x=IFF2Impl(newx)
function newx=DCT2Impl(x)
function x=IDCT2Impl(newx)
```

which implement the two-dimensional DCT, FFT, and their inverses as described in this section. Base your code on the algorithm at the end of Section 11.1.

2. Implement functions

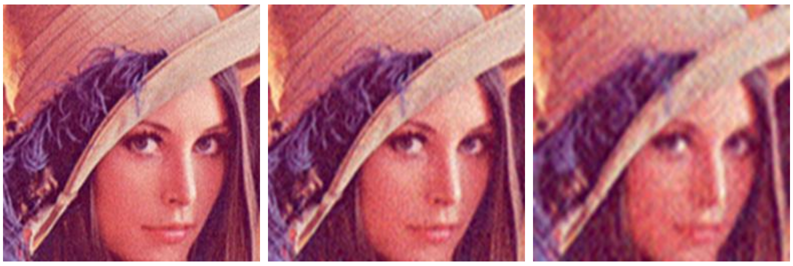


(a) Threshold 30. 93.1% of the DCT-values were neglected

(b) Threshold 50. 96.6% of the DCT-values were neglected

(c) Threshold 100. 98.8% of the DCT-values were neglected

Figure 11.3: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected.



(a) Threshold 30. 93.2% of the DCT-values were neglected

(b) Threshold 50. 95.8% of the DCT-values were neglected

(c) Threshold 100. 97.7% of the DCT-values were neglected

Figure 11.4: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold were neglected. The image has not been split into blocks here.

```
function samples=transform2jpeg(x)
function samples=transform2invjpeg(x)
```

which splits the image into blocks of size 8 × 8, and performs the DCT2/IDCT2 on each block. Finally run the code

```
function showDCThigher(threshold)
  img=double(imread('lena.png','png'));
  zeroedout=0;
  for k=1:3
      newimg=transform2jpeg(img(:,:,k));
      thresholdmatr=(abs(newimg)>=threshold);
      zeroedout=zeroedout+size(img,1)*size(img,2)-sum(sum(thresholdmatr));
      img(:,:,k)=transform2invjpeg(newimg.*thresholdmatr);
  end
  imshow(uint8(255*mapto01(img)));
  fprintf('%i percent of samples zeroed out\n',...
      100*zeroedout/(3*size(img,1)*size(img,2)));
```

for different threshold parameters, and check that this reproduces the test images of this section, and prints the correct numbers of values which have been neglected (i.e. which are below the threshold) on screen.

3. (Exam UIO V2012) Suppose that `FFTImpl` and `IFFTImpl` are implementations of the FFT and IFFT algorithms. Suppose also that we have given an image by the matrix `X`. Consider the following Matlab code:

```
threshold=30;
for k=1:2
  for s=1:size(X,2)
    X(:,s)=FFTImpl(X(:,s));
  end
  X=X';
end
X=X.*(abs(X)>=threshold);
for k=1:2
  for s=1:size(X,2)
    X(:,s)=IFFTImpl(X(:,s));
  end
  X=X';
end
```

Comment what the code does. Comment in particular on the meaning of the parameter `threshold`, and what effect this has on the image.

11.3 Summary

We defined the tensor product, and saw how this could be used to define operations on images in a similar way to how we defined operations on sound. It turned out that the tensor product construction could be used to construct some of the operations on images we looked at in the previous chapter, which now could be factorized into first filtering the columns in the image, and then filtering the rows in the image. We went through an algorithm for computing the tensor product, and established how we could perform change of coordinates in tensor products. This enables us to define two-dimensional extensions of the DCT and the DFT and their inverses, and we used these extensions to experiment on images.

We mentioned that the JPEG standard essentially follows the approach described in this chapter. A more detailed description of this standard can be found in [11].

Chapter 12

Tensor products in a wavelet setting

In Chapter 11 we defined tensor products in terms of vectors, and we saw that the tensor product of two vectors is in fact a matrix. The same construction can be applied to other vector spaces, in particular to vector spaces that are function spaces. As we will see, the tensor product of two univariate function spaces will be a space of functions in two variables. Recall that wavelets are defined in terms of function spaces, so this construction will allow us to define tensor products of wavelets. Through this we will be able to define wavelet transforms that can be applied to images.

Definition 12.1 (Tensor product of function spaces)**.** Let U_1 and U_2 be two vector spaces of functions defined on the intervals $[0, M)$ and $[0, N)$, respectively, and suppose that $f_1 \in U_1$ and $f_2 \in U_2$. The tensor product of f_1 and f_2, denoted $f_1 \otimes f_2$, denotes the function in two variables defined on $[0, M) \times [0, N)$ given by $f_1(t_1)f_2(t_2)$. The function $f_1 \otimes f_2$ is also referred to as the separable extension of f_1 and f_2 to two variables. The tensor product of the two spaces $U_1 \otimes U_2$ denotes the set of all functions in two variables defined on $[0, M) \times [0, N)$ and on the form $f_1(t_1)f_2(t_2)$, where $f_1 \in U_1$ and $f_2 \in U_2$.

We will always assume that the spaces U_1 and U_2 consist of functions which are at least integrable. In this case $U_1 \otimes U_2$ is also an inner product space, with the inner product given by a double integral,

$$\langle f, g \rangle = \int_0^N \int_0^M f(t_1, t_2)g(t_1, t_2)dt_1 dt_2. \tag{12.1}$$

In particular, this says that

$$\begin{aligned}\langle f_1 \otimes f_2, g_1 \otimes g_2\rangle &= \int_0^N \int_0^M f_1(t_1)f_2(t_2)g_1(t_1)g_2(t_2)dt_1dt_2 \\ &= \int_0^M f_1(t_1)g_1(t_1)dt_1 \int_0^N f_2(t_2)g_2(t_2)dt_2 = \langle f_1, g_1\rangle\langle f_2, g_2\rangle.\end{aligned} \tag{12.2}$$

This means that for tensor products, a double integral can be computed as the product of two one-dimensional integrals.

The tensor product space defined in Definition 12.1 is useful for approximation of functions of two variables if each of the two spaces of univariate functions have good approximation properties.

Idea 12.2. If the spaces U_1 and U_2 can be used to approximate functions in one variable, then $U_1 \otimes U_2$ can be used to approximate functions in two variables.

We will not state this precisely, but just consider some important examples.

Example 12.3. Let $U_1 = U_2$ be the space of all polynomials of finite degree. We know that U_1 can be used for approximating many kinds of functions, such as continuous functions, for example by Taylor series. The tensor product $U_1 \otimes U_1$ consists of all functions on the form $\sum_{i,j} \alpha_{i,j} t_1^i t_2^j$. It turns out that polynomials in several variables have approximation properties analogous to univariate polynomials. ♣

Example 12.4. Let $U_1 = U_2 = V_{N,T}$ be the Nth order Fourier space which is spanned by the functions

$$e^{-2\pi iNt/T}, \dots, e^{-2\pi it/T}, 1, e^{2\pi it/T}, \dots, e^{2\pi iNt/T}$$

The tensor product space $U_1 \otimes U_1$ now consists of all functions on the form $\sum_{k,l=0}^N \alpha_{k,l} e^{2\pi ikt_1/T} e^{2\pi ilt_2/T}$. One can show that this space has approximation properties similar to $V_{N,T}$. This is the basis for the theory of Fourier series in two variables. ♣

In the following we think of $U_1 \otimes U_2$ as a space which can be used for approximating a general class of functions. By associating a function with the vector of coordinates relative to some basis, and a matrix with a function in two variables, we have the following parallel to Theorem 11.12:

Theorem 12.5. If $\{f_i\}_{i=0}^{M-1}$ is a basis for U_1 and $\{g_j\}_{j=0}^{N-1}$ is a basis for U_2, then $\{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $U_1 \otimes U_2$. Moreover, if the bases for U_1 and U_2 are orthogonal/orthonormal, then the basis for $U_1 \otimes U_2$ is orthogonal/orthonormal.

Proof. The proof is similar to that of Theorem 11.12: if

$$\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(f_i \otimes g_j) = 0,$$

we define $h_i(t_2) = \sum_{j=0}^{N-1} \alpha_{i,j} g_j(t_2)$. It follows as before that $\sum_{i=0}^{M-1} h_i(t_2) f_i = 0$ for any t_2, so that $h_i(t_2) = 0$ for any t_2 due to linear independence of the f_i. But then $\alpha_{i,j} = 0$ also, due to linear independene of the g_j. The statement about orthogonality follows from Equation 12.2. □

We can now define the tensor product of two bases of functions as before: if $\mathcal{B} = \{f_i\}_{i=0}^{M-1}$ and $\mathcal{C} = \{g_j\}_{j=0}^{N-1}$, we set $\mathcal{B} \otimes \mathcal{C} = \{f_i \otimes g_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$. Coordinate matrices can also be defined as before: if $f(t_1, t_2) = \sum X_{i,j}(f_i \otimes g_j)(t_1, t_2)$, the coordinate matrix of f is the matrix X with elements $X_{i,j}$. Theorem 11.14 can also be proved in the same way in the context of function spaces. We state this as follows:

Theorem 12.6 (Change of coordinates in tensor products of function spaces)**.** Assume that $\mathcal{B}_1, \mathcal{C}_1$ are bases for U_1, and $\mathcal{B}_2, \mathcal{C}_2$ are bases for U_2, and that S is the change of coordinates matrix from $\mathcal{C}_1$ to $\mathcal{B}_1$, and that T is the change of coordinates matrix from $\mathcal{C}_2$ to $\mathcal{B}_2$. Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $U_1 \otimes U_2$, and if X is the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, and Y the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, then

$$Y = SXT^T. \tag{12.3}$$

12.1 Adopting the tensor product terminology to wavelets

In the remaining part of this chapter we will apply the tensor product construction to wavelets. In particular the spaces U_1, U_2 from Definition 12.1 are defined from function spaces V_m, W_m, constructed from a given wavelet. We can in particular form the tensor products $\phi_{0,n_1} \otimes \phi_{0,n_2}$. We will assume that

1. the first component ϕ_{0,n_1} has period M (so that $\{\phi_{0,n_1}\}_{n_1=0}^{M-1}$ is a basis for the first component space),

2. the second component ϕ_{0,n_2} has period N (so that $\{\phi_{0,n_2}\}_{n_2=0}^{N-1}$ is a basis for the second component space).

When we speak of $V_0 \otimes V_0$ we thus mean an MN-dimensional space with basis $\{\phi_{0,n_1} \otimes \phi_{0,n_2}\}_{(n_1,n_2)=(0,0)}^{(M-1,N-1)}$, where the coordinate matrices are $M \times N$. This difference in the dimension of the two components is done to allow for images where the number of rows and columns may be different. In the following we will implicitly assume that the component spaces have dimension M and N, to ease notation. If we use that $\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}$ also is a basis for V_m, we get the following corollary to Theorem 12.5:

Corollary 12.7. Let ϕ, ψ be a scaling function and a mother wavelet. Then the two sets of tensor products given by

$$\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m = \{\phi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$$

and

$$\begin{aligned}
&(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \\
&= \{\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\
&\qquad \phi_{m-1,n_1} \otimes \psi_{m-1,n_2}, \\
&\qquad \psi_{m-1,n_1} \otimes \phi_{m-1,n_2}, \\
&\qquad \psi_{m-1,n_1} \otimes \psi_{m-1,n_2}\}_{n_1,n_2}
\end{aligned}$$

are both bases for $V_m \otimes V_m$. This second basis is orthogonal/orthonormal whenever the first basis is.

From this we observe that while the one-dimensional wavelet decomposition splits V_m into a direct sum of the two vector spaces V_{m-1} and W_{m-1}, the corresponding two-dimensional decomposition splits $V_m \otimes V_m$ into a direct sum of four tensor product vector spaces which deserve individual names.

Definition 12.8. We define the following tensor product spaces:

1. The space $W_m^{(0,1)}$ spanned by $\{\phi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$,
2. The space $W_m^{(1,0)}$ spanned by $\{\psi_{m,n_1} \otimes \phi_{m,n_2}\}_{n_1,n_2}$,
3. The space $W_m^{(1,1)}$ spanned by $\{\psi_{m,n_1} \otimes \psi_{m,n_2}\}_{n_1,n_2}$.

The splitting of $V_m \otimes V_m$ into a direct sum of vector spaces can now be summed up as

$$V_m \otimes V_m = (V_{m-1} \otimes V_{m-1}) \oplus W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}. \tag{12.4}$$

Also in the setting of tensor products we refer to $V_{m-1} \otimes V_{m-1}$ as the space of low-resolution approximations. The remaining parts, $W_{m-1}^{(0,1)} \oplus W_{m-1}^{(1,0)} \oplus W_{m-1}^{(1,1)}$, is refered to as the detail space. Note that the coordinate matrix of

$$\begin{aligned}
\sum_{n_1,n_2=0}^{2^{m-1}N} &(c_{m-1,n_1,n_2}(\phi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(0,1)}(\phi_{m-1,n_1} \otimes \psi_{m-1,n_2}) + \\
&w_{m-1,n_1,n_2}^{(1,0)}(\psi_{m-1,n_1} \otimes \phi_{m-1,n_2}) + w_{m-1,n_1,n_2}^{(1,1)}(\psi_{m-1,n_1} \otimes \psi_{m-1,n_2}))
\end{aligned} \tag{12.5}$$

in the basis $(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1})$ is

$$\left(\begin{array}{cc|cc} c_{m-1,0,0} & \cdots & w^{(0,1)}_{m-1,0,0} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \hline w^{(1,0)}_{m-1,0,0} & \cdots & w^{(1,1)}_{m-1,0,0} & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{array}\right). \tag{12.6}$$

We see that the coordinate matrix is split into four submatrices:

- The c_{m-1}-values, i.e. the coordinates for $V_{m-1} \oplus V_{m-1}$. This is the upper left corner in (12.6), and is also called the 00-subband.
- The $w^{(0,1)}_{m-1}$-values, i.e. the coordinates for $W^{(0,1)}_{m-1}$. This is the upper right corner in (12.6).
- The $w^{(1,0)}_{m-1}$-values, i.e. the coordinates for $W^{(1,0)}_{m-1}$. This is the lower left corner in (12.6).
- The $w^{(1,1)}_{m-1}$-values, i.e. the coordinates for $W^{(1,1)}_{m-1}$. This is the lower right corner in (12.6).

The $w^{(i,j)}_{m-1}$-values are as in the one-dimensional situation often refered to as wavelet coefficients. Let us consider the Haar wavelet as an example.

Example 12.9. If V_m is the vector space of piecewise constant functions on any interval of the form $[k2^{-m}, (k+1)2^{-m})$ (as in the piecewise constant wavelet), $V_m \otimes V_m$ is the vector space of functions in two variables which are constant on any square of the form $[k_1 2^{-m}, (k_1+1)2^{-m}) \times [k_2 2^{-m}, (k_2+1)2^{-m})$. Clearly $\phi_{m,k_1} \otimes \phi_{m,k_2}$ is constant on such a square and 0 elsewhere, and these functions are a basis for $V_m \otimes V_m$.

Let us compute the orthogonal projection of $\phi_{1,k_1} \otimes \phi_{1,k_2}$ onto $V_0 \otimes V_0$. Since the Haar wavelet is orthonormal, the basis functions in (12.4) are orthonormal, and we can thus use the orthogonal decomposition formula to find this projection. Clearly $\phi_{1,k_1} \otimes \phi_{1,k_2}$ has different support from all except one of $\phi_{0,n_1} \otimes \phi_{0,n_2}$. Since

$$\langle \phi_{1,k_1} \otimes \phi_{1,k_2}, \phi_{0,n_1} \otimes \phi_{0,n_2} \rangle = 1/2$$

when the supports intersect, we obtain

$$\text{proj}_{V_0 \otimes V_0} \phi_{1,k_1} \otimes \phi_{1,k_2} = \begin{cases} \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1, k_2 \text{ are even} \\ \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1 \text{ is even, } k_2 \text{ is odd} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,k_2/2}) & \text{when } k_1 \text{ is odd, } k_2 \text{ is even} \\ \frac{1}{2}(\phi_{0,(k_1-1)/2} \otimes \phi_{0,(k_2-1)/2}) & \text{when } k_1, k_2 \text{ are odd} \end{cases}$$

So, in this case there were 4 different formulas, since there were 4 different combinations of even/odd. Let us also compute the projection onto the orthogonal complement of $V_0 \otimes V_0$ in $V_1 \otimes V_1$, and let us express this in terms of the

$\phi_{0,n}, \psi_{0,n}$, like we did in the one-variable case. Also here there are 4 different formulas. When k_1, k_2 are both even we obtain

$$
\begin{aligned}
&\phi_{1,k_1} \otimes \phi_{1,k_2} - \text{proj}_{V_0 \otimes V_0}(\phi_{1,k_1} \otimes \phi_{1,k_2}) \\
&= \phi_{1,k_1} \otimes \phi_{1,k_2} - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \left(\frac{1}{\sqrt{2}}(\phi_{0,k_1/2} + \psi_{0,k_1/2})\right) \otimes \left(\frac{1}{\sqrt{2}}(\phi_{0,k_2/2} + \psi_{0,k_2/2})\right) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) \\
&+ \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}) - \frac{1}{2}(\phi_{0,k_1/2} \otimes \phi_{0,k_2/2}) \\
&= \frac{1}{2}(\phi_{0,k_1/2} \otimes \psi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \phi_{0,k_2/2}) + \frac{1}{2}(\psi_{0,k_1/2} \otimes \psi_{0,k_2/2}).
\end{aligned}
$$

Here we have used the relation $\phi_{1,k_i} = \frac{1}{\sqrt{2}}(\phi_{0,k_i/2} + \psi_{0,k_i/2})$, which we have from our first analysis of the Haar wavelet. Checking the other possibilities we find similar formulas for the projection onto the orthogonal complement of $V_0 \otimes V_0$ in $V_1 \otimes V_1$ when either k_1 or k_2 is odd. In all cases, the formulas use the basis functions for $W_0^{(0,1)}$, $W_0^{(1,0)}$, $W_0^{(1,1)}$. These functions are shown in Figure 12.1, together with the function $\phi \otimes \phi \in V_0 \otimes V_0$. ♣

Example 12.10. If we instead use any of the wavelets for piecewise linear functions, the wavelet basis functions are not orthogonal anymore, just as in the one-dimensional case. The new basis functions are shown in Figure 12.2 for the alternative piecewise linear wavelet. ♣

An immediate corollary of Theorem 12.6 is the following:

Corollary 12.11. Let

$$
\begin{aligned}
A_m &= P_{(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \leftarrow \boldsymbol{\phi}_m} \\
B_m &= P_{\boldsymbol{\phi}_m \leftarrow (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1})}
\end{aligned}
$$

be the stages in the DWT and the IDWT, and let

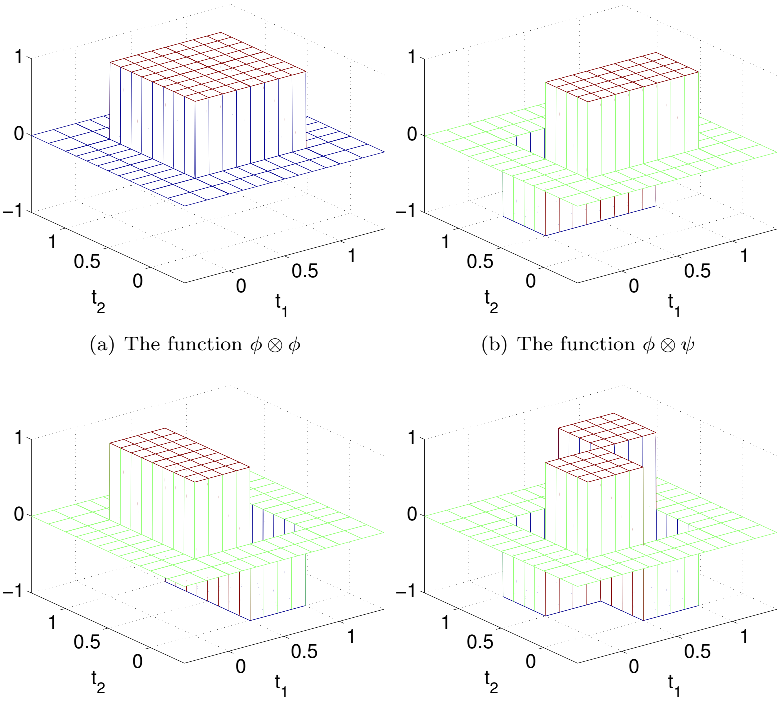
$$
X = (c_{m,i,j})_{i,j} \qquad Y = \begin{pmatrix} (c_{m-1,i,j})_{i,j} & (w^{(0,1)}_{m-1,i,j})_{i,j} \\ (w^{(1,0)}_{m-1,i,j})_{i,j} & (w^{(1,1)}_{m-1,i,j})_{i,j} \end{pmatrix} \tag{12.7}
$$

be the coordinate matrices in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$, and $(\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1}) \otimes (\boldsymbol{\phi}_{m-1} \oplus \boldsymbol{\psi}_{m-1})$, respectively. Then

$$
Y = A_m X A_m^T \tag{12.8}
$$

$$
X = B_m Y B_m^T \tag{12.9}
$$

By the m-level two-dimensional DWT/IDWT (or DWT2/IDWT2) we mean the change of coordinates where this is repeated m times as in a DWT/IDWT.



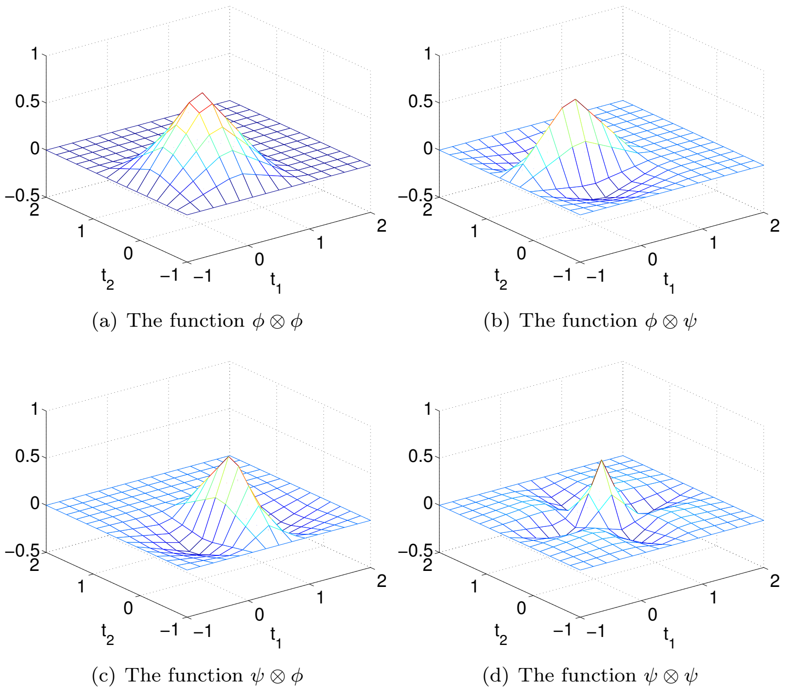
(a) The function $\phi \otimes \phi$

(b) The function $\phi \otimes \psi$

(c) The function $\psi \otimes \phi$

(d) The function $\psi \otimes \psi$

Figure 12.1: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the Haar wavelet.



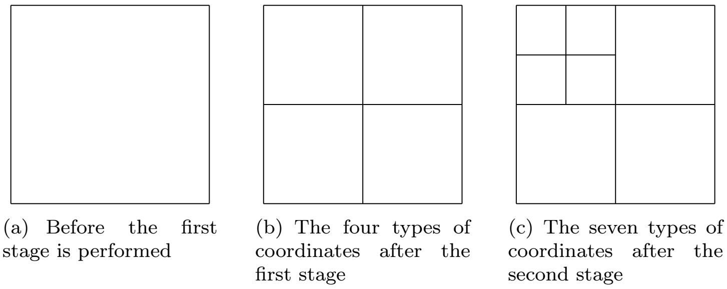
(a) The function $\phi \otimes \phi$

(b) The function $\phi \otimes \psi$

(c) The function $\psi \otimes \phi$

(d) The function $\psi \otimes \psi$

Figure 12.2: The basis functions for $(V_0 \otimes V_0) \oplus W_0^{(0,1)} \oplus W_0^{(1,0)} \oplus W_0^{(1,1)}$ for the alternative piecewise linear wavelet.



(a) Before the first stage is performed

(b) The four types of coordinates after the first stage

(c) The seven types of coordinates after the second stage

Figure 12.3: Illustration of the different coordinates in a two level DWT2.

Each stage in DWT2 and IDWT2 can now be implemented by substituting the matrices A_m, B_m above into Theorem 11.15. This implementation can reuse an efficient implementation of the one-dimensional DWT/IDWT. When using many levels of the DWT2, the next stage is applied only to the upper left corner of the matrix, just as the DWT at the next stage only is applied to the first part of the coordinates. At each stage, the upper left corner of the coordinate matrix (which gets smaller at each iteration), is split into four equally big parts. This is illustrated in Figure 12.3, where the different types of coordinates which appear in the first two stages in a DWT2 are indicated. It is also instructive to see what information the different types of coordinates in an image represent. If we discard a given type of coordinates, we will illustrate this by coloring the corresponding region black. If we perform a two-level DWT2, i.e. we start with a coordinate matrix in the basis $\boldsymbol{\phi}_2 \otimes \boldsymbol{\phi}_2$, Figure 12.4 illustrates first the collection of all coordinates, and then the resulting collection of coordinates after removing subbands at the first level successively. The subbands which have been removed are indicated with a black colour. Figure 12.5 illustrates in the same way incremental removal of the subbands at the second level.

Let us make some experiments with images using the wavelets we have considered [1]. Our theory is applied to images in the following way: We visualize the pixels in the image as coordinates in the basis $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ (so that the image has size $(2^m M) \times (2^m N)$), and perform change of coordinates with the DWT2. We can then, just as we did for sound, and for the DCT/DFT-values in images, either set the the part from the $W_k^{(i,j)}$-spaces (the detail) to zero, or the part from $V_0 \otimes V_0$ (the $M \times N$-low-resolution approximation) to zero, depending on whether we want to inspect the detail or the low-resolution approximation in the image. Finally we apply the IDWT2 to end up with coordinates in $\boldsymbol{\phi}_m \otimes \boldsymbol{\phi}_m$ again, and display the image with pixel values being these coordinates.

Example 12.12 (Applying the Haar wavelet to a very simple example image)**.** Let us return to the interpretation of the different corners in the image after

[1] Note also that Matlab has a wavelet toolbox which could be used for these purposes, but we will not go into the usage of this.

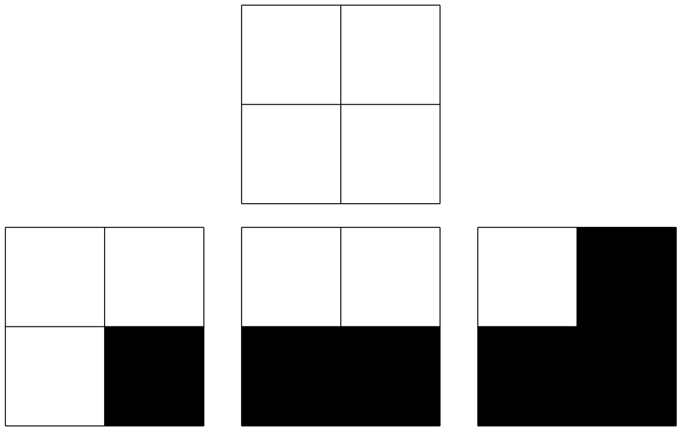


Figure 12.4: Graphical representation of neglecting the wavelet coefficients at the first level. After applying DWT2, the wavelet coefficients are split into four parts, as shown in the first figure. In the following figures we have removed coefficients from $W_1^{(1,1)}$, $W_1^{(1,0)}$, and $W_1^{(0,1)}$, in that order.

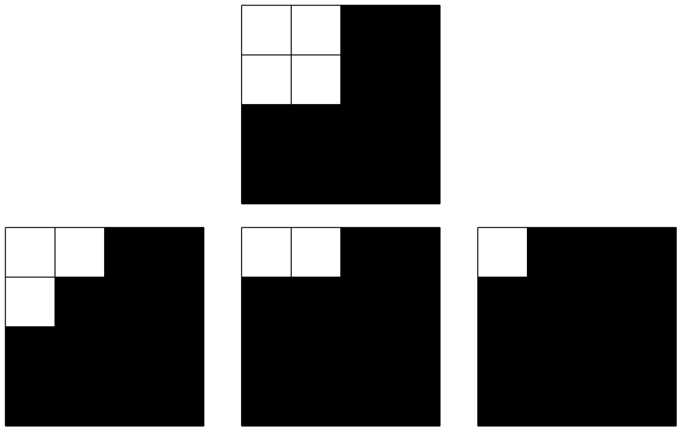


Figure 12.5: Graphical representation of neglecting the wavelet coefficients at the second level. After applying the second stage in DWT2, the wavelet coefficients from the upper left corner are also split into four parts, as shown in the first figure. In the following figures we have removed coefficients from $W_2^{(1,1)}$, $W_2^{(1,0)}$, and $W_2^{(0,1)}$, in that order.

application of the DWT2. Since the first half of the coordinates in a DWT are outputs from the lowpass filter H_0, the upper half of the matrix has been subject to a lowpass filter to the columns. Similarly, the left half of the matrix has been subject to the same lowpass filter to the rows. Due to this, the following names are given.

- The upper left corner is called the LL-subband,
- The upper right corner is called the LH-subband,
- The lower left corner is called the HL-subband,
- The lower right corner is called the HH-subband.

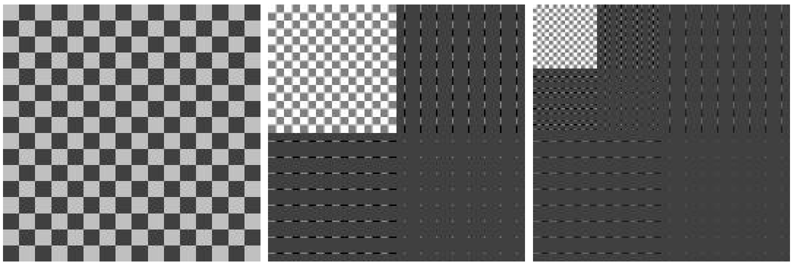
The two letters indicate the kind of filters which have been applied (L=lowpass, H=highpass). The first filter indicates the filter which is applied to the columns, the second indicates which is applied to the rows. The order is therefore important. The name *subband* comes from the interpretation of these filters as being selective on a certain frequency band.

Let us apply the Haar wavelet to a sample image with a simple chess pattern, as shown in Figure 12.6(a). The lowpass filter of the Haar wavelet was essentially a smoothing filter with two elements. Also, as we have seen, the highpass filter essentially computes an approximation to the partial derivative. Therefore, after the DWT2 we should see the following:

- In the upper left corner, the image is smoothed both in the vertical and horizontal direction.
- In the upper right corner, the image is smoothed in the horizontal direction, and edges (points with abrupt changes) in the vertical direction should be visible.
- In the lower left corner, the image is smoothed in the vertical direction, and edges in the horizontal direction should be visible.
- In the lower right corner, only points where we have abrupt changes in both directions should be visible.

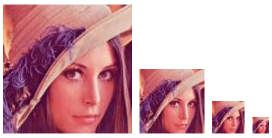
The sample image is chosen so that all these effect are easily seen, as shown in Figure 12.6(b) and (c). In particular, only vertical edges are visible in the upper right corner at the expected places in the chess pattern, and only horizontal edges are visible in the lower left corner. Finally, only the grid points in the image display abrupt changes in value in both direction, so that only these are visible in the lower right corner. Note that the values after DWT2 may not lie in the legal range of pixel values. The figure above has taken this into account. ♣

Example 12.13 (Creating thumbnail images)**.** Let us apply the Haar wavelet to our sample image. In Exercise 2 you will be asked to implement a function which computes DWT2 for the Haar wavelet. After the DWT2, the upper left



(a) Original image (b) $m = 1$ (c) $m = 2$

Figure 12.6: A simple image before and after application of the DWT2. The Haar wavelet was used.



(a) (b) (c) (d)

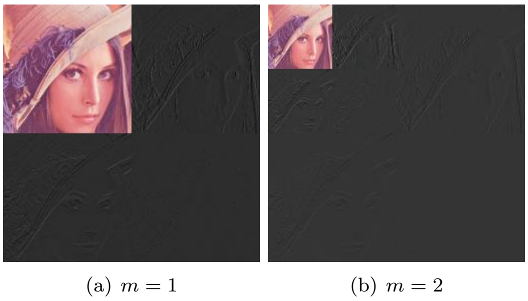
Figure 12.7: The corresponding thumbnail images for the Image of Lena, obtained with a DWT of 1, 2, 3, and 4 levels.

submatrices represent the low-resolution approximations from $V_{m-1} \otimes V_{m-1}$, $V_{m-2} \otimes V_{m-2}$, and so on. We can now use the following code to store the low-resolution approximation for $m = 1$:

```
img = double(imread(’lena.png’,’png’));
[l1,l2]=size(img);
x=DWT2HaarImpl(img,1);
x=x(1:(l1/2),1:(l2/2));
imwrite(uint8(x),’mm1thumbnail.jpg’,’jpg’);
```

In Figure 12.7 the results are shown up to 4 resolutions. In Figure 12.8 we have also shown the entire result after a 1- and 2-stage DWT2 on the image. The first two thumbnail images can be seen as the the upper left corners of the first two images. The other corners represent detail. ♣

Example 12.14 (Detail and low-resolution approximations with the Haar wavelet). In Exercise 3 you will be asked to implement a function `showDWTlowerHaar` which displays the low-resolution approximations to our test image for the Haar wavelet, using functions we implement in the exercises. Let us take a closer look at the images generated. Above we viewed the low-resolution approximation as a smaller image. Let us compare with the image resulting from setting


(a) $m = 1$

(b) $m = 2$

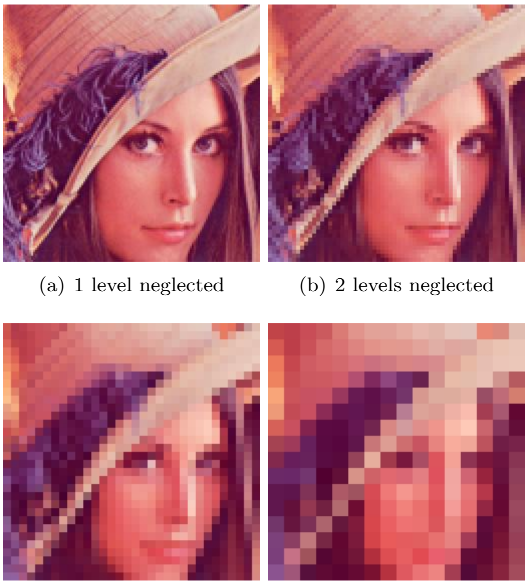
Figure 12.8: The corresponding image resulting from a wavelet transform with the Haar-wavelet.

the wavelet detail coefficients to zero, and viewing the result as an image of the same size. In particular, let us neglect the wavelet coefficients as pictured in Figure 12.4 and Figure 12.5. Since the Haar wavelet has few vanishing moments, we should expect that the lower order resolution approximations from V_0 are worse when m increase. Figure 12.9 confirms this for the lower order resolution approximations. Alternatively, we should see that the higher order detail spaces contain more information. In Exercise 4 you will be asked to implement a function `showDWTlowerdifferenceHaar` which displays the detail components in the image for a given resolution m for the Haar wavelet. The new images when this function is used are shown in Figure 12.10. The black colour indicates values which are close to 0. In other words, most of the coefficients are close to 0, which reflects one of the properties of the wavelet. ♣

Example 12.15 (The Spline 5/3 wavelet and removing bands in the detail spaces)**.** In Exercise 6 you will be asked to implement functions `showDWTlower53` and `showDWTlower97` which display the low-resolution approximations to our image test file `lena.png`, for the Spline 5/3 and CDF 9/7 wavelets (these call functions we also implement in the exercises). With these functions we can display the result for all the wavelets we have considered up to now, in succession, and at a given resolution, with the following code:

```
function showDWTall(m)
  disp('Haar wavelet');
  showDWTlowerHaar(m);
  disp('5/3 wavelet');
  showDWTlower53(m);
  disp('9/7 wavelet');
  showDWTlower97(m);
```

The call to `showDWTlowerHaar` first displays the result, using the Haar wavelet. The code then moves on to the function `showDWTlower53` which uses the Spline



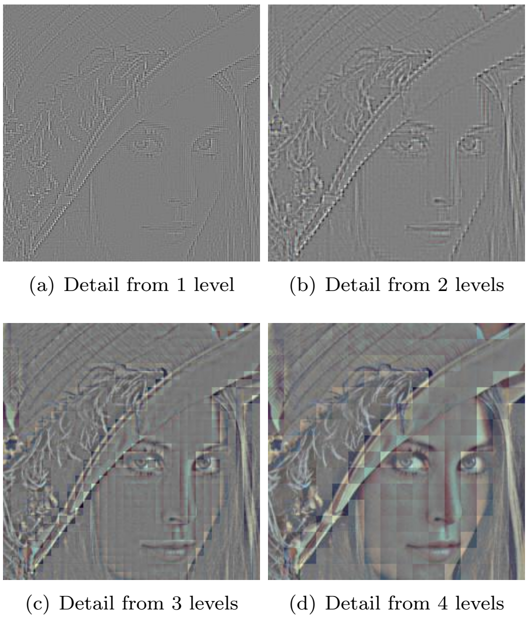
(a) 1 level neglected

(b) 2 levels neglected

(c) 3 levels neglected

(d) 4 levels neglected

Figure 12.9: Image of Lena, with higher levels of detail neglected for the Haar wavelet.



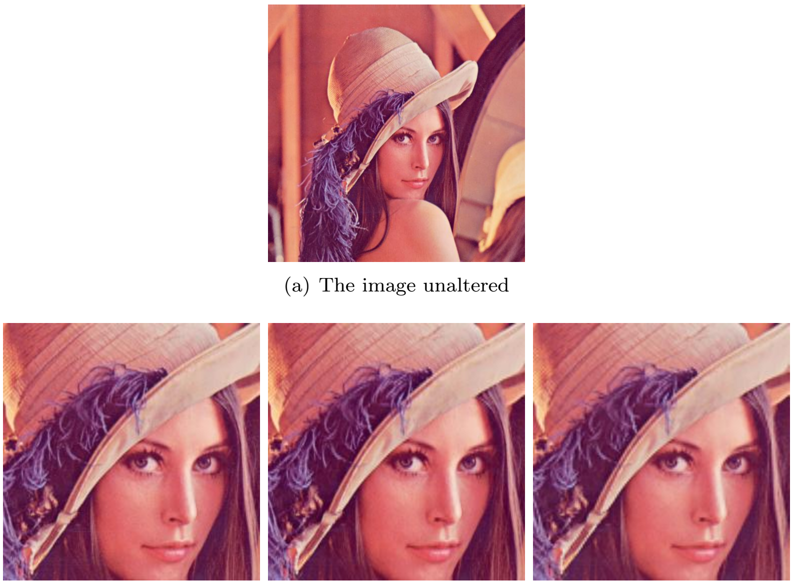
(a) Detail from 1 level

(b) Detail from 2 levels

(c) Detail from 3 levels

(d) Detail from 4 levels

Figure 12.10: The corresponding detail for the images in Figure 12.9, with the Haar wavelet.



(a) The image unaltered

(b) Resulting image after neglecting detail in $W_1^{(1,1)}$, as illustrated in Figure 12.4(b)

(c) Resulting image after neglecting also detail in $W_1^{(1,0)}$, as illustrated in Figure 12.4(c).

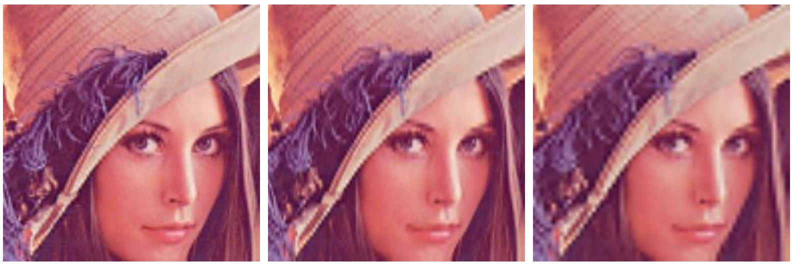
(d) Resulting image after neglecting also detail in $W_1^{(0,1)}$, as illustrated in Figure 12.4(d).

Figure 12.11: Image of Lena, with various bands of detail at the first level neglected. The Spline 5/3 wavelet was used.

5/3 wavelet, and the function `showDWTlower97` which uses the CDF 9/7 wavelet. In the `show`-functions, the image is first read from file, and then code of the following form is called to neglect m levels of detail:

```
img=DWT2Impl53(img,m);
tokeep=img(1:(l1/2^m),1:(l2/2^m))
img=zeros(size(img));
img(1:(l1/2^m),1:(l2/2^m))=tokeep(1:(l1/2^m),1:(l2/2^m));
img=IDWT2Impl53(img,m);
```

Here the Spline 5/3 wavelet was used, and the image had size $l_1 \times l_2$. We can repeat this for various number of levels m, and compare the different images. We can also neglect only parts of the detail, since it at each level is grouped into three bands ($W_m^{(1,1)}$, $W_m^{(1,0)}$, $W_m^{(0,1)}$), contrary to the one-dimensional case. Let us use the Spline 5/3 wavelet. The resulting images when the bands on the first level indicated in Figure 12.4 are removed are shown in Figure 12.11. The resulting images when the bands on the second level indicated in Figure 12.5 are removed are shown in Figure 12.12. The image is seen still to resemble the



(a) Resulting image after also neglecting detail in $W_2^{(1,1)}$, as illustrated in Figure 12.12(a).

(b) Resulting image after also neglecting detail in $W_2^{(1,0)}$, as illustrated in Figure 12.12(b).

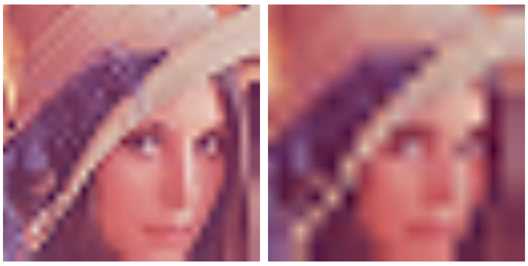
(c) Resulting image after also neglecting detail in $W_2^{(0,1)}$, as illustrated in Figure 12.12(c).

Figure 12.12: Image of Lena, with various bands of detail at the second level neglected. The Spline 5/3 wavelet was used.

original one, even after two levels of wavelets coefficients have been neglected. This in itself is good for compression purposes, since we may achieve compression simply by dropping the given coefficients. However, if we continue to neglect more levels of coefficients, the result will look poorer. In Figure 12.13 we have also shown the resulting image after the third and fourth level of detail have been neglected. Although we still can see details in the image, the quality in the image is definitely poorer. Although the quality is poorer when we neglect levels of wavelet coefficients, all information is kept if we additionally include the detail/bands. In Figure 12.14, we have shown the corresponding detail for Figure 12.11(d), Figure 12.12(c), and Figure 12.13. Clearly, more detail can be seen in the image when more of the detail is included. ♣

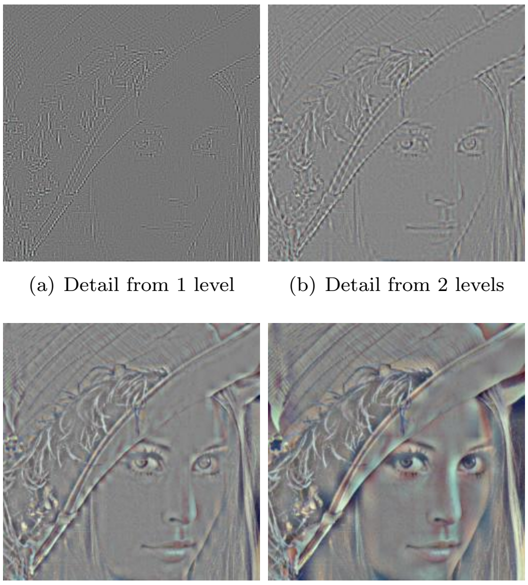
Example 12.16. Let us repeat the previous example for the CDF 9/7 wavelet, using the function `showDWTlower97` you implemented in Exercise 6. We should now see improved images when we discared the detail in the images. Figure 12.15 confirms this for the lower resolution spaces, while Figure 12.16 confirms this for the higher order detail spaces. ♣

As mentioned, the procedure developed in this section for applying a wavelet transform to an image with the help of the tensor product construction, is adopted in the JPEG2000 standard. This lossy (can also be used as lossless) image format was developed by the Joint Photographic Experts Group and published in 2000. After significant processing of the wavelet coefficients, the final coding with JPEG2000 uses an advanced version of arithmetic coding. At the cost of increased encoding and decoding times, JPEG2000 leads to as much as 20 % improvement in compression ratios for medium compression rates, possibly more for high or low compression rates. The artefacts are less visible than in JPEG and appear at higher compression rates. Although a number of



(a) 3 levels neglected (b) 4 levels neglected

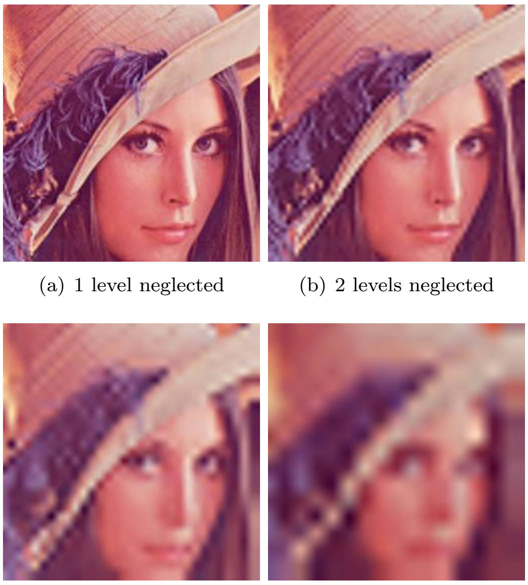
Figure 12.13: Image of Lena, with higher levels of detail neglected. The Spline 5/3 wavelet was used.



(a) Detail from 1 level (b) Detail from 2 levels

(c) Detail from 3 levels (d) Detail from 4 levels

Figure 12.14: The corresponding detail for the image of Lena. The Spline 5/3 wavelet was used.



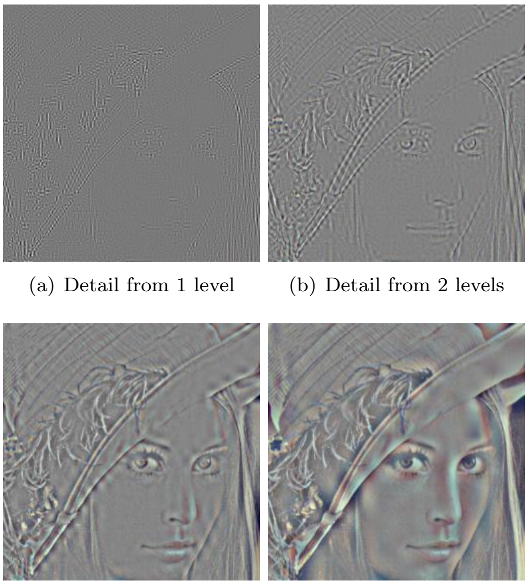
(a) 1 level neglected

(b) 2 levels neglected

(c) 3 levels neglected

(d) 4 levels neglected

Figure 12.15: Image of Lena, with higher levels of detail neglected. The CDF 9/7 wavelet was used.



(a) Detail from 1 level (b) Detail from 2 levels

(c) Detail from 3 levels (d) Detail from 4 levels

Figure 12.16: The corresponding detail for the image of Lena. The CDF 9/7 wavelet was used.

components in JPEG2000 are patented, the patent holders have agreed that the core software should be available free of charge, and JPEG2000 is part of most Linux distributions. However, there appear to be some further, rather obscure, patents that have not been licensed, and this may be the reason why JPEG2000 is not used more. The extension of JPEG2000 files is `.jp2`.

Exercises for Section 12.1

Note that there are three colour components in the test image 'lena.png'. In the following exercises you must therefore run the DWT and IDWT2 on all three components.

1. In this exercise we will use the filters $G_0 = \{\underline{1}, 1\}$, $G_1 = \{1, \underline{-1}\}$.

a. Let X be a matrix which represents the pixel values in an image. Define $\boldsymbol{x} = (1, 0, 1, 0)$ and $\boldsymbol{y} = (0, 1, 0, 1)$. Compute $(G_0 \otimes G_0)(\boldsymbol{x} \otimes \boldsymbol{y})$.

b. For a general image X, describe how the images $(G_0 \otimes G_0)X$, $(G_0 \otimes G_1)X$, $(G_1 \otimes G_0)X$, and $(G_1 \otimes G_1)X$ may look.

c. Assume that we run the following code on an image represented by the matrix `X`:

```
[l1,l2]=size(X);
for s=1:l2
  c=(X(1:2:(l1-1),s)+X(1:l1,s));
  w=(X(1:2:(l1-1),s)-X(1:l1,s));
  X(1:l1,s)=[c w];
end
X=X';

for s=1:l1
  c=(X(1:2:(l2-1),s)+X(1:l2,s));
  w=(X(1:2:(l2-1),s)-X(1:l2,s));
  X(1:l2,s)=[c w];;
end
X=X';
```

Comment the code. Describe what will be shown in the upper left corner of `X` after the code has run. Do the same for the lower left corner of the matrix. What is the connection with the images $(G_0 \otimes G_0)X$, $(G_0 \otimes G_1)X$, $(G_1 \otimes G_0)X$, and $(G_1 \otimes G_1)X$?

2. Implement functions

```
function xnew=DWT2HaarImpl(x,m)
function x=IDWT2HaarImpl(xnew,m)
```

which implements DWT2 and IDWT2 for the Haar-wavelet. Your functions should call the functions `DWTHaarImpl` and `IDWTHaarImpl` from Section 5.3.

3. In this exercise we will experiment with applying an m-level DWT to a sound file.

a. Write a function

```
function showDWTlowerHaar(m)
```

which

1. reads the image file `lena.png`,
2. performs an m-level DWT2 to the image samples using the function `DWT2HaarImpl`,
3. sets all wavelet coefficients representing detail to zero (i.e. keep only wavelet coefficients from $V_0 \otimes V_0$),
4. performs an IDWT2 on the resulting coefficients using the function `IDWT2HaarImpl`,
5. displays the resuting image.

b. Run the function `showDWTlowerHaar` for different values of m. Describe what you see for different m. degraded? Compare with what you saw with the function `showDCThigher` in Exercise 2, where you performed a DCT on the image samples instead, and set DCT coefficients below a given threshold to zero.

c. Do the image samples returned by `showDWTlowerHaar` lie in $[0, 255]$?

4. Repeat Exercise 3, but this time instead keep only wavelet coefficients from the detail spaces. Call the new function `showDWTlowerdifferenceHaar`. What kind of image do you see? Can you recognize the original image in what you see?

5. In this exercise we will implement DTW2 for the CDF 9/7 wavelet.

a. Implement functions

```
function Xnew=DWT2Impl53(X,m)
function X=IDWT2Impl53(Xnew,m)
```

where `DWT2Impl53` performs the m-level DWT2 on the image given by `X`, and `IDWT2Impl53` performs the m-level IDWT2, when the Spline 5/3 wavelet is used. The functions should at each stage call `DWTImpl53` and `IDWTImpl53` with $m = 1$, which you implemented in Exercise 2, and each call to these functions should alter the appropriate upper left submatrix in the coordinate matrix, in the way we have described in this chapter.

b. Implement functions

```
function Xnew=DWT2Impl97(X,m)
function X=IDWT2Impl97(Xnew,m)
```

which does for the CDF 9/7 wavelet, what the functions in (a) did for the Spline 5/3 wavelet. The same comments apply here, but you should instead call the functions `DWTImpl97` and `IDWTImpl97`, which you implemented in Exercise 3.

6. Write functions

```
function showDWTlower53(m)
function showDWTlower97(m)
```

which reimplements the function `showDWTlowerHaar` from Exercise 3 when the Spline 5/3 wavelet and the CDF 9/7 wavelet are used instead. Look at the result using the different wavelets we have encountered and for different m, using the code from Example 12.14. Can you see any difference from the Haar wavelet? If so, which wavelet gives the best image quality?

7. In this exercise we will change the code in Example 12.14 so that it instead only shows the contribution from the detail spaces.

a. Reimplement the functions you made in Exercise 6 so that they instead show the contribution from the detail spaces. Call the new functions `showDWTlowerdifference53` and `showDWTlowerdifference97`.

b. In Exercise 4 we implemented a function `showDWTlowerdifferenceHaar` for looking at the detail/error when the Haar wavelet is used. In the function `showDWTall` from Example 12.14, replace `showDWTlowerHaar`, `showDWTlower53`, and `showDWTlower97`, with `showDWTlowerdifferenceHaar`, `showDWTlowerdifference53`, and `showDWTlowerdifference97`. Describe the images you see for different m. Try to explain why the images seem to get clearer when you increase m.

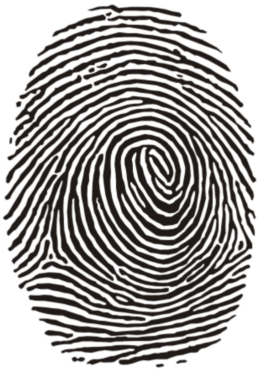


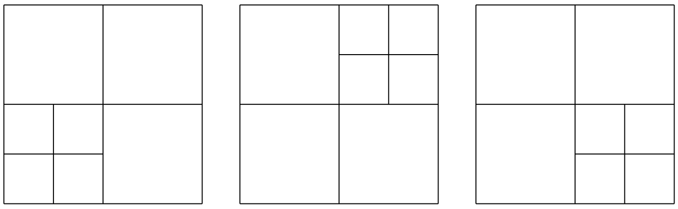
Figure 12.17: A typical fingerprint image.

12.2 An application to the FBI standard for compression of fingerprint images

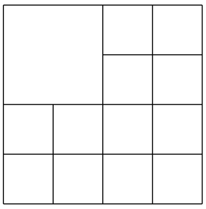
In the beginning of the 1990s, the FBI realized that it had a major problem when it came to their archive of fingerprint images. With more than 200 million fingerprint records, their digital storage exploded in size, so that some compression strategy needed to be employed. Several strategies were tried, for instance the widely adopted JPEG standard. The problem with JPEG had to do with the blocking artefacts, which we saw in Section 11.2. Among other strategies, FBI chose a wavelet-based strategy due to its nice properties. The particular way wavelets are applied in this strategy is called *Wavelet transform/scalar quantization* (WSQ).

Fingerprint images are a very specific type of images, as seen in Figure 12.17. They differ from natural images by having a large number of abrupt changes. One may ask whether other wavelets than the ones we have used up to now are more suitable for compressing such images. After all, the technique of vanishing moments we have used for constructing wavelets are most suitable when the images display some degree of regularity (such as most natural images do). Extensive tests were undertaken to compare different wavelets, and the CDF 9/7 wavelet used by JPEG2000 turned out to perform very well, also for fingerprint images. One advantage with the choice of this wavelet for the FBI standard is that one then can exploit existing wavelet transformations from the JPEG2000 standard.

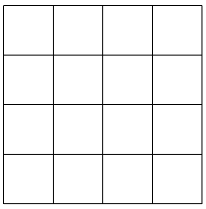
Besides the choice of wavelet, one can also ask other questions in the quest to compress fingerprint images: What number of levels is optimal in the application of the DWT2? And, while the levels in a DWT2 (see Figure 12.3) have an interpretation as change of coordinates, one can apply a DWT2 to the other subbands as well. This can not be interpreted as a change of coordinates, but if we assume that these subbands have the same characteristics as the original image, the DWT2 will also help us with compression when applied to them. Let us illustrate how the FBI standard applies the DWT2 to the different subbands. After one stage of the DWT2, we get the image shown in Figure 12.18(a). If we after this also apply a DWT2 to the bands where highpass filters have been applied, we get the following illustrations of the new subbands:



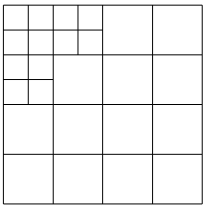
If we apply all these, we get the following subband structure:



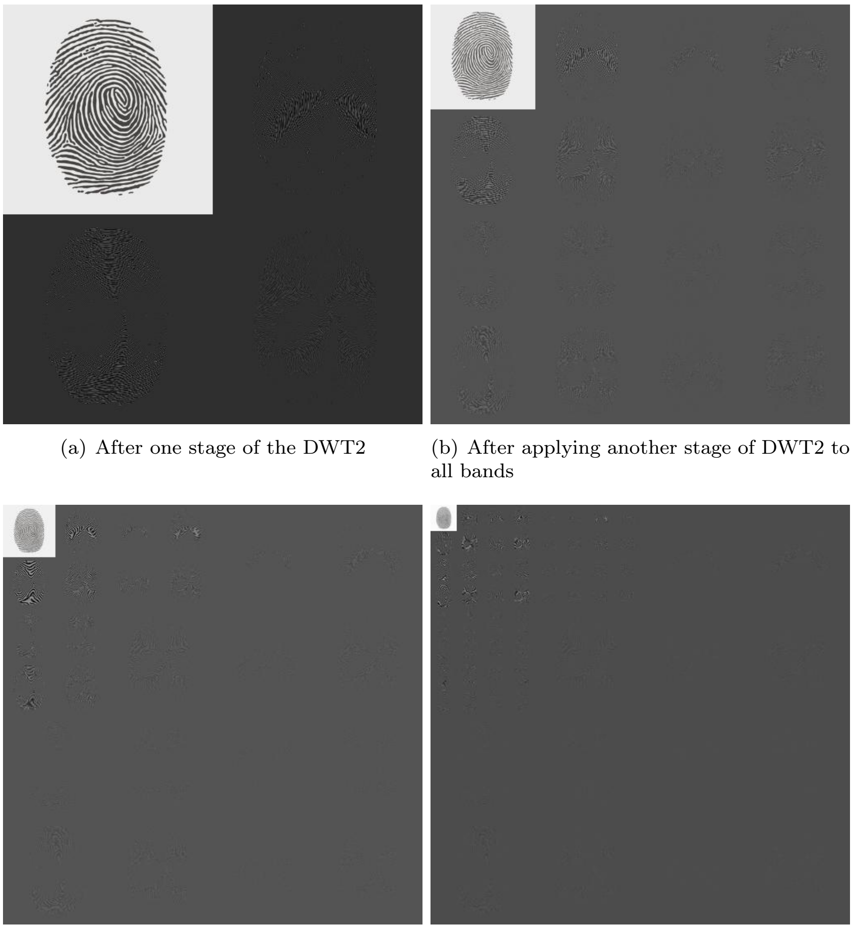
If we also apply the second stage in a DWT2 we arrive at



The resulting image is shown in Figure 12.18(b). Now the FBI standard applies a DWT2 in three of the four resulting subbands, to arrive at the subband structure



The resulting image is shown in Figure 12.18(c). In all the remaining subbands, the DWT2 is now again applied:



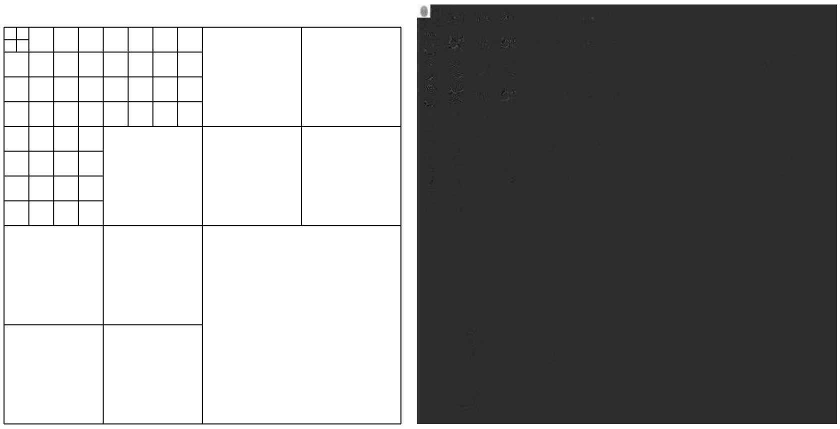
(a) After one stage of the DWT2

(b) After applying another stage of DWT2 to all bands

(c) After one stage of the DWT

(d) After one stage of the DWT

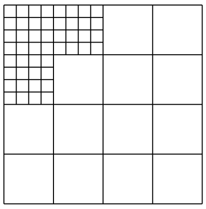
Figure 12.18: The fingerprint image after several DWT's



(a) Subband decomposition

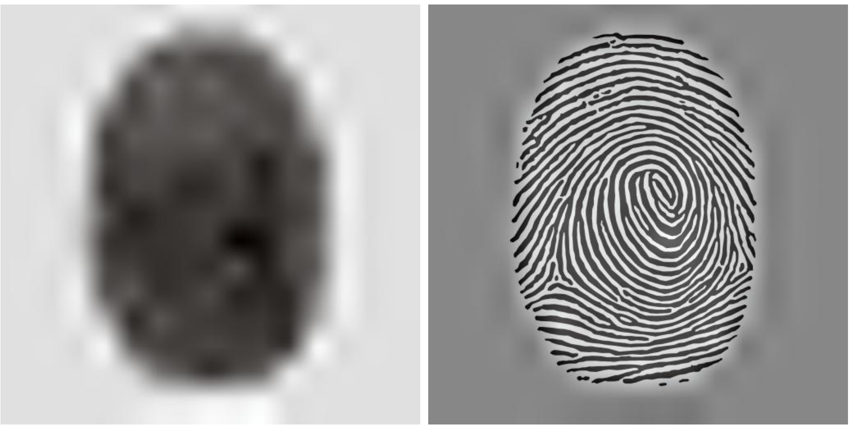
(b) Resulting image

Figure 12.19: The wavelet subband decomposition with the resulting image, as employed by the FBI



The resulting image is shown in Figure 12.18(d). Finally, a DWT2 is again applied, but this time only to the upper left corner. In Figure 12.19 the resulting subband decomposition is shown, together with the resulting image. When establishing the standard for compression of fingerprint images, the FBI chose this subband decomposition. In Figure 12.20 we also show the corresponding low resolution approximation and detail. As can be seen from the subband decomposition, the low-resolution approximation is simply the approximation after a five stage DWT2.

The original JPEG2000 standard did not give the possibility for this type of subband decomposition. This has been added to a later extension of the standard, which makes the two standards more compatible. IN FBI's system, there are also other important parts besides the actual compression strategy, such as *fingerprint pattern matching*: In order to match a fingerprint quickly with the records in the database, several characteristics of the fingerprints are stored, such as the number of lines in the fingerprint, and points where the lines split or join. When the database is indexed with this information, one may not need to decompress all images in the database to perform matching. We will not go into details on this here.



(a) Low resolution approximation

(b) Detail

Figure 12.20: The low-resolution approximation and the detail obtained by the FBI standard for compression of fingerprint images, when applied to our sample fingerprint image.

Exercises for Section 12.2

1. Write code which generates the images shown in figures 12.18, 12.19, and 12.20. Use the functions `DWT2Impl97` and `IDWT2Impl97` from Exercise 12.1. 5 to achieve this.

12.3 Summary

We extended the tensor product construction to functions by defining the tensor product of functions as a function in two variables. We explained with some examples that this made the tensor product formalism useful for approximation of functions in several variables. We extended the wavelet transform to the tensor product setting, so that it too could be applied to images. We also performed several experiments on our test image, such as creating low-resolution images and neglecting wavelet coefficients. We also used different wavelets, such as the Haar wavelet, the Spline 5/3 wavelet, and the CDF 9/7 wavelet. The experiments confirmed what we previously have proved, that wavelets with many vanishing moments are better suited for compression purposes.

The specification of the JPGE2000 standard can be found in [1]. In [14], most details of this theory is covered, in particular details on how the wavelet coefficients are coded (which is not covered here).

One particular application of wavelets in image processing is the compression of fingerprint images. The standard which describes how this should be performed can be found in [5]. In [2], the theory is described.